Eurographics Conference on Visualization (EuroVis) 2017
J. Heer, T. Ropinski and J. van Wijk
(Guest Editors)

Volume 36 (2017), Number 3

Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images

Jorge Poco[1] and Jeffrey Heer[1]

[1]University of Washington

Abstract

*We investigate how to automatically recover visual encodings from a chart image, primarily using inferred text elements. We contribute an end-to-end pipeline which takes a bitmap image as input and returns a visual encoding specification as output. We present a text analysis pipeline which detects text elements in a chart, classifies their role (*e.g.*, chart title, x-axis label, y-axis title, etc.), and recovers the text content using optical character recognition. We also train a Convolutional Neural Network for mark type classification. Using the identified text elements and graphical mark type, we can then infer the encoding specification of an input chart image. We evaluate our techniques on three chart corpora: a set of automatically labeled charts generated using Vega, charts from the Quartz news website, and charts extracted from academic papers. We demonstrate accurate automatic inference of text elements, mark types, and chart specifications across a variety of input chart types.*

1. Introduction

Charts and graphs are commonly used to present quantitative information. They are pervasive in scientific papers, textbooks, economic reports, news articles and webpages. In many cases these visualizations are the only publicly available representation of the underlying data. When well-designed, visualizations leverage human visual processing to convey information efficiently and effectively. But, such depictions are not designed for machine consumption. While people can readily decode data in charts and graphs, machines cannot directly access them. This is unfortunate, as centuries of publications (both printed and online) depict data visually. A vast store of information is locked inside visualizations. The lack of machine readability hinders analysis, reuse and indexing.

Prior work on computational interpretation of chart images typically assumes all text localization and content as given, relying on manual annotation or optical character recognition (OCR) engines that perform poorly on chart images. We present an automated end-to-end system that performs specialized text localization and extraction for chart images, and uses inferred text elements to recover visual encodings. Given a bitmap image as input, our system returns a chart specification as output (sketched in Figure 1).

Our primary contribution is a text analysis pipeline that identifies text elements in a chart image, determines their bounding boxes, recognizes the text content using OCR, and classifies their role in the chart (*e.g.*, chart title, x-axis label, y-axis title, *etc.*). We also train a Convolutional Neural Network that classifies the type of graphical mark used to encode data in a chart (*e.g.*, bars, lines, points, or areas). Together, we leverage the inferred text and chart type information to recover a visual encoding specification in a declarative grammar similar to Vega-Lite [SMWH17] or Tableau's VizQL [STH02]. This chart specification can then be used for indexing, search, or retargeting of the input visualization.

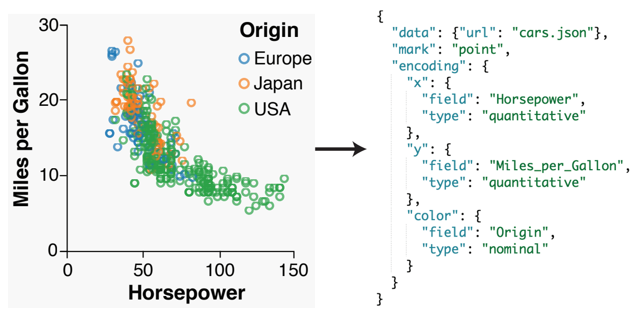

Figure 1: *Inferring a visual specification from a chart image. Given a bitmap image as input, we seek to recover visual encodings for purposes of indexing, search, and retargeting.*

We evaluate our techniques on three chart corpora: a set of automatically annotated charts generated using the Vega language, charts from the Quartz news website, and charts extracted from academic papers in the field of computational linguistics. We demonstrate accurate automatic inference of text elements, content and chart specifications across a variety of input chart types. For text localization and OCR, we achieve minimum F1-scores of 80%; for both text role and mark type classification, we achieve a minimum F1-score of 98%. We share examples of successfully recovered chart specifications and discuss recurring error cases. We conclude with a brief discussion of visualization applications enabled by our automated chart interpretation pipeline.

2. Related Work

Our work draws on prior research in the areas of text localization and computational chart interpretation.

2.1. Text Localization and Optical Character Recognition

Text localization and recognition in documents have been investigated extensively over the past decades. State-of-the-art tools include Microsoft OCR [mic] and Tesseract [Smi07]. Localization for natural images has also been studied (*e.g.*, photos, Google Street View) [HLYW13, NM16]. However, these methods do not achieve acceptable accuracy on chart images. Siegel et al. [SDF16] report an F1-score of 60.3% using Microsoft OCR on a corpus of academic charts. In this paper, we evaluate the same service across three chart corpora and obtain F1-scores ranging from 44% to 61%.

Other text localization attempts for chart images usually follow a bottom-up, region-based approach: find connected components and merge them according to rules such as proximity. Huang and Tan [HT07] separate text and graphical elements using this approach; however, they focus on the graphical elements and manually fix OCR failures. Jayant et al. [JRW*07] and Böschen and Scherp [BS15] include additional steps to infer the text orientation. These techniques assume that geometric relationships among connected components are sufficient to suppress false positives. Based on our experience using a larger data set with charts from multiple sources, we found this assumption to be faulty. Our approach is more robust, for example by using a Convolutional Neural Network to first identify and remove non-text pixels.

2.2. Computational Chart Interpretation

There are two basic component types in a chart image: text and graphical elements. In order to successfully deconstruct a visualization, one must be able to recognize both. However, most prior work focuses on graphical elements. Harper and Agrawala [HA14] present a system for deconstructing D3 [BOH11] visualizations that extracts data, marks and mappings between them. The technique exploits the fact that one can access both SVG elements and data directly in the web browser. We similarly aim to recover visual encoding specifications, but for static chart images, which are both more common and more difficult to interpret.

Multiple systems attempt to classify chart images, identify graphical marks and recover the underlying data. Huang and Tan [HT07] describe a mark extraction method for *vectorized* graphic marks, which can be difficult to obtain from bitmap images. Savva et al. [SKC*11] introduce ReVision, a system to classify bitmap chart images and extract data from pie charts and bar charts. Using ReVision, Kong and Agrawala [KA12] demonstrate how to add interactivity to static pie and bar charts. Siegel et al. [SDF16] and Choudhury et al. [CWG16, RCWG16] present techniques to extract data from line charts using bitmap and vectorial images, respectively. Jung et al.'s ChartSense [JKS*17] uses a semi-automatic approach to extract data from line, pie and bar charts, while Méndez et al.'s iVoLVER [MNV16] relies on manual annotation to extract data and encodings. In all these cases, the researchers treat the text localization and content as given. In contrast, we perform accurate inference of text elements from bitmap input.

