Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations

Jorge Poco, Angela Mayhua, and Jeffrey Heer

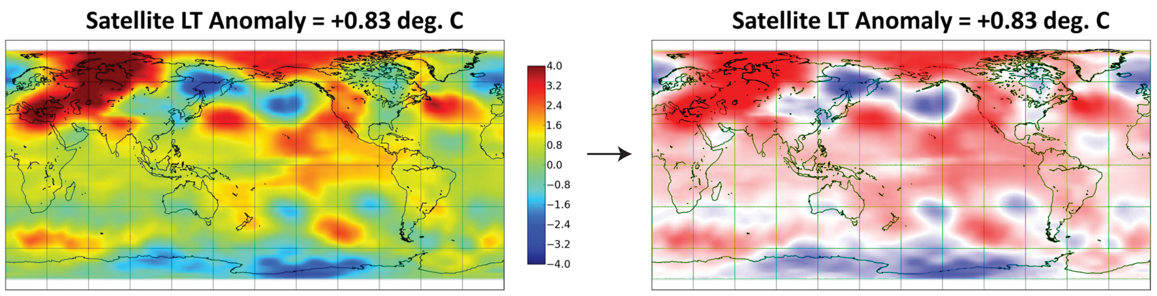


Fig. 1. Automatic extraction and redesign of color mappings for a geographic heatmap. The bitmap image on the left uses a questionable rainbow color palette. Our methods automatically recover the color mapping, enabling applications such as automatic recoloring. The generated image on the right replaces the original color palette with a perceptually-motivated diverging color scheme.

Abstract— Visualization designers regularly use color to encode quantitative or categorical data. However, visualizations "in the wild" often violate perceptual color design principles and may only be available as bitmap images. In this work, we contribute a method to semi-automatically extract color encodings from a bitmap visualization image. Given an image and a legend location, we classify the legend as describing either a discrete or continuous color encoding, identify the colors used, and extract legend text using OCR methods. We then combine this information to recover the specific color mapping. Users can also correct interpretation errors using an annotation interface. We evaluate our techniques using a corpus of images extracted from scientific papers and demonstrate accurate automatic inference of color mappings across a variety of chart types. In addition, we present two applications of our method: automatic recoloring to improve perceptual effectiveness, and interactive overlays to enable improved reading of static visualizations.

Index Terms—Visualization, color, chart understanding, information extraction, redesign, computer vision.

1 Introduction

Color mappings are commonly used to encode data and enhance visualization aesthetics. Distinct hues can be used to effectively convey category values, while changes in luminance or saturation can encode ordinal or quantitative differences. Designing effective color encodings, however, can prove difficult. Designers must balance issues including perceptual discriminability, cultural conventions, and aesthetic preferences [4, 11, 13, 18]. Poorly designed palettes can lead to imprecise and inaccurate readings of the data [1, 2, 8]. Even when well-designed, color encodings often afford less precise comparisons than alternative visual channels such as position or size, and may suffer from issues such as limited discriminability or simultaneous contrast.

Given the complexities of color design, analysts often rely on the default palettes provided by visualization software packages. In many cases these palettes — including ubiquitous rainbow palettes [2] — have not been subjected to perceptual design and analysis. As a result, viewers could benefit from tools for automatic recoloring and interactive querying of visualizations with color encodings.

However, many visualizations "in the wild" are available only as static images in a bitmap or vector format, including charts found in media such as web sites, presentation slides, textbooks, and academic papers. While these static images are useful for viewing by people, their content is largely inaccessible to computers, limiting our ability to perform automated analysis and retargeting.

In response, some recent projects explore the use of computer vision and machine learning techniques to (semi-)automatically interpret chart images [15, 22, 25, 26]. These systems can successfully identify chart types, recover spatial encodings, and read off data values for basic plots such as bar, pie, or line charts. However, these systems do not address the task of automatically recovering color mappings.

We contribute a method to semi-automatically extract color encodings from a bitmap visualization image. Given an image and legend location, we classify the legend as describing either a discrete or continuous color encoding, identify the colors used, and extract legend text using OCR methods. We then combine color and text information to recover the color mapping. We evaluate our techniques using a corpus of images extracted from scientific papers and demonstrate accurate automatic inference of color mappings across a variety of chart types. We also present a legend annotation interface that can be used to label images and correct interpretation errors.

Extracted color mappings can be used to aid visualization indexing, search, and redesign. We present two user-facing applications of our color encoding extraction methods. Our first application performs *automatic recoloring*: given bitmap images as input, we produce new images that use effective, perceptually-motivated color encodings. Inspired by Kong & Agrawala [16], our second application adds *interactive overlays* to a static image, enabling data querying and highlighting. Users can brush a color legend to select corresponding data values, hover over data points to highlight matching legend values, and select regions to view automatically-produced statistical summaries. Many of these features require accurate extraction of legend color values only (not OCR text), and are applicable to a variety of visualization images.

• *Jorge Poco is with University of Washington and Universidad Católica San Pablo. E-mail: jpocom@ucsp.edu.*
• *Angela Mayhua is with Universidad Católica San Pablo. E-mail: angela.mayhua@ucsp.edu.pe.*
• *Jeffrey Heer is with University of Washington. E-mail: jheer@uw.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

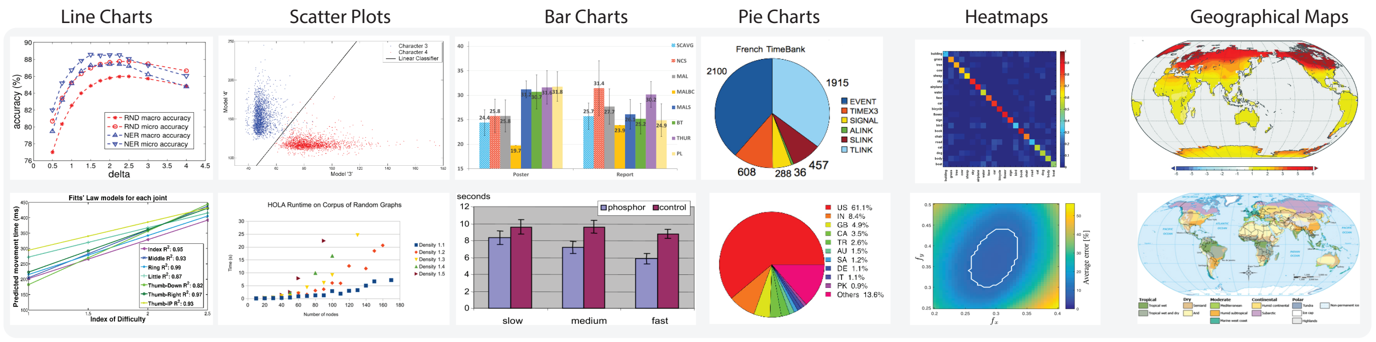


Fig. 2. Example visualization images from our corpus, each including an explicit color legend.

2 Related Work

Our work on color mapping extraction draws on two streams of prior work: color design for visualization and automatic chart interpretation.

2.1 Color Design for Visualization

Colors play a central role in data visualization, as they are commonly used to visually encode data. However, there is often a disconnect between visualization research and visualization practice "in the wild". For example, Dasgupta et al. [8] report mismatches with the climate research community based on a two year collaboration with domain scientists. Despite the availability of decades-old design guidelines [2, 24] in the visualization literature, rainbow color maps are still ubiquitous. Such design choices can have serious consequences: Borkin et al. [1] found that changing the color encoding used for visualizations of arterial stress (switching from a rainbow scale to a perceptually-motivated palette) led to marked improvements in doctors' diagnostic accuracy. Unfortunately, a number of existing commercial visualization tools still provide default color palettes of questionable quality.

Meanwhile, a number of visualization projects have sought to provide both stock palettes and palette-generation tools to promote more effective mappings between data values and colors. Cynthia Brewer's color-use guidelines [4] and popular ColorBrewer palettes are widely used for coloring both maps and more general information visualizations. Heer & Stone [13] demonstrate how models of color naming can be used to assess and improve color palettes. Lin et al. [18] contribute an algorithm for generating color palettes that respect "semantically resonant" color-concept associations. More recently, Gramazio et al.'s Colorgorical [11] tool allows designers to interactively balance concerns such as discriminability, naming similarity, and aesthetic preferences.

