Extracting Visual Encodings from Map Chart Images with Color-encoded Scalar Values

Angela Mayhua*, Erick Gomez-Nieto*, Jeffrey Heer† and Jorge Poco*‡
*Universidad Católica San Pablo, Arequipa, Peru
E-mails: {angela.mayhua, emgomez, jpocom}@ucsp.edu.pe
†University of Washington, Seattle, WA, USA
E-mail: jheer@uw.edu
‡Fundação Getulio Vargas, Rio De Janeiro, Brazil

***Abstract*—Map charts are used in diverse domains to show geographic data (*e.g.*, climate research, oceanography, business analysis, *etc*.). These charts can be found in news articles, scientific papers, and on the Web. However, many map charts are available only as bitmap images, hindering machine interpretation of the visualized data for indexing and reuse. We propose a pipeline to recover both the visual encodings and underlying data from bitmap images of geographic maps with color-encoded scalar values. We evaluate our results using map images from scientific documents, achieving high accuracy along each step of our proposal. In addition, we present two applications: *data extraction* and *map reprojection* to enable improved visual representations of map charts.**

I. Introduction

Geographic maps are a popular form of data visualization, used to convey information within a geo-spatial context. The use of maps is not limited to experts such as geographers or cartographers: millions of maps are produced and used by scientists, students, governments, and companies for a variety of analytical purposes. A well-designed map encodes information so as to be interpretable by human viewers; however, these maps are often published as bitmap images, without access to the original data. Having access only to pixel-level information impedes automatic processing for tasks such as indexing, searching, and analysis [1], [2]. For that reason, it is difficult to find and reuse map data using either spatial queries (*e.g.*, find all maps involving a specific country) or semantic queries (*e.g.*, find all maps with temperature values in a particular range) [3]. We need computational solutions to automatically process maps due to the existence of millions of maps that have been digitally scanned or digitally created [4].

Existing methods for automatic chart interpretation focus on analyzing common statistical graphics such as bar, line, area, or pie charts. Some projects attempt to recover the underlying data [1], [2], [5]–[9], while others focus on recovering the visual encodings [10], [11]. However, these systems do not support analysis of geographic maps. In this paper, we extend these prior approaches to recover both visual encodings and underlying data for map images with color-encoded scalar values.

Our primary contribution is a map image analysis pipeline that automatically extracts spatial information from map region and color information from legend region to recover a visual encoding specification in a declarative grammar similar to Vega-Lite [12], a high-level grammar of graphics. We also present two applications of our map image analysis pipeline: extraction of color-encoded data values to generate alternative visualizations, such as bar charts of aggregate values by continent, and a reprojection method to generate redesigned map images conveying the same original data.

II. Related Work

Our work draws on prior research in the areas of map interpretation and automatic chart interpretation.

A. Map Interpretation

The literature shows a varied collection of methods to perform automatic *map interpretation* [13] to extract information from maps and analyze their content. For instance, Dhar and Chanda [14] analyze scanned topographic maps to extract and recognize symbols (*e.g.*, trees, forests, rivers, huts, *etc*.) and text contained within the map. One of the steps is to separate the image into four layers: (i) green elements (trees, forests), (ii) red elements (streets), (iii) blue elements (rivers, lakes) and (iv) black elements (text). The map scale and range of latitude/longitude coordinates are entered by the user in order to locate points on the map given their geographical coordinates. Pezeshk and Tutwiler [15] also worked on scanned topographic maps to automatically extract each component of the map in separate layers and recognize the text contained. They propose an algorithm for extracting linear features to generate a layer containing map lines (streets, roads, *etc*.); they then use the RANSAC algorithm to improve the text preprocessing and a hidden Markov model to recognize texts and generate a text output layer.

These previous works focus mainly on topographic maps (*i.e.*, maps characterized by contour lines and road lines [15]) and recognizing their symbols. Our approach automatically extracts spatial information from the geographical map contained in a map chart; this information includes the type of geographic projection used by the map and the range latitude and longitude values of the displayed region.

B. Automatic Chart Interpretation

A growing number of techniques focus on the "inverse problem" of data visualization: given a visualization, recover

the underlying visual encoding and its corresponding data values [16]. Some of these approaches have focused on *data extraction*, for instance, ReVision [5], VIEW [6] and ChartSense [1] classify previously the chart images. ReVision [5] and VIEW [6] use Support Vector Machine (SVM) [17] as classifier and ChartSense [1] uses GoogLeNet [18]. Depending on the chart type, they applied a different approach to extract data, generating as output a data table. Al-Zaidy *et al.* [8] proposed a system that extracts data values from bitmap images of bar charts and using the label roles (*e.g.*, x-title, x-labels, y-title, *etc.*) generates a semantic graph to create a summary which describes the input image. The above mentioned approaches extract data from charts that contain discrete legends (*e.g.*, bar, pie, area, line, or radar charts); however, our work is focused on data extraction from map charts that contain continuous and quantized color legends, which have not been addressed so far.