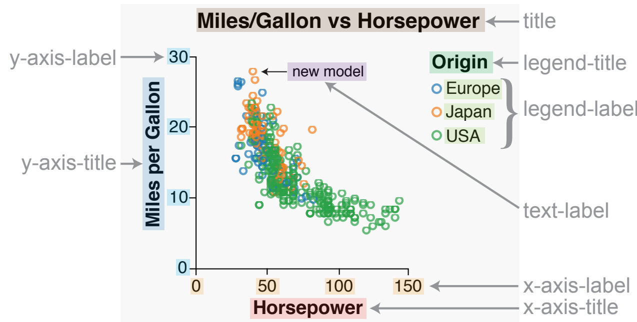

Figure 2: *Text role labels, shown for a Vega-generated scatter plot.*

Text role classification (*e.g.*, identifying axis labels, legend labels, titles, *etc.*) was also explored by Huang and Tan [HT07]. They use spatial relationships between text and graphical elements to generate feature vectors, but again require vectorization of the chart. In DiagramFlyer, Chen et al. [CCA15] use feature vectors based only on text bounding boxes. Choudhury et al. [CWG16] use text bounding boxes and also text content. We follow a similar approach, but do not incorporate text content, as it may propagate OCR errors. Instead, we exploit structural information such as the alignment and grouping of text boxes. Siegel et al. [SDF16] propose a two step technique to classify text. First, they scan for vertically or horizontally aligned boxes (with numeric content) to infer axis labels. Next, legends are classified using a feature vector with geometric information. The first step makes several assumptions, such as axes anchored on the bottom-left side, restricting its use.

Mark type classification has been studied in multiple prior projects. In ReVision, Savva et al. [SKC*11] sample image patches to learn visual "bag of words" features for a Support Vector Machine (SVM) classifier. Both FigureSeer [SDF16] and ChartSense [JKS*17] use Convolutional Neural Networks (CNNs) for classification. We similarly use a CNN and demonstrate superior performance to ReVision and ChartSense.

To the best of our knowledge, no prior work has produced an end-to-end system that accurately extracts text information from bitmap chart images to recover visual encoding specifications. The closest works (FigureSeer [SDF16] and ReVision [SKC*11]) assume that text information is given *a priori*.

3. Data Collection and Generation

We use training and test data drawn from three corpora: a combination of automatically generated charts (using Vega [SRHH16]) and manually annotated charts from 3rd party sources (Quartz news and academic papers in computational linguistics). Automatic generation using Vega allows us to create an arbitrarily large data set that systematically varies the visual encodings. Given Vega's underlying use of D3 [BOH11], the resulting images mirror many charts on the web. The other corpora consist of real-world charts used online (Quartz) and in print (academic papers).

For each image, our data includes the *bounding boxes* and transcribed *content* of all text elements. Each element is labeled with its *role*, drawn from the label set in Figure 2. Figure 3 shows examples from each corpus; Table 1 tallies images by corpus and type.

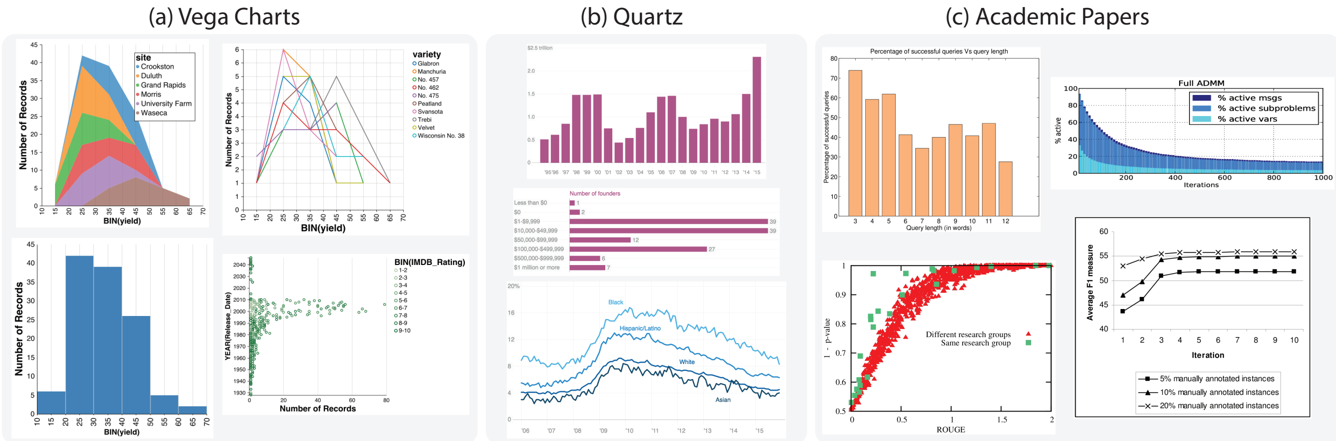

Figure 3: *Examples chart images from the (a) Vega, (b) Quartz, and (c) Academic Papers corpora.*

	Vega Charts	Quartz	Academic Papers
Area Charts	477	0	1
Bar Charts	1,358	191	57
Line Charts	360	283	248
Scatter Plots	2,123	1	26
Total	4,318	475	332

Table 1: *Chart Corpus Statistics.*

In this work, we make some simplifying assumptions to constrain the chart types considered. First, we assume there are *no composed figures* with multiple plots; prior work by Lee and Howe [LH15] focuses on the specific problem of segmenting composed figures. Second, we assume that *plots are not layered*. For instance, we do not yet support dual axis charts, or those with multiple mark types (*e.g.*, Pareto charts). Finally, we assume a *Cartesian coordinate space*, which still affords a variety of chart types such as bar charts, line charts, scatter plots and area charts.

3.1. Vega Charts (VEG)

Our first corpus consists of chart images generated using the Vega visualization grammar [SRHH16] and associated tools. We developed a system that takes a data table as input and outputs a set of chart images and annotations. We use the Compass recommendation engine (part of Voyager [WMA*16]) to generate charts visualizing combinations of 1-3 data variables, filtered according to perceptual expressiveness criteria. We applied this process to 11 data sets, resulting in 5,542 Vega specifications. To increase the variability of our chart images, we randomly selected values for fonts, font size, presence of grid lines, and legend & axis positions from a curated set of options. We then reviewed the results to remove problematic instances — such as charts with interior legends that occlude data and aggregate plots with only a single data point — leading to 4,318 charts. Finally, for each chart we analyzed Vega's scenegraph to automatically extract text bounding box and role labels. Figure 3(a) shows examples from this corpus. A full, replicable description of our generation procedure and data sets is included in supplemental material (Appendix A).

3.2. Quartz (QTZ)

Our second corpus contains chart images from the news website Quartz (http://qz.com/). All visualizations in Quartz's articles are available in SVG format from the Atlas search engine (https://www.theatlas.com/). We implemented a crawler to retrieve all charts from Atlas. We collected 500 SVG files and excluded 25 (5%) that did not satisfy our assumptions. Figure 3(b) shows images from this corpus; most of these images are line and bar charts (Table 1). We then processed the SVG files to extract text elements. Though we can trivially parse the SVG structure to extract text, we must still refine the bounding boxes and assign a role to each element. For example, if an axis title is composed of two words specified as separate text elements, we must merge them. To accelerate manual labeling, we created a graphical interface to display the charts and overlay text bounding boxes. The interface supports operations such as adding, deleting, merging and resizing boxes, as well as assigning role labels to each (Appendix B).

