Data and Trend Extraction from Time Series Graphs

Xiaoyi Liu [1] **Diego Klabjan** [2] **Patrick N Bless** [3]

Abstract

Time-series graphs are commonly used to present sequential data and are effective in visualizing trends. However, it is much more difficult for machines to extract the trends than humans. Knowledge extraction that extracts textual and numerical data and trends hidden in numerical values from time-series graphs is important in knowledge management. There are many rule-based and machine learning methods for reverse-engineering different data plots. However, they focus only on data extraction without trend understanding and deal with only one of the two aspects. In this paper, we consider single time-series, and propose a new integrated method that combines conventional image analysis and deep learning techniques that considers both effectiveness and efficiency. Experiments show that this integrated method extracts knowledge from time-series graphs both accurately and efficiently.

1. Introduction

Time-series data is everywhere, such as stock prices or city temperatures, and time-series graphs can be found from business reports to academic papers. The main benefit of a time-series graph is that humans have the ability to understand the trends in time-series data intuitively. But it is much more difficult for machines to understand time-series graphs and their trends only from an image without the raw data, which is often missing in documents. An automatic knowledge extraction approach from time-series graphs can benefit us in data mining from scientific documents, newspapers, and financial reports.

Many studies have been done in the area of extracting data from graphs. The previously mainstream methods for information extraction from charts (Savva et al., 2011; Huang & Tan, 2007) are based on traditional computer vision methods, which rely on complicated human-defined rules and thus are not robust. With the development of deep learning, many models have been proposed to solve the data extraction problem by neural networks in an end-to-end manner.

However, the challenges of this task, information extraction from time-series graphs, are far from solved because styles in graphs have enormous diversity, and time-series lines lack high-level features that are suitable to neural networks.

In our work, we propose a new integrated framework that takes advantage of both conventional computer vision techniques and deep learning techniques. The first step in this framework is to segment the graph area, which is a critical step. We define a straightforward rule using conventional computer vision techniques while adopting an auto-encoder for other edge cases not covered by the rule. As textual features are most high-level in data graphs, we also design an approach to segment the graph area with the help of OCR (optical character recognition). Line detection inside the graph is also done by a combination of rule-based methods and neural networks. With the detected line and OCR results, the data conversion is done by getting the x-y values of the line, in which every pixel inside the graph is projected to the detected axes and their tick marks. The last step is to analyze the time-series trends. Two options are explored, one is by image classification, and the other is by brute force search. Image classification is proposed to reduce the computational time of the brute force search. The fully integrated method predicts the graph bounding box with IoU greater than 97.4% on two data sets. The time-series line is captured with accuracy higher than 95%, and the function type is classified with an accuracy of 99.8%. After data conversion, the extracted data is predicted with less than 3.4% error on both data sets. The average inference time for each image is 12.2 seconds, which means that data mining within a large volume of time-series graphs is possible.

Our main contributions are as follows.

- We propose a new integrated method that combines conventional image analyses methods and deep neural networks. The combination of the two kinds of meth-

[1]Department of Mechanical Engineering, Northwestern University, Evanston, USA [2]Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, USA [3]Intel Corporation, Chandler, USA. Correspondence to: Xiaoyi Liu <xiaoyiliu2021@u.northwestern.edu>, Diego Klabjan <d-klabjan@northwestern.edu>, Patrick N Bless <patrick.n.bless@intel.com>.

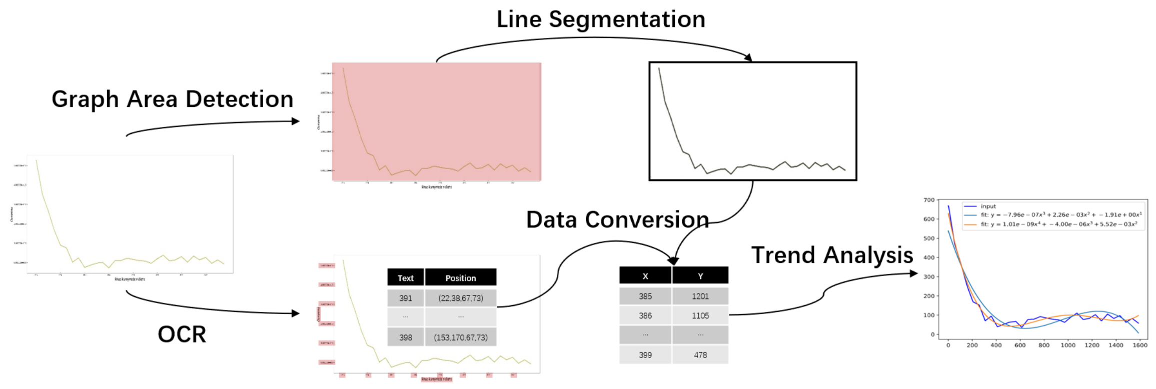

Figure 1. **The framework of the integrated method**: The graph area detection takes an image as input and predicts the bounding box of the graph area; line segmentation then takes the cropped graph area as input and outputs the line pixel locations, which are combined with the OCR results to provide the x-y values based on the axis tick marks. The final step is to analyze trend based on the x-y values.

ods helps it extract knowledge accurately and quickly.

