Computerized Detection of Trends in Line Charts
6.869 - Fall 2017
Final Project

Anelise Newman
apnewman@mit.edu
6.869

Nathan Landman
landmann@mit.edu
6.869

Abstract

With the massive growth of internet media outlets and news feeds, the amount of data available through visual media, like charts and graphs, is increasing. In order to fully harness this data, we need to be able to extract information from a graph represented as an image. In this paper, we attempt to automatically deduce the salient trend from a line graph. To do this, we generate a synthetic line chart dataset containing 125,000 training images and 12,500 validation images with a variety of styles and underlying data patterns. We also curate a test set containing 527 labeled images from the web. We produce two models–one using traditional vision techniques and the other using deep learning methods–to classify graphs as showing an 'increasing', 'decreasing' or 'neutral' trend. Lastly, we use Tesseract's OCR system coupled with our deep neural network to automatically generate a relevant, searchable, data-driven caption for a graph that will improve communication and recall. [1]

1. Introduction

1.1. Motivation

Graphs and charts are a data-rich way to present information on topics ranging from economic numbers to the popularity of a new song over time. Unfortunately, such data is often stored in a pixel-based graphical format. These visuals are of limited use unless the underlying data, or some properties of it, can be extracted.

The aim of this paper is to develop an automated system that detects the most important information about a line graph: the trend shown by the data. If this information can be efficiently extracted and put to use, the message contained in the visuals can be reinforced.

[1] Code and data can be found at `https://github.mit.edu/landmann/graph-trend-understanding`

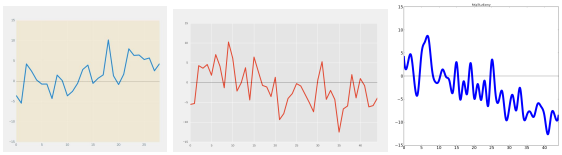

Figure 1: Synthetic images taken from our training set, generated using an underlying linear function. From left to right, the labels are: increasing, neutral, decreasing.

1.2. Related Work

1.2.1 Classifying and extracting graph data

Significant work has been done towards classifying and extracting data from charts. [9] and [1] classify charts based on their design (bar graph, line graph, pie chart, etc.). [9] separates carefully extracted feature vectors using an SVM and achieves an accuracy of 96%.

Progress has also been made towards extracting graph data directly from bitmap images. [9] presents a system for extracting specific numerical data from bar and pie charts. They do not cover how to extract underlying data from a line graph. [7] presents a highly engineered method for extracting textual components, like labels, titles, and axes, from charts.

1.2.2 Datasets and Synthetic Data Generation

[7] provides a precedent for using synthetic data to train and evaluate a model. A large fraction of both their train and test set is composed of visualizations synthesized using the Vega descriptive styling language [8] mixed with graphs taken from other sources. However, with their setup, the train and test data was taken from the same pool of largely-synthetic charts, avoiding the problem of training a model on a set of abundant synthetic data in the hopes that it will generalize to a more varied population of visualizations.

[9] also contributes a data set of charts labeled by type.

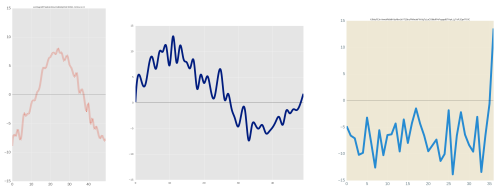

Figure 2: Examples of additional graphs added to our dataset to increase the diversity of the line shapes. The two graphs to the left are generated from a sinusoid, while the one on the right is exponential. From left to right, the labels are: neutral, neutral, decreasing.

We make use of the line graphs from this set as part of our test set.

2. Dataset

In order to train and evaluate our model, we need a large dataset of line graphs labeled with the associated trend. We define three trend classifications:

1. **Increasing:** A significant increase in the plotted value over time
2. **Decreasing:** A significant reduction in the plotted value over time
3. **Neutral:** Either there is no significant difference in the plotted value at the start and end of the graph, or the graph is too noisy to discern a significant trend. Examples include a flat line, one period of a sinusoidal function, or random noise.