On the other hand, some methods have been focused on *recovering visual encodings* from charts. Harper and Agrawala [10] present a tool to decompose and redesign visualizations created by the D3 library [19] (*e.g.*, bar charts, line charts, scatter plots, donut charts, and choropleth) analyzing the SVG elements of the chart via JavaScript. Poco and Heer [11] trained a Convolutional Neural Network (CNN) for classifying ten chart types, achieving an accuracy of 96% for classifying maps. Their pipeline identifies textual elements in the bitmap image to recover a visual encoding specification. However, their work does not include extraction of color encodings or geographic projections.

The most related work is the method presented by Poco *et al.* [16]. They proposed a technique to extract the color encoding from discrete and continuous legends of chart images, including geographic maps. Their work identifies the colors used and the legend text, then recovers the full color mapping (*i.e.*, associating value labels with their corresponding colors); however, their system classifies quantized legends as continuous and does not work when each color of the quantized legend represents a range of values. We build upon this approach in our current work, particularly for color extraction. Nevertheless, our work is focused on map charts, thus, we have to tackle different challenges such as identifying map projections.

III. Data Collection and Annotation

In order to train the Machine Learning (ML) techniques used in our work, we collected images and manually annotated them to build a ground-truth corpus of map charts.

A. Image Collection

We collected 474 documents from three well-known geoscience journals in the field of climate change — Nature, the Journal of Climate, and Geophysical Research Letters. Then, the *pdffigures* tool [20] was used to extract 2,018 figures. To select only map charts, we applied chart type classifier by Poco and Heer [11]; obtaining 1,351 map images. We then manually applied two constraints: map images must have a color legend, and the map region must have text labels indicating the latitude and longitude values.

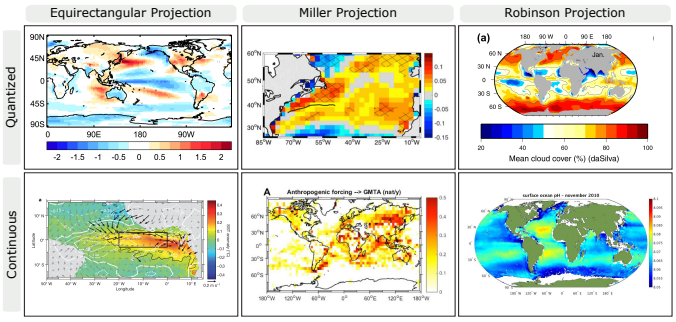


Fig. 1. Examples of map chart images from our corpus, covering three projections and two color legend types. We extracted map images from geoscience journals in the field of climate change.

TABLE I
Counts of map charts per projection and per color legend type, taken from our corpus of map images.

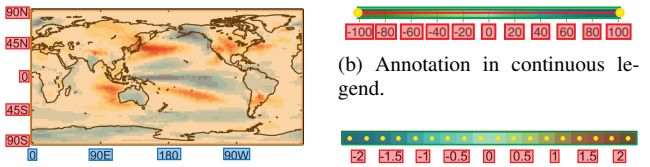
	Equirectangular	Miller	Robinson	Total
Continuous	29	45	37	111
Quantized	71	50	63	189
Total	100	100	100	300

A preliminary analysis shows that the most common projections are *Equirectangular* 46%, *Robinson* 16%, and *Miller* 13% (see columns in Figure 1). In addition, we identified three color legend types: *discrete*, *continuous* and *quantized* (which maps a continuous quantitative domain to a discrete set of colors). We noticed that a few percent of the collected map images contain discrete color legends. Given these results, we decided to focus on these three projections, as well as the *continuous* and *quantized* legends (see rows in Figure 1). Finally, we used uniform random selection of 100 map images for each projection, including both quantized and continuous legends. Table I shows a summary of our map image corpus. Figure 1 shows map images from our corpus that span the world or a specific region using different projections and legend types. After selecting the 300 map images, we annotated each of them, following the process below.

B. Image Annotation

The annotation for *map regions* includes: (i) the map projection type (*i.e.*, Equirectangular, Miller or Robinson), (ii) map location inside the image as represented by a pixel-coordinate bounding box (orange rectangle in Figure 2a), and (iii) the location, content, and role of textual elements (*i.e.*, *latitude*, *longitude* or *other*). In Figure 2a, we can see textual elements indicated with red (latitudes) and blue boxes (longitudes).

For *legend regions*, we annotate the color bar location (green rectangle in Figure 2b and Figure 2c) and the textual elements including their image location, content and role (*i.e.*, *label*, *other*). For continuous color legends, we also annotate the minimum and maximum pixel coordinates (yellow circles in Figure 2b). For quantized color legends, we mark a representative pixel inside each bin (yellow circles in Figure 2c).



(a) Annotation in map region.

(b) Annotation in continuous legend.

(c) Annotation in quantized legend.