3.3. Academic Paper Figures (ACA)

Our third corpus is composed of chart images extracted from scientific documents. In particular, we downloaded 21,142 papers from the ACL Anthology repository, which includes a large number of published charts from a single, coherent domain. To extract figures from source PDF files, we used the `pdffigures` [CD16] utility, which outputs a JSON document with figure location, text location, text rotation, and text content information. We extracted 26,134 images and selected an initial random subset of 500 images for parity with QTZ. However, we found that more than 50% of the figures lay outside our current scope (*e.g.*, general diagrams, workflows, syntactic trees, tables). We removed these figures and randomly selected more figures; after two iterations, we had a set of 350 figures. We decided to work with this subset once we stopped seeing new chart designs. As with the Quartz corpus, we used our annotation interface to manually refine bounding boxes and assign role labels.

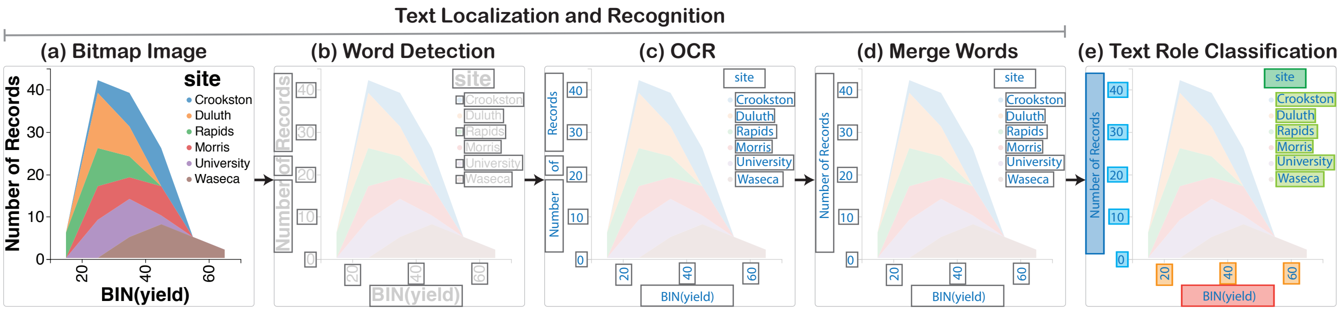

Figure 4: *Text analysis pipeline. (a) Initial bitmap image. (b) Identify bounding boxes for words; note that some boxes are incorrect. (c) Use OCR to recover text for each box. (d) Merge words in lines. (e) Use geometric features of bounding boxes to classify their role in the chart.*

In this step, we discovered charts with very low resolution and others with Chinese text. We removed these, leaving us with 332 total annotated images. Figure 3(c) shows examples from this corpus.

4. Text Localization and Recognition

Figure 4 summarizes our text analysis pipeline, which given an input image outputs a set of labeled text elements. In this section, we present techniques for locating and extracting text content (steps b, c and d in Figure 4). We first detect candidate words to produce a set of bounding boxes. We then individually apply OCR to these boxes to extract text content, and use OCR confidence values to filter erroneous non-text boxes. Finally, we merge adjacent boxes to consolidate multi-word phrases and titles.

Unlike prior work that applies a strong filter to candidate words in the initial steps, we use multiple weak filters across text pixel prediction, word detection, and OCR to minimize errors that might propagate through the pipeline. We only discard a box once we are highly confident that it is not text.

4.1. Word Detection

To detect candidate words in chart images, we first identify likely text pixels and remove non-text pixels. This initial pass removes elements that can confuse standard region-based text localization methods, which we then apply. We also leverage the assumption that letters in words should have the same color.

Preprocessing: In order to standardize chart images, we perform some preprocessing steps. First, we resize images, preserving aspect ratio, so that they fit within a rectangle of 1200×1200 pixels. Then, we binarize the image using a global threshold approach. The optimum threshold is calculated using Otsu's method [Ots79], which assumes that pixels belong to two classes and attempts to maximize the inter-class variance. Figure 5(a) shows the binary image of the area chart shown in Figure 4(a).

Text Pixel Classification: In this stage, we remove pixels that do not correspond to text. We use Darknet [Red16], a Convolutional Neural Network (CNN) framework written in C and CUDA. Darknet includes a network for predicting text pixels, trained using 500k figures from scientific papers (arXiv, Pubmed and ACL Anthology). The network takes as input a 256×256 pixel image and outputs a 64×64 heatmap of text pixel probabilities [Mor17]. To remove text, we resize the output heatmap back to the original size,

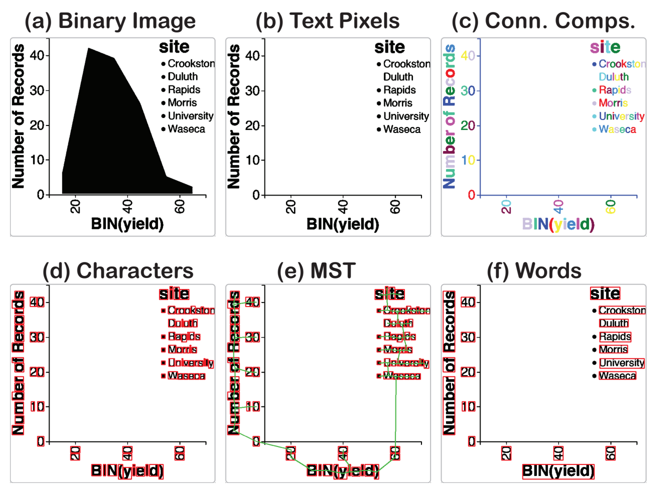

Figure 5: *Word detection process. (a) Initial binarized image. (b) Clean image with non-text pixels removed. (c) Connected components. (d) Filter region candidates using geometric properties. (e) Minimal Spanning Tree using boxes as vertices and distances as edges weights. (f) Detected words after pruning edges in the MST.*

binarize the image using a threshold of 0.6, and use it to mask the original image via bitwise AND. In some cases, parts of letters are removed due to the low resolution of the heatmap. To solve this issue, we run a flood fill algorithm on the filtered image using the binary image as a mask. In this way, we can complete those missing letters. Figure 5(b) shows the results of removing non-text pixels.

Text Localization: Once non-text pixels have been removed, we employ a region-based approach to localize text. We proceed in a bottom-up fashion, first detecting small components and consecutively merging them to form words. Figure 5 shows the main steps.

To find candidate *characters*, we run the connected components algorithm. Figure 5(c) shows each connected region in a random color; at this point the axis lines (shown in blue) are included as candidate regions. We then filter regions based on their geometric properties. Given a bounding box, we retain a region if its aspect ratio $\in [1/15, 15]$ and its area $\in [4, 1000]$. These criteria were determined in accordance with prior work and our own experiments. In Figure 5(d), this step removes the axis lines but we still have the legend symbols (colored circles) as candidates.

To then identify candidate *words*, we compute a minimal spanning tree (MST) by applying Kruskal's algorithm [Kru56] to a fully connected graph in which the nodes are the candidate characters and the edge weights are the Euclidean distances between bounding box centers. Figure 5(e) shows a resulting MST. The MST captures proximity information between characters, but requires additional pruning to isolate words. To do so, we first calculate the most common element (l) in a *length set* containing the heights and widths of all candidate characters. The intuition is that l represents the font height, which we can use to determine which MST edges to filter. We discard edges with length $> 2l$. We also filter edges according to the *alignment* of the connected bounding boxes. An edge is preserved if the bounding boxes are vertically or horizontally aligned. We consider two boxes vertically aligned if $overlap_{horiz}(b_i, b_j) > min(w_i, w_j)/2$; horizontal alignment is defined similarly. Finally, we assume that all characters in a word should have the same *color*. We calculate the mean CIE LAB color (c_i) for each pixel in a connected component, and discard an MST edge if $dist(c_i, c_j) > 20$ according to CIEDE2000 color difference. In Figure 5(f), this stage prunes MST edges connecting legend symbols (colored circles) and legend labels.