In this work, we contribute techniques for extracting color encodings from visualization images, as well as applications for automatic recoloring and interactive overlays. To be clear, we do not contribute methods for color *design*. For example, our recoloring application assumes an appropriate target color scheme is provided as an input.

2.2 Automatic Chart Interpretation & Retargeting

A growing body of work focuses on the "inverse problem" of data visualization: given a visualization, can one recover the underlying encodings and data values? Solutions to this problem can enable automated analysis, indexing and redesign of published visualizations — in some cases without recourse to an original specification or dataset.

Harper and Agrawala [12] introduce a system to deconstruct D3 [3] visualizations within a web browser. By exploiting the live data binding between vector graphics and backing data elements, they extract data, marks, and mappings between them. This information can then be used for tasks such as redesign and style transfer.

A number of projects focus on the more difficult problem of interpreting visualizations available only as static images. Savva et al. [25] introduce ReVision, a system to classify bitmap chart images by chart type, automatically extract data from pie charts and bar charts, and generate redesigns using the extracted data table. Siegel et al. [26] and Choudhury et al. [6, 23] present techniques to extract data from line charts using bitmap and vectorial images, respectively.

Other projects — including ChartSense [15] and iVoLVER [21] — focus on semi-automated approaches, leveraging interactive annotation to perform accurate chart interpretation. We similarly provide an annotation interface that we use for corpus annotation; this interface can also be applied to correct automated extraction errors.

While reading *data* from a chart image is one potential goal of automatic interpretation, one might also wish to infer the *program* that generated the visualization. Extracting encoding specifications can be valuable for indexing and can often be performed even when occlusion and visual clutter render precise data extraction impossible. Poco & Heer [22] present a pipeline that specializes in accurate text localization and recognition within bitmap chart images, and classifies recovered text labels according to their role (*e.g.*, x-axis label, y-axis label, legend label, etc). Using these components they are able to perform accurate extraction of spatial encodings. In this paper we follow a similar aim, but focus squarely on the problem of color mapping extraction. We adopt methods from Poco & Heer [22] to recognize the text content of legend labels. Siegel et al. [26] also contribute a classifier to assign semantic roles to each text element, and use inferred legend labels to identify legend symbols. However, they assume that text information is given *a priori*, focus only on discrete color legends, and do not recover the color mapping. To the best of our knowledge, no prior work has proposed a technique to semi-automatically recover color mappings from a static visualization image.

In addition to indexing and redesign applications, automatic chart interpretation can enable new interactions with previously static media. Kong & Agrawala [16] demonstrate how to add interactivity to static pie and bar charts. They use the ReVision system [25] to interpret the chart and generate *graphical overlays* to highlight marks and provide guideline annotations to assist chart reading. Inspired by this application, we introduce a novel application that leverages extracted color mappings to support interaction with either plotted data or color legends to highlight and summarize values of interest. Elmqvist et al. propose Color Lens [9], a technique that dynamically optimizes color scales based on a set of sampling lenses. Using this technique we can emulate some of the features of interactive overlays; however, Color Lens has a different goal and does not recover color mappings from bitmap images.

3 Data Collection and Annotation

To develop techniques for color mapping extraction, we compiled an annotated corpus of visualization images using color encodings.

3.1 Image Collection

We collected visualization images from both academic papers and the web sites of scientific institutions (*e.g.*, NASA, universities). First we downloaded 16 million images from papers indexed by the Semantic Scholar[1] search engine. These chart images come from academic publications in Computer Science. In order to reduce the number of images and promote design variability, we selected figures from the areas of visualization, human computer interaction, computer vision, machine learning, and natural language processing. From an initial

[1]http://www.semanticscholar.org/

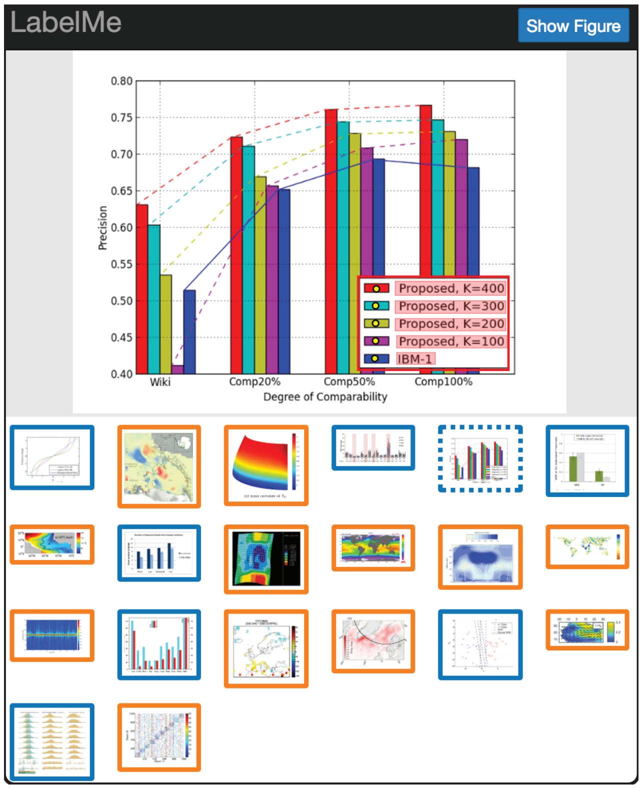


Fig. 3. Graphical user interface for legend annotation. The bottom panel shows chart images in our corpus. Outline colors represent the color legend type: orange for continuous and blue for discrete.

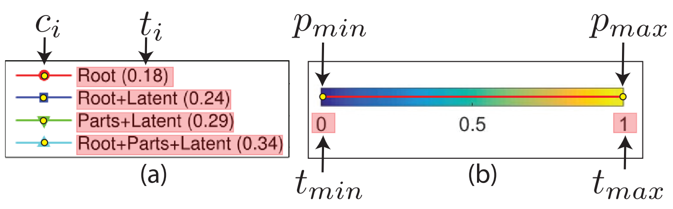


Fig. 4. Annotations on color legends. Note that we extract different information for (a) discrete and (b) continuous legend types.

collection of 330,000 figures, we manually selected 1,000 chart images containing explicit color legends. We initially selected a random subset of 200 images per category. We then removed figures without a color legend and randomly selected more figures to replace them; after some iteration, we arrived at a set of 1,000 figures. We then downloaded 275 papers from three journals influential in earth sciences: Nature, the Journal of Climate, and Geophysical Research Letters. To extract figures from journal PDF files, we used the `pdffigures` [7] utility. We extracted 994 images and manually selected 500 chart images with color legends, using the same process for manual selection described previously. To further augment our collection, we crawled scientific websites such as NASA and university departments (*e.g.*, climatology & oceanography), netting 300 additional images with color legends.

In sum, our corpus contains 1,800 visualization images with color legends. We manually classified each image as representing discrete or continuous data, and selected 800 charts uniformly at random for each type. We use these 1,600 chart images for training and testing throughout this paper. Figure 2 shows examples from our corpus, including line charts, scatter plots, bar charts, area charts, pie charts, heat maps, and geographic maps. Our only constraint for selecting chart images was the inclusion of a color legend.

3.2 Legend Annotation

For each visualization image, we classified the color legend as either discrete or continuous, and then manually labeled the legend elements using a custom graphical interface (Figure 3). Two annotators performed these tasks, requiring roughly 20 hours to complete using our graphical interface.

We define discrete color legends as consisting of a set of symbol elements $l_{disc} = \{e_1, e_2, ..., e_n\}$. Each symbol element is a 2-tuple $e_i = (c_i, t_i)$, where c_i is a color and t_i is the text associated with the color. Using our annotation interface, one can click a legend symbol to extract the representative color c_i and its location p_i. Additionally, to annotate the text t_i associated with c_i, one can draw a rectangle to cover the whole text element. Given the text bounding box, we attempt to recover the text content using the Tesseract OCR engine [27] and manually correct the text as needed. We store both the text content and bounding box information. In Figure 4(a) the yellow circles represent the selected pixels and the red boxes represent the text elements.