Fig. 2. For each map chart we annotate the map and legend regions. (a) A map region has the map location (orange rectangle) and textual elements representing latitude (red boxes), longitude (blue boxes), or other text. (b) A continuous color legend has the color bar location (green rectangle), textual elements such as labels (red boxes) or other texts, and the minimum and maximum pixel coordinates (yellow circles); the red line represents all the colors inside the color bar. (c) A quantized color legend also has the color bar location and textual elements; yellow circles indicate representative pixels for each bin.

In the following sections, we explain how the annotated corpus is used to train our techniques in order to recover the visual encoding for each map image.

IV. Method Overview

Our map image interpretation pipeline is comprised of four main steps (see Figure 3). We first segment the input map image into two main regions (Section V). Second, we extract spatial information for the map region, based on recovered latitude and longitude labels (*map analysis*, Section VI). Third, given the legend region, we extract color encoding information (*legend analysis*, Section VII). Finally, we perform *encoding inference* to generate a visual encoding specification using the information extracted in the two previous steps (Section VIII). In the following four sections, we explain and validate each step in detail.

V. Map and Legend Segmentation

Given a map chart image as input, we seek to decompose it into two main regions: map and legend, because each one requires a different approach to extract information.

The intuition is that the two largest components correspond to the map and the color bar, as we can clearly see in Figure 3a. We convert the input image to grayscale, apply simple binarization, flood fill the holes, erode the binary image, and then run the connected components algorithm [21]. Next, we sort the connected components by area and select the two largest components.

At this point, we have a map/legend region *without* textual information. To attach this information, we first analyze the aspect ratio of the color bar to determine the legend orientation. We consider four cases: when the legend is vertical, the map may be on the left or right side of the legend; when the legend is horizontal, the map may be on the bottom or top side of the legend. When the legend is vertical and the map is on the left side of the legend, we compute the center (c_x, c_y) of each component and compare the distance from the c_x to the right of the map region and the left of the color bar. The lowest distance indicates if the component belongs to the map region or the legend region. We use an analogous approach for the other three cases. In Figure 3b we see the result of applying our approach to recover the map (orange rectangle) and the legend (green rectangle) regions.

Validation. To evaluate this method, we apply the matching score between two rectangles defined by Lucas *et al.* [22]. For each image in our corpus (§III), we compute the matching score between the ground truth and the estimated rectangles (map and legend regions); then, the accuracy is defined as the average of the matching scores by region. Our region identification technique achieves an accuracy of 97.96% for map regions and 92.35% for legend regions. The strong prevalence of the legends placed outside the plotting area contributes to this high performance.

VI. Map Analysis

Given the map region as input, our pipeline automatically extracts the spatial information, including textual elements and projection type, following these steps: (a) text bounding box identification, (b) text bounding box classification, (c) text extraction, (d) value inference, and (e) projection inference.

A. Text Bounding Box Identification

Our first goal is to identify the bounding boxes of text elements. To improve text localization performance, we first remove the largest connected component that in this case corresponds to the map (orange region in Figure 4b). Once the image is clean, we apply the *text localization* method of Poco and Heer [11].

Validation. We compute the F1-score in the same way as Poco and Heer [11]. They showed that even if all text boxes are correctly identified, the F1-score can still vary from 80%-100% by adjusting the box size by 1–2 pixels. Here, we achieve an F1-score of 80.09%, which indicates that this technique is acceptable to identify bounding boxes within the map region.

B. Text Bounding Box Classification

Once the text bounding boxes are identified, we classify each one according to its role. Our classifier is based on the *text role classifier* of Poco and Heer [11] and was trained to distinguish between three text roles, as illustrated in Figure 4b: *latitude* (red boxes), *longitude* (blue boxes) and *other* (*i.e.*, any other text inside the map region).

Validation. We evaluate our classifier using our ground-truth corpus (§III). We use 5-fold cross validation with stratified sampling to ensure an equal proportion of the classes in each fold. Table IIA shows the total number of boxes for each text role within a map region and our corresponding F1-score, which ranges from 95% to 100%.

C. Text Extraction

We apply the Tesseract OCR engine [23] to extract text from the bounding boxes. For the *other* class we apply the default Tesseract OCR configuration; however, to improve the text extraction for latitudes and longitudes, we created our own dictionary for Tesseract OCR to reduce the number of