- In time-series graph area segmentation, the huge diversity of time-series graphs prevents conventional rule-based methods from extracting the lines well. We train a neural network to segment a time-series graph, which can be generalized to time-series graphs with arbitrary plotting styles.

- We implement an OCR-based graph area segmentation method, which is the first one to utilize textual features to segment the graph area in data plots.

- We treat the trend analysis as an image classification problem and demonstrate that regression with the guidance of trend classification performs better than without guidance.

In Section 2, related work and methods for data extraction from graphs are reviewed. We show all components of our framework and data generation methods in Section 3. The computational results are discussed in Section 4.

2. Literature Review

2.1. Rule-based Data Plot Knowledge Extraction

Traditional computer vision techniques based on features and rules were mainstream for chart component extraction before the wave of deep learning. Zhou & Tan (2000) combine the Hough transform and boundary tracing to detect bars. Huang & Tan (2007) employ rules to detect chart components using edge maps. Savva et al. (2011) propose Revision, in which bars or pies are detected by their shapes and color information in pixels. These color searching and edge detection methods are efficient but rely on specific features engineered by experts. An advantage of these methods is that the accuracy is guaranteed if the rules apply to the graphs well. However, there cannot be fixed rules that apply to all of the time-series graphs. Due to the enormous diversity of the time-series graphs, many edge cases can not be solved by human-defined rules.

2.2. Deep Learning-based Data Plot Knowledge Extraction

Deep learning provides end-to-end solutions for many computer vision problems. In the area of data extraction from plots, some matured neural networks that were firstly proposed for other computer vision tasks are used to detect graph components, text and recognize them. With the help of deep learning techniques, all the text and chart components can be automatically detected in a model. Cliche & Yee (2017) train three separate object detection models (Stewart & Ng, 2016) to detect tick marks, tick labels, and points in different resolutions, which are finally combined to extract data from a scatter plot. Poco & Heer (2017) employ a CNN to classify each pixel as text or not in a chart and then remove all non-text pixels. The incredible power of neural networks eliminates the complicated procedure of defining rules, but compared to the rule-based methods, neural networks are hard to interpret and lose accuracy since the graphs consist of lower-level features.

However, there is no existing model taking advantage of both rule-based and deep learning-based methods. We propose an integration method that combines the accurate rule-based results and general appliable deep-learning results.

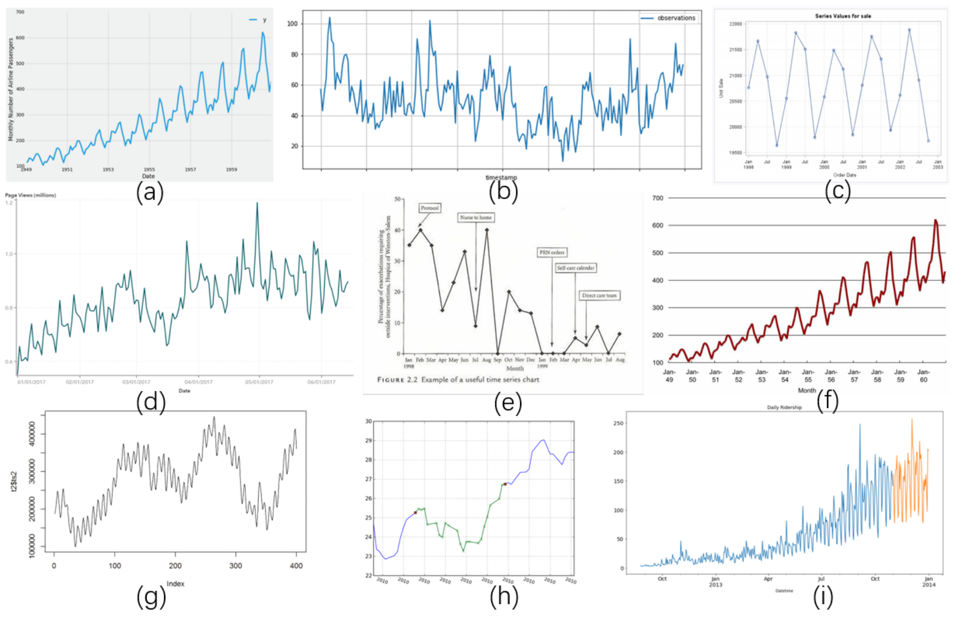

Figure 2. **Diversity of time-series graphs**: The line of time-series can be in one color, or black, or two colors; the outlines of time-series graph can be rectangles, or one or two lines, or missing; the grid lines can be both vertical and horizontal, or only horizontal.