We model continuous color legends as spatially contiguous gradients parameterized by the minimum and maximum extents. Thus, $l_{cont} = \{(c_i, t_i), p_{min} \leq p_i \leq p_{max}\}$, where p_i is the i-*th* pixel position. In other words, a continuous color legend is represented by all the colors along the line between the minimum and maximum positions. To annotate continuous color legends within our interface, one can simply click the minimum and maximum positions. In Figure 4(b), these points are represented by the yellow circles. To recover the colors, we scan from p_{min} to p_{max} (for example, along the red line in Figure 4(b)). One can also draw rectangles next to p_{min} and p_{max} to cover the legend labels (red boxes in Figure 4(b)). We then apply OCR and (as needed) manual correction to recover the text content of the labels.

4 Color Mapping Extraction Method

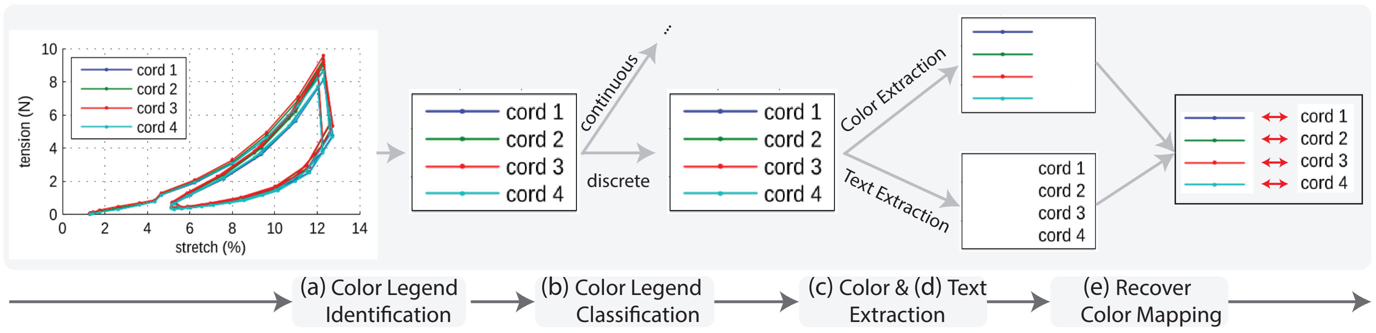
Our approach comprises 5 steps: (1) color legend identification, (2) legend classification, (3) color extraction, (4) legend text extraction, and (5) color mapping recovery. These steps are illustrated in Figure 5. In step 1, we automatically detect legends that occur outside the main plotting area; otherwise, users can manually indicate the legend region. In step 2, we classify the legend as *discrete*, *continuous*, or *other* (to capture images not supported by our technique). In steps 3 and 4, we automatically identify and process legend regions to extract colors and associated text, using different algorithms for the discrete and continuous cases. Finally, in step 5 we combine color and text information to recover color mappings. Across each step, we model colors using the CIE LAB color space. All the techniques described in this section are trained and validated using the annotated corpus of visualization images described in §3.

4.1 Color Legend Identification

Automatic legend identification is a difficult problem, as there is significant variation in both placement and arrangement (*i.e.*, vertical, horizontal, grid). While legends are sometimes placed outside the plotting area, it is also common to place them inside the plot area, particularly in academic papers where the amount of space is limited. In prior work, both Siegel *et. al.* [26] and Poco & Heer [22] contribute classifiers that assign semantic roles to each text element (*i.e.*, legend label, axis title, axis label, etc.). Legend regions could then be inferred based on recognized legend label and legend title elements. A shortcoming of Siegel et al.'s method is that they assume that text information is given *a priori*, which is not true for bitmap images. Poco & Heer do not make this assumption and propose text localization and recognition methods; however, these steps can still produce errors that propagate forward.

Given the difficulty of accurate automatic legend extraction, and the relative ease with which users can simply indicate a legend region with a rectangular brush, our color extraction pipeline expects the legend region to be given as an input. Nevertheless, in this section we present a simple automatic method that does not require labeled text elements and is applicable when the legend is outside the plotting area. This method can be used to initialize a candidate legend region that can be interactively adjusted as needed.

Method: Detection of continuous color legends is typically straightforward, as the color gradient is commonly placed outside the plotting area.



tension (N)
stretch (%)
cord 1
cord 2
cord 3
cord 4
continuous
discrete
Color Extraction
Text Extraction
(a) Color Legend Identification
(b) Color Legend Classification
(c) Color & (d) Text Extraction
(e) Recover Color Mapping

Fig. 5. Steps to infer color mappings from a chart image. (a) Localize the legend region in the chart images. (b) Use a CNN to classify the color legend as discrete, continuous, or other. (c) Depending on the legend type, process the legend to extract colors. (d) Use OCR to recover legend text. (e) Merge color and text information to recover the color mapping.

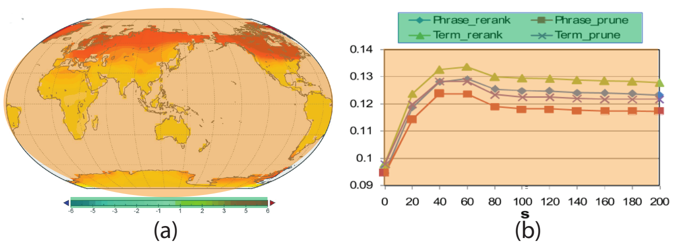


Fig. 6. Automatically identified plotting areas (orange) and legends (green) for both (a) a continuous legend and (b) a discrete legend.

The intuition behind our method is that the plotting area will typically be the largest connected component in the chart, and the second largest component will be the color gradient. For example, see the heatmaps and geographic maps in Figure 2. We first binarize an input image, flood fill holes, and run the connected components algorithm. We then sort the connected components by total area and remove the largest one (*i.e.*, plotting area). Next, we search through the other components in order. If the next largest component fits a rectangle, we return it as the color gradient. Figure 6(a) shows the largest component in orange, and the automatically recognized legend area in green.

Discrete color legends are often placed inside the plotting area. However, if the legend has a rectangular border and is placed outside a well-delimited plotting area, we can apply the identical method described above for continuous legends. Figure 6(b) shows the identified plotting area and discrete legend for a line chart.

Obviously, our simple method will fail for cases that violate our assumptions. Our approach could be further augmented using the methods of Poco & Heer [22], identifying legends based on text element classification. Within user-facing applications, one can also manually indicate a legend position by simply dragging a rectangle, as in our annotation interface.

Validation: Across our visualization image corpus, our simple legend identification technique correctly identifies 50% (400/800) of continuous legends and 10% (77/800) of discrete legends. As expected, the high prevalence in our corpus of legends placed within the plotting area contributes to poor performance, particularly in the discrete case. Other error cases include graphics that do not have a single dedicated plotting area (e.g., heatmaps over multiple 3D physical models) or the presence of variable background colors.

4.2 Color Legend Classification

Once we have identified the legend region in an image, we seek to classify the legend type in order to apply appropriate extraction methods. We trained a classifier that takes a legend region sub-image as input and classifies the image into one of three color legend types: *discrete*, *continuous* and *other*. This last class is included to recognize legend images that are not supported by our approach. In our current version, the *other* class includes random sub-images from chart images. We decided to train our classifier using these sorts of images to ensure the classifier results are useful for seeking user intervention if automated legend identification fails (*i.e.*, it returns an incorrect legend region).

	Precision	Recall	F1-score	# Test Images
Discrete	96%	96%	96%	165
Continuous	97%	98%	97%	159
Other	95%	94%	95%	156
Average / Total	96%	96%	96%	480

Table 1. Legend classification performance.

Method: We use a Convolutional Neural Network (CNN) model for classification. CNNs achieve state-of-the-art performance for many computer vision tasks and have been successfully used to classify the chart type (bar, line, scatter, *etc.*) of visualization images [15, 22, 26]. Training a CNN from scratch requires a large amount of labeled training data, yet we have only a few thousand images in our corpus. A common solution to this mismatch is to fine-tune a pre-trained network via backpropagation with additional images. Here, we fine-tune using the Caffe [14] implementation of AlexNet [17], pre-trained on the millions of images contained in the ImageNet dataset.

