Parsing Line Chart Images Using Linear Programming

Hajime Kato[1] Mitsuru Nakazawa[1] Hsuan-Kung Yang[2,*] Mark Chen[3,*] Björn Stenger[1]
[1]Rakuten Institute of Technology, Rakuten Group, Inc.
[2]National Tsing Hua University
[3]The University of Tokyo
{hajime.a.kato, mitsuru.nakazawa, bjorn.stenger}@rakuten.com

Abstract

This paper proposes a method for automatically recovering data from chart images. In particular we focus on the task of estimating line charts, as the most common chart type, in a fully automatic way that handles line occlusions, as well as lines of different styles, e.g. *dashed or dotted. For this, we first train a single semantic segmentation network to predict probability maps for each different line styles. We then construct a graph based on this output and formulate the line tracing task as a minimum-cost-flow problem, optimizing a cost function using linear programming. From the traced lines, the axes, and text labels, we recover the numerical values used to generate the chart. In experiments on six datasets, containing both synthesized and crawled images, we show significant improvements over prior work.*

1. Introduction

Chart images are an abundant source of numerical data on the web. These charts were typically created for data visualization, however the original input data may not be directly accessible. Automatically recovering this data allows further processing and analysis, chart redesign, as well as building large-scale systems to automatically reason and provide insights based on this data [30, 39, 41, 42]. Chart components form spatial and semantic relationships, which need to be understood in order to interpret the data [30]. These relationships depend on the chart type. Chart components themselves vary in appearance and location, or can simply be absent. Reconstruction of the underlying data is challenging due to differences in chart layout, occlusion of chart components, and image compression artifacts [11, 44]. Many systems require clean images for value extraction, such as the *ReVision* system, which is able to parse different chart types and returns chart visualizations of the same data in alternative styles [39]. Numerous practical systems require manual input, allowing reliable data entry at the expense of being time consuming[1]. Seminal work on full automatic chart analysis is *FigureSeer*, a question-answering system for figures in academic papers [41]. It localizes figures and classifies them into different categories using a ResNet-50. Line charts are further analyzed by extracting lines and associating each of them with the legend data, which is required for the method to work. Lines are then iteratively traced and removed using dynamic programming. In this work we propose a method that (a) estimates all lines simultaneously by optimizing a cost function using linear programming, and (b) removes the requirement of figure legends. As shown in Fig. 1, we estimate pixel-wise probabilities for drawing styles and chart components using a semantic segmentation network. Using these probability maps, we construct a graph by estimating the value ranges of lines, and placing nodes at pixels of high foreground probability. Numerical values are obtained by estimating axis ticks, if available, or the coordinates of text labels, as estimated by text detection and analyzed by OCR. We show that the proposed method advances the state of the art in line chart extraction, automating the process to obtain values from images.

In summary, the main contributions of this paper are as follows: (1) We propose the first framework to automatically extract values from line chart images using a single segmentation network. (2) We introduce a new line-tracing method based on graph flow cost minimization. (3) We do not require figure legends or axis tick-marks, but use them for increased precision when available. (4) Our method achieves state-of-the-art accuracy on six datasets.

2. Prior work

In this section, we review related work on chart analysis and its applications. More comprehensive surveys can be found in [17, 31].

*work conducted during an internship at Rakuten Group, Inc.

[1] *e.g.*, DigitizeIt, graphreader.com, UN-SCAN-IT, WebPlotDigitizer

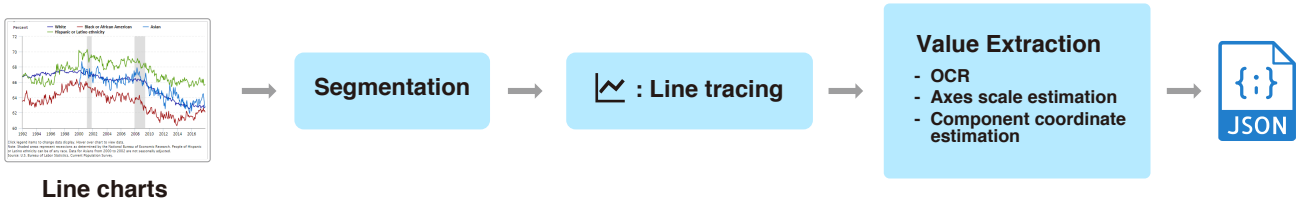


Figure 1: Overview of our method. *We estimate pixel-wise probabilities of different drawing styles using a segmentation network, build a graph and apply line tracing using linear programming. In order to obtain precise coordinates and estimate the chart values, we read the text labels using OCR and estimate the axes scales.*

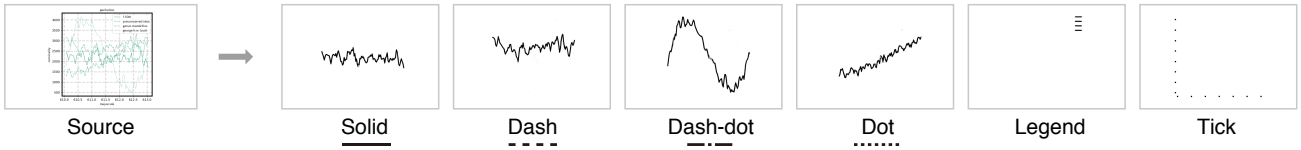


Figure 2: Semantic segmentation network. *Our segmentation model for tracing line charts outputs six probability maps, four representing different line plotting styles, and two for legends and tick marks, respectively.*