2.3. Image segmentation

Image segmentation is one of the critical problems in the field of computer vision. In image segmentation, every pixel is classified as the class of its enclosing object or background. The most commonly used neural network architecture for image segmentation is auto-encoder. The encoder is usually a pre-trained classification network like ResNet (He et al., 2016), or VGG (Simonyan & Zisserman, 2012) without the last fully connected layer. The decoder is usually a sequence of transposed convolutional layers for upsampling. FCN (Fully convolutional network) (Noh et al., 2015) is one of the earliest image segmentation models following the auto-encoder architecture. U-Net (Ronneberger et al., 2015) is then proposed for biomedical image segmentation, which modifies the original FCN and performs better with fewer training samples. We treat the graph area as a special object in the time-series graphs, so the extraction of the graph area is a segmentation problem.

3. Methodology

The overall framework of our proposed method is presented in Figure 1. The framework consists of five modules: graph area segmentation, line detection, OCR, data conversion, and trend analysis. There are both conventional and deep-learning-based methods for the graph area segmentation, line detection, and trend analysis modules. In the following sections, we first introduce the detailed approaches for these five modules. This is followed by the details of training data and annotation generation.

3.1. Graph Area Segmentation

The first task in this problem is to identify the graph area in a time-series graph image. Accurate segmentation of the graph area is essential for all of the following procedures. The most direct approach is to locate the outlines of the graph area since most time-series graphs have the outlines (see Figure 2 (b,c)). However, the outlines can be vague or even missing in some graphs (see Figure 2 (a)). In such cases, the segmentation module must rely on other clues like texts and legends, or it can be a neural network without human-designed rules. All three kinds of methods are investigated and implemented in our work.

Algorithm 1: Straight line-based graph segmentation

```
Input: Raw Image $I$
Output: Bounding box of graph
        $B = \{h_1, h_2, w_1, w_2\}$
Parameter : $\sigma$ = standard deviation bound
Function Main:
    $\{h_1, h_2, w_1, w_2\}, s \leftarrow$ GetBBoxStd($I$)
    Let $B = \{h_1, h_2, w_1, w_2\}$
    $I \leftarrow I[h_1 = +1, h_2 = -1, w_1 = +1, w_2 = -1]$
    while $s > \sigma$ and $h_1 < h_2$ and $w_1 < w_2$ do
        $B, s \leftarrow$ GetBBoxStd($I$)
        $I \leftarrow I[h_1 = +1, h_2 = -1, w_1 = +1, w_2 = -1]$
    return B
Function GetBBoxStd($I$):
    $L \leftarrow$ set of straight vertical or horizontal lines in $I$
    $B = \{h_1, h_2, w_1, w_2\} \leftarrow$ bounding box of $L$
    for $w \in [w_1, w_2]$ do
        for $h \in [h_1, h_2 - 1]$ do
            $dif(h, w) \leftarrow \sum_{i=1}^{3}[M(h, w, i) - M(h + 1, w, i)]$
        $d_w \leftarrow \sum_h dif(h, w)$
    $s \leftarrow \underset{w}{\text{std}}(d_w)$
    return $B, s$
```

3.1.1. Outlines detection

In many time-series graphs, there are outlines of the graph area. The outlines become the most critical features to segment the graph area. The outlines are determined by the bounding boxes of the straight lines in the time-series images. In Algorithm 1, the function GetBBoxStd of an RGB image or region of an image $I \in R^{H \times W \times 3}$ is to find the bounding box of all the straight lines inside I. The straight lines to find include any vertical and horizontal lines in the image. Value $dif(h, w)$ of each pixel is the RGB value difference between itself and the pixel below it. Summing up dif in the vertical direction, d_w provides the total color changes of column w. For a column in the bounding box, the ideal case is that the column only changes color around the time-series lines and grid lines.

Assuming that the time-series and grid lines are continuous, the standard deviation of d_w should be moderate. We set the value for σ empirically to stop iterative bounding box localization. Otherwise, the algorithm would keep shrinking the bounding box until the standard deviation is small. This iterative procedure is designed to address many time-series images consisting of outlines of the whole image.

3.1.2. OCR and Legend Classification

In time-series graphs, texts are the components with the most high-level features. The state-of-the-art OCR tools show great power in detecting and recognizing these printed texts. We use Tesseract 3.05 to recognize all the texts and their locations. The locations of the tick marks along the x-axis and y-axis can be employed as clues to locate the graph area without any additional computation. The disadvantage is also apparent: missing detections of tick marks would lead to a reduced graph area; some x-axis or y-axis are a bit longer than the range of tick marks. We make the assumption that the x-axis is always on the bottom of the graph, and the y-axis is always on the left.