To train our CNN, we first extract the image regions (x_l, y_l, w_l, h_l) for each *discrete* and *continuous* color legend to use as examples of those classes. To provide examples of the *other* class, for each chart image we additionally extract an image region (x_o, y_o, w_o, h_o), where (x_o, y_o) is a random coordinate and $w_o = r \times w_l$, $h_o = r \times w_l$, where r is a random (uniform) number between $[0.9, 1.1]$. We resize each extracted sub-image to a 256×256 pixel square, preserving aspect ratio and filling empty space with a white color.

Validation: We evaluate our classification approach using 2,400 images across 3 categories. To maintain parity among classes, for the *other* class we sample 800 of the 1,600 randomized image regions generated above. We then randomly split the data into training (80%) and test (20%) sets. Table 1 shows the resulting precision, recall and F1-scores for test set classification. Across all classes, we find that our classifier exhibits an average F1-score of 96%.

4.3 Color Extraction

Given the color legend region and type we can proceed to extract colors, using different algorithms for the discrete and continuous cases.

4.3.1 Discrete Legends

As described in Section 3, we represent a discrete color legend as a set of elements $l_{disc} = \{e_1, e_2, ..., e_n\}$, where $e_i = (c_i, t_i)$. In this section, we present a technique to extract the colors c_i. As shown in Figure 7, we first mask a legend image to isolate colored legend symbols and then use clustering to extract representative colors.

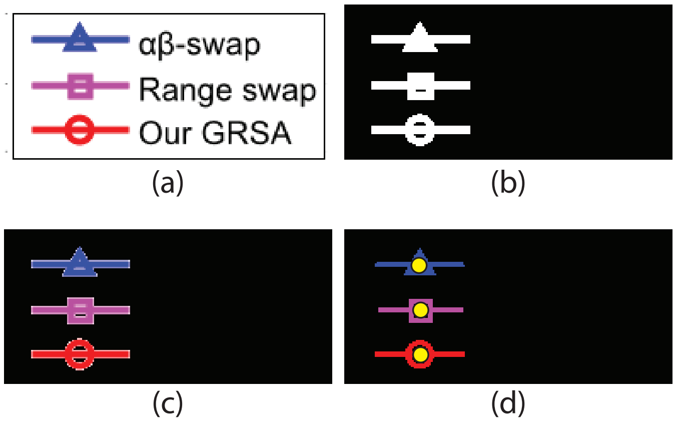


Fig. 7. Extracting color from discrete color legends. (a) Original legend image. (b) Mask combining *background* and *grayscale* masks. (c) Color pixels after applying the mask. (d) Yellow circles indicate pixels with representative colors found via cluster analysis.

Method: We apply two masks to the legend image to filter extraneous pixels: a *background* mask and a *grayscale* mask. We infer the background color c_{bg} by computing a color histogram for all pixels in the legend image and selecting the most frequent color. In most cases the background color is white. As pixel colors may be noisy, we consider a pixel part of the background if its color c_i is in the range $c_{bg} \pm T_{bg}$, where we set $T_{bg} = 5$ (or approximately two JNDs [20]).

We consider a color c_i to be grayscale if both its a and b LAB components lie in the range $0 \pm T_{gray}$ (again, setting $T_{gray} = 5$). The grayscale mask is useful for removing grid lines and text; however, it can also remove pixels that belong to a symbol encoded with a gray color. We describe how we recover those pixels later in this section.

Applying these masks leaves the pixels for the legend colors. Figure 7(b) shows the combined mask, and Figure 7(c) shows the resulting colored pixels. Note that the colors for each symbol might not be homogeneous, especially along the symbol edges. As the color values are noisy and also potentially discontinuous (see inline figure), we can not use simple color binning or connected components analysis. We need a more robust approach for grouping pixels with similar color.

We apply a clustering algorithm to identify the legend colors. We represent each colored pixel using a five-dimensional feature vector consisting of the three LAB color channel values plus the x and y image pixel coordinates: (L,a,b,x,y). In most cases, color information is enough to groups the pixels. However, some legends have colors with the same hue but varying luminance (see inline figure). In such cases our clustering algorithm could fail if the colors are near each other in the CIELAB space. Pixel coordinates help to separate these cases.

We then use the DBSCAN [10] clustering algorithm. DBSCAN does not require that the number of clusters be chosen a priori, and accepts parameters that are well defined for our setting. The ε parameter sets the maximum distance for two points to be considered in the same neighborhood; we use $\varepsilon = 5$. The minPoints parameter sets the minimum number of points in a cluster. We assume that a color legend has at most 20 symbols and so minPoints $= |\text{color pixels}|/20$.

For each resulting cluster g_i, we then select the most common color to be the representative legend color c_i. Figure 7(d) shows the locations of pixels with representative colors (yellow circles).

As mentioned earlier, our *grayscale* mask may erroneously remove pixels that belong to gray-colored legend symbols. To recover such pixels, we run the connected components algorithm within the *grayscale* mask, and retrieve components with an area (*i.e.*, number of pixels) similar to the area of the discovered clusters g_i.

Validation: We evaluate our discrete color extraction method against our visualization corpus. We use precision, recall and F1-score metrics typically used for comparing inferred bounding boxes in the context of text localization [19]. However, in our case, we compare two sets of colors (estimated and ground truth) instead of bounding boxes.

For a color c we find the best match in a set of colors S such that:

$$m(c,S) = \min_{c' \in S} dist_{color}(c,c') \qquad (1)$$

where $dist_{color}(c_i,c_j)$ is the CIEDE2000 color difference.

We apply this *best matching* to align the ground truth legend colors T and estimated colors E. We define $\delta(c_i,c_j) = dist(c_i,c_j) < 5$. Note that δ is 1 if two colors are within 5 CIE LAB units, 0 otherwise. We noticed that comparing colors (c_i, c_j) in the representative pixels (p_i, p_j) can be very sensitive, specially when legend symbol is a segment line. This problem arises specially in annotated data. To address this issue, we use patch regions centered at representative pixels (p_i) and width 3. Thus, $dist(c_i,c_j)$ returns the minimum color difference between patches centered at p_i and p_j.

We define precision, recall and F1-score as follows:

$$p = \frac{\sum_{e \in E} \delta(e,m(e,T))}{|E|},\ r = \frac{\sum_{t \in T} \delta(t,m(t,E))}{|T|},\ F1 = 2\frac{p \cdot r}{p+r} \qquad (2)$$

Our technique has a precision of 92%, recall of 89%, and F1-score of 90% in our 800 discrete color legends.

4.3.2 Continuous Legends

In this section, we describe how to extract colors for a continuous legend. To characterize a continuous color legend, we must identify the color gradient end points p_{min} and p_{max}. We record the pixel locations for each end point and extract colors by scanning the line between the two. Figure 8 illustrates each step in this technique.

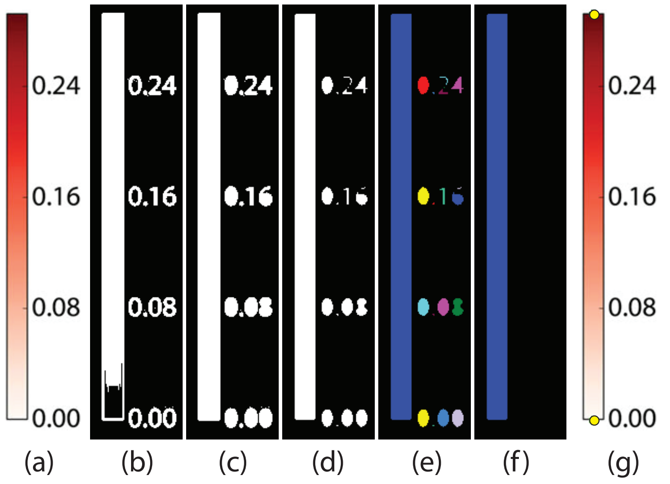
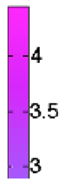


Fig. 8. Extracting color from continuous color legends. (a) Original legend image. (b) Binarized image. (c) Flood fill to recover color gradient. (d) Morphological erosion to separate text from legends. (e) Connected components. (f) Color gradient is the largest connected component. (g) Yellow circles indicate positions of minimum and maximum values.