Chart classification and detection. For digitizing general chart images crawled from the web, classification is essential to remove non-chart images first, and select the value extraction method depending on the chart type. Early work took a rule-based approach to detect shapes such as lines or ellipses [18, 19, 28]. Support vector machines have also been applied, using patch-based image features [38, 39], local binary pattern features [35] or line features [32] as input. Deep networks have been shown to achieve high accuracy, using a range of different architectures, such as a simple three-layer CNN [12], an AlexNet-style deep belief network [44], GoogLeNet [16, 21], VGG [9, 30], ResNet [11, 41], MobileNet [10] and Xception [7]. These networks achieve near perfect classification accuracy, however, there can be ambiguous cases, *e.g.* images containing multiple different chart types, or if a chart is embedded within a larger different image. To handle such cases, chart extraction has been formulated as an object detection task, estimating the region of each chart [42].

Data extraction. Value extraction from line charts is challenging, because lines can be thin, overlapping, and plotted in different drawing styles. Early work used image binarization and thinning [36] for single line extraction. Linear and non-linear regression based on extracted data points from the chart region was used in [26]. For the extraction of multiple solid lines, Lu *et al*. [32] proposed a method based on primitive chain coding and a curve construction algorithm. Radhakrishnan *et al*. [35] used a multiple-line tracing method that handles dotted and dashed line as well. In their method, the tracing direction was determined based on color consistency and line smoothness. FigureSeer [41] extracts main chart components using a Siamese network, followed by dynamic programming for optimal path-finding based on both the likelihood of a pixel to belong to a path and the path smoothness. To trace multiple lines, the number of lines is first estimated from the chart legend and each line is traced individually. ChartOCR [33] handles line charts by attaching a convolutional embedding layer to the keypoint extraction branch, to obtain an embedding where features for points on the same line are close compared to points on different lines. Recently, Ma *et al*. [34] applied keypoint detection to line charts using a U-Net like segmentation network. Although these methods based on keypoint extraction work well for simple cases, the accuracy drops for more complex cases where lines are partially occluded.

For data extraction from general chart images consisting of 2D components (*e.g.* rectangles or pie slices), early work detected such elements using line or curve detection [18, 19, 25, 26, 28, 49]. ReVision [39] applied connected components analysis (CCA) and RANSAC-based ellipse fitting to extract bar and pie charts, respectively, and numerous works have followed a similar approach [4, 5, 6, 10, 16, 21]. Liu *et al*. proposed a method based on Faster R-CNN to detect bar and pie chart components [30]. Similarly, the method by Choi *et al*. [15] proposed using YOLO v2 object detectors for chart components. Such detection approaches do not generalize well to line charts. Recently, ChartOCR [33] simplified the task of component detection by detecting keypoints independent of the chart type by using a modified CornerNet with an Hourglass Net backbone. Chart components were detected using rules depending on the chart type. These deep learning methods accurately detect uniformly colored chart components, as shown in [30, 15, 33].

However, detecting components filled with dot or striped patterns remains challenging.

We note that there are a number of practical systems for extracting values from chart images based on user interaction[1]. Line charts can be digitized by manually labeling points and entering at least two values per axis. Value extraction with these systems is reliable, but time consuming [13].

Visual Question Answering (VQA). To answer binary questions from images, Kahou *et al*. [24] released a synthesized dataset consisting of chart images with annotation and question-answer pairs. A VQA model was proposed to extract image features and embed questions via an LSTM and predict the answer using a Relation Network. Kafle *et al*. [22] proposed a method for answering open-ended questions about bar charts by extracting ResNet features, embedding questions with an LSTM. Two sub-networks composed of a classification and an OCR sub-network were adapted for answering generic and chart-specific questions. Other recent work proposes extracting chart components using a Mask-RCNN [14], or image features using a ResNet [40] or Dense-Net [23]. Note that current VQA methods take encoded visual features as input and do not require precise chart value recovery.

3. Method

3.1. Chart classification

We first classify input images into six different classes, including four chart types: *vertical bar chart (v-bar)*, *horizontal bar chart (h-bar)*, *pie chart* and *line chart*. We use two classes for *other charts* and *other images*, respectively, which will not be analyzed further. Given the high accuracy of deep learning classifiers, we use a NASNetMobile [50] as the base network and train it with random weight initialization without pre-training.

3.2. Component probability maps

We estimate probability maps of line chart components [41]. This approach allows us to group areas and lines of different styles, *e.g*. striped or dotted patterns, and provides pixel-accurate estimation for further processing in the following stages. A single segmentation model is used for all chart types. We use a modified FastFCN [47] model with an EfficientNetB3 [43] as backbone to estimate probabilities of six labels: four different line styles (solid, dashed, dash-dotted, dotted), as well as legend marks and tick marks, shown in Fig. 2. An input image of shape $(H, W, 3)$ is first encoded by the backbone network, progressively reduced to a shape of $(H/32, W/32, 1536)$. The activation outputs at each resolution are fed into a joint pyramid upsampling (JPU) module with an output of shape of $(H/8, W/8, 256)$ as the standard structure of the FastFCN. For more accurate segmentation, we introduce a second JPU module with an output of shape $(H/2, W/2, 64)$, which we upsample using convolution layers to obtain a 32-channel image of the original input size. We transform the 32 channels into the 6 target output channels using a convolution and sigmoid activation layer.