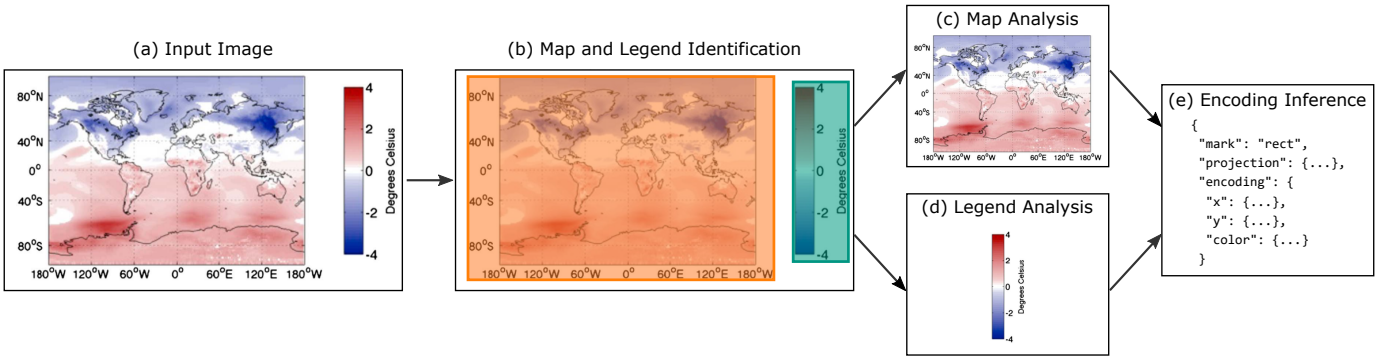


Fig. 3. Our approach to analyzing an (a) input map chart is comprised of four main steps: (b) We segment the image into map and legend regions. (c) The map region is analyzed to extract spatial encoding information. (d) The legend region is processed to extract color encoding information. (e) Finally, we combine the information extracted in the previous steps to infer a visual encoding specification.

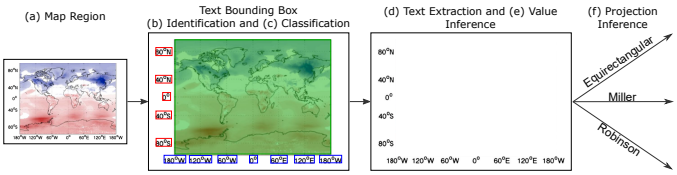


Fig. 4. Map analysis pipeline for extracting spatial information. (a) The map region is given as input. (b) We identify the text bounding boxes and (c) they are classified depending on their text role. Next, (d) we extract the text content and (e) infer the label values. Finally, (f) we infer the map projection type.

TABLE II
EVALUATION PER TEXT ROLE. (A) F1-SCORE OF OUR CLASSIFIER OF TEXT BOUNDING BOXES COMPUTED USING 5-FOLD CROSS VALIDATION. (B) ACCURACY OF OUR TEXT EXTRACTION TECHNIQUE COMPUTED USING EXACT MATCHING OF TEXT STRINGS.

Text role	# boxes	F1-score
latitude	2168	99%
longitude	2309	100%
other	254	95%
Total / Avg	4731	99%

(A) Text bounding box classification.

Text role	# boxes	Accuracy
latitude	2168	79.38%
longitude	2309	81.90%
other	254	12.60%
Total / Avg	4731	57.96%

(B) Text extraction.

characters. For each bounding box, we convert the subimage to grayscale, resize it using Lanczos interpolation [24], and binarize the output using Otsu's method [25]. We then apply the Tesseract OCR configuration corresponding to the text role.

Validation. To evaluate OCR performance, we use the similarity function sim_{exact} that seeks an exact matching between ground-truth and estimated text strings, $sim_{exact}(s_i, s_j)$ returns "1" if the strings are equal and "0" if they are not.

For each bounding box within map regions in our ground-truth corpus (§III), we extract the text using our technique and compare with the real text. Table IIB shows the accuracy for each text role in the map region. The *other* class has the lowest accuracy due to some map regions have superindices, subindices or Greek symbols as part of their titles.

D. Value Inference

In this step, we infer numeric values from the extracted texts; latitude values will range from -90 to 90 and longitude values will range from -180 to 180. To infer the value, we extract the numeric part of the text and sort latitude bounding boxes by their center y-coordinates to determine the sign: when the number decreases it means the values are positive. However, if the number increases the value is negative. We use a similar approach to determine longitude values.

Validation. For each image of our corpus (§III), we select the latitudes and longitudes from the ground-truth bounding boxes, then, we infer the values from the ground-truth texts. Finally, we compare the ground-truth values with the inferred values. Our method achieves an accuracy of 100% for both latitude and longitude ground-truth texts; this result indicates that as long as the text is correct, the value will be inferred correctly.

E. Projection Inference

Once we have inferred latitude and longitude values, we use these values and their bounding box centers $c_i = (c_{ix}, c_{iy})$ to infer the projection type. As we show in Figure 4c, longitudes (blue boxes) are located on the x-axis and latitudes (red boxes) along the y-axis. Moreover, they are mutually independent and their positions depend on the geo projection used by the map. As we see in Section III, the selected geo projections have constant distance between longitude texts, for that reason we focus here on analyzing latitude texts.

As first step, we identify the *first latitude* (*i.e.*, located on the top-left or top-right side of the map region) with center c_0, then we compute the distribution of latitude labels along the y-axis by computing the vertical distance in pixels between latitude label i and c_0. The relationship between those distances and latitude values generates a distribution of points that can be fitted to a curve. The second row in Figure 5 shows examples of distributions (red points) and curve fitting (blue curve) of these points: the x-axis represents latitude values and the y-axis represents distances (normalized) to the *first latitude*.