Method: Akin to the discrete case, we first infer the background color and remove background pixels. Next, we binarize the legend image using a global threshold approach. Note that the bottom part of the color gradient in Figure 8(b) is incomplete. To recover such regions, we use a flood fill algorithm. Figure 8(c) shows the completed region.



We noted that in some charts the text elements are very close to the colorbar (see inline figure). To separate them, we apply a morphological erosion operation (Figure 8(d)). Then we run the connected components algorithm (Figure 8(e)). Our intuition

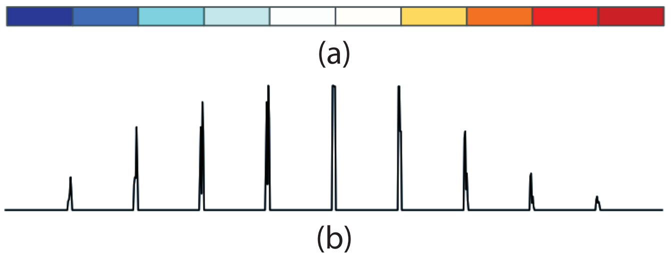


Fig. 9. Analysis of quantized legends for continuous data. (a) Some legends discretize a quantitative domain. (b) Breaks between colors correspond to peaks in the absolute derivative of pixel color.

is that the color gradient should be the largest connected component and all the text characters should be smaller. Thus, we select the largest component (Figure 8(f)) and fit a rectangle that covers these pixels. We note if the rectangle has a vertical or horizontal orientation based on its aspect ratio. If vertical, we determine p_{min} and p_{max} by selecting two aligned points, one at the top and one at the bottom side of the rectangle, each horizontally centered. Scanning a line that join these two points, we recover the colors belonging to this legend (See Figure 8(g)). An analogous approach applies for the horizontal case.

Validation: We evaluate continuous legend color extraction on the 800 continuous examples in our corpus. For each image, we estimate p_{min} and p_{max} using the method above and compare these two points with the ground truth data p'_{min} and p'_{max}. For a vertical rectangle, we deem two points similar if $|p_y - p'_y| < 3$ (*i.e.*, the y-coordinates are at most 3 pixels away). If both p_{min} and p_{max} lie near the ground truth values, we consider the color gradient to have been successfully identified. The horizontal case is treated similarly.

Our method achieves accurate extraction for 83% (662/800) of the continuous legends in our corpus. Recurring error cases involve confusion between the color gradient and background colors (*i.e.*, when a color gradient is placed *within* the plotting area) and confusion due to text labels placed so close to the gradient that they become part of the same connected component. Perhaps ironically, this first error case is most prevalent in the visualization literature, as other disciplines tend to place the color gradient *outside* the plotting area.

Quantized Continuous Domains: It is common to use discretized version of continuous legends (Figure 9(a)). Our color legend classifier classifies these legends as continuous, and we process them using our continuous color extraction methods (*i.e.*, determining p_{min} and p_{max}). However, we can then perform a post-processing step to properly handle this legend type.

Consider the horizontal color gradient in Figure 9(a). First we compute the horizontal derivative to identify strong changes along the x-axis. We apply a Sobel filter with a kernel of size 3. Figure 9(b) shows the absolute values of this derivative (computed as the sum of r, g, and b color channel derivatives). We next convolve the data with a wavelet of size 10 (this value is the expected range that should cover the peaks of interesest) and extract the peaks. Given k identified peaks, we extract a set of $k+1$ colors by picking the colors at the midpoints between peaks, or between the legend boundary and nearest peak.

4.4 Legend Text Extraction

The previous sections describe how we extract colors for discrete and continuous legends. We now present a method for recovering text elements from a legend image (Figure 5(d)).

Method: We first apply the *text localization* method of Poco & Heer [22]. This approach consists of three stages: (1) word detection, (2) optical character recognition, and (3) word merging. In stage 1, a CNN trained to classify text vs. non-text pixels is used to mask non-text pixels. Then, standard region-based text localization methods are applied to output bounding boxes for individual words. In stage 2,

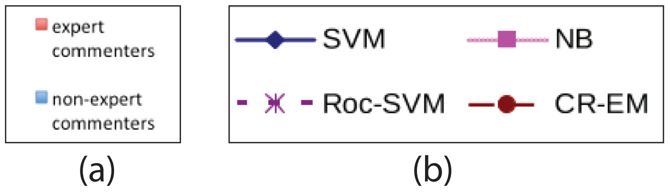


Fig. 10. Recovering color-value mappings for discrete legends. In (a), we associate text with the nearest legend symbol. For multi-column legends (b), we first identify if the text is on the right or left of the legend symbols.

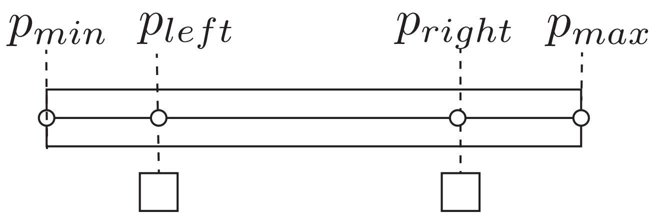


Fig. 11. Recovering color-value mappings for continuous color legends. Values in bounding boxes b_{left} and b_{right} are mapped to positions p_{left} and p_{right}, respectively. We then extrapolate values for p_{min} and p_{max}.

we apply optical character recognition to each candidate word using the open source Tesseract [27] engine. Finally, in stage 3, words are merged into phrases based on their orientation and geometric relationships.

Before submitting legend images to the text localization pipeline, we remove all pixels that belong to either discrete legend symbols or continuous color gradients. This preprocessing step cleans the input images in order to improve text localization performance.

Validation: To evaluate the text extraction technique, we use the definition of *best matching* depicted in Section 4.3.1 (Equation 1), but in our case, we compare two sets of boxes (estimated and ground truth) instead of colors. For a box b, we find the best match in a set of boxes S such that:

$$m(b,S) = \max_{b' \in B} dist_{box}(b,b') \qquad (3)$$

Where, $dist_{box}(b_i,b_j) = \frac{2\, area(b_i \cap b_j)}{area(b_i)+area(b_j)}$. Note that $dist_{box}$ is 1 for equal boxes and 0 for boxes without intersection. We then apply this matching to our bounding boxes T (ground truth boxes) and E (estimated boxes) in a legend image. We define precision, recall and F1-score as follows:

$$p = \frac{\sum_{e \in E} m(e,T)}{|E|}, \; r = \frac{\sum_{t \in T} m(t,E)}{|T|}, \; F1 = 2\frac{p \cdot r}{p+r} \qquad (4)$$

Using only the discrete color legends we obtain a precision of 82%, recall 91% and F1-score 86%. If we do not remove the colored pixels beforehand the F1-score is 83%; the OCR results are meaningfully improved by this filter. In addition, these performance results are very sensitive to the bounding boxes. For example, if we shrink or expand the ground truth boxes by 2 pixels, the F1-scores reduce to values between 82% and 85%. The main errors arise when legend symbols contain multiple colors, in which case our preprocessing can fail to remove those pixels.

Using the continuous color legends we obtain a precision of 78%, recall 79% and F1-score 78%. The main issues arise with tick marks. A tick may be confused with the '-' character, extending the text bounding box. In some cases ticks connect the text with the color gradient, causing the *text localization* step to throw out text along with the color gradient. Across both legend types, we obtain an overall precision of 80%, recall 85% and F1-score 82%.