Algorithm 2 first utilizes OCR models to recognize all texts and their locations in the image. If the algorithm finds more than two bounding boxes sharing the same right bound, these bounding boxes are determined as the y-ticks. x-ticks bounding boxes are determined by finding the most frequent top bounds. Then the graph area is determined by these x- and y-ticks. The left bound of the graph area is the same as the right bound of the y-ticks and the bottom bound of the graph area is determined by the upper bounds of x-ticks. The most right x-tick shares the same right bound as the graph area and the top y-tick shares the same top bound.

Algorithm 2: OCR-based graph segmentation

```
Input: Raw Image $I$
Output: Bounding box of graph
        $B = \{h_1, h_2, w_1, w_2\}$
$\{B_i\} = \{h_1^i, h_2^i, w_1^i, w_2^i\} \leftarrow OCR(I)$
$S_w = \{w_2^i\}_i$ – a multiset
$S_h = \{h_2^i\}_j$ – a multiset
$C_w$ = set of all distinct $(\bar{w}, \bar{n})$ with $\bar{w} \in S_w$ and $\bar{n}$ is its multiplicity
$C_h$ = set of all distinct $(\bar{h}, \bar{m})$ with $\bar{h} \in S_h$ and $\bar{m}$ is its multiplicity
if $\max_i(\bar{n}_i) = 1$ or $\max_j(\bar{m}_j) = 1$ then
    Algorithm fails
else
    $(\bar{w}^*, \bar{n}^*) = \underset{(\bar{w}, \bar{n}) \in C_w}{\text{argmax}}\ \bar{n}$
    $I_{yticks} = \{i | w_2^i = \bar{w}^*\}$
    $(\bar{h}^*, \bar{m}^*) = \underset{(\bar{h}, \bar{m}) \in C_h}{\text{argmax}}\ \bar{m}$
    $I_{xticks} = \{i | h_2^i = \bar{h}^*\}$
    $w_1 = \bar{w}^*$
    $h_1 = \bar{h}^*$
    $w_2 = \max_{i \in I_{xticks}} (w_2^i)$
    $h_2 = \max_{i \in I_{yticks}} (h_2^i)$
return $\{h_1, h_2, w_1, w_2\}$
```

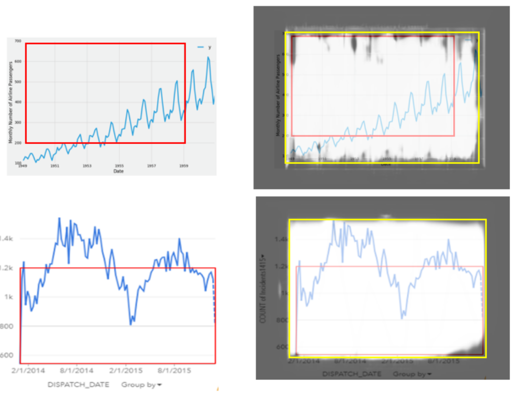

Figure 3. Segmentation cases: The red rectangles are generated by the straight lines-based method, which miss part of the time-series lines, while the yellow rectangles are generated by the U-Net-based method.

3.1.3. Auto-Encoder-based Segmentation

Although the above two methods have the ability to handle most of the time-series graphs, there are cases not covered by them. Another neural network-based approach is necessary to improve coverage: a U-Net to segment the time-series graph image at the pixel level. The task of segmentation is to detect the graph area, but it could be more flexible since the final target is to include the time-series line in this segment. The training details are covered in Section 4.2. Figure 3 shows a real example of the performance of the two methods. The final box is the larger of the two.

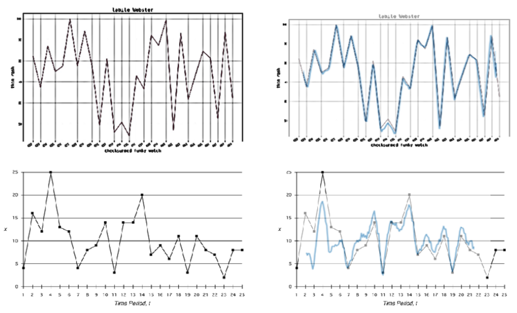

Figure 4. Line detection cases: These two images consist of time-series and grid lines all in black, thus the color-based method fails but the CNN-based method predicts reasonable results, with the lower example more challenging.

3.2. Line Detection

The second task following graph area segmentation is to detect the line inside the graph. This detection procedure is critical to the accuracy of data extraction. In many time-series graphs, the colors of the time-series lines are different from the colors of the background or grid lines. However, there are also some challenging cases such as 1) one single time-series line has more than one color (Figure 2 (h, i)); 2) time-series lines are in the same (black) color as the grid lines (Figure 4). We design two methods to handle the simple cases and general cases.