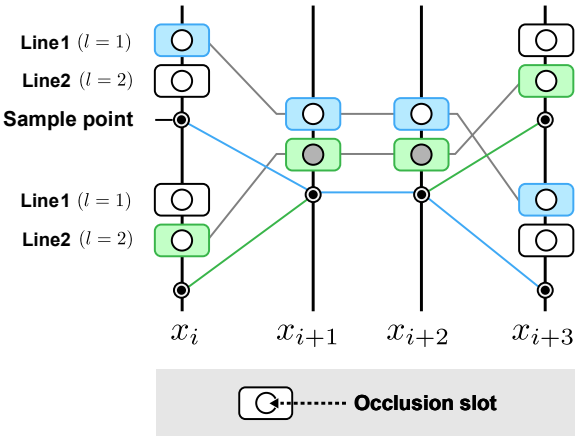


Figure 3: Occlusion model schematic. *At each line connecting two sample points, we consider four possible combinations of occlusion status of two connecting nodes ($z, z' = \{0, 1\}$). The figure represents the final estimate for the case of the blue line partially occluding the green line. Circles represent the occlusion status, where gray means occluded and white means visible (best viewed in color).*

3.3. Line Tracing Using Linear Programming

The line tracing problem is formulated as a network flow problem, where each line corresponds to a path in a graph. The graph is constructed by sampling pixels with high line probability values as nodes, and connecting nodes adjacent in x-direction by edges, as shown in Fig. 3 We solve for pixel-to-line assignments using linear programming (LP) [20]. In contrast to prior work in FigureSeer [41], which uses dynamic programming to trace individual lines sequentially, our method optimizes a cost function for all lines simultaneously.

The input to the line tracing algorithm are the source image $I \in [0, 1]^{H \times W \times 3}$ and the probability maps of the four line styles $S \in [0, 1]^{H \times W \times 4}$. We average the pixel-wise responses of the four line probability maps into a single map as $\overline{S} \in [0, 1]^{H \times W}$. The x-coordinate range of lines is estimated by summing $\overline{S}$ in the y-direction (vertical) for each x-coordinate (horizontal) and binarizing the result by normalization and thresholding.

We estimate the number of lines n_{line} by binarizing $\overline{S}$ as $\overline{S}_{\text{bin}}$

and for each x position counting the number of connected pixel clusters in its y-direction. This results in a vector containing the estimated number of lines at each x position. To account for segmentation errors due to grid lines or legends, we take the number at the top 15 percentile as a robust estimate for the overall number of lines. If a chart contains a legend, we obtain a second estimate for n_{line} from the number of connected components of the legend probability map after binarization. We take the maximum of both estimates as more reliable estimate, as either method tends to underestimate the number of lines.

With estimates of the x-coordinate range and the number of lines, we can formalize the line tracing task using linear programming (LP), analogous to the minimum cost flow problem. We use a binary variable $b \in \{0, 1\}$ to estimate the connection of two points in the binarized segmentation map $\overline{S}_{\text{bin}}$. To reduce computation time, we subsample the x-range use a skip value, d, in our case $d = 3$, at n_{smpl} points $x_1 + kd$, with $k = 0, ..., n_{\text{smpl}} - 1$. Starting from the leftmost x-coordinate x_1, we scan pixels vertically, selecting up to $2n_{\text{line}}$ points for each x_i by progressive farthest point sampling. We minimize the following LP function composed of the cost term c and binary variable b as follows:

$$\sum_{i=1}^{n_{\text{smpl}}-1} \sum_{\substack{y \in Y_i \\ y' \in Y_i'}} \sum_{l=1}^{n_{\text{line}}} \sum_{\substack{z,z' \\ \in \{0,1\}}} c\left(x_i, y, y', l\right) b\left(x_i, y, y', l, z, z'\right), \quad (1)$$

where i is the sample point index. Y_i and Y_i' are the set of y-coordinates at sample points x_i and x_{i+1}, respectively (namely, $Y_i' = Y_{i+1}$). l is the line index. Variables z and z' indicate whether or not the first point (x_i, y) or second point (x_{i+1}, y'), respectively, is occluded. The binary variable $b\left(x_i, y, y', l, z, z'\right)$ indicates whether or not the first and the second point are both on the l-th line.

Figure 3 shows a line tracing example, where a green line is partially occluded by a blue line in the interval $[x_{i+1}, x_{i+2}]$. For each line connecting two sample points, we consider four possible combinations: whether or not the first or the second point is occluded or not ($z, z' \in \{0, 1\}$). The figure represents the correctly estimated z values after solving the linear program defined in (1).

The cost term c of (1) is defined as follows:

$$\begin{aligned} c\left(x_i, y, y', l\right) = \lambda_{\text{col}} c_{\text{col}}\left(\cdot\right) + \lambda_{\text{sty}} c_{\text{sty}}\left(\cdot\right) \\ + \lambda_{\text{fg}} c_{\text{fg}}\left(\cdot\right) + \lambda_{\text{sm}} c_{\text{sm}}\left(\cdot\right) + \lambda_{\text{occ}}, \quad (2) \end{aligned}$$

where $c_{\text{col}}\left(\cdot\right)$ is a color consistency term, $c_{\text{sty}}\left(\cdot\right)$ is a style consistency term, $c_{\text{fg}}\left(\cdot\right)$ is a foreground probability term, and $c_{\text{sm}}\left(\cdot\right)$ is a smoothness term. Each term is multiplied by a corresponding weight, along with an additional bias weight λ_{occ} depending on occlusion flags z and z'. Note that our cost function is similar to that of FigureSeer [41], but with novel foreground and occlusion terms, to enable simultaneous line fitting.

