Scatteract: Automated Extraction of Data from Scatter Plots

Mathieu Cliche[1(✉)], David Rosenberg[1], Dhruv Madeka[2], and Connie Yee[1]

[1] Bloomberg LP, 731 Lexington Ave., New York, NY, USA
{mcliche,drosenberg44,cyee3}@bloomberg.net
[2] Amazon.com Inc., 7 West 34th Street, New York, NY, USA
maded@amazon.com

Abstract. Charts are an excellent way to convey patterns and trends in data, but they do not facilitate further modeling of the data or close inspection of individual data points. We present a fully automated system for extracting the numerical values of data points from images of scatter plots. We use deep learning techniques to identify the key components of the chart, and optical character recognition together with robust regression to map from pixels to the coordinate system of the chart. We focus on scatter plots with linear scales, which already have several interesting challenges. Previous work has done fully automatic extraction for other types of charts, but to our knowledge this is the first approach that is fully automatic for scatter plots. Our method performs well, achieving successful data extraction on 89% of the plots in our test set.

Keywords: Computer vision · Information retrieval
Data visualization

1 Introduction

Charts are used in many contexts to give a visual representation of data. However, there is often reason to extract the data represented by the chart back into a numeric form, which is easy to use for further analysis or processing. This problem has been studied for many years, and the solution process may be divided into four steps: chart detection [4,17], chart classification [9,15,18], text detection and recognition [5,14], and data extraction [14,18]. Our work focuses on the latter two areas.

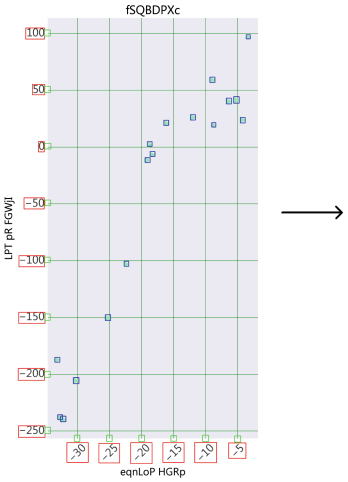
We have built Scatteract[1], a system that takes the image of a scatter plot as input and produces a table of the data points represented in the plot, in the coordinate system defined by the plot axes. There are two main steps in this process. First, we localize three key types of objects in the image: (1) *tick marks*, the visual representation of units on an axis (e.g. short lines on the axis),

D. Madeka—Work done while the author was at Bloomberg L.P.

[1] https://github.com/bloomberg/scatteract.

M. Ceci et al. (Eds.): ECML PKDD 2017, Part I, LNAI 10534, pp. 135–150, 2017.
https://doi.org/10.1007/978-3-319-71249-9_9

(2) *tick values*, the representation of scale on the axis (e.g. numbers written near the tick marks), and (3) *points*, the visual elements representing data tuples of the form[2] (x, y). In the chart in Fig. 1, we show the bounding boxes for objects of these three types, as produced by our object detection system. The locations of these bounding boxes are initially described in *pixel coordinates*, as is natural for image processing. The second key step in our process is to map the pixel coordinates of the points into *chart coordinates*, as defined by the scale on the chart axes. The end result of our process, a table of data points, is also illustrated in Fig. 1, along with the ground truth (x, y) coordinates of the data.



X_{pred}	Y_{pred}
-3.4	96.4
-9.0	58.5
-6.4	39.9
-5.2	40.8
-16.1	20.6
-12.0	25.4
-8.7	18.8
-4.2	23.0
-18.7	2.0
-18.3	-6.7
-19.0	-12.1
-22.4	-103.2
-25.2	-150.7
-33.1	-187.6
-30.2	-206.0
-32.2	-239.6
-32.7	-238.1

X_{true}	Y_{true}
-3.3	96.9
-8.9	59.1
-6.3	40.2
-5.1	41.2
-16.1	21.1
-12.0	26.2
-8.7	19.2
-4.1	23.8
-18.7	2.2
-18.3	-6.4
-19.0	-11.6
-22.3	-102.9
-25.2	-150.5
-33.1	-187.1
-30.2	-205.6
-32.2	-239.7
-32.7	-238.3

Fig. 1. Illustration of input and output of the end-to-end system. Bounding boxes are shown to illustrate the result of the object detection and are not present on the original image. Red bounding boxes identify the tick values, green bounding boxes identify the tick marks, and blue bounding boxes identify the points. (Color figure online)

While information extraction from charts and other infographics has been widely studied, to our knowledge, our system is the first to extract data from an image of a scatter plot and represent it in the coordinate system of the chart, fully automatically. The closest in aim is [3], an unpublished work that describes an approach to such a system, but only gives test results for the point detection part. Semi-automatic software tools are available, but they require the user to manually define the coordinate system of the chart and click on the data points, or to provide metadata about the axes and data [11,14,20,26].

There exist several systems that attempt to extract key components of charts, but do not attempt to convert from pixel coordinates to chart coordinates. Many use heuristics, such as connected component analysis, edge detection and k-median filtering [8,12,13,15,16]. Recent work has taken a machine learning approach. [2] classifies the role of extracted chart components (e.g. bar or legend).

[2] As is traditional, we refer to the horizontal axis as "X" and the vertical axis as "Y".