4.2. Optical Character Recognition

Next, we perform optical character recognition for each candidate word, using the open source Tesseract [Smi07] engine (Figure 4(c)). An important aspect of OCR performance is the quality of the input image. In order to archive the best performance, it is recommended to have high resolution images with horizontal text, high contrast and little noise. However, text in chart images may be small and have varied orientations.

First, we crop a word rectangle from the binary image and scale it by a factor of 3. We run Tesseract multiple times with the image rotated at different angles, including 0°, 90°, and −90°. Tesseract returns both a text string and a confidence score. We select the string with the highest confidence. In case of a tie, we count the number of connected components in the cropped image, excluding small regions that represent diacritics or dot marks over a lowercase 'i' or 'j', and select the text string with the number of characters closest to the number of components. The orientation information is used in the next stage to merge related words. We also filter word candidates with confidence $< 25\%$. For example, in Figure 4(c) we remove the legend symbols (colored circles).

After inspection of OCR results, we found some common errors that can be fixed by some simple heuristics. The vowel 'O' or 'o' was the output in multiples cases instead of the number '0'. Given the nature of our charts, it is more likely to be a number than the vowel when appearing as a single-character string. We found a similar confusion among the letter 'l' and the number '1'.

4.3. Word Merging

Text elements in charts may contain multiple words; for example, the variable "Miles per Gallon" in the y-axis title of Figure 4. In this stage, we seek to detect such cases and merge associated words to produce a final set of bounding boxes and extracted text strings. We merge words if they are *aligned* and have the same *orientation*. Consider Figure 4(b), where the three words ('Number', 'of', 'Records') are vertically oriented. Let b_i and b_j denote the bounding boxes for 'Number' and 'of' respectively. In this case, we are testing vertically aligned boxes, but the same approach applies for the horizontal case. We then check the following conditions:

- b_i and b_j have the same orientation if $angle(b_i) = angle(b_j)$
- b_i and b_j are near each other if $dist_{ext}(b_i, b_j) < min(w_i, w_j)$, where $dist_{ext}$ is the external separation between b_i and b_j.
- b_i and b_j align if $overlap_{vert}(b_i, b_j) > min(w_i, w_j)/2$

4.4. Validation

We evaluate our text localization and extraction methods against our three chart corpora. We first report the accuracy of bounding box identification. We then evaluate OCR performance against estimated and ground truth bounding boxes, using both exact matching and edit distance to compare text strings.

Text Localization Performance: To validate our approach we use precision, recall and F1-score metrics as defined in [Luc05]. These metrics are widely used in text localization competitions (*e.g.*, ICDAR 2003 and 2005).

For a box b we find the best match $\hat{b}$ in a set of boxes B using:

$$\hat{b} = m(b, B) = \max m_a(b, b') \mid b' \in B| \quad (1)$$

Where $m_a(b_i, b_j) = \frac{2\, area(b_i \cap b_j)}{area(b_i) + area(b_j)}$. Note that m_a is 1 for equal boxes and 0 for boxes without intersection.

Then, we apply this definition of *best matching* to our bounding boxes T (ground truth boxes) and E (estimated boxes) in a chart image. We define precision, recall and F1-score as follows:

$$p = \frac{\sum_{b_e \in E} m(b_e, T)}{|E|},\ r = \frac{\sum_{b_t \in T} m(b_t, E)}{|T|},\ F1 = 2\frac{p \cdot r}{p + r} \quad (2)$$

Table 2 shows the average value of F1 over all images. The ACA corpus has the lowest F1-score (80%) due to a higher variation in visual styles, intersection of text content with other elements in the chart, and text with very small font sizes.

As it is unlikely to infer boxes that *exactly* match the ground truth, the F1-score can vary from 80%-100% even if all text is correctly localized. For example, when simply shrinking or expanding the ground truth boxes by 2 pixels, the F1-scores for the QTZ corpus are 82% and 85%, respectively. Our estimated boxes are very tight to the letters. We calculated F1-scores while expanding the boxes from 1 to 5 pixels and found a peak at 3 pixels. The results in Table 2 were computed using this padding.

	Ours	Microsoft OCR [mic]
VEG	**88%**	44%
QTZ	**86%**	68%
ACA	**80%**	61%

Table 2: *Text localization performance (F1-scores).*

	Ground Truth Boxes		Estimated Boxes	
	Exact	Edit	Exact	Edit
VEG	95%	98%	87%	93%
QTZ	99%	100%	92%	96%
ACA	93%	98%	82%	88%

Table 3: *OCR performance. F1-scores for ground truth and estimated bounding boxes, using exact matching or edit distance.*

In addition, we compared our results with the state-of-the-art services provided by Microsoft OCR [mic]. As we can see in Table 2, we obtain superior F1-scores for all three chart corpora. We analyzed the output of Microsoft OCR and noted problems with text in multiple orientations. In particular, this leads to a low F1-score for VEG, as in many cases the x-axis labels are vertically oriented. We also noted problems with single-character text strings.

OCR Performance: To evaluate OCR performance, we use two similarity functions for text strings. The first, $sim_{exact}(s_i, s_i)$, simply returns 1 if the strings are equal and 0 if they are not. The second function is $sim_{edit}(s_i, s_i) = 1 - \text{lev}(s_i, s_j)/\max(|s_i|, |s_j|)$, where $\text{lev}(s_i, s_j)$ is the Levenshtein edit distance between two strings and the denominator is a normalization factor.

Similar to before, we define precision, recall and F1-score as:

$$p = \frac{\sum_{b_e \in E} \text{sim}(b_e, \hat{b_e})}{|E|}, \; r = \frac{\sum_{b_t \in T} \text{sim}(b_t, \hat{b_t})}{|T|}, \; F1 = 2\frac{p \cdot r}{p + r} \quad (3)$$

Here, sim(.) can be any of the two similarity functions and $\hat{b_e}$ and $\hat{b_t}$ are the *best matchings* to b_e and b_t respectively. Table 3 shows F1-scores for both similarity functions. OCR performance for ground truth bounding boxes is at least 93% using sim_{exact} and 98% using sim_{edit}.

In order to evaluate OCR performance for estimated boxes, we use the *best matching* rule (Equation 1) and consider two boxes the same if $m_a(b_i, b_j) > 0.5$. This rule is commonly used to evaluate text localization techniques. If there is not a perfect matching, we penalize it by 1, the maximum value returned by the similarity functions. The last two columns in Table 3 report F1-scores for our estimated boxes. In the case of sim_{exact} the lowest value is 82% in the ACA corpus. Using sim_{edit} our lowest F1-score is 88%.