To calculate each cost term, we first estimate the centroids of n_{line} lines in 7D space $[I, S]^{\text{T}}$, the direct sum of source image I and 4D line style space S, where each centroid corresponds to a different line. The centroid for the l-th line is denoted as $[I^l, S^l]^{\text{T}}$. Cost terms in (2) are calculated as

$$c_{\text{col}}(x_i, y, y', l) = \sum_{x^*, y^* = (x_i, y)}^{(x_{i+1}, y')} w_{\text{col}}(x^*, y^*, l)/\Delta \quad (3)$$

$$c_{\text{sty}}(x_i, y, y', l) = \sum w_{\text{sty}}(x^*, y^*, l)/\Delta \quad (4)$$

$$c_{\text{fg}}(x_i, y, y') = \sum w_{\text{fg}}(x^*, y^*)/\Delta \quad (5)$$

$$c_{\text{sm}}(x_i, y, y') = |h|/H\,, \quad (6)$$

where $h = y' - y$, $\Delta = (h^2 + d^2)^{\alpha}$ with a constant α, H is the image height, and (x^*, y^*) are linear interpolations between two sample points (x, y) and (x_{i+1}, y'), where each y^* is an integer.

Weights are calculated as follows:

$$w_{\text{col}}(x^*, y^*, l) = ||I(x^*, y^*) - I^l||_1 \quad (7)$$

$$w_{\text{sty}}(x^*, y^*, l) = ||S(x^*, y^*) - S^l||_1 \quad (8)$$

$$w_{\text{fg}}(x^*, y^*) = \max(0, 1 - ||S(x^*, y^*)||_1), \quad (9)$$

Note that we use bilinear interpolation for $I(x^*, y^*)$ or $S(x^*, y^*)$ as x^* and y^* are floating point numbers.

To complete the LP formulation, equation 1 is minimized subject to the following constraints: (1) The incoming and outgoing degrees for each point and each line are equal. (2) The incoming degree for each non-occluded point for each line is at most 1. (3) The value of b during optimization must be within the range $[0, 1]$. (4) The number of outgoing degrees at the leftmost x-index is n_{line}.

3.4. Value extraction

In the final step we extract values from line chart components. We estimate values using correspondences between pixel coordinates and the numbers obtained by optical character recognition (OCR).

OCR is a two-stage process, first detecting text areas, followed by recognition. For text detection, we use the CRAFT detector [8] and for text recognition, we use STAR-Net [29], a deep residual network with a spatial attention mechanism. To handle rotated text, we apply STAR-Net to the CRAFT output region and its 90-degree rotated version

and select the result with higher confidence score. We experimentally validated the particular choice of detection and recognition networks.

Axis scale estimation. We estimate the scale of coordinate axes to extract numerical values. Applying OCR, we obtain locations of text boxes with numbers, and group them vertically and horizontally using their respective centroid coordinates. If tick marks are detected in the segmentation step, we obtain their locations with high accuracy. In case of missing tick marks the method falls back to using the centroid coordinates of the bounding boxes containing numbers. For robust scale estimation, we use RANSAC regression for each axis, using the pixel coordinates and numbers read by OCR.

Value estimation for chart components. Given the correspondence between image coordinates and values, we can map the chart coordinates to numerical ones to acquire numerical value of all points on extracted lines.

4. Datasets

In our experiments we use synthesized and real image datasets. Synthesized images were generated using a graph plotting library, with automatic ground truth annotation.

4.1. Synthesized image datasets

CHART2019-S (CH-S) dataset. This dataset was published for the 2019 Competition on HArvesting Raw Tables [45]. The images were generated with Matplotlib[2], and chart types include bar charts with simple, grouped or stacked bars, line charts, pie charts, doughnut charts, box charts and scatter charts. Due to the small size of this dataset we used 1,000 line charts for evaluation only.

ExcelChart400K (EXC) dataset. This dataset was published with the *ChartOCR* work [33]. It contains bar, pie and line charts and was generated by crawling public Excel files and generating chart images via API calls. From the EXC dataset, we used 110,000, 10,000 and 4,000 images for training, validation and testing, respectively.

SYN dataset. In order to test our method on a larger variety of drawing styles in terms of layouts, values, colors, and line styles, we generated a new dataset containing bar, pie and line charts using Matplotlib[2], together with ground truth annotations. Title, legends, labels, and grid lines were added, each with 50% probability. The SYN dataset contains 54,000 training images, 6,000 validation images and 4,000 test images, see Fig. 4 for examples. Compared with the CH-S and EXC datasets, the SYN dataset contains more images with different patterns such as dotted or striped element areas, as well as lines plotted with different line styles in the same color.