In [25], a deep learning object detection model is trained to detect sub-figures in compound figures. FigureSeer uses a convolutional neural network to extract features for the localization of lines and heuristics for the localization of tick values [21]. The end result is the localization of all the relevant chart components, but does not include the pixel-to-chart coordinate transformation.

For charts other than scatter plots, there are two systems that aim to convert from pixel coordinates to chart coordinates. The automatic data extraction in [1] is for bar charts. It extracts the text and numerical values using OCR, and then recovers the numerical values for each bar by multiplying its height in pixels by the pixel-per-data ratio. Inaccuracies of the OCR tool resulted in a significant number of the charts having incorrectly calculated Y-axis scale values. The ReVision system proposed in [18] recovers the raw data encoded in bar and pie charts by extracting the labels with OCR and using a scaling factor to map from image space to data space. All these systems are highly dependent on the accuracy of the OCR.

In this paper, we propose Scatteract, an algorithm that leverages deep learning techniques and OCR to retrieve the chart coordinates of the data points in scatter plots. Scatteract has three key advantages over the systems mentioned above. First, to our knowledge, Scatteract is the only system to use deep learning methods to identify all the key components of a scatter plot. The second novel aspect of our work is that we created a system for procedurally generating a large training dataset of scatter plots, for which the ground truth is known, without requiring any manual labeling. This allows our system to be much more extensible than heuristic methods built around a set of assumptions. The plot distribution can be easily modified to accommodate new plot aesthetics, while heuristic methods may need a complete redesign to accommodate these changes. The third key advantage of our system is that our method for determining the mapping from pixel coordinates to chart coordinates is fairly robust to OCR errors.

This paper is organized as follows. In Sect. 2 we describe the datasets used to train and test Scatteract. In Sect. 3 we expand on the methodology and present test results for the building blocks of Scatteract. In Sect. 4 we present a performance analysis of the end-to-end system, and we outline our main conclusions in Sect. 5.

2 Datasets

Scatteract uses an object detection model, which requires a large amount of annotated data for training. One way to achieve this is to collect a large sample of scatter plots from the web and manually label the bounding boxes for the objects of interest (points, tick marks, and tick values). A more efficient approach is to generate the scatter plots procedurally, so that the bounding boxes are known. Using this approach, we generated 25,000 train images and 600 test images. Some examples are shown in Figs. 6 and 8. Besides these artificial charts, we scraped an additional 50 scatter plots from the web (Figs. 7 and 9). More details on how our datasets were obtained are below.

2.1 Procedurally Generated Scatter Plots

To achieve randomness in the scatter plots, we developed a script to randomly select values that affect the aesthetics and the data of the scatter plot. We used the Python library Matplotlib [10] to generate the plots since it allows for easy extraction of the bounding boxes of the points, tick marks, and tick values. The factors we used to randomize the plot aesthetics and data are listed below.

1. Plot aesthetics
   (a) Plot style (default styling values e.g. "classic", "seaborn", "ggplot", etc.)
   (b) Size and resolution of the plot
   (c) Number of style variations on the points
   (d) Point styles (markers, sizes, colors)
   (e) Padding between axis and tick values, axis and axis-labels, and plot area and plot title
   (f) Axis and ticks locations
   (g) Tick sizes if using non-default ticks
   (h) Rotation angle of the tick values, if any
   (i) Presence and style of minor ticks
   (j) Presence and style of grid lines
   (k) Font and size of the tick values
   (l) Fonts, sizes and contents of the axis-labels and plot title
   (m) Colors of the plot area and background area
2. Plot data
   (a) Data points distribution (uniformly random, or random around a linear or quadratic distribution)
   (b) Number of points
   (c) Order of magnitude of the X and Y coordinates
   (d) X and Y coordinates ranges around the selected order of magnitude
   (e) Actual values of the X and Y coordinates given the order of magnitude, ranges and distribution

These parameters allow us to build a wide variety of scatter plots, for which we have full ground-truth labels. Some of the plots generated from this procedure are very difficult and sometimes impossible to read, even for a human. For example, the tick values can overlap with each other or with the data points, and the randomly selected font for the tick values can be unreadable. Although we did not eliminate such unrealistic plots from the training set, we did manually remove them from the test set.

2.2 Scatter Plots from the Web

To see how our system, trained on randomly generated scatter plots, generalizes to real charts, we also collected a small test set of scatter plots that were generated by humans. We downloaded 50 scatter plots from a Google image search for "scatter plot". The only inclusion criteria was the presence of tick values without units.

3 Methodology

We take as input the image of a scatter plot, and the output is a set of the $(X_{\text{pred}}, Y_{\text{pred}})$ chart coordinates of the data points detected. The pipeline is as follows:

1. Use the object detection model to find bounding boxes for the tick marks, tick values, and points.
2. Apply OCR on the images inside the bounding boxes of the tick values.
3. Find the closest tick mark to each tick value.
4. Use clustering to assign each (tick mark, tick value) pair to either the X or Y axis.
5. Apply robust regression to determine the mapping from pixel coordinates to chart coordinates, for each axis.
6. Apply the mapping to the pixel locations of the detected points to build the table of chart coordinates.