We define a map template of the whole world for each projection (see the first row of Figure 5). For each map template, we fit a curve that will be used to infer the projection type for any map region. After testing different functions, our final curves are: linear function for Equirectangular projection

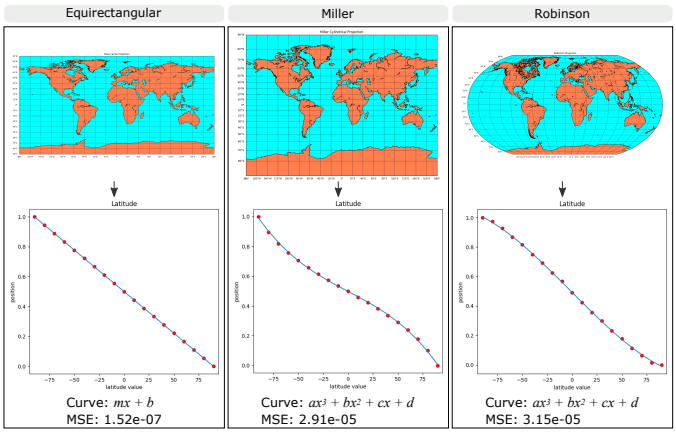


Fig. 5. Map templates for each geo projection and their corresponding fit curves. The curve fits points of the relationship between latitude values and their position into the map region; red points represent the distribution of latitudes and the blue curve represents the fit.

and cubic function for Miller and Robinson projections; we can see that mean squared errors (MSE) are very small which means a good fit.

After applying the prior steps to the input image, we first compute the distribution of distances between its latitudes. Let l_i and l_{i+1} be the values of two consecutive latitude labels, pos_i and pos_{i+1} the corresponding values in the y-axis of their distribution, pos'_i and pos'_{i+1} are the estimated values for l_i and l_{i+1} using the fit curve, respectively. We compute a scale ratio $r_i = \frac{pos_{i+1} - pos_i}{pos'_{i+1} - pos'_i}$ for each pair of consecutive latitude labels.

A distribution of latitude labels can be fitted to an existing curve (template) when their scale ratios have the same or near values for all latitude labels. We verify latitude alignment, if they are vertically aligned, we select as possible geo projections Equirectangular and Miller; as otherwise it should be Robinson. For each curve that represents a possible projection type, we compute pos'_i and the scale ratio r_i, then we obtain the standard deviation for that set of r_i values. Finally, we choose as output the projection with the smallest standard deviation.

Validation. For each image in our corpus (§III), we select the latitudes from the ground-truth bounding boxes, then we infer the projection type and compare with the annotated projection type. The accuracies are Equirectangular projection 99%, Miller 97% and Robinson 100%. Robinson projection achieves a 100% because it is the only projection in our corpus which does not have latitude labels aligned vertically.

VII. Legend Analysis

Given the legend region as input, our pipeline extracts the color information, including textual elements and representative pixels for color extraction. To recover this information, we apply the following steps: (a) legend classification, (b) text bounding box identification, (c) text bounding box classification, (d) text extraction using OCR, and (e) color extraction.

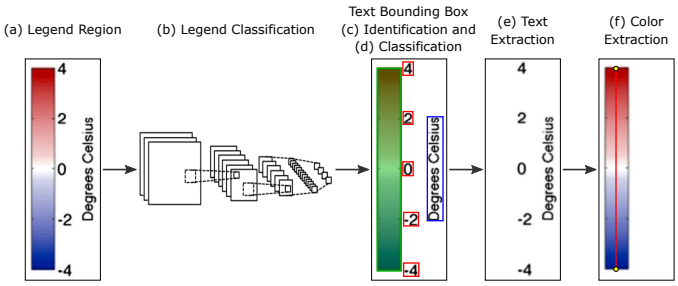


Fig. 6. Legend analysis pipeline for extracting color information. (a) The legend region is given as input and (b) it is classified by type. Then (c) we identify the text bounding boxes, (d) classify them, (e) extract their text using OCR, and finally (f) extract colors from the color bar.

TABLE III
Legend Classification Performance for Test Set.

Legend type	Precision	Recall	F1-score	# Images
Continuous	92%	82%	86%	93
Quantized	85%	92%	88%	99
Discrete	98%	100%	99%	107
Other	96%	96%	96%	101
Avg / Total	93%	93%	93%	400

A. Legend Classification

First, we classify the color legend to apply an appropriate method in the color extraction. This classifier takes a legend region as input and classifies it into one of four color legend types: *continuous*, *quantized*, *discrete* or *other* (*i.e.*, legend images that are not supported by our proposal). Color legends can vary in terms of number of colors, color scales, orientation, and size. For that reason, our classifier is based on the CNN AlexNet [26], because the features and weights for classifying are all learned from end-to-end.

We obtained the legend corpus of Poco *et al.* [16], then legend region subimages were extracted from our corpus to add the *quantized* class. In total, our legend corpus has 2,000 images across 4 categories (80% for training and 20% for testing). Next, to train with our small corpus of legends, we fine-tune the Caffe [27] implementation of AlexNet [26], modifying the number of classes in the last layer [28] to specify our four classes.