3.2.1. Color-based clustering

The color is the most distinctive feature for time-series lines in many time-series images. Assuming the time-series lines are in different colors from other components, the pixels inside the graph area are clustered by K-Means (MacQueen, 1967) using the RGB values.

The graph area is cropped from the input image and a K-Means clustering is conducted on all the pixels with respect to their colors (3 feature clustering). Due to the rendering of images, there can be some pixels around the time-series lines, so the selection from clusters is made by counting the number of consecutive pixel sequences in each column in a cluster.

Algorithm 3 exhibits the strategy. For a given clustering with k clusters and a cluster, we find the number of consecutive pixels in the cluster for each column in the box. If the mean of the numbers is close to 1 and their standard deviation is low, the cluster is a candidate. Among all candidate clusters, we select those pertaining to the smallest number of clusters. If there are still many candidates, we select one uniformly at random. When Algorithm 3 does not find a single cluster, then this process fails.

Based on this cluster for each column we designate the line pixel as the average of all y-values of the pixels in the cluster in a column. The values of columns with no pixels in the cluster are interpolated.

3.2.2. CNN-based sequential data prediction

As stated before, black grid lines can prevent the color-based methods from detecting the time-series line (Figure 4). We design a CNN to convert the graph area image $I \in R^{H \times W \times 3}$ to a sequential value $O \in [0, 1]^W$ directly. The sequential value O is the relative position of the time-series pixel in each column. This procedure is only triggered if color-based clustering fails.

3.3. Data Conversion

After the generation of the time-series line by the color-based method if it succeeds, the coordinates of pixels are calculated based on the position of the graph area. If CNN-based sequential data prediction is triggered, the predicted relative position O is combined with the bounding box of the graph area to compute the x-y coordinates for the time-

Algorithm 3: Clustering-based line detection

```
Input: Bounding box $B$
Output: A cluster of pixels
Parameter: $K$ = maximum number of clusters
           $\tau$ = tolerance of mean
           $\sigma$ = standard deviation bound
for $k$=1,...,$K$ do
    Run K-Means with $k$ clusters by clustering the
      pixels in $B$
    $u = \emptyset$
    for each cluster $c$ do
        for each column $w$ in $B$ do
            $M_{c,w}$= number of seqeunces of pixels
              in $c$ in column $w$
        $\bar{m_c} = \underset{w}{\text{mean}} M_{c,w}$
        $s\bar{t}d_c = \underset{w}{\text{std}} M_{c,w}$
        if $\bar{m_c} \in [2-\tau, 2+\tau]$ and $s\bar{t}d_c \leq \sigma$ then
            $u = u \cup \{c\}$
    if $u \neq \emptyset$ then
        Randomly return a cluster from $u$
The algorithm fails.
```

series line.

The x-y coordinates are then converted to the actual x-y values by projecting the coordinate space to the real value space, generated by the OCR results of tick marks. RANSAC interpolation (Fischler & Bolles, 1981) is employed to generate the x-y value of each coordinate.

3.4. Trend Analysis

Trend analysis is the last step in the framework. Most of the previous works in data extraction from plots stop with the step of data conversion. However, the trend is the unique characteristic of time-series graphs and is complicated for machines to understand.

The simplest way to recognize the trend is to test any candidate function by brute force. However, this does not scale with the size of candidate functions. It would be faster if a neural network can determine a function type. The function types of time-series graphs are classified by a lightweight CNN, SqueezeNet (Iandola et al., 2016). The network selects a function type (such as Polynomial, Logarithmic, Exponential, Sine) and then we use MSE fitting to determine the function parameters. For example, the network might select Sine and fitting then determines a, b, c in $a\sin(b*x+c)$.

3.5. Data Generation

We generate data by using the Matplotlib Python library. All titles and labels are assumed to have less than three words from a vocabulary size of 25,000. The font type can be any one of the 35 commonly used ones. Possible font size ranges for each type of text (titles and legend labels) and color choices for time-series lines are arbitrary. Note that in the simulation code, we output all of the necessary ground truth information, e.g., segmentation for the graph area, pixel coordinates of the time-series lines.

The x-y data of the time-series lines are not fully randomized. We use Sine, Logarithm, Exponential, and Polynomial functions with random parameters as the base functions, adding by different levels of noises. The polynomial functions can be of order at most 4.

Twenty thousand simulated time-series images are generated by the above strategy. These images are split into 18,000 training samples, 1,000 validation samples, and 1,000 testing samples. This dataset is referred to as SIMUL in the following sections.

We also manually annotate 20 images downloaded from Google Image, which are used as testing samples to show the generalization of our model. This dataset is referred to as ANNOT in the following sections. For each image in ANNOT, the x-y data is annotated as a sequence consisting of 20 points.

4. Experiments and Results

4.1. Implementation details of conventional methods

In outlines detection, a horizontal straight line must be longer than 0.6 times the width, and a vertical straight line must be longer than 0.6 times the height. Otherwise, short straight lines are not likely to be outlines.