Below we expand on the most important steps.

3.1 Object Detection

Object detection is the task of putting bounding boxes around objects that appear in an image. For our task, we use object detection to localize points, tick marks, and tick values. There is a wide variety of object detection models, but all the state-of-the-art methods use deep learning. We chose ReInspect [23] as our object detection method because it is very effective in crowded scenes, and the points in scatter plots are often very close together, even overlapping. ReInspect uses the OverFeat algorithm [19] but adds a long short-term memory network at the end of it, such that predictions are not made independently of each other. We used the Tensorflow implementation of ReInspect, TensorBox[3]. It is standard practice to initialize the core of the model with a pretrained convolutional network, for which we selected Google Inception V1 [24]. We trained three separate models: one for tick marks, one for tick values, and one for points. We resized our scatter plot images to 800×800 pixels for the tick mark and tick value models, and to 1440×1440 for the point model to help resolve a high density of points. Each model was trained for 5 epochs on a Geforce GTX Titan X GPU, and the total training time was about 60 h. Given an image, a model outputs a set of bounding boxes for the object it is detecting, along with a confidence score for each. We only keep detections with a confidence above 0.3, based on validation experiments.

We can evaluate the performance of the object detection on the procedurally generated test set. We define a true positive as a predicted bounding box which has an intersection-over-union ratio with a true bounding box above 50%. We can then vary the threshold on the confidence of the predicted bounding boxes to generate recall-precision curves, see Fig. 2. We get an average precision of 87.1%

[3] https://github.com/TensorBox/TensorBox.

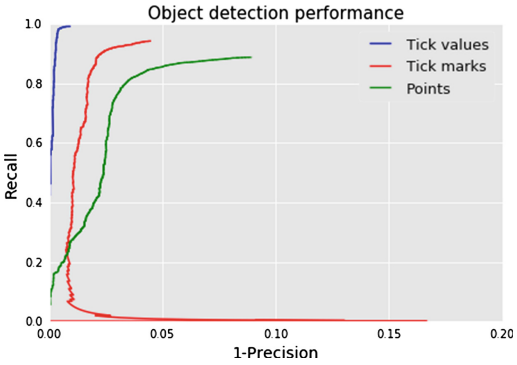


Fig. 2. Recall-Precision curves for the object detection.

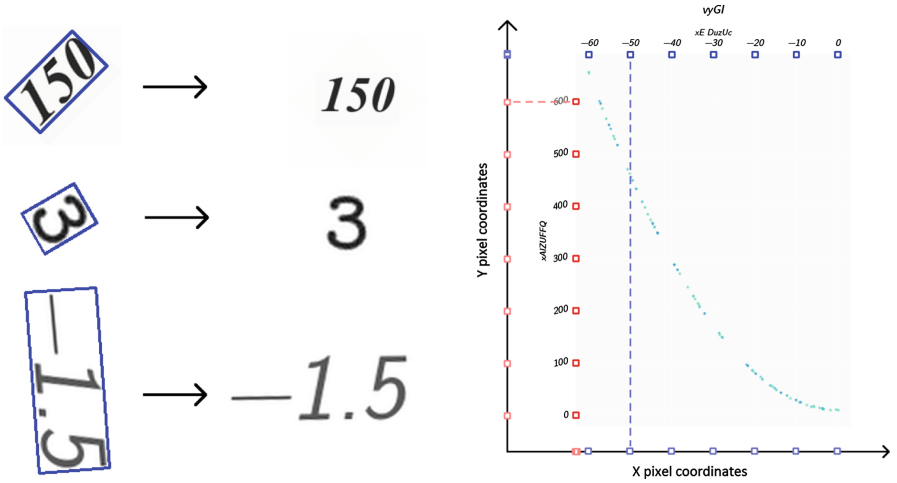
for the detection of points, 99.1% for the detection of tick values and 93.0% for the detection of tick marks. The detection of points is the most difficult, particularly for plots with many overlapping points. However, in the end-to-end system, failures on the detection of tick marks and tick values can be much more costly, since they are essential for determining the pixel-to-chart coordinate conversion.

3.2 Optical Character Recognition

To extract the tick values, we apply OCR to the content of the bounding boxes produced by the tick-value model. We use Google's open source OCR engine, Tesseract [22], because of its ease of use. It comes pretrained on several fonts, and while it is possible to retrain it on custom fonts, we used it as is. However, we found that performing several preprocessing steps on a tick-value image before applying Tesseract significantly improves OCR accuracy. We first convert the image to gray scale and then rescale it so that its height is 130 pixels, while maintaining its aspect ratio. The most important transformation, however, is a heuristic procedure to rotate the tick-value images such that they are horizontally aligned, since this is what Tesseract is expecting. This procedure[4] finds the minimum area rectangle that encloses the thresholded tick-value image, and then rotates the image by an angle that takes into account the angle of the rectangle and the number of characters in the tick-value image. The result of this procedure is illustrated in Fig. 3(a). It works well except for tick values that are rotated by precisely $\pm 90°$ or upside-down.

To test the OCR piece, we used true tick-value images along with their corresponding tick values, for each of the plots in the procedurally generated test set. Then, for each tick-value image, we ran our image preprocessing technique,

[4] The procedure is largely inspired by this blog post: http://felix.abecassis.me/2011/10/opencv-rotation-deskewing/.