Validation. We evaluate our classifier using 20% of images from the legend corpus presented previously. Table III shows the resulting precision, recall and F1-score measures for test set classification. Across all classes, we find that our classifier exhibits an average F1-score of 93%.

B. Text Bounding Box Identification

In this step, we apply the same procedure of §VI-A. First, removing the color bar in the legend region (green rectangle in Figure 6c); then, we identify text bounding boxes.

Validation. We evaluate this technique against the ground-truth bounding boxes of legend regions in our corpus (§III) in the manner as §VI-A. In this case, for our corpus the F1-score is 78.30%.

C. Text Bounding Box Classification

Our classifier is based on *text role classifier* of Poco and Heer [11] and was trained to distinguish two text roles inside legend region (see Figure 6d): *label* (*e.g.*, red boxes) and *other* (*i.e.*, any other text into legend region).

Validation. We applied the same validation process as in §VI-B, using the ground-truth bounding boxes of legend regions of our corpus (§III). Our classifier achieves an average F1-score of 99%, with an F1-score of 100% (2589/2589) for *label* role and 98% (178/182) for *other* role.

D. Text Extraction

We extract their text content. For each bounding box, we apply the same image processing explained in §VI-C, and then apply the Tesseract OCR engine [23].

Validation. To evaluate OCR performance, we perform exact matching of extracted text with the ground-truth text from our corpus. The accuracy using exact matching is 91.59%.

E. Color Extraction

Given the color legend region and its type, we extract colors using different approaches for each color legend type.

1) Continuous Legends: We identify the color bar end points p_{min} and p_{max} (yellow circles in Figure 6f) and record their pixel locations. By scanning the line between these points, we can recover all colors.

Validation. We evaluate this technique using the same evaluation process of Poco *et al.* [16] on the 111 images that contain continuous color legend in our corpus (see Table I). This technique achieves an accuracy of 90.99% (101/111).

2) Quantized Legends: We identify p_{min} and p_{max} for extracting the line between these points. For a horizontal color bar, a Sobel filter of size 3 is applied to compute the horizontal derivative and identify strong changes along the x-axis. Then, we compute the absolute values of the horizontal derivative (*i.e.*, the sum of r, g, and b color channel derivatives) and extract the peaks. If we identify k peaks, we will have $k+1$ colors We use a similar approach to extract colors from a vertical color bar.

Validation. We evaluate color extraction of quantized legends on the 189 images that contain this legend type (see Table I). For each image we estimate pixel locations of peaks and compare the pixel colors in those locations with the pixel colors of ground-truth points using CIEDE2000 color difference [29] as distance measure between colors, where $dist_{color} < 2.5$ [30]. Our technique achieves an F1-score of 93.59%.

VIII. Visual Encoding Generation

We obtain different information of the image after analyzing the map and legend regions (see Figure 7). Using that information, we can generate a visual encoding specification, setting values directly for some entries and inferring others.

Fig. 7. Recovery of visual encoding using data extracted by map analyzer and legend analyzer in a declarative grammar similar to Vega-Lite [12]. Some values need to be inferred and others are assigned directly (colored).

Direct specification: We see in Figure 7 that *labels* are assigned directly in the visual encoding to x/y-channel (map analysis output) or color channel (legend analysis output). The projection type also is assigned directly. We noticed that while some information is not used directly, we need it to infer other fields required in our visual encoding.

Inferring color scale type: We infer the color scale type (*i.e.*, linear, logarithmic, power, or sqrt) using the centers of text bounding boxes (x-coordinate for horizontal legends and y-coordinate for vertical legends). We use non-linear least squares regression to fit multiple functions to these values and pick the model with the minimum mean squared error.

Inferring domain and range of color channel: The `legend.type` entry in our visual encoding (see Figure 7) depends directly on the legend type, *i.e.*, if the legend type is continuous, the entry value is *gradient* and if it is quantized, the entry value is *binned*. For both legend types, `range` is an array of hexadecimal colors extracted from representative pixel positions.

For a continuous legend, the domain is defined as $[v_{min}, v_{max}]$ that represent extreme values of the legend. The inverse function S^{-1} of scale type is used to estimate them, having as input the representative pixel positions (attribute *colors* in the legend analysis output).

For quantized legends we have two situations, when each

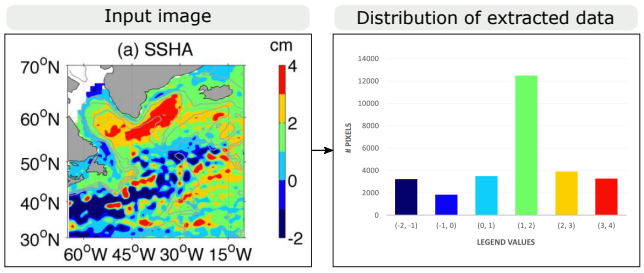


Fig. 8. Analysis of extracted encoded data on map charts. User can know how distribution of legend values is inside map area.