5. Text Role Classification

Given a set of text elements (*i.e.*, bounding boxes and text content), we want to classify each according to its role in the chart. We consider the 8 text types illustrated in Figure 2. This classification may be performed for text elements inferred using the methods of the previous section, or for text elements directly extracted from vector graphics (*e.g.*, SVG or PDF). Our technique uses geometric properties of the bounding boxes to define feature vectors which we then classify using Support Vector Machines (SVM) [CV95]. In contrast to prior work, we include structural information based on a global analysis of bounding boxes. We also perform post-processing to further improve the results.

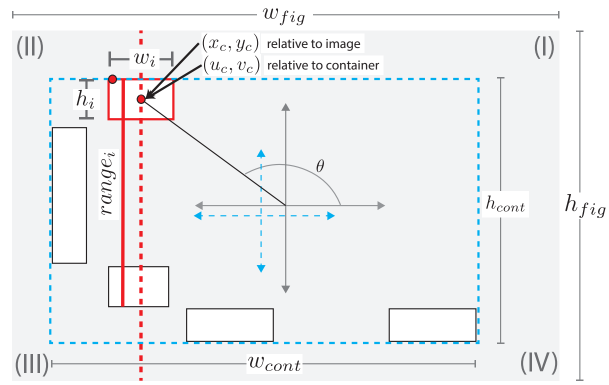

Figure 6: *Bounding box features using geometric information, showing the information to compute features for the red box.*

5.1. Feature Engineering & Classification

We use the geometric properties of the bounding boxes to define a feature vector of 14 dimensions. These features are grouped in three categories. The first category consists of 5 features concerning the position and aspect ratio of a bounding box. The idea is that some text types should be located in specific regions of the chart. For instance, an x-axis title will normally be located near the bottom or top of a chart. The features are the *normalized center coordinates* (x_c/w_{fig}, y_c/h_{fig}), *aspect ratio* (w_i/h_i), *angle* with respect to the image center (θ), and *quadrant* ($quad \in \{I, II, III, IV\}$). Figure 6 illustrates these concepts.

The second category contains 3 features that also encode the spatial information relative to the global bounding box for all elements (the blue rectangle in Figure 6). The features are the *normalized container size* (h_{cont}/h_{fig}, w_{cont}/w_{fig}), *normalized center coordinates* relative to the container (u_c, v_c), and the angle (rad) with respect to the container center (blue arrows in Figure 6).

The third category has 4 features that exploit structural information. For example, if box b_i is a y-axis label, then it should lie on a vertical line that intersects other boxes (*i.e.*, other y-axis labels). We define the *vertical score* ($score_v = n_i/n$), where n is the number of boxes in the image and n_i is the number of boxes that intersect b_i vertically. In Figure 6, the red box intersects with one box below it, so has $score_v = 1/5$. The same reasoning applies to the horizontal case ($score_h$). However, labels in legends might also be aligned vertically. So, $score_v$ may not distinguish between y-axis labels and legend labels. Nevertheless, the range of y-coordinates in y-axis labels might be greater than range of y-coordinates in legend labels, as y-axis labels typically extend across the height of the image. We define the *vertical span* ($span_v = range_i/h_{fig}$), where $range_i$ is the vertical range of the boxes that intersect with b_i. For instance, in Figure 6 the range of the red box is the length of the red solid line. Again, similar reasoning applies to the horizontal case ($span_h$).

Using these features, we train a multi-class SVM [CV95] using a radial basis function kernel [MMR*01]. Each class is assigned a weight inversely proportional to its frequency in the input data. We use parameter values $\gamma = 0.1$ and $C = 100$, found via cross-validation on the training data.

	Number of Boxes			F1-score		
	VEG	QTZ	ACA	VEG	QTZ	ACA
x-axis-label	35,361	4,092	2,473	100%	100%	100%
x-axis-title	4,371	43	270	99%	94%	97%
y-axis-label	45,169	2,982	2,473	100%	100%	100%
y-axis-title	4,371	—	280	100%	—	97%
legend-label	8,250	308	960	98%	99%	97%
legend-title	1,423	—	2	95%	—	0%
text-label	—	565	147	—	99%	79%
title	—	—	50	—	—	93%
Average	98,945	7,990	6,558	100%	100%	98%

Table 4: *Text role classifier performance. Columns 2-4 show the number of boxes in our three chart corpora by text type. The latter columns present F1-scores using 5-fold cross validation.*

5.2. Validation

We evaluate our classifier using ground truth bounding boxes from our three chart corpora. Columns 2-4 in Table 4 show the number of boxes for each text type. Given the unbalanced number of boxes among corpora, we decided to evaluate each corpus independently. We performed 5-fold cross validation, using stratified sampling to ensure each class was represented with approximately equal proportions in each fold. As shown in Table 4, our classifier performs well, with average F1-scores ranging from 98% to 100%. Note the 0% value for `legend-title` in the `ACA` corpus; this arises because there are only two instances of legend titles in that corpus.

Post-processing: Performing error analysis, we found that the main confusion was `y-axis label` vs. `legend-label` (100 boxes), followed by `legend-title` vs. `x-axis-title` (32 boxes), and then `legend-title` vs. `legend-label` (31 boxes). To improve our results, we post-process the SVM output. Our heuristic rules reduce the numbers of incorrect charts in `VEG` from 54 to 4, in `QTZ` from 5 to 1, and in `ACA` from 16 to 4.

We first detect the legend orientation (vertical or horizontal) using `legend-labels`. If not all boxes align in one direction, we choose the orientation with the most aligned boxes. We then select boxes with role `legend-label` that are not aligned and check if they align with `y-axis labels`. If so, we change their role to `y-axis-label`. Next, we verify that all `y-axis-labels` vertically align. If any of these boxes do not align, we check if they align with `legend-labels` or `x-axis-labels`, and change their role accordingly. If they do not align with any of these, we assign the role `text-label`. Finally, if there are multiple `legend-titles`, we check if they are on the top on the `legend-labels` and select the one on the top. For the others, we check if they are aligned with the `legend-labels`, `x-axis-labels`, or `y-axis-labels` and reassign their text type accordingly. If not aligned, we assign role `text-label`.

6. Mark Type Classification

Text elements for axes and legends provide valuable information about how data is encoded in a chart. However, to infer a full chart specification, we must also identify the type of graphical marks

	Ours	ReVision	ChartSense	# Images
Area Graphs	**95%**	88%	67%	90
Bar Graphs	**97%**	78%	93%	169
Curve Plots	**94%**	73%	78%	318
Maps	**96%**	84%	88%	249
Pareto Charts	**89%**	85%	85%	168
Pie Charts	**98%**	79%	92%	210
Radar Plots	**93%**	88%	86%	137
Scatter Plots	**92%**	79%	86%	372
Tables	**98%**	86%	94%	263
Venn Diagrams	**91%**	75%	67%	108
Average / Total	**94%**	80%	90%	2,084

Table 5: *Mark type classification accuracies for our classifier, ReVision [SKC*11], and ChartSense [JKS*17].*

	Ours (All)	Ours (500)
Area Marks	98%	98%
Bar Marks	99%	98%
Line Marks	95%	98%
Plotting Symbols	99%	98%
Other	97%	100%
Average / Total	98%	98%

Table 6: *Mark type classification performance. F1-scores using 5,125 and 2,500 chart images, grouped in 5 categories.*

used to encode data. Here we present a classifier that takes a bitmap chart image as input and estimates the mark type used. Our primary classifier is trained to recognize five mark types: bars, lines, areas, scatter plot symbols, and "other". This last class is included to recognize chart types that are not supported by our larger pipeline. To evaluate our approach, we also train a 10-class model and compare our results with ReVision [SKC*11] and ChartSense [JKS*17].