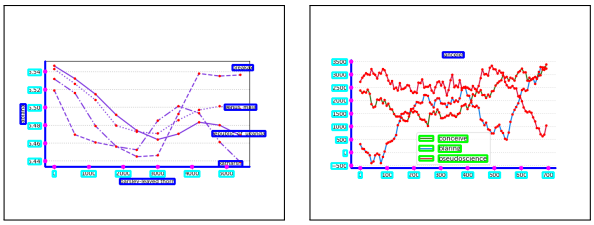


Figure 4: Examples from the SYN dataset, *containing 40,000 automatically generated chart images with ground truth annotation.*

4.2. Real image datasets

FigureSeer (FS) dataset. This dataset contains 997 line chart images crawled from the CiteSeerX website [41]. The images include legend information, which the method in [41] requires for line tracing. Annotations of line point coordinates, text positions, and axis tick marks are available. Due to the limited size of this dataset, we used it only for evaluating line chart analysis.

ICPR-2020-CHART-UB_PMC (PMC) dataset. This dataset was collected and manually annotated from biomedical and life sciences journals, and published for the ICPR2020 CHART-Infographics competition [46]. The dataset includes 15 chart types, from which we select horizontal and vertical bar charts and line charts for the evaluation. Pie chart images were excluded because of the lack of annotations of pie slice components. We used 1,450, 162 and 1,449 images for training, validation and testing, respectively.

GOV dataset. We collected an additional dataset by crawling 4,100 images from government websites containing statistical data[3]. It contains 900 vertical bar charts, 450 horizontal bar charts, 350 pie charts, 1,100 line charts as well as 600 other charts and 700 other images (photos and illustrations). We annotated chart components with bounding boxes in 400 images, 100 for each of the four chart types. Compared to other chart datasets, our web-crawled dataset includes more text and color variations.

[2] https://matplotlib.org/

[3] https://www.usa.gov/statistics

5. Results

5.1. Line Segmentation

Experimental Setup. To train the segmentation model, we use 110,000 chart images from the SYN, EXC and PMC datasets. Some of them are over-sampled or sub-sampled depending on their dataset size.

Data augmentation includes changes of rotation, brightness, contrast, and color balance. We also add compression and Gaussian noise, and add text boxes of different sizes at random locations in the image.

We train the model for 50 epochs using the standard cross entropy loss function and the Ranger optimizer (RAdam [27] and Lookahead [48]) with an initial learning rate of 0.001 and a batch size of 8. After training the segmentation model, the parameters for estimating the plot area and the number of lines, as well as the weights in equation (2) are determined using Optuna [3] within uniform ranges for each weight parameter so that it maximizes the F1 score for line tracing, using 100 images each from the SYN, EXC and PMC datasets, respectively.

Evaluation: To check whether a line is detected correctly, we compute the y-coordinate error for each point on the predicted line. To determine whether or not a line was traced correctly, we apply an error threshold on the F1 score, the same metric as used in [41]. For matching pairs of predicted and ground truth lines, we iterate over the predicted lines and calculate F1 scores with respect to each ground truth line. The pair with the highest F1 score is selected as matching pair and we continue processing the remaining lines. Unmatched predicted lines are counted as false positives, and unmatched ground truth lines are counted as false negatives.

We compare our method to existing methods, FigureSeer [41] and ChartOCR [33]. For a fair evaluation, we use the same experimental settings and evaluation metrics as FigureSeer [41]. The FigureSeer results are taken from the paper [41], and ChartOCR results are obtained using public code provided by the authors [33].

Since the performance of the segmentation network is important for our method, we evaluate different backbone models within the segmentation network in an ablation study, *i.e.*, UNet [37] + EfficientNetB3, and FastFCN + EfficientNetB0. We train these models using the same procedure as FastFCN + EfficientNetB3.

5.2. Qualitative evaluation

Results: Table 1 shows the F1 scores at the line component level. The error tolerance rate is set to 2%. Our method shows a significant improvement over FigureSeer and ChartOCR on all datasets, with an average F1 score of 67.3%. In the comparison of synthesized datasets (the first three: CH-S, EXC and SYN) and real datasets (the last three: FS, PMC, and GOV), the performance of synthesized datasets is higher than that of real datasets. Table 2 shows the result of ablation study comparing alternative segmentation networks. The proposed model (FastFCN+B3) performed best on the most chart types across datasets.

Table 1: Segmentation results *comparing methods in terms of F1-scores on synthesized datasets (the first three: CH-S, EXC and SYN) and real datasets (the last three: FS, PMC, and GOV), where the error tolerance rate is set to 2%.*

Method	Dataset						
	CH-S [45]	EXC [33]	SYN	FS [41]	PMC [46]	GOV	Avg.
FigureSeer [41]	n/a	n/a	n/a	26.4	n/a	n/a	n/a
ChartOCR [33]	16.0	62.8	25.8	25.1	27.6	48.0	34.2
Ours	**67.8**	**76.2**	**72.2**	**65.8**	**68.0**	**53.5**	**67.3**

Table 2: Ablation study results *to evaluate backbone architectures of the segmentation model. The experimental setup is same as that of Table 1. FastFCN+B3 achieved the best performance on average in terms of F1 score.*

Backbone	Dataset						
	CH-S [45]	EXC [33]	SYN	FS [41]	PMC [46]	GOV	Avg.
UNet+B3	**81.7**	66.9	**82.5**	59.8	53.8	40.9	64.3
FastFCN+B0	71.3	65.9	82.1	54.5	49.5	42.1	60.9
FastFCN+B3	67.8	**76.2**	72.2	**65.8**	**68.0**	**53.5**	**67.3**