(a) Illustration of the results from the rotation-fixing heuristics. The blue bounding box represents the minimum area rectangle that encloses the thresholded image.

(b) Illustration of the clustering problem used to assign the tick marks to either the X- or Y-axis.

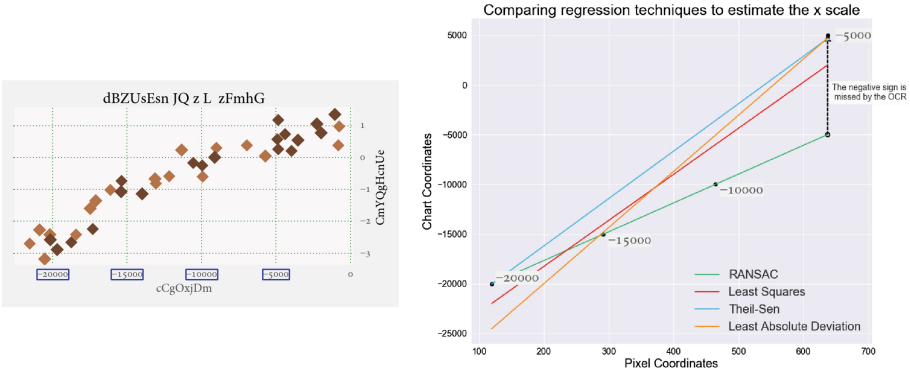
Fig. 3. .

followed by the Tesseract engine, and compared the predicted tick values with the ground truth. This gives an accuracy of 91.2%, but without the image preprocessing the accuracy drops to 63.8%.

3.3 Axis Splitting

Now that we have extracted the tick value associated with each tick-value image, we need to associate each value with its corresponding tick mark and assign it to the X-axis or Y-axis. We make use of the observation that if we project the centers of the tick-mark bounding boxes to the X-axis, the projection of the ticks from the Y-axis are clustered in one area, while the projection of ticks from the X-axis are spread out, as illustrated in Fig. 3(b). This therefore effectively becomes a clustering problem. The algorithm is as follows:

1. Assign each tick value to its closest tick mark.
2. Perform a 1-dimensional DBSCAN [6] clustering algorithm twice: once to obtain a set of clusters for the X pixel coordinates of the ticks, and another one to obtain a set of clusters for the Y pixel coordinates of the ticks. The expected result from the two DBSCANs is one cluster with the X coordinates, these are the ticks that belong on the Y-axis, and one cluster with the Y coordinates, these are the ticks that belong on the X-axis.



(a) (b)

Fig. 4. In Fig. 4(a) above, the OCR fails to detect the minus sign in front of the number −5000. While other regression techniques are thrown off by this outlier, Fig. 4(b) shows that RANSAC regression (visualized by the green line) provides a more robust estimation of the transformation from pixel to chart coordinates. (Color figure online)

3.4 RANSAC Regression

At this point, we need to find the mapping from pixel to chart coordinates. We assume the scales are linear, so the conversion is an affine transformation. For the X-axis, it takes the form $X_{\text{chart}} = \alpha_x X_{\text{pixel}} + \beta_x$, and similarly for the Y-axis. We now create a pair of the form $(X_{\text{pixel}}, X_{\text{chart}})$ for each tick mark[5]/tick value pair we find on the X-axis, and similarly for the Y-axis. For the chart in Fig. 4(a), we have extracted these pairs for the X-axis and plotted them as black points in Fig. 4(b). In principle, we can determine our affine transformation just by performing linear regression on these points. However, standard least-squares regression is very sensitive to outliers, which means that a single OCR error can cause a complete failure to estimate the transformation. We therefore explored alternatives that are more robust to outliers, such as least absolute deviations (LAD) regression, Theil-Sen Regression, and RANSAC regression. While all three fared better than linear regression, we obtained the best results with RANSAC regression [7]. RANSAC regression is an iterative algorithm in which linear regression is applied to samples of the data points in order to detect inliers and outliers from the out-of-sample data points. The model with the most inliers and the smallest residuals on the inliers is chosen. For RANSAC we need to choose a loss function, as well as a maximum residual $R_{\max}$ beyond which a point is classified as an outlier. We use a square loss function, and choose $R_{\max}$ as the median absolute deviation of the tick values squared and divided by 50, such that for the X-axis:

$$R_{\max,x} = \frac{\text{median}\left(X_{\text{chart},j} - \text{median}(X_{\text{chart}})\right)^2}{50}. \tag{1}$$

[5] We use the center of the tick-mark bounding box as the tick mark location.

We apply RANSAC regression for both the X- and Y-axes, such that we end up with a set of 4 parameters $(\alpha_x, \alpha_y, \beta_x, \beta_y)$ for each scatter plot. It is then straightforward to apply these affine transformations to the pixel coordinates of the center of the bounding box of each point detected, and thus obtain their location in chart coordinates. Figure 4 shows how RANSAC improves the estimate of the affine transformation compared to other regression techniques.