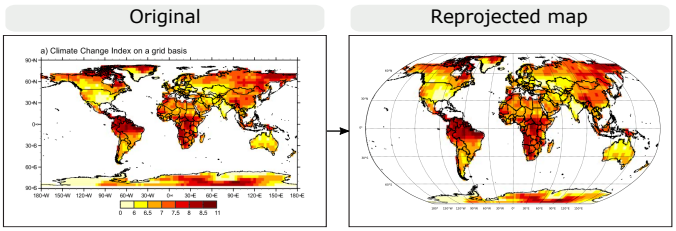


Fig. 9. An example of map reprojection: given a bitmap image and a target map projection, we generate a new image that contains the reprojected map which maintains original data.

color represents a value or it represents a range of values. In the first case, representative pixels should be aligned with a legend label; however, it does not always happen and we use the S^{-1} function to estimate the values for the missing legend labels. The domain will be an array with all values (*i.e.*, estimated and label values). For the second case, we verify that a legend label exists between two representative pixels, if it does not exist, we estimate its value using S^{-1}. The domain will be an array of tuples (min_i, max_i) where min_i and max_i represent the minimum and maximum values of the range.

Inferring domain and range of x/y***-channel****:* Some maps show their minimum and maximum longitudes and latitudes, but it is not always the case. For that reason, we infer those values using the inverse function of the current projection type.

For latitudes (y-channel), let $\{y_1, y_2, \ldots, y_n\}$ denote the center of latitude bounding boxes in the y-axis, and let $\{lat_1, lat_2, \ldots, lat_n\}$ denote the latitude values on each box. Then, $P^{-1}(y_i) = lat_i$ denotes the inverse function of latitudes to the current projection type used to infer the minimum lat_{min} and maximum lat_{max} latitudes given $y_{max} = b_y$ and $y_{min} = b_y + b_h$, where b is bounding box of the map region. We then infer the y-channel domain as $[lat_{min}, lat_{max}]$ and its range as $[y_{min}, y_{max}]$ (see y-channel in Figure 7). A similar procedure, using x-coordinates, is applicable for longitudes (x-channel).

Inferring projection data*:* As we see in Figure 7, `lucorner` will be $[lon_{min}, lat_{max}]$ and `rlcorner` will be $[lon_{max}, lat_{min}]$. `center` is defined as $[lon_c, lat_c]$ that represents the geographical coordinate of the middle point of map bounding box.

IX. Applications

Our method can be used to support a variety of applications. In this section, we present how to use our inferred visual encoding to extract data encoded in a map chart and to reproject a map image maintaining its original data.

A. Data Extraction

Visual encoding can be used to extract data encoded on a map chart, which can then be used to redesign the visualization itself [5]. We have a domain $D = \{d_1, ..., d_n\}$ and a range $R = \{c_1, ..., c_n\}$ from the color channel entry in our visual encoding. First, we create a colormap $CM = \{(c_i, d_i)\}$ mapping a color (c_i) to a data value (d_i). Then, for each pixel p in the map area, we infer the latitude lat and longitude lon values using the inverse projection function. After that, we get the color p_c and find the nearest color in CM such that $dist_{color}(p_c, c_i) < 2.5$ [30]. If a nearest color exists, we recover its value d_i as val. Finally, we export those values into a CSV file, where each row contains a (lat, lon, val) tuple. This file can be used by the analyst to create other visualizations and to extract trends in data. For instance, the input image in Figure 8 contains a quantized legend where each legend value is a tuple and the bar chart shows the distribution of pixels for each legend value.

B. Map Reprojection

Our second application performs *map reprojection*: given a bitmap image of a map chart as input, we produce a new image which uses a different projection, maintaining the original data (colors). Figure 9 shows an example, using a map with the Equirectangular projection as input, our tool automatically create a new map chart with the Robinson projection.

This application uses the `projection` field from the inferred visual encoding and a target geo projection. For each pixel p_i in the map area, we infer its latitude lat_i and longitude lon_i using the inverse function of the geo projection. Then, we create a color map CM that given a pixel position pos_i (the position in the vectorized version of the image), returns the pixel color c_i. In addition, we create a list with the elements (lat_i, lon_i, pos_i).

Once the coordinates, positions, and CM have been generated, we can reproject a map chart. To create this new map, we use the list with (lat_i, lon_i, pos_i) and the color map CM to plot the same colors as the original image. The center and parameters of map projection are specified using the field `projection` of the visual encoding, and the projection type is the only parameter we change to create the new map.

X. Limitations

Multiple geographic coordinates*:* Our approach assumes that map charts contain latitudes and longitudes. However, we found that other coordinates systems exist such as the *Universal Transverse Mercator (UTM)* and *Military Grid Reference System*. Our technique does not support those systems because we did not find enough map images in the selected geoscience journals; however, we intend to explore new data sources and, if necessary, generalize our techniques to handle those cases.