The threshold for terminating Algorithm 1 is set as 1.5. Higher values could result in bigger bounding boxes, while smaller values could shrink into a small rectangle constructed by grid lines.

Another hyper-parameter is the value of K in Algorithm 2, which is set to be 10. There are other parameters including the tolerance of mean τ and standard deviation bound σ, which are set as 0.5 and 1.5, respectively.

4.2. Neural network details

There are three deep-learning models to train in the framework: 1. graph area segmentation U-Net; 2. line detection CNN; and 3. function type classification SqueezeNet. The U-Net follows the same architecture as in the original paper (Ronneberger et al., 2015), and is trained with L_2 loss,

$$L_{seg} = \sum_{x \in \Omega} L_2(l_x, p_x)$$

where l is the ground truth for each pixel and p is the prediction of the pixel class generated by the U-Net.

The bounding box of the graph area is determined by

$$x_1, x_2, y_1, y_2 = \underset{x_1, x_2, y_1, y_2}{\text{argmax}} \, [\underset{x \in [x1,x2], y \in [y1,y2]}{\text{mean}} p(x,y) + \lambda \cdot \underset{x \in [x1,x2], y \in [y1,y2]}{\text{sum}} p(x,y)]. \tag{1}$$

During inference, the bounding box of the graph area is selected by the maximum area of the three different methods. The reason is that we do not want to lose any part of the time-series line in the following steps, while it is tolerable that some additional components are included.

The line detection CNN takes input image I and resizes it to a fixed scale $I_{resize} \in R^{256,256,3}$. The CNN follows the same architecture as ResNet-18 (He et al., 2016) while it consists of filters only with size of [3,1] in every layer and generates a vector $O \in [0,1]^{256}$, trained with L_2 loss,

$$L_{line} = \sum_{i=1}^{256} L_2(O_i^*, O_i) \tag{2}$$

where O_i^* is the ground truth of the relative height for each column in the resized image.

SqueezeNet is trained as a four-class image classification problem with cross-entropy as loss. We consider 4 function types: Polynomial, Logarithmic, Exponential, Sine.

All the implementations are in PyTorch. Because graphs are very different from natural images, we initialize the weights and biases of all three models by Xavier uniform initialization (Glorot & Bengio, 2010).

The three models are trained separately, with the same batch size of 16 and maximum epochs of 50. The initial learning rates are set as $3 \cdot 10^{-5}$ for the Adam algorithm (Kingma & Ba, 2014) in the first two models, while the third model has the initial learning rate of $1 \cdot 10^{-3}$.

4.3. Metrics

We have tested the framework on both SIMUL and ANNOT datasets. There are four metrics designed to measure the model performance. "Graph IoU" is the average IoU between the predicted graph area and the ground truth. "Line Inclusive" is the percentage of the images that the line is entirely inside the predicted graph area. These two metrics are designed to measure the graph area segmentation modules. The accuracy of line detection is measured by "Line L_2" defined in (2). Another metric is "MAPE$_y$," which is MAPE between the extracted values and real data values after data conversion. "L_2" and "MAPE$_y$" take into account every point of the time-series line for the SIMUL data set while they are computed based on 20 points annotated in ANNOT.

4.4. Generalization for Segmentation Model

In order to test the generalization capability of the segmentation model, we generated three more training and validation data sets with fewer varieties in time-series graphs.

The first data set S1 reduces the varieties of fonts and colors: the font type is fixed as Times Roman, font size kept as 12 pixels, and the color for time-series lines can only be red, blue, green, or black. The second data set S2 reduces the varieties of graph styles: there are always border frames bounding the graph area while there is no gridline, and the line width is set as 1 point. The third data set S3 has none of all the above varieties.

Each of these data sets has 18,000 training samples and 1,000 validation samples. Three segmentation models are trained with the same settings as the SIMUL data set, and the trained models are tested on the SIMUL and ANNOT data sets.

4.5. Results

This section summarizes and analyzes the results for generalization experiments and model training, following a computational cost analysis. In the end, we run an inference test on a multi-line time-series graph.

4.5.1. Graph Area Segmentation

In Table 1, we summarize the three metrics for the fully integrated model and ablation of each module in Section 3.1 and Section 3.2. Training and validation sets are based on SIMUL (but are clearly disjoint from test used in Table 1).

Ablation of straight line-based and OCR-based outlines decreases the graph IoU significantly for the ANNOT data set, which means the straight line-based and OCR-based outlines detection is critical. However, the OCR-based outlines detection is not helpful for the SIMUL data set. This implies that our simulated data set can be fully handled by the other two methods, which is not as challenging as the ANNOT data set.

Omission of Auto-Encoder only brings a minor performance loss in the SIMUL dataset but a more significant loss in the ANNOT dataset. This implies that each method does not perform ideally on the ANNOT dataset, but the integrated method performs much better.