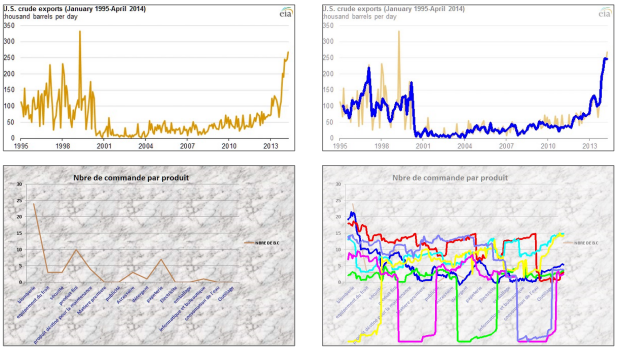
5.3. Value extraction

Evaluation: To evaluate how accurately numerical values can be estimated from chart images, we measure the percentage of charts that have a maximum error below a threshold value. An error tolerance (2%) is used to calculate F1 scores for each line, where a line is regarded as correctly parsed, if the F1 score is higher than 95%. If this is the case for all lines, the complete chart is considered correctly parsed. For the numerical value extraction, the pixel coordinates of line charts obtained in Subsection 5.1 are mapped to the y-axis scale using the value extraction process. In a similar way of evaluation in pixel coordinates (Subsection 5.1), errors between predicted values and ground truth are calculated in numerical coordinates as absolute differences divided by the range of the y-axis, as estimated by minimum and maximum values of y-axis tick labels .

Results: Table 4 shows the percentage of charts for which values could be successfully estimated, given different error thresholds. The low performance on the GOV dataset is mainly caused by the low resolution of its images. In summary, the percentages of charts that can be correctly analyzed at 5% error tolerance ranges from 23-46%, depending

(a) *Original image* **(b)** *ChartOCR [33]* **(c)** *Proposed Method*

Figure 5: Example results. *(Top) A line chart (a) from the EXC dataset [33] with touching lines is parsed correctly using the proposed method (c), while ChartOCR [33] leads to an over-smoothed estimate. (Center) A monotone colored chart from EXC with different line styles is parsed correctly using the proposed method (c). (Bottom) The method is able to successfully parse this line chart from the PMC dataset [46].*



(a) *Original image* **(b)** *Proposed Method*

Figure 6: Failure case examples. *(Top) The method fails to accurately trace a line chart from the GOV dataset with large y-value oscillations due to insufficient number of sample points in the x-direction. (Bottom) Complex patterned backgrounds, as in this example from the EXC dataset [33], can lead to segmentation failures from which the method is unable to recover.*

on the dataset.

For a clearer understanding of which types of line chart images the proposed method can process, we analyze success and failure cases. Figure 5 shows successfully parsed example cases for cases in which keypoint extraction methods fail [14]. In particular, the proposed model is able to handle

Table 3: Percentage of correctly parsed charts *vs.* **error tolerance values**, *in terms of extracted value accuracy. Note that the results of EXC [33] and FS [41] do not exist in this table because they do not contain ground truth of chart values.*

Error	Dataset				
Tolerance	CH-S [45]	SYN	PMC [46]	GOV	Avg.
10%	44.6	54.9	27.5	42.0	42.3
5%	43.8	46.2	22.6	28.0	35.2
2%	37.8	32.4	13.3	4.0	21.9

Table 4: Percentage of correctly parsed charts *vs.* **error tolerance values**, *in terms of extracted value accuracy. Note that the results of EXC [33] and FS [41] do not exist in this table because they do not contain ground truth of chart values.*

Error	Dataset				
Tolerance	CH-S [45]	SYN	PMC [46]	GOV	Avg.
10%	44.6	54.9	27.5	42.0	42.3
5%	43.8	46.2	22.6	28.0	35.2
2%	37.8	32.4	13.3	4.0	21.9

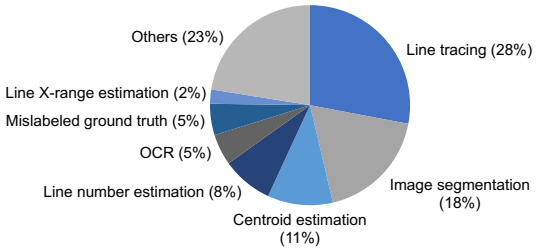


Figure 7: Statistical error analysis. *We analyzed the causes of failure cases on the CH-S, SYN and GOV datasets. Two major reasons are line tracing errors and image segmentation, with examples shown in Fig. 6.*

challenging cases of monotone chart images where lines are plotted in different styles, see bottom of Fig. 5.

Failure example cases are shown in Figure 6. The method does not work well for chart images containing jagged lines, shown in Fig. 6, top. The main reason is the fixed subsampling of the x-value range. The proposed method sometimes also fails to parse chart images with complicated background patterns, see Fig. 6, bottom. The underlying reason is the failure of the segmentation model to estimate accurate probability maps. In such cases, adding a variety of background images during the data augmentation process may improve the results.

To quantify the causes of failures, we analyzed the results of 400 images from CH-S, SYN and GOV datasets. As shown in Fig. 7, the major reasons are line tracing and image segmentation errors, two examples cases shown in Fig. 6. In addition, we found that the method can fail due to inaccurate centroid estimation in 7D space or due to incorrect line number estimates, particularly in the case of background grids. Another error source is the OCR engine, which has a large impact on value extraction as mentioned in Subsection 5.3. In summary, there remain several challenges in this task, and our error analysis shows that multiple improvements are necessary to address these.