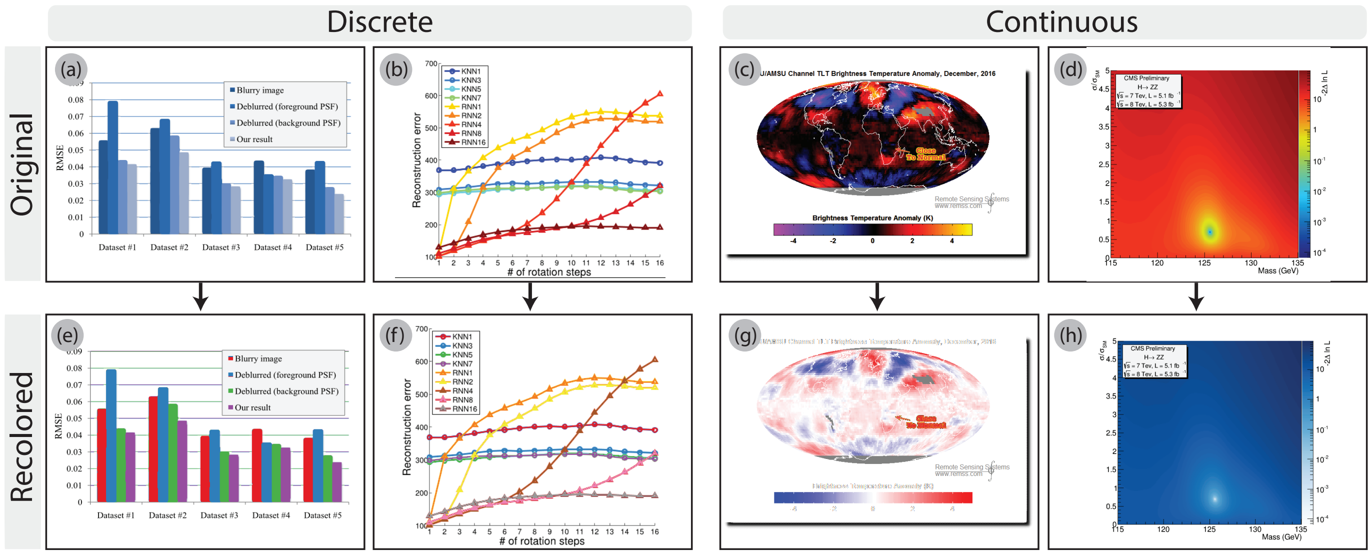


Fig. 12. Automatic Recoloring: Given a chart image and target color scheme, we generate a recolored image with an alternative color encoding.

4.5 Recovering Color Mapping

Given the color information and the text extracted in the last two sections, we can recover the full color mapping. To do so, we need to associate the text values with the color information (Figure 5(e)).

Discrete color legends: Here, we connect each text bounding box with a legend symbol — represented by a pixel inside the legend symbol. For each text bounding box, we calculate the closest legend symbol and associate this text with the color. This strategy also works for legends with multi-line text (Figure 10(a)). However, for horizontally-aligned elements or multiple columns, this strategy can fail. In Figure 10(b), the legend text "Roc-SVM" might be considered closest to the brown legend symbol for "CR-EM". To address this issue, we must determine if text lies to the left or right of the legend symbols. We first sort by x-coordinate of the representative pixels for each legend symbol. Next we check if the left-most and right-most pixel are closer to the left or right side of the legend bounding box, respectively. If the left-most pixel is closer, then text lies to the right (See Figure 10(b)), otherwise text lies to the left.

Continuous color legends: We first identify if the legend gradient is vertically or horizontally oriented. We do so by checking the alignment of the p_{min} and p_{max} coordinates. For the horizontal case, we sort the text bounding boxes by the x-coordinate of their centers. We denote the center of the left-most and right-most text bounding boxes with p_{left} and p_{right} respectively (see Figure 11). We map the value represented by the text in both bounding boxes to p_{left} and p_{right}, respectively. Finally, we extrapolate the values out to p_{min} and p_{max}. An analogous approach applies for the vertical case.

In order to extract the values represented by the text, we first attempt to parse the label text as number, including scientific notation (*e.g.*, '-10', '15', '4.5', '3e-4'). Next, we verify if there is a modifier, such as characters indicating units. We check if the text is using International System units (*e.g.*, '1M, '1k, '10M) and parse these as numbers (*e.g.*, '1k' is converted to 1,000). We leave text parsing of other cases, such as physical units of measurement, as future work.

5 End-to-End Performance

At first blush our pipeline might look complex, given the multiple processing stages. An alternative approach might be to analyze a color histogram for each image and skip the color legend identification step. However, we found that the unbalanced number of pixels for each color makes subsequent color extraction less accurate. Moreover, to automatically infer a color mapping one must identify the legend region. This step could increase the complexity of our pipeline and suffer from error propagation (see discussion by Poco & Heer [22]). Instead, we assume the legend regions are provided as input, as it is relatively easy for a user to drag a rectangle over them.

The validations presented in §4 assume that output from a previous step is perfect. However, without user intervention, an error generated in an earlier step can propagate through the pipeline, reducing the final accuracy. As mentioned above, we present our results without interdependence of the pipeline steps. Though users can manually fix interpretation errors using our annotation interface, for some applications (*e.g.*, analysis of visualizations practices "in the wild") a fully automatic method is required.

For an end-to-end analysis, we assume legend regions are given as input to our pipeline. Our color and text extraction components then depend on the output of legend type classification. However, there is no interdependence between color and text extraction: both steps can be performed in parallel as shown in Figure 5. For discrete color legends, we achieve local F1-scores of 96% in the *legend classification* step, 90% in the *color extraction* step, and 87% in the *legend text extraction* step. If we propagate errors, we have final F1-scores of 82% for *color extraction* and 80% for the *legend text extraction*. For continuous color legends, we have local F1-scores of 96% in the *legend classification* step, 82% in the *color extraction* step, and 80% in the *legend text extraction* step. If we propagate errors, we have final F1-scores of 79% for *color extraction* and 77% for the *legend text extraction*.

6 Applications of Color Mapping Extraction

Our extraction methods can be used to support a variety of visualization applications. For example, our work can be used to extend existing visualization reverse-engineering tools [15,22,25,26] with support for color encodings, improving tasks such as indexing for visualization search engines [5]. Here, we contribute two novel user-facing applications: *automatic recoloring* and *interactive overlays*.

6.1 Automatic Recoloring

Our first application performs *automatic recoloring*: given bitmap images as input, we produce new images that use perceptually-motivated color encodings. This application works for both discrete and continuous color legends. Figure 12 shows examples for four chart images: the first row contains original visualizations and the second row shows output images with new color encodings.

Discrete color legends. Given representative colors $C = \{c_1, \ldots, c_n\}$ inferred by our extraction methods and a target color scheme $T = \{t_1, \ldots, t_n\}$, we create a transfer function $t_i = f_{disc}(c_i)$ that simply maps from one indexed color to another. For each pixel p_i in the image, we find the nearest color in C such that $dist_{color}(c_i, c_s) < 2.5$ and then set