4 Results

4.1 Performance Analysis

To evaluate the performance of the end-to-end system, we need to have an evaluation metric. We define true positives as predicted points that are within 2% of true points in chart coordinates:

$$\frac{|X_{\text{pred}} - X_{\text{true}}|}{\Delta X_{\text{true}}} <= 0.02 \quad \text{and} \quad \frac{|Y_{\text{pred}} - Y_{\text{true}}|}{\Delta Y_{\text{true}}} <= 0.02 \tag{2}$$

where $\Delta X_{\text{true}} = \max_j(X_{\text{true},j}) - \min_j(X_{\text{true},j})$ and $\Delta Y_{\text{true}} = \max_j(Y_{\text{true},j}) - \min_j(Y_{\text{true},j})$ are the true ranges of values for the X and Y chart coordinates. We compute all pairs of distances between predicted and true points, and we first select the predicted points closest to each true point in order to find true positives. If a pair of nearby predicted and true points satisfy the true positive criteria, both true and predicted points are removed from the true and predicted set such that we do not overcount true positives. We repeat this process until either no points are remaining or none of the closest points satisfy the true positive criteria. With this definition we can evaluate the precision and recall for each plot. Taking the average across all plots in the procedurally generated test set, we end up with an average precision of 88% and an average recall of 87%. We can also compute the F_1 score for each plot. Figure 5 shows the distribution of F_1 scores across all plots in the procedurally generated test set. Based on Fig. 5, we can see that plots either have a very high F_1 score or a very low one. The latter case indicates that the pixel/chart coordinate transformation was not determined correctly. We can therefore define a successful data extraction for a plot if the F_1 score is above 80% for that plot. By this definition, our method is successful for 89% of the plots in the procedurally generated test. We can also use this definition to evaluate alternative versions of our method. In Table 1, note that the version that uses linear regression on 2 tick values instead of RANSAC is analogous to what [3] used to find the conversion factors between pixel and chart coordinates. RANSAC has a net advantage over the other methods, giving an absolute performance boost of 5.4% over Theil-Sen, the next best method. We note that in its current implementation, Scatteract requires 3.5 s to extract the data from a single plot.

Without the ground truth available for the test set from the web, we cannot evaluate it as systematically as the procedurally generated one. However, a visual inspection of the plots is sufficient to determine if the data extraction is successful. To quickly see if the axes were correctly decoded, we sampled predicted

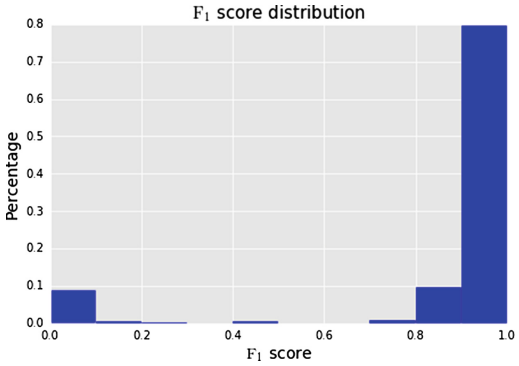


Fig. 5. F_1 score distribution across the plots in the procedurally generated test set.

Table 1. Test results on variations of our system.

System	Success rate
Scatteract	**89.2**%
Scatteract without image preprocessing before OCR	43.3%
Scatteract with Theil-Sen instead of RANSAC	83.8%
Scatteract with LAD instead of RANSAC	82.8%
Scatteract with linear regression on 2 tick values instead of RANSAC	70.7%

points and verified that they correspond to actual points on the plots. Moreover, to see if most points were detected correctly, we inspected the predicted bounding boxes of the points directly on the plot image. Through visual inspection we conclude that our method gave a successful data extraction on 39 plots out of the 50, or 78%. As one might expect, this is lower than on the procedurally generated test set due to the fact that these plots occasionally exhibit features which are not present in the training data. These aspects make identifying all the relevant objects more challenging for the object detection.

4.2 Error Analysis

Let us now comment on results for individual plots. First, Fig. 6 shows six successful data extraction plots from the procedurally generated test set, while Fig. 7 displays six successful data extraction plots from the web test set. Note that these plots cover a wide range of features occasionally seen in scatter plots. Second, Fig. 8 displays two unsuccessful data extraction plots from the procedurally generated test set, while Fig. 9 displays two unsuccessful data extraction plots from the web test set. By examining the picture with the predicted bounding boxes and