Inference speed is another key factor for practical use. Segmentation takes under 1 second on an Intel Xeon Silver 4214 machine with RTX 2080 Ti GPU. Note that this is significantly faster than the FigureSeer approach [41], which requires approximately 40 seconds to compute line probability maps by iterating over each line found in the figure legend. In our case, probability maps are obtained in a single forward pass. The processing time for linear programming rapidly grows with the number of detected lines. We measured average run times of 0.2s, 0.9s, 9.6s, and 285.3s for 1, 2, 4, and 8 lines, respectively. Note that the LP solver we use [1, 2] runs on a single CPU thread and run time may be improved by changing the number of CPU cores or the solver itself.

6. Discussion

This work focuses on extracting main components in line chart images and the values. Currently the method does not aim to extract chart captions or legends. One simple approach could be to associate text with its closest detected component. Value extraction from tick labels can be more complex in the general case. For example, date values need to first be converted to units of time that can be continuously interpolated. Axes may also have a multiplier value, declared separately on the graph, which is applied to all axis values. Values may also skip a range interval, indicated by axis slash marks, in order to reduce the space needed. Handling such special cases can be addressed by post-processing the extracted chart data.

7. Conclusion

This paper proposed a novel framework to automatically recover data from line chart images using a single segmentation network to extract line type probabilities. Line fitting was formulated as cost function optimization using linear programming. An explicit occlusion model allowed us to trace lines simultaneously, leading to higher accuracy on six different chart datasets. We introduced two new datasets with realistic variations, one generated synthetically and one crawled from the web. We also relaxed the requirement for legends or axes tick marks to be present. Finally, we assessed the overall performance of correct data extraction for different error tolerances, analyzed error sources, and highlighted potential areas of improvement.

References

[1] CBC (coin-or branch and cut) mixed integer linear programming solver. `https://github.com/coin-or/Cbc`.

[2] PuLP linear programming library. `https://github.com/coin-or/pulp`.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *In: CoRR*, 2019.

[4] Rabah A. Al-Zaidy, Sagnik Ray Choudhury, and Clyde Lee Giles. Automatic summary generation for scientific data charts. *In: AAAI*, WS-16-01–WS-16-15:658–663, 2016.

[5] Rabah A. Al-Zaidy and C. Lee Gile. A machine learning approach for semantic structuring of scientific charts in scholarly documents. *In: AAAI*, pages 4644–4649, 2017.

[6] Rabah A. Al-Zaidy and Clyde Lee Giles. Automatic extraction of data from bar charts. *In: K-CAP*, pages 1–4, 2015.

[7] Tiago Araujo, Paulo Chagas, João Alves, Carlos Santos, Beatriz Santos, and Bianchi Meiguins. A real-world approach on the problem of chart recognition using classification, detection and perspective correction. *Sensors*, 20:4370, 08 2020.

[8] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. *In: CVPR*, pages 9365–9374, 2019.

[9] Filip Bajic, Josip Job, and Kresimir Nenadic. Chart classification using simplified vgg model. *In: IWSSIP*, pages 229–233, 2019.

[10] Abhijit Balaji, Thuvaarakkesh Ramanathan, and Venkateshwarlu Sonathi. Chart-text: A fully automated chart image descriptor. *In: arXiv*, 2018.

[11] Paulo Chagas, Rafael Daisuke Akiyama, Aruanda Simoes Goncalves Meiguins, Carlos Gustavo Resque dos Santos, Filipe de Oliveira Saraiva, Bianchi Serique Meiguins, and Jefferson Morais. Evaluation of convolutional neural network architectures for chart image classification. *In: IJCNN*, 2018.

[12] Paulo Chagas, Alexandre Freitas, Rafael Daisuke, Brunelli Miranda, Tiago Araújo, Carlos Santos, Bianchi Meiguins, and Jefferson Morais. Architecture proposal for data extraction of chart images using convolutional neural network. *In: IV*, pages 318–323, 2017.

[13] Chengliang Chai, Guoliang Li, Ju Fan, and Yuyu Luo. Crowdchart: Crowdsourced data extraction from visualization charts. *In: TKDE*, pages 1–12, 2020.

[14] Ritwick Chaudhry, Sumit Shekhar, Utkarsh Gupta, Pranav Maneriker, Prann Bansal, and Ajay Joshi. Leaf-qa: Locate, encode & attend for figure question answering. *In: WACV*, pages 3512–3521, 2020.

[15] Jinho Choi, Sanghun Jung, Deok Gun Park, Jaegul Choo, and Niklas Elmqvist. Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization. *Computer Graphics Forum*, 38(3):249–260, 2019.

[16] Wenjing Dai, Meng Wang, Zhibin Niu, and Jiawan Zhang. Chart decoder: Generating textual and numeric information from chart images automatically. *Journal of Visual Languages & Computing*, 48:101–109, 2018.

[17] Kenny Davila, Srirangaraj Setlur, David Doermann, Urala Kota Bhargava, and Venu Govindaraju. Chart mining: A survey of methods for automated chart analysis. *In: TPAMI (Early Access)*, pages 1–1, 2020.

[18] Weihua Huang and Chew Lim Tan. A system for understanding imaged infographics and its applications. *In: DocEng*, pages 9–18, 2007.