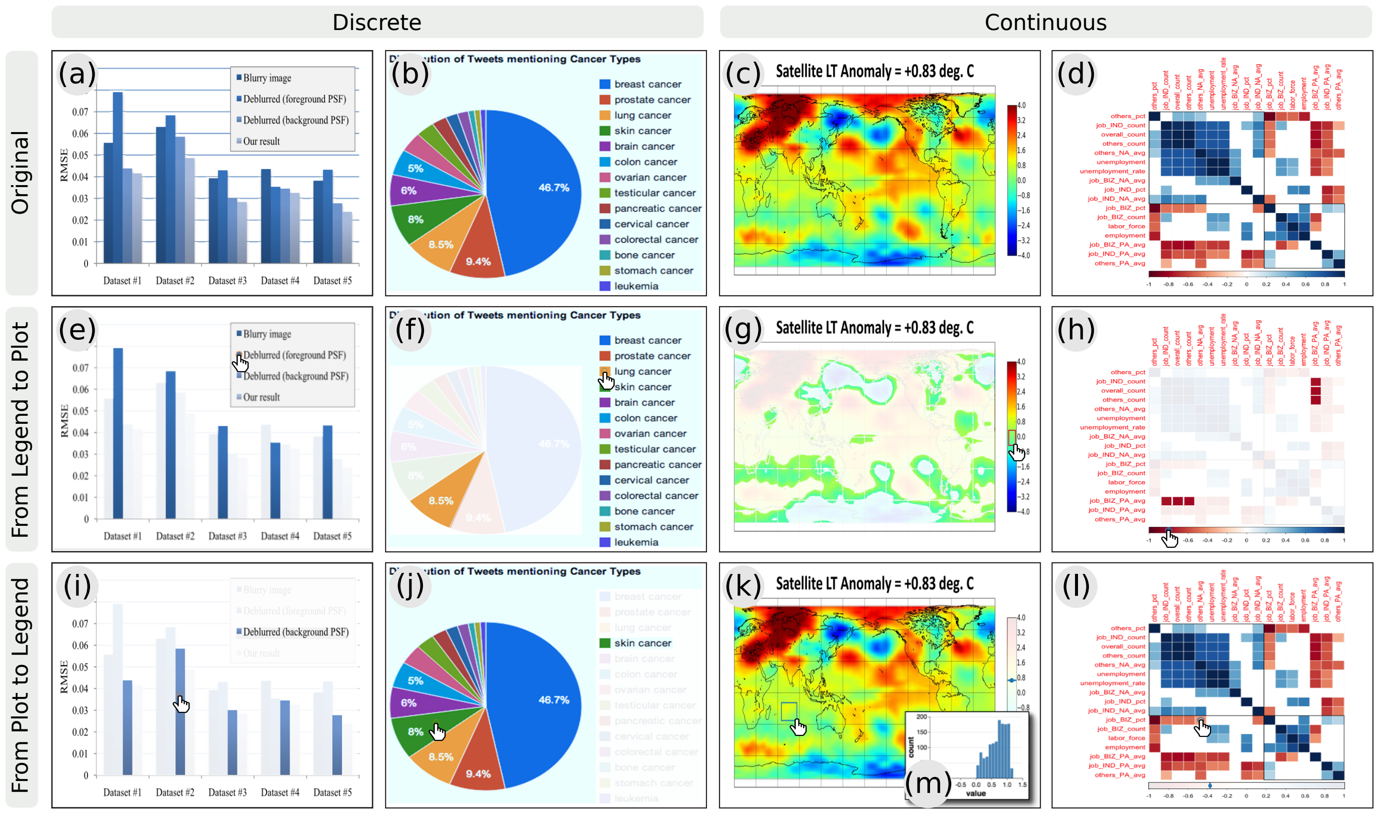


Fig. 13. Interactive Overlays. The first row contains input chart images. The second row shows interactive highlights of the data in response to selecting legend values. The third row illustrates highlights in response to selecting marks in the plotting area. Sub-figure (m) shows a value histogram for the selected pixels, generated by inverting the extracted color mapping.

p_i to the color $f_{disc}(c_i)$. The distance constraint helps avoid recoloring pixels that do not belong to the color legend (*e.g.*, grid lines or text).

The bar chart in Figure 12(a) uses a sequential color encoding where a categorical encoding might be more appropriate. In Figure 12(e), our tool replaces the original scheme with a ColorBrewer palette. The line chart in Figure 12(b) uses a discrete rainbow color map. Some legend items (*e.g.*, "RNN1" and "RNN4") are difficult to discriminate. In Figure 12(f) we retarget the color encoding to a more perceptually effective scheme.

Continuous color legends. For continuous legends, we begin with the extracted colors spanning the minimum and maximum points in the color gradient ($\{c_i \mid p_{min} < p_i < p_{max}\}$), and a target continuous color palette T parameterized on the interval $[0,1]$. We define a transfer function $t_i = f_{cont}(c_i)$, such that c_i and t_i occur at the same relative index in their respective color ramps. For each pixel p_i in the image, we search for the nearest color in C such that $dist_{color}(c_i, c_s) < 2.5$ and recolor the pixel with $f_{cont}(c_i)$.

The map in Figure 12(c) uses a multi-hue color gradient across a domain spanning both negative and positive values. In response, we recolor the image using a diverging color scheme as shown in Figure 12(g). Figure 12(d) uses a rainbow color scheme; in Figure 12(h) we replace the rainbow with a perceptually-informed sequential scheme.

For these recoloring tasks, we need only extract the color information provided in the legend. The actual legend text is not necessary, making our tool robust to issues such as OCR error. However, as our next application demonstrates, recovering the legend values can enable more sophisticated interactions.

6.2 Interactive Overlays

Our second application adds *interactive overlays* [16] to static images to support data querying and highlighting. Our web application generates an interactive visualization from a static image input.

First, the system must locate the color legend. If automated identification (§4.1) fails, a user can simply draw a rectangle to isolate the legend. Next we recover legend color and text information using our extraction techniques. If a user notes any extraction errors, they can use our annotation interface (§3.2) to correct them manually. The application then generates an interactive visualization that supports two-way interactions between the legend and plotting area. Users can brush in either region to see corresponding information highlighted in the other component. Figure 13 shows examples of such interactions across four different chart types.

From legend to plot area. We provide two types of interactions: point and range selection. These interactions are designed to highlight selected data values, as illustrated in the second row of Figure 13.

Point selection is supported for both discrete and continuous color legends. For discrete legends, users can click either a legend symbol or the text associated with the symbol. We then identify the associated representative color c_{sel}. Next, we overlay the plotting area with a translucent white layer, using full transparency for pixels p_i that satisfy $dist_{color}(c_{sel}, c_i) < 2.5$. If the legend is inside the plot area (as in Figure 13(e)), we make that region transparent as well. For continuous color legends, a user can click anywhere within the color gradient to select the color c_{sel}. Then, we similarly add a translucent overlay, but with full transparency for pixels p_i that satisfy $c_i = c_{sel}$.

Range selection is often more appropriate for continuous color legends. When a user draws a rectangle (R) within the color gradient, we select the set of colors $C_{sel} = \{c_i \mid p_i \in R\}$. As with point selection, we then add a translucent overlay with full transparency for pixels that satisfy $c_i = c_s$, $\forall c_s \in C_{sel}$.

From plot area to legend. For interactions with the plot area, we similarly support both point and range selections. These interactions are depicted in the third row of Figure 13.

Point selections can be performed visualizations with either discrete or continuous color legends. For discrete legends, users can click a data-encoding mark in the plot area. We then select the color c_{sel} at the click position p_{sel}, and use it to identify the legend element e_{sel} with representative color nearest to c_{sel}. Next, we overlay the legend region with a translucent white layer. We use the information in e_{sel} (*i.e.*, text and color) to make the layer transparent over pixels that lie inside the text bounding box or satisfy $c_{sel} = color(e_{sel})$. We also add a layer over the plot area, with full transparency for pixels that satisfy $dist_{color}(c_{sel}, c_i) < 2.5$. In this second layer highlights all marks associated with the same e_{sel} (for instance, the multiple bars highlighted in Figure 13(i)). For continuous legends, when a user clicks in the plot area we select the color c_{sel} and search for the nearest color in the legend. We proceed as before to highlight pixels in the plot area. However in the legend region, we highlight only the nearest color in the color gradient.

Our tool also supports range selections for continuous encodings. A user can draw a rectangle (R) in the plot area, for which we retrieve all contained colors $C_{sel} = \{c_i \mid p_i \in R\}$. Then, for all colors in C_{sel}, we find the nearest colors in the legend gradient and highlight them (Figure 13(k)). In this case, we do not add an overlay to the plot area.

The interactions described above require only color information from the legend. However, we can use the full mapping (*i.e.*, inferred data values) to enable additional features. Our tool calculates and displays descriptive statistics for the data values indicated by the selected pixels. For example, Figure 13(m) uses the data values associated with the colors in C_{sel} to create a histogram for the selected region.

7 Limitations

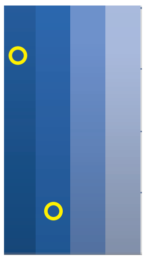
Our color legend classifier includes the category *other*, which we train using random sub-images sampled from our corpus. An improvement would be to collect real legend images for other encoding channels (*e.g.*, size, shape) and train our classifier to discriminate those.