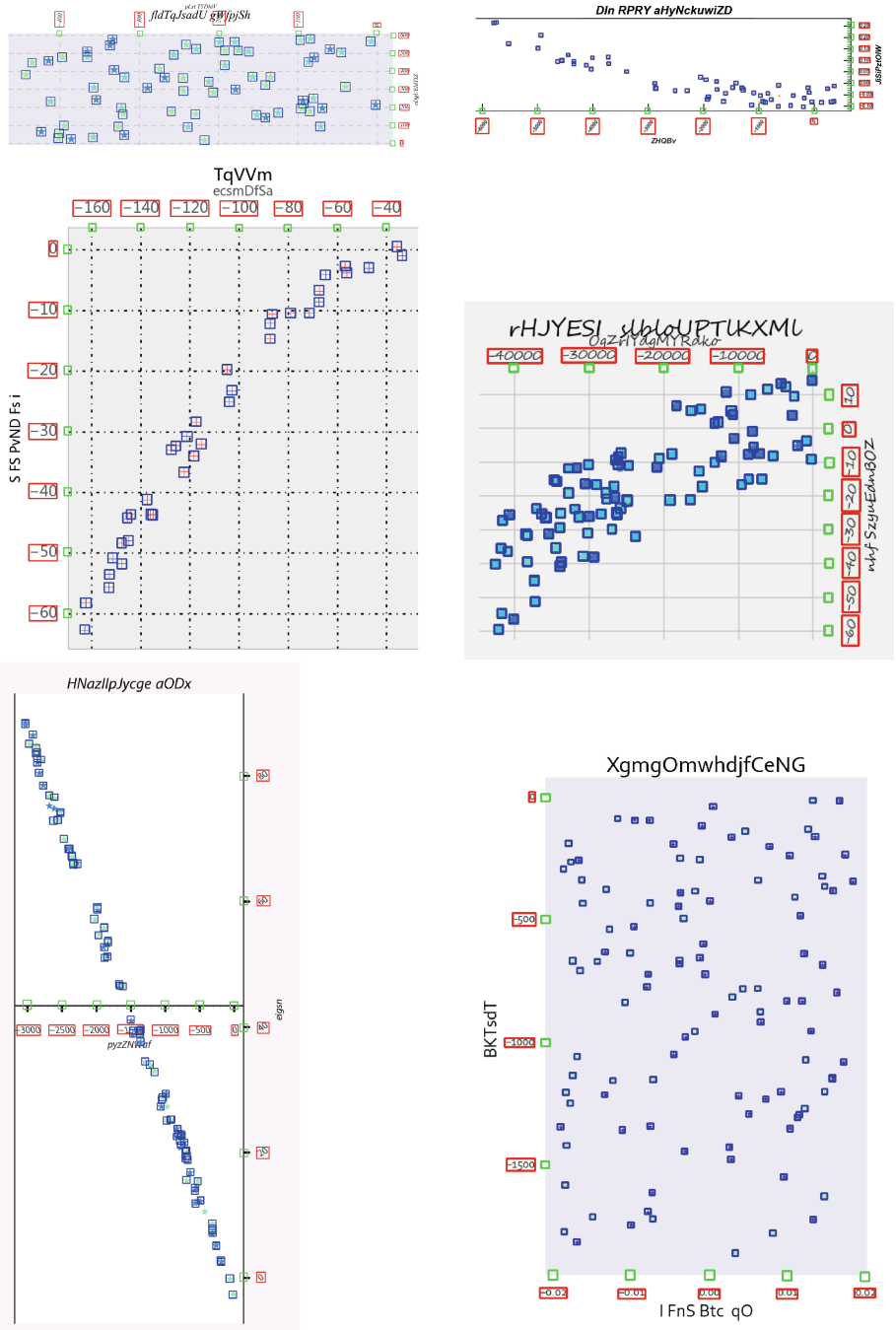
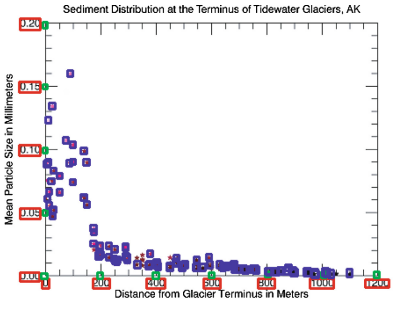
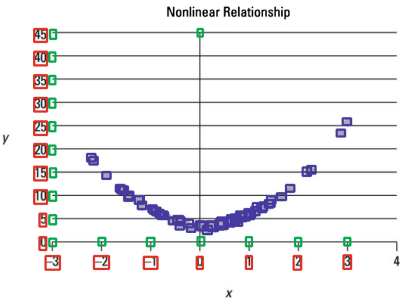


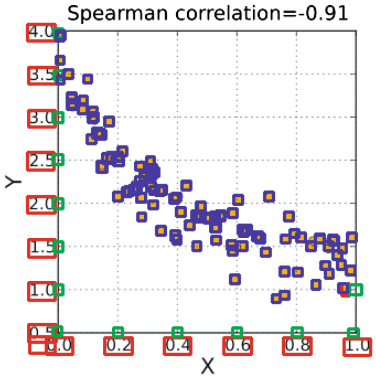
Fig. 6. Successful examples from the procedurally generated test set.



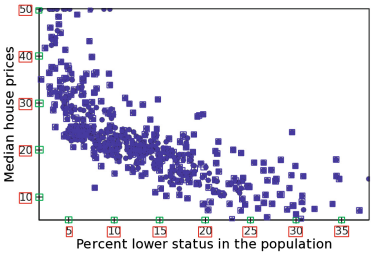
(a) https://www.harrisgeospatial.com/docs/html/images/SedData_Scatterplot.png



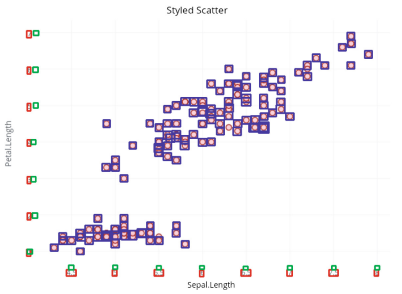
(b) http://d2r5da613aq50s.cloudfront.net/wp-content/uploads/460860.image0.jpg