Maps without coordinates: It is also common to render geographic maps without coordinates, *i.e.*, without textual information indicating latitude and longitude values. Analyzing our image corpus, we did not find many of these cases; however, we know that we can find many of them on the Internet. A possible solution would be to apply techniques from shape analysis and match the map boundaries with prebuilt maps in order to identify spatial locations and map projection types.

Automatic chart interpretation: This work is part of a more ambitious objective. The goal is to create an automatic chart interpretation system that, given any chart image, can automatically infer the visual encodings and, as is feasible, extract the underlying data or approximate distributions thereof. If we accomplish this goal, we could create more impactful applications such as improving figure indexing & search, make chart images more accessible for people with disabilities, and perform large-scale analysis of visualization practices.

XI. Conclusion

In this paper, we present a novel approach to extract the visual encodings from map chart images. Given a bitmap map image as input, we generate a visual encoding specification in a format similar to Vega-Lite. We trained and validated each component of our pipeline using real data collected from scientific documents, and our results show high accuracies on each task. Moreover, we describe two useful applications: data extraction from map charts, and reprojection to change the design of the map using the inferred visual encoding specification.

Acknowledgment

This work was supported by grant 234-2015-FONDECYT (Master Program) from Cienciactiva of the National Council for Science,Technology and Technological Innovation (CONCYTEC-PERU).

References

[1] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo, "ChartSense: Interactive data extraction from chart images," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 6706–6717.

[2] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, and A. Farhadi, *FigureSeer: Parsing Result-Figures in Research Papers*. Cham: Springer International Publishing, 2016, pp. 664–680.

[3] V. Walter, F. Luo, and D. Fritsch, *Automatic Map Retrieval and Map Interpretation in the Internet*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 209–221.

[4] Y.-Y. Chiang, S. Leyk, and C. A. Knoblock, "A survey of digital map processing techniques," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–44, May 2014.

[5] 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*. ACM, 2011, pp. 393–402.

[6] J. Gao, Y. Zhou, and K. E. Barner, "View: Visual information extraction widget for improving chart images accessibility," in *19th IEEE International Conference on Image Processing*, Sep. 2012, pp. 2865–2868.

[7] R. A. Al-Zaidy and C. L. Giles, "Automatic extraction of data from bar charts," in *Proceedings of the 8th International Conference on Knowledge Capture*, ser. K-CAP 2015. ACM, 2015, pp. 30:1–30:4.

[8] R. A. Al-Zaidy, S. R. Choudhury, and C. L. Giles, "Automatic summary generation for scientific data charts," in *AAAI Workshop: Scholarly Big Data*, 2016.

[9] B. Tummers, "DataThief III," http://datathief.org/, 2006.

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

[11] J. Poco and J. Heer, "Reverse-engineering visualizations: Recovering visual encodings from chart images," *Computer Graphics Forum*, vol. 36, no. 3, pp. 353–363, jun 2017.

[12] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, vol. 23, no. 1, pp. 341–350, 2017.

[13] V. Walter and F. Luo, "Automatic interpretation of digital maps," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 4, pp. 519–528, Jul. 2011.

[14] D. B. Dhar and B. Chanda, "Extraction and recognition of geographical features from paper maps," *International Journal of Document Analysis and Recognition*, vol. 8, no. 4, pp. 232–245, Sep. 2006.

[15] A. Pezeshk and R. L. Tutwiler, "Automatic feature extraction and text recognition from scanned topographic maps," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 12, pp. 5047–5063, Dec 2011.

[16] J. Poco, A. Mayhua, and J. Heer, "Extracting and retargeting color mappings from bitmap images of visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 637–646, Jan. 2018.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[19] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-driven documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[20] 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*, 2016, pp. 143–152.

[21] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-56864, Jan. 2005.

[22] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young, K. Ashida, H. Nagai, M. Okamoto, H. Yamamoto, H. Miyao, J. Zhu, W. Ou, C. Wolf, J.-M. Jolion, L. Todoran, M. Worring, and X. Lin, "Icdar 2003 robust reading competitions: Entries, results, and future directions," *International Journal of Document Analysis and Recognition*, vol. 7, no. 2-3, pp. 105–122, Jul. 2005.

[23] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, Sept 2007, pp. 629–633.

[24] K. Turkowski, "Filters for common resampling tasks," in *Graphics Gems*, A. S. Glassner, Ed. San Diego, CA, USA: Academic Press Professional, Inc., Apr. 1990, pp. 147–165.

[25] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979.

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

[27] 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.

[28] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, May 2016.

[29] G. Sharma, W. Wu, and E. N. Dalal, "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations," *Color Research & Application*, vol. 30, no. 1, pp. 21–30, Dec. 2005.

[30] M. Stokes, M. D. Fairchild, and R. S. Berns, "Precision requirements for digital color reproduction," *ACM Transactions on Graphics*, vol. 11, no. 4, pp. 406–422, Oct. 1992.