Given the variability of color legend styles, we constrained the color legend types supported. For example, we do not support grayscale legends. Our color extractor for discrete color legends uses a grayscale mask to remove text and grid elements from the legend. As a result, all legend symbols in grayscale are also removed. However, if not all the legend symbols are grayscale, we can recover them as explained in § 4.3.1. We also do not support bi/trivariate color maps, which are uncommon in the figures we collected. In the case of legend symbols with repeated colors (*e.g.*, "red" symbol with a solid line and "red" symbol with a dashed line), our color extractor might fail because we are using LAB coordinates in the clustering, and all the red pixels will be near each other in the CIELAB space.

In the legend text recognition step, we use the techniques of Poco & Heer [22]. However, OCR errors remain an issue in some cases. Recurring challenges for text localization include the resolution of the legend labels, spacing between labels and color elements, and mathematical notation. Moreover, in this work we are not using the "units" information included in some continuous color legends. For applications such as indexing figures (*e.g.* for a scientific database), accurately identifying units may be important.

We currently do not support accurate data value interpolation for non-linear color gradients. To do so, our method must also (a) extract intermediate value labels (not only minimum and maximum), (b) correctly associate the labels with positions in the color gradient, and (c) infer the the type of scale function (*e.g.*, log, square root) based on the label values and spacing. Step (c) is relatively straightforward and supported in prior work [22]. However, step (a) depends heavily on OCR accuracy.

For our recoloring application, errors arise when the legend includes colors that also occur in other chart components (*e.g.*, black for text and grid lines, white for background). In Figure 12(g), the chart title was affected by our transfer function because the initial color gradient contains black as well.



In addition, we found that some bar charts contain elements with variable colors (*e.g.*, gradient fills). This can lead to the same color occurring across marks associated with different legend elements. For instance, in the inline figure, the colors inside the two yellow rectangles are the same, even though the two bars correspond to different legend entries. As a result, our interactive overlay application does not highlight the bars correctly. A more sophisticated color matching model is needed to handle cases such as these shading gradients.

8 Conclusion

In this paper we presented methods for recovering color mappings from static visualization images. To evaluate our approach we compiled an annotated corpus of images and color legend information for 1,600 visualizations from academic documents, and demonstrated accurate extraction of both discrete and continuous color encodings. While we focused on the general case of bitmap images, components of our system are directly applicable to structured image formats such as vector PDF or SVG documents, where objects and text content might be trivially extracted.

We also presented applications of our method for automatic recoloring and generating interactive visualizations from static images. Conveniently, many of the features of these applications do not require the full mapping from color to data value, for example supporting recoloring and highlighting based on extracted colors alone. However, as demonstrated by interactive overlay application, access to the full color mapping can provide even richer interactive capabilities.

To the best of our knowledge, this research contributes the first approach to recover full color mappings from visualization images. Going forward, we hope to improve our methods and explore additional applications enabled by our approach.

An immediate future work item is to further improve the performance of each stage in our pipeline, for example by addressing the limitations enumerated above. Looking further out, we might also extend our techniques to recover mappings for other non-spatial channels, such as size, shape, and texture encodings. We might also extend our work to support bi/trivariate color maps. Combining legend analysis with reverse-engineering pipelines for spatial encodings [15, 22, 25, 26] might enable rich indexing of visualizations, in turn supporting new visualization search and retrieval applications.

Another area for future work is to perform a more thorough comparison with previous approaches. The closest work is Siegel et al.'s approach [26] to extract legend symbols from *discrete color legends*. An appropriate comparison would be to run our pipeline on Siegel et al.'s annotated image dataset (1,000 annotated charts). For that, we will need to parse the figure metadata and infer the legend regions using the bounding boxes of the legend text and legend symbols. However, comparing text elements would be unfair because Siegel et al. assume that the text information is given as input. Also, in Siegel et al.'s dataset the legend symbols are annotated using bounding boxes; to compare with our color extraction method, we will need to process each bounding box region and extract the colors used.

We might also use our color extraction techniques to perform a large scale analysis of the use of colors "in the wild". We can analyze a large collection of visualization images from academic documents across different communities, identify common color schemes, and assess how they may have evolved or disseminated over time.

To help enable such future applications, both our annotated visualization image corpus and the source code for our color mapping extraction methods are freely available at `https://github.com/uwdata/rev`.

Acknowledgments

This work was supported by a Paul G. Allen Family Foundation Distinguished Investigator Award and the Moore Foundation Data-Driven Discovery Investigator program. The second author gratefully acknowledges CONCYTEC for a scholarship in support of graduate studies.

References

[1] M. Borkin, K. Gajos, A. Peters, D. Mitsouras, S. Melchionna, F. Rybicki, C. Feldman, and H. Pfister. Evaluation of artery visualizations for heart disease diagnosis. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2479–2488, 2011.

[2] D. Borland and R. M. Taylor. Rainbow color map (still) considered harmful. *Computer Graphics and Applications*, 27(2):14–17, 2007.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[4] C. A. Brewer. Color use guidelines for data representation. In *Proceedings of the Section on Statistical Graphics, American Statistical Association*, pp. 55–60, 1999.

[5] Z. Chen, M. Cafarella, and E. Adar. DiagramFlyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pp. 183–186. ACM, New York, NY, USA, 2015.

[6] S. R. Choudhury, S. Wang, and C. L. Giles. Scalable algorithms for scholarly figure mining and semantics. In *Proceedings of the International Workshop on Semantic Big Data*, SBD '16, pp. 1:1–1:6. ACM, New York, NY, USA, 2016.

[7] C. Clark and S. Divvala. Pdffigures 2.0: Mining figures from research papers. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, JCDL '16, pp. 143–152. ACM, New York, NY, USA, 2016.

[8] A. Dasgupta, J. Poco, Y. Wei, R. Cook, E. Bertini, and C. Silva. Bridging theory with practice: An exploratory study of visualization use and design for climate model comparison. *IEEE Transactions on Visualization and Computer Graphics*, 21(9):996–1014, Sept 2015.

[9] N. Elmqvist, P. Dragicevic, and J.-D. Fekete. Color lens: Adaptive color scale optimization for visual exploration. *IEEE Trans. Vis. Comput. Graph.*, 17(6):795–807, 2011.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pp. 226–231, 1996.

[11] C. C. Gramazio, D. H. Laidlaw, and K. B. Schloss. Colorgorical: Creating discriminable and preferable color palettes for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):521–530, 2017.

[12] J. Harper and M. Agrawala. Deconstructing and restyling d3 visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pp. 253–262. ACM, New York, NY, USA, 2014.

[13] J. Heer and M. Stone. Color naming models for color selection, image editing and palette design. In *ACM Human Factors in Computing Systems (CHI)*, 2012.

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

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

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

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[18] S. Lin, J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. Selecting semantically-resonant colors for data visualization. *Computer Graphics Forum (Proc. EuroVis)*, 2013.

[19] S. M. Lucas. ICDAR 2005 text locating competition results. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pp. 80–84 Vol. 1, Aug 2005.

[20] M. Mahy, L. Van Eycken, and A. Oosterlinck. Evaluation of uniform color spaces developed after the adoption of cielab and cieluv. *Color Res. Appl.*, 19(2):105–121, 1994.

[21] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pp. 4073–4085. ACM, New York, NY, USA, 2016.

[22] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Computer Graphics Forum (Proc. EuroVis)*, 36(3):353–363, 2017.

[23] S. Ray Choudhury, S. Wang, and C. L. Giles. Curve separation for line graphs in scholarly documents. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, JCDL '16, pp. 277–278. ACM, New York, NY, USA, 2016.

[24] B. E. Rogowitz, L. A. Treinish, and S. Bryson. How not to lie with visualization. *Computers in Physics*, 10(3):268–273, 1996.

[25] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. ReVision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pp. 393–402. ACM, New York, NY, USA, 2011.

[26] N. Siegel, S. Divvala, and A. Farhadi. FigureSeer: Parsing result-figures in research papers. In *Proceedings of the European Conference on Computer Vision*, ECCV '16, 2016.

[27] R. Smith. An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, pp. 629–633. IEEE Computer Society, Washington, DC, USA, 2007.