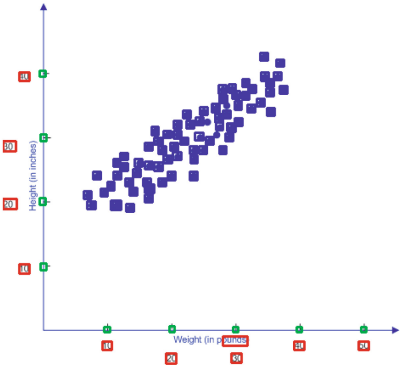
(c) http://www.statisticshowto.com/wp-content/uploads/2013/08/spss-scatter-plot-4.png



(d) http://docs.enthought.com/chaco/_images/scatter_plot.png

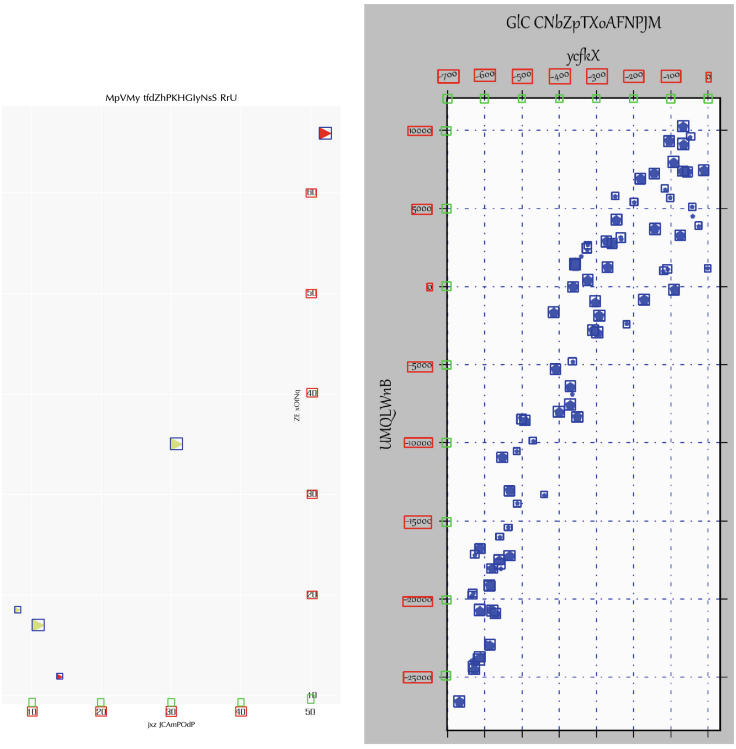


(e) https://plot.ly/~RPlotBot/4326/styled-scatter.png



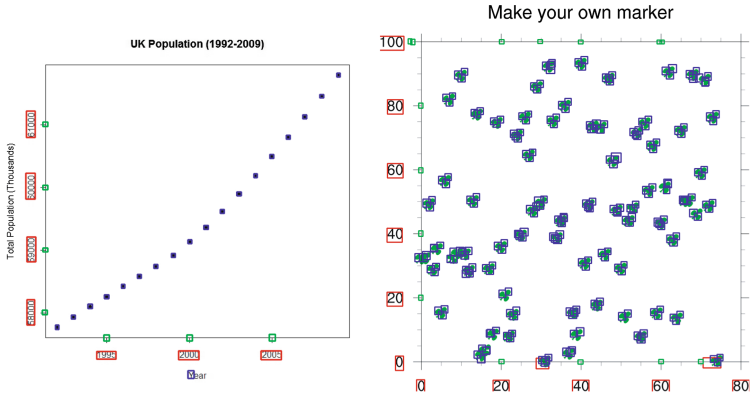
(f) https://statsmethods.files.wordpress.com/2013/05/scatter-plot-2.png

Fig. 7. Successful examples from the web test set.



(a) (b)

Fig. 8. Unsuccessful examples from the procedurally generated test set.



(a) http://www.wekaleamstudios.co.uk/wp-content/uploads/2010/04/scatterplot-base.jpeg (b) https://www.ncl.ucar.edu/Applications/Images/scatter_2_1_lg.png

Fig. 9. Unsuccessful examples from the web test set.

the result of the OCR we can see which part of the system failed on the unsuccessful examples. On Fig. 8(a) the tick detection failed on the Y-axis because it is ambiguous whether the ticks should be on the left or right of the tick values, and on Fig. 8(b) the OCR failed to pick up the minus sign on several tick values of the X-axis. Similarly, on Fig. 9(a) our rotation fixing procedure fails on the 90°-rotated Y tick values, and on Fig. 9(b) the shamrock marker is detected as three separate points by the object detection. Those errors represent well the kind of failure that our method can encounter.

5 Conclusion

In this paper we introduced Scatteract, a method used to automatically extract data from scatter plots. Arguably the most important innovation in Scatteract is the deep learning object detection models used to locate points, tick marks, and tick values, combined with the ability to train these models on procedurally generated scatter plots, for which the ground truth is known. The other main contribution of Scatteract is the pipeline, from the OCR preprocessing to the RANSAC regression, for finding the affine transformation between pixel coordinates and chart coordinates. There seems to be relatively little prior work on this, and we have shown our method to be more robust than the other methods we are aware of in the literature. The end-to-end system is able to successfully extract data from 89% of procedurally generated scatter plots and 78% of scatter plots from the web.