6.1. Method

To perform mark type classification we use Convolutional Neural Networks (CNNs), which achieve state-of-the-art performance for many computer vision tasks. However, CNNs require large amounts of data and computation to train, while our chart corpora contain a total of only 5,125 images. One strategy to address this mismatch is to *fine-tune* a pre-trained network via back-propagation. We use the Caffe [JSD*14] implementation of AlexNet [KSH12]. This model was trained on the ImageNet dataset, which contains millions of images across 1,000 classes. This same fine-tuning approach is used by the FigureSeer [SDF16] system. ChartSense [JKS*17] also uses a CNN, but with a different architecture (GoogLeNet [SLJ*15]) trained from scratch.

6.2. Validation

We evaluate our classification approach using two different data sets. To compare with prior work, we first trained a model using the ReVision [SKC*11] corpus of 2,084 images across 10 categories (Table 5). We split the data into 75% and 25% for training and testing, respectively. Table 5 shows multi-class classification

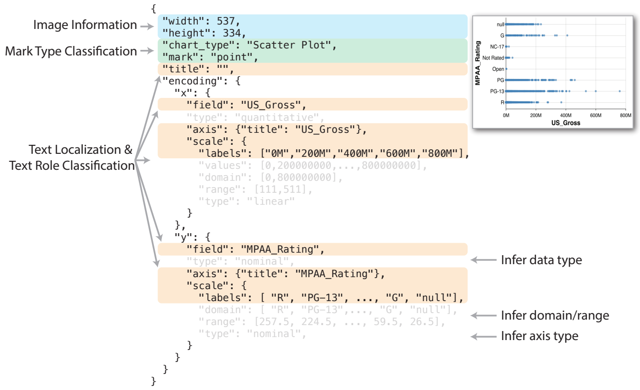

Figure 7: *Specification induction. A chart specification is recovered for an input scatter plot, using information directly from extracted text and mark type information (colored regions). The grey regions show additional sections that must be inferred (*i.e.*, data type, axis type, and scale domain & range).*

accuracy per chart type as well as the average accuracy. Our classifier exhibits superior performance, with an average classification accuracy of 94% compared to ReVision's 80% and ChartSense's 90%. We did not compare with Siegel et al. [SDF16], as they use the same fine-tuning strategy. They report an accuracy of 84% using a 7-class model on a different chart image corpus.

For our second data set, we merged the images from our three chart corpora (5,125) with the ReVision dataset (2,084) for a total of 7,209 images. We grouped the charts into 5 categories: line marks (1209), bar marks (1775), area marks (586), plotting symbols (2522), and all "others" (1135). We again split the data into 75% and 25% for training and testing, respectively. The second column in Table 6 shows F1-scores for this model. We achieve highly accurate classification with an F1-score of 98%.

To assess the effects of unbalanced groups, we randomly filtered our second dataset down to 500 images per category, and trained another model. The last column in Table 6 shows the F1-scores for this model. We again achieve an average F1-score of 98%.

7. Specification Induction

Given mark type information, text content, and bounding boxes, we can produce a visual encoding specification for a chart image. As shown in Figure 7, we use the results of earlier pipeline stages to directly specify components such as the chart width & height, mark type, data field names, and chart & axis titles. However, other elements — including data types for each encoded field and axis specifications (domain, range, scale type) — must still be inferred.

Inferring Data Types: We use axis or legend label text to classify data types as either *quantitative* or *nominal* (here encompassing both categorical and ordinal fields). To check for quantitative data, we first attempt to parse the label text as numbers (*e.g.*, '-100', '1000', '4.5', '1e-10'). However, in many cases this is insufficient due to additional modifiers, such as characters indicating units. If naïve parsing fails, we further check if the text uses the International System of Units (*e.g.*, '1M', '1k', '10M') and parse these as floating point numbers. For instance, '1M' is converted to $1,000,000$. Figure 7 includes x-axis labels that use SI notation. If this fails, we attempt to parse the test using a library of common units, including percentages ('10%') and bytes ('10MB'). If any of these stages succeed, we assign the *quantitative* data type, otherwise we assign the *nominal* type. Currently date-time types are treated as *nominal*; in future work we plan to additionally add string parsers to recognize *temporal* data as a dedicated type.

Inferring Axis Domain & Range: For quantitative variables, we let $\{x_1, x_2, \ldots, x_n\}$ denote the center x-coordinates of the label bounding boxes in the x-axis and $\{v_1, v_2, \ldots v_n\}$ denote the values of the text contents on each box. We then infer the x-axis domain as $[v_1, v_n]$ and axis range as $[x_1, x_n]$. (See x-axis in Figure 7). The same procedure (using y-coordinates) is applicable for the y-axis.

For nominal variables, we let $\{x_1, x_2, \ldots, x_n\}$ denote the center x-coordinates of the label bounding boxes in the x-axis and $\{t_1, t_2, \ldots t_n\}$ denote text content on each box respectively. Then we infer the x-axis domain as $[t_1, t_2, \ldots, t_n]$ and the axis range as $[x_1, x_2, \ldots, x_n]$ (and similarly for the y-axis, as in Figure 7).

Inferring Axis Scale Type: Once we know that an axis encodes a *quantitative* field, we can infer the axis scale type (*linear*, *logarithmic*, *power* or *sqrt*) using the *domain* and *range* values. We use non-linear least squares to fit multiple functions to the data (*i.e.*, *linear*, *log*, *power* and *sqrt* functions) and pick the model with the minimum mean squared error.

8. Example Results

Figure 8 shows successful outputs of our system: given only bitmap images as input, we are able to successfully recover full visual encoding specifications for a variety of mark types, data field types, and axis scales. Figure 8(e) includes a log-transformed x-axis scale. Note that inferred specifications may have empty fields; for example Figures 8(c & d) lack axis titles.

We also noted some recurring errors in our inferred chart specifications, which can arise due to failures of text localization, OCR, or role classification. Incorrectly merged labels due to tight spacing occur in the `QTZ` and `ACA` corpora (Figure 9(a)), notably with x-axis labels. The minimal separation between labels makes it difficult for our techniques to isolate them. One approach to address this error may be to process the text content of unusually long boxes, detect any repetitive patterns, and separate the labels accordingly.

Another error arises due to the use of text characters as plotting symbols (Figure 9(b)), which occurs in all three corpora. Our neural network text pixel classifier in the word detection stage reduces these errors relative to standard region-based localization schemes, but we still get a few symbols confused with text labels.

In addition, bounding boxes in non-standard locations occur in the `QTZ` corpus. In the left chart in Figure 9(c), the legend label is misclassified as a y-axis label. First, our technique fails to separate the legend symbol (top-left box) and legend label because both have the same color. Second, the box is left-aligned with the y-axis labels and so the text role classifier confuses its text role. In the right chart, the y-axis title is placed above the y-axis labels and the role classifier fails to recognize it as a title.