[19] Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. Model-based chart image recognition. *In: International Workshop on Graphics Recognition*, pages 87–99, 2003.

[20] Hao Jiang, Sidney Fels, and James J. Little. A linear programming approach for multiple object tracking. *In: CVPR*, pages 1–8, 2007.

[21] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. *In: CHI*, pages 6706—-6717, 2017.

[22] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. *In: CVPR*, 2018.

[23] Kushal Kafle, Robik Shrestha, Scott Cohen, Brian Price, and Christopher Kanan. Answering questions about data visualizations using efficient bimodal fusion. *In: WACV*, pages 1498–01507, 2020.

[24] Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Akos Kadar, Adam Trischler, and Yoshua Bengio. Figureqa: An annotated figure dataset for visual reasoning. *In: ICLR Workshop*, 2018.

[25] Jagadish S Kallimani, K G Srinivasa, and Reddy B Eswara. Extraction and interpretation of charts in technical documents. *In: International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 22–25, 2013.

[26] Sarunya Kanjanawattana and Masaomi Kimura. Extraction of graph information based on image contents and the use of ontology. *International Association for Development of the Information Society*, pages 19–26, 2016.

[27] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *In: ICLR*, 2020.

[28] Ruizhe Liu, Weihua Huang, and Chew Lim Tan. Extraction of vectorized graphical information from scientific chart images. *In: ICDAR*, pages 521–525, 2007.

[29] Wei Liu, Chaofeng Chen, Kwan-YeeK Wong, Zhizhong Su, and Junyu Han. Star-net: A spatial attention residue network for scene text recognition. *In: BMVC*, pages 43.1–43.13, 2016.

[30] Xiaoyi Liu, Diego Klabjan, and Patrick N Bless. Data extraction from charts via single deep neural network. *In: arXiv*, 2019.

[31] Yan Liu, Xiaoqing Lyu, Yeyang Qin, Zhi Tang, and Jianbo Xu. Review of chart recognition in document images. *In: International Society for Optical Engineering*, page 865410, 2013.

[32] Xiaonan Lu, Saurabh Kataria, William J. Brouwer, James Z. Wang, Prasenjit Mitra, and C. Lee Giles. Automated analysis of images in documents for intelligent document search. *In: IJDAR*, 12:65–81, 2009.

[33] Junyu Luo, Zekun Li, Jinpeng Wang, and Chin-Yew Lin. Chartocr: Data extraction from charts images via a deep hybrid framework. *In: WACV*, pages 1917–1925, 2021.

[34] Weihong Ma, Hesuo Zhang, Shuang Yan, Guangshun Yao, Yichao Huang, Hui Li, Yaqiang Wu, and Lianwen Jin. Towards an efficient framework for data extraction from chart images. *In: arXiv*, abs/2105.02039, 2021.

[35] Rathin Radhakrishnan Nair, Nishant Sankaran, Ifeoma Nwogu, and Venu Govindaraju. Automated analysis of line plots in documents. *In: ICDAR*, pages 796–800, 2015.

[36] Viswanath K. Reddy and C. M. Kaushik. Image processing based data extraction from graphical representation. *In: International Conference on Computer Graphics, Vision and Information Security (CGVIS)*, pages 190–194, 2015.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *In: CoRR*, 2015.

[38] C. Lee. Giles Sagnik Ray Choudhury, Shuting Wang. Scalable algorithms for scholarly figure mining and semantics. *In: International Workshop on Semantic Big Data (SBD)*, pages 1–6, 2016.

[39] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. ReVision: automated classification, analysis and redesign of chart images. *In: UIST*, pages 393–402, 2011.

[40] Monika Sharma, Shikha Gupta, Arindam Chowdhury, and Lovekesh Vig. Chartnet: Visual reasoning over statistical charts using mac-networks. *In: International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2019.

[41] Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. Figureseer: Parsing result-figures in research papers. *In: ECCV*, pages 664–680, 2016.

[42] Noah Siegel, Nicholas Lourie, Russell Power, and Waleed Ammar. Extracting scientific figures with distantly supervised neural networks. *In: JCDL*, page 223–232, 2018.

[43] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *In: ICML*, pages 6105–6114, 2019.

[44] Binbin Tang, Xiao Liu, Jie Lei, Mingli Song, Dapeng Tao, Shuifa Sun, and Fangmin Dong. Deepchart: Combining deep convolutional networks and deep belief networks in chart classification. *In: Signal Processing*, 124:156–161, 2016.

[45] Christopher Tensmeyer. Competition on HArvesting Raw Tables (CHART) 2019-Synthetic (CHART2019-S) , CHART2019-S_1 `http://tc11.cvc.uab.es/datasets/CHART2019-S_1`.

[46] Christopher Tensmeyer. ICPR2020 Competition on HArvesting Raw Tables (ICPR-2020-CHART-UB_PMC) (ICPR2020-CHART-Info),1,ICPR2020-CHART-Info_1, `http://tc11.cvc.uab.es/datasets/ICPR2020-CHART-Info_1`.

[47] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. *In: CoRR*, 2019.

[48] Michael R. Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. *In: NeurIPS*, pages 9593–9604, 2019.

[49] Yan Ping Zhou and Chew Lim Tan. Hough technique for bar charts detection and recognition in document images. *In: ICIP*, 2:605–608, 2000.

[50] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *In: CVPR*, pages 8697–8710, 2018.