The L_2 loss also shows a big difference between the fully integrated model and the models without any graph area segmentation, which is the side effect of inaccurate graph area segmentation.

Table 1. Experiment results and Ablation study

DATASET	METHODS	GRAPH IOU	LINE INCLUSIVE	LINE L_2	MAPE$_y$
SIMUL	FULLY INTEGRATED	**100**	**100**	**0.00012**	**0.4**
	W/O STRAIGHT-LINE DETECTION	98.2	100	**0.00012**	**0.4**
	W/O OCR-BASED SEGMENTATION	**100**	**100**	**0.00012**	**0.4**
	W/O AUTO-ENCODER SEGMENTATION	99.5	100	**0.00012**	**0.4**
	W/O COLOR-BASED LINE CLUSTERING	**100**	**100**	0.00109	1.8
	W/O CNN-BASED SEQUENTAIL DATA PREDICTION	**100**	**100**	0.00045	1.1
ANNOT	FULLY INTEGRATED	**98.3**	**100**	**0.00117**	**3.4**
	W/O STRAIGHT-LINE DETECTION	94.2	95	0.00172	4.9
	W/O OCR-BASED SEGMENTATION	95.6	**100**	**0.00117**	**3.4**
	W/O AUTO-ENCODER SEGMENTATION	88.7	95	0.00201	6.5
	W/O COLOR-BASED LINE CLUSTERING	**98.3**	**100**	0.00398	10.7
	W/O CNN-BASED SEQUENTAIL DATA PREDICTION	**98.3**	**100**	0.00275	8.2

Table 2. Graph area segmentation results for generalization experiments

DATASET	IOU FOR SIMUL	IOU FOR ANNOT
S1	97.8	91.2
S2	93.6	85.7
S3	93.9	85.4
SIMUL	98.2	92.0

In Fig 5, we show two problematic samples that either fail on straight-line based or U-Net based segmentation. The first sample has no vertical straight lines, so the straight-line-based method fails, while the U-Net successfully segments the graph area. The second sample shows that the U-Net misses the left part of the time-series line and the straight-line-based method also makes a wrong prediction by taking the outlines of the whole image. However, the time-series lines in both images are captured accurately. This also implies great robustness of the integrated method.

4.5.2. GENERALIZATION

The "Graph IoU" results for different training data sets with different levels of varieties are shown in Table 2. The similar performance between trained models for S1 and SIMUL, and S2 and S3 demonstrates that the font styles and time-series colors do not augment the data set a lot. However, there is a big difference between S2 and SIMUL, which shows that the varieties of border frames, gridlines, and line width are essential to improve generalization capability.

4.5.3. LINE DETECTION

The significant L_2 loss difference on both datasets between the fully integrated method and the method without color-based line clustering shows that CNN-based sequential data prediction is a good supplement. The integrated framework performs much better than any single method. The annotation errors also lead "LINE L_2" being higher for the ANNOT data set.

4.5.4. DATA CONVERSION

There are also significant performance differences of each method on "MAPE$_y$." Besides the error of line detection, "MAPE$_y$" includes the error of tick marks alignment, OCR, and interpolation. The fully integrated method predicts the real y-value with 0.4% error for the SIMUL and 3.4% error for the ANNOT data set.

4.5.5. FUNCTION TYPE CLASSIFICATION

The function type classification model achieves 99.8% accuracy on the SIMUL data set, which means most function types are predicted the same as brute force searching. We run the classification model on ANNOT data set although there are no labeled function types for these images. The model makes reasonable fittings as shown in Fig 5.

4.5.6. COMPUTATIONAL COST AND TEST ON MULTI-LINE TIME-SERIES GRAPH

Another advantage of this integrated method is its efficiency. We run an inference experiment on a desktop PC with an Intel i5-10400 CPU and a Geforce RTX 3060 GPU. The GPU is used to run the three neural networks, and all other steps are done within the CPU. The average inference time for each image in ANNOT is 12.2 seconds. The most time-consuming step is color-based clustering for line detection, which takes 9.4 seconds on average. Each of the other steps takes less than a half-second.

The results in Fig 6 show that the model has the capacity to be expanded although it is designed to detect a single time-series line. Our method detects the graph area accurately and captures one of the three lines with its legend.