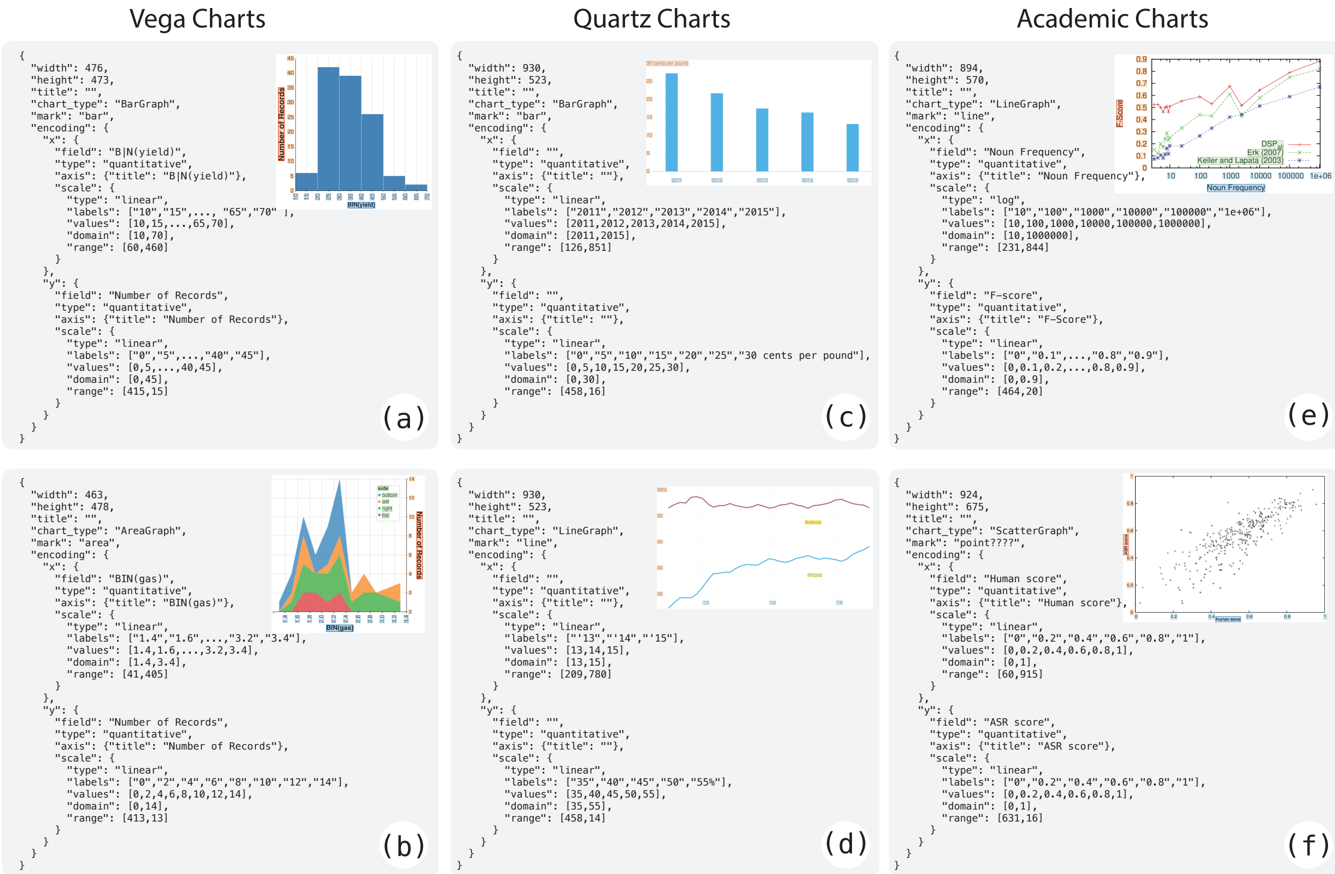

Figure 8: *Successful examples of inferred specifications for bitmap inputs from each corpus.*

Finally, some chart designs in the QTZ corpus use direct labeling of elements (*e.g.*, bars in a bar chart) in lieu of standard x-axis labels. Our system correctly identifies these elements as text labels, but does not use them to infer the correct x-axis domain. Augmenting our system with graphical mark extraction (which is beyond the scope of this paper), might allow us to resolve this issue.

9. Discussion

We presented multiple components that comprise a pipeline for reverse-engineering visualizations, each performing with high accuracy. Our *text localization and recognition* methods outperform Microsoft OCR by at least 19% in the ACA corpus. While we did not directly compare our *text role classifier* with other approaches, we obtain a F1-score of 98%. This result is higher than the 92% F1-score reported by Choudhury et al. [CWG16] using different features and a smaller data set (165 charts, 4,363 boxes). Our *mark type classification* approach exhibits superior performance to ReVision's classifier (F1-scores 94% vs. 80%) on their corpus. However, we are not the first to use Convolutional Neural Networks for mark type classification; Siegel et al. [SDF16] use a similar approach in FigureSeer. Finally, we demonstrated how to use the results of these components to recover a complete visual specification for a chart image. To the best of our knowledge, this is the first end-to-end system to automatically recover visual encoding specifications from a bitmap image. Going forward, we hope to improve our methods and explore novel applications enabled by our pipeline.

9.1. Limitations and Future Improvements

An immediate future work item is to further improve the performance of each stage in our pipeline. Though our text localization and recognition outperforms state-of-the-art OCR, we consider this an open problem that requires more attention for complex chart images, as well as the varied characters (mathematical notation, *etc.*) common to scientific papers. For text role classification, we encode structural information in the feature vectors based on a global analysis of bounding boxes, and perform additional post-processing based on SVM output. More robust techniques might be used. An interesting future research direction would be to apply structured prediction techniques to this task.

While our methods identify legend titles and labels, we do not perform *interpretation* of legends. As a result, our pipeline does not infer scale mappings for visual channels such as color, shape, and size. Legend analysis remains an important area for future work.

To scope this work, we made simplifying assumptions regarding possible chart types. However, many components of our pipeline can be applied to more general situations. Future work might expand our pipeline to cover multiple mark types (*e.g.*, layers), trellis plots, maps, error bars, box plots and radial charts. Currently

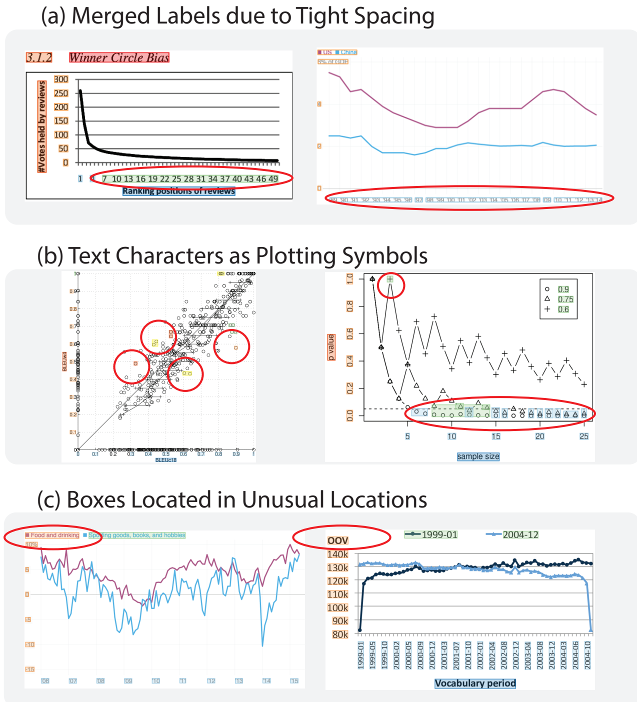

Figure 9: *The three most common chart inference failure cases.*

we consider mark information only at the level of whole-image classification. Combining our methods with mark extraction methods [SKC*11,SDF16,JKS*17] might enable more accurate specification inference for a wider array of chart types. A related research direction is to integrate our work with data extraction techniques.