Extending our models to handle new chart aesthetics is straightforward if we can generate training data with the new aesthetics. While the performance on the web test set is promising, it is somewhat worse than performance on the procedurally generated test set. We leave to future work an investigation of how to improve Scatteract's generalization ability. We will also pursue a deeper extraction of information from scatter plots, including point categories, legends, axis titles, nonlinear scales, scales with units, and non-numeric scales, such as dates. The ultimate goal is to extend the method to other chart types as well, so that an end-to-end system would detect charts, classify them (line chart, bar chart, pie chart, scatter plot, etc.), and then use an appropriate pipeline to extract the data from the chart.

References

1. Al-Zaidy, R.A., Giles, C.L.: Automatic extraction of data from bar charts. In: Proceedings of the 8th International Conference on Knowledge Capture, p. 30. ACM (2015)
2. Al-Zaidy, R.A., Giles, C.L.: A machine learning approach for semantic structuring of scientific charts in scholarly documents. In: Twenty-Ninth IAAI Conference (2017)
3. Baucom, A., Echanique, C.: Scatterscanner: data extraction and chart restyling of scatterplots (2013)

4. Browuer, W., Kataria, S., Das, S., Mitra, P., Giles, C.L.: Segregating and extracting overlapping data points in two-dimensional plots. In: Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 276–279. ACM (2008)
5. Chen, Z., Cafarella, M., Adar, E.: Diagramflyer: a search engine for data-driven diagrams. In: Proceedings of the 24th International Conference on World Wide Web, pp. 183–186. ACM (2015)
6. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, vol. 96, pp. 226–231 (1996)
7. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
8. Huang, W., Tan, C.L.: A system for understanding imaged infographics and its applications. In: Proceedings of the 2007 ACM Symposium on Document Engineering, pp. 9–18. ACM (2007)
9. Huang, W., Tan, C.L., Leow, W.K.: Model-based chart image recognition. In: Lladós, J., Kwon, Y.-B. (eds.) GREC 2003. LNCS, vol. 3088, pp. 87–99. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25977-0_8
10. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007)
11. Jung, D., Kim, W., Song, H., Hwang, J.i., Lee, B., Kim, B., Seo, J.: ChartSense: interactive data extraction from chart images. ACM (2017)
12. Kataria, S., Browuer, W., Mitra, P., Giles, C.L.: Automatic extraction of data points and text blocks from 2-dimensional plots in digital documents (2008)
13. Lu, X., Kataria, S., Brouwer, W.J., Wang, J.Z., Mitra, P., Giles, C.L.: Automated analysis of images in documents for intelligent document search. Int. J. Document Anal. Recogn. (IJDAR) **12**(2), 65–81 (2009)
14. Mishchenko, A., Vassilieva, N.: Chart image understanding and numerical data extraction. In: 2011 Sixth International Conference on Digital Information Management (ICDIM), pp. 115–120. IEEE (2011)
15. Nair, R.R., Sankaran, N., Nwogu, I., Govindaraju, V.: Automated analysis of line plots in documents. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 796–800. IEEE (2015)
16. Poco, J., Heer, J.: Reverse-engineering visualizations: recovering visual encodings from chart images. In: Computer Graphics Forum, vol. 36, pp. 353–363. Wiley Online Library (2017)
17. Ray Choudhury, S., Giles, C.L.: An architecture for information extraction from figures in digital libraries. In: Proceedings of the 24th International Conference on World Wide Web, pp. 667–672. ACM (2015)
18. Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., Heer, J.: Revision: automated classification, analysis and redesign of chart images. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, pp. 393–402. ACM (2011)
19. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
20. Shadish, W.R., Brasil, I.C., Illingworth, D.A., White, K.D., Galindo, R., Nagler, E.D., Rindskopf, D.M.: Using ungraph to extract data from image files: verification of reliability and validity. Behav. Res. Methods **41**(1), 177–183 (2009)

21. Siegel, N., Horvitz, Z., Levin, R., Divvala, S., Farhadi, A.: FigureSeer: parsing result-figures in research papers. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9911, pp. 664–680. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46478-7_41
22. Smith, R.: An overview of the Tesseract OCR engine. In: Ninth International Conference on Document Analysis and Recognition, ICDAR 2007, vol. 2, pp. 629–633. IEEE (2007)
23. Stewart, R., Andriluka, M., Ng, A.Y.: End-to-end people detection in crowded scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2325–2333 (2016)
24. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
25. Tsutsui, S., Crandall, D.: A data driven approach for compound figure separation using convolutional neural networks. arXiv preprint arXiv:1703.05105 (2017)
26. Yang, L., Huang, W., Tan, C.L.: Semi-automatic ground truth generation for chart image recognition. In: Bunke, H., Spitz, A.L. (eds.) DAS 2006. LNCS, vol. 3872, pp. 324–335. Springer, Heidelberg (2006). https://doi.org/10.1007/11669487_29