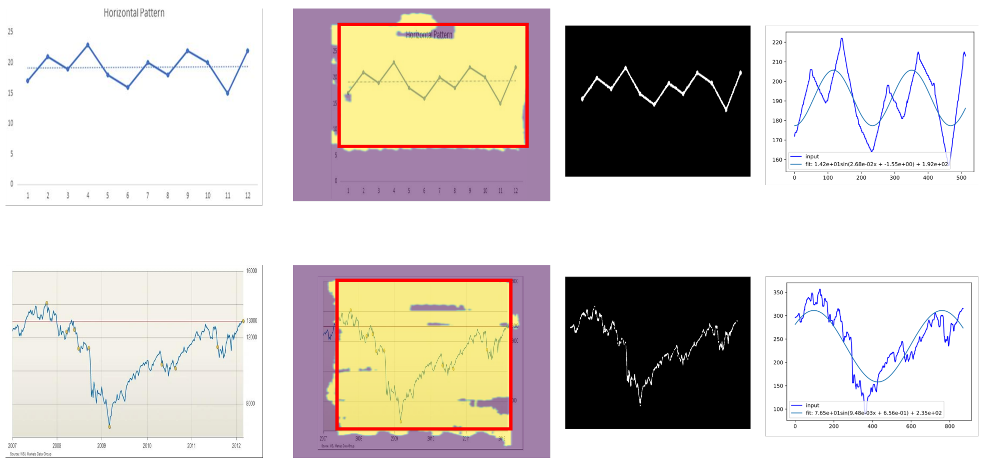

Figure 5. **Results for two sample images from ANNOT**: The first column shows each image; the second column shows U-Net segmentation (yellow) and the corresponding box (red); the third column is the detected line (white); and the fourth column is the fitting result.

5. Conclusion

In this work, a new integrated framework that takes advantage of both conventional computer vision techniques and deep-learning techniques is proposed for the consideration of efficiency and generalization purposes. The experiments show the potential of integrating conventional methods and deep-learning techniques in computer vision, especially for data extraction from graphs. Complexity and diversity of scientific graphs make the task challenging for conventional computer vision techniques, with lower-level features more suitable to be understood by conventional techniques than deep neural networks.

Our model is also efficient by using simple rules and lightweight neural networks, which is much faster than any data extraction tool that needs user interactions.

In future work, it will be interesting to extend our framework to cover multiple time-series, and to include more graph styles in the training data set.

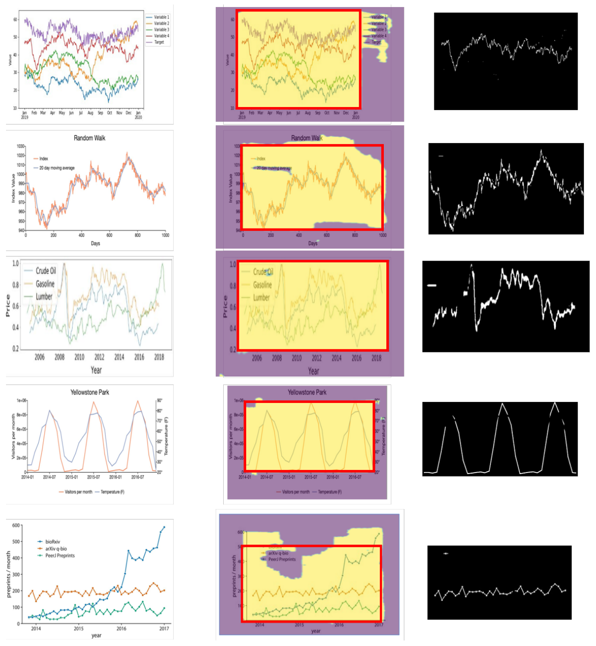

Figure 6. Inference results for multi-line time-series graphs.

References

Matplotlib: Python Plotting. `https://matplotlib.org`.

Tesseract-OCR 3.05.02. `https://github.com/tesseract-ocr/tesseract/releases/tag/3.05.02`.

Cliche, M., Rosenberg D. Madeka D. and Yee, C. Scatteract: Automated extraction of data from scatter plots. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 135–150, 2017.

Fischler, M.A. and Bolles, R.C.J. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Communications of the ACM*, pp. 381–395, 1981.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Huang, W. and Tan, C.L. A system for understanding imaged infographics and its applications. In *ACM Symposium on Document Engineering*, pp. 9–18, 2007.

Iandola, F.N., Han, S., Moskewicz M.W., Ashraf, K., Dally, W.J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 MB model size. In *arXiv preprint arXiv:1602.07360*, 2016.

Kingma, D.P. and Ba, J. Adam: A method for stochastic optimization. In *arXiv preprint arXiv:1412.6980*, 2014.

MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

Noh, H., Hong, S., and Han, B. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015.

Poco, J. and Heer, J. Reverse engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, pp. 353–363, 2017.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pp. 234–241, 2015.

Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., and 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, 2011.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2012.

Stewart, R., Andriluka M. and Ng, A.Y. End-to-end people detection in crowded scenes. In *Computer Vision and Pattern Recognition*, pp. 2325–2333, 2016.

Zhou, Y. and Tan, C.L. Hough-based model for recognizing bar charts in document images. In *Document Recognition and Retrieval VIII*, pp. 333–341, 2000.