9.2. Potential Applications

While this paper focuses primarily on the application of computer vision and machine learning methods to interpret chart images, our results can enable a variety of visualization applications. For example, our inferred chart specifications enable indexing of chart images based on mark type, visualized data fields, and data ranges. A straightforward application is to use this information to improve search engines [CCA15] by better incorporating figures.

Another application area is to restyle or retarget visualizations, an initial motivation of the ReVision system [SKC*11]. This task is important as many published charts exhibit poor perceptual design choices that may hamper understanding. In addition, the lack of accessibility information leaves many charts unusable by people with vision impairment. Our pipeline allows automatic extraction of valuable metadata; paired with access to the backing data, it could enable a variety of redesign tools.

We are particularly eager to perform large-scale analyses of visualization practices. Using the information extracted by our system, we hope to chart the development, deployment, and dissemination of visual encoding conventions across various literatures. By reverse-engineering visualizations, we can perform an automated census and analyze visualization use in the wild.

To help enable future applications, our pipeline and chart corpora are available at https://github.com/uwdata/rev.

Acknowledgements

We thank Dominik Moritz for providing a pre-trained CNN model for text pixel classification. This work was supported by a Paul G. Allen Family Foundation Distinguished Investigator Award and the Moore Foundation Data-Driven Discovery Investigator program.

References

[BOH11] Bostock M., Ogievetsky V., Heer J.: D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2301–2309. 2

[BS15] Böschen F., Scherp A.: Multi-oriented text extraction from information graphics. In *Proceedings of the 2015 ACM Symposium on Document Engineering* (2015), pp. 35–38. 2

[CCA15] Chen Z., Cafarella M., Adar E.: DiagramFlyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web* (2015), pp. 183–186. 2, 10

[CD16] Clark C., Divvala S.: PDFFigures 2.0: Mining figures from research papers. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries* (2016), pp. 143–152. 3

[CV95] Cortes C., Vapnik V.: Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297. 6

[CWG16] Choudhury S. R., Wang S., Giles C. L.: Scalable algorithms for scholarly figure mining and semantics. In *Proceedings of the International Workshop on Semantic Big Data* (2016), pp. 1:1–1:6. 2, 9

[HA14] Harper J., Agrawala M.: Deconstructing and restyling D3 visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (2014), pp. 253–262. 2

[HLYW13] Huang W., Lin Z., Yang J., Wang J.: Text localization in natural images using stroke feature transform and text covariance descriptors. In *Proceedings of the 2013 IEEE International Conference on Computer Vision* (2013), pp. 1241–1248. 2

[HT07] Huang W., Tan C. L.: A system for understanding imaged infographics and its applications. In *Proceedings of the 2007 ACM Symposium on Document Engineering* (2007), pp. 9–18. 2

[JKS*17] Jung D., Kim W., Song H., Hwang J.-I., Lee B., Kim B., Seo J.: ChartSense: Interactive data extraction from chart images. In *ACM Human Factors in Computing Systems (CHI)* (2017). 2, 7, 10

[JRW*07] Jayant C., Renzelmann M., Wen D., Krisnandi S., Ladner R., Comden D.: Automated tactile graphics translation: In the field. In *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility* (2007), pp. 75–82. 2

[JSD*14] Jia Y., Shelhamer E., Donahue J., Karayev S., Long J., Girshick R., Guadarrama S., Darrell T.: Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014). 7

[KA12] Kong N., Agrawala M.: Graphical overlays: Using layered elements to aid chart reading. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2631–2638. 2

[Kru56] Kruskal J. B.: On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society 7*, 1 (1956), 48–50. 5

[KSH12] Krizhevsky A., Sutskever I., Hinton G. E.: ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012, pp. 1097–1105. 7

[LH15] Lee P., Howe B.: Dismantling composite visualizations in the scientific literature. In *4th International Conference on Pattern Recognition Applications and Methods* (2015). 3

[Luc05] LUCAS S. M.: ICDAR 2005 text locating competition results. In *Eighth International Conference on Document Analysis and Recognition* (2005), vol. 1, pp. 80–84. 5

[mic] Microsoft Project Oxford. https://www.projectoxford.ai/vision. 2, 5, 6

[MMR*01] MÜLLER K.-R., MIKA S., RÄTSCH G., TSUDA S., SCHÖLKOPF B.: An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks 12*, 2 (2001), 181–202. 6

[MNV16] MÉNDEZ G. G., NACENTA M. A., VANDENHESTE S.: iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (2016), pp. 4073–4085. 2

[Mor17] MORITZ D.: Text detection in screen images with a convolutional neural network. *The Journal of Open Source Software* (2017). https://github.com/domoritz/label_generator. 4

[NM16] NEUMANN L., MATAS J.: Real-time lexicon-free scene text localization and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 38*, 9 (2016), 1872–1885. 2

[Ots79] OTSU N.: A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics 9*, 1 (1979), 62–66. 4

[RCWG16] RAY CHOUDHURY S., WANG S., GILES C. L.: Curve separation for line graphs in scholarly documents. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries* (2016), pp. 277–278. 2

[Red16] REDMON J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. 4

[SDF16] SIEGEL N., DIVVALA S., FARHADI A.: FigureSeer: Parsing result-figures in research papers. In *Proceedings of the European Conference on Computer Vision* (2016), pp. 664–680. 2, 7, 8, 9, 10

[SKC*11] SAVVA M., KONG N., CHHAJTA A., FEI-FEI L., AGRAWALA M., HEER J.: ReVision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (2011), pp. 393–402. 2, 7, 10

[SLJ*15] SZEGEDY C., LIU W., JIA Y., SERMANET P., REED S., ANGUELOV D., ERHAN D., VANHOUCKE V., RABINOVICH A.: Going deeper with convolutions. In *Computer Vision and Pattern Recognition* (2015), pp. 1–9. 7

[Smi07] SMITH R.: An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition* (2007), vol. 2, pp. 629–633. 2, 5

[SMWH17] SATYANARAYAN A., MORITZ D., WONGSUPHASAWAT K., HEER J.: Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 341–350. 1

[SRHH16] SATYANARAYAN A., RUSSELL R., HOFFSWELL J., HEER J.: Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (2016), 659–668. 2, 3

[STH02] STOLTE C., TANG D., HANRAHAN P.: Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (2002), 52–65. 1

[WMA*16] WONGSUPHASAWAT K., MORITZ D., ANAND A., MACKINLAY J., HOWE B., HEER J.: Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (2016), 649–658. 3