2.1. Training and Validation Sets

In order to obtain the amount of data we needed for training, we decided to synthetically generate training and validation data using variations of Matplotlib's pre-set styles [5]. This allowed us to quickly generate graphs with randomly varied characteristics, including background color, presence or absence of grid-lines, aspect ratio, number of points, presence and length of a title, as well as the thickness, color, smoothness, and style (dashed vs dotted) of a line.

We generated data by selecting an underlying function and adding a variable amount of Gaussian noise. Originally, we used lines with a random slope and offset as the underlying function. Because we had the ground-truth data for each graph, we were able to deterministically calculate the appropriate label. The graphs were classified by calculating a linear regression of the noisy data. A graph was considered 'neutral' if it either a) had a p-value of greater than .05, indicating no conclusive correlation, or b) if the slope of the best-fit line was such that the total change in the line over the domain of the graph was less than $1/100^{th}$ of the range of the graph. If the correlation was both significant and large enough, the line was labeled 'increasing' or 'decreasing' based on the slope of the best-fit line. See Figure **??** for examples.

We later expanded our data set to include sinusoidal and exponential graphs (see Figure 2. Sinusoidal graphs covered one period and thus were labeled as 'neutral'. Exponential graphs were labeled as either 'increasing' or 'decreasing' based on the transformations applied to the function.

In total, we generated 125,000 train and 12,500 validation images (for the train set, 100,000 of the images were lines, 12,500 were sinusoidal, and 12,500 were exponents, with a similar breakdown for the validation set). These images are saved as size 224×224, and we provide code to generate full-size figures.

2.2. Test Set

To evaluate our methods, we curated a set of real-world graphs from several sources (Figure 3. The sources include the news site Atlas [6], the dataset put forth in [9], and Google image search results for queries like 'line chart' and other variants. We originally scraped approximately 2531 images. After manually filtering for line charts with a single trend line, we had approximately 599 images. We then each hand-labeled these images. Since the labeling is somewhat subjective, we only kept images where the two of us independently assigned the same label. This gave us a final test set of 527 images. The test set includes images with an enormous variety of styles, including both digitized and hand-drawn representations.

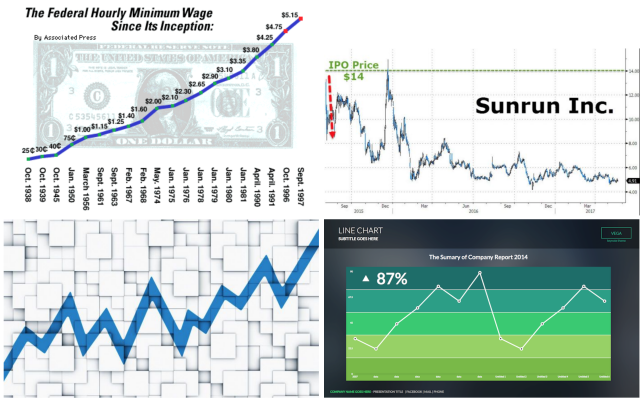

Figure 3: Real-world images from the test set. From left to right, top to bottom, the labels are: increasing, decreasing, increasing, and neutral.

3. Approach

We present two methods for extracting trend data from charts: a pipeline based on traditional computer vision methods, and a deep learning model in the form of a neural network. While the traditional pipeline is more brittle and has more constraints on the input, it is able to extract more data from eligible graphs. The neural network model only outputs the trend classification, yet it is designed to generalize better, relaxing most, if not all, constraints on the styling of the graph.

3.1. Traditional Model

In an attempt to determine how necessary deep learning models are for graphs, we developed a method to process single line charts using traditional machine vision techniques. This model was entirely tuned to fit the synthetic data generated, and later assessed both in and out-of-sample.

The processing pipeline consists of ten traditional image processing techniques that attempt to cleanly segment out the trendline and subsequently classify it using statistical techniques. First, we crop the image to the chart area, crop the title text if necessary, and re-size it to a uniform dimension. We then color quantize the figure in order to reduce noise for a clean separation of background and foreground pixels. The sixth step is to process the foreground by removing grid lines. Seventh, we further reduce noise by applying an original line segmentation algorithm that groups pixels in terms of the segments they compose. Lastly, we convert pixel data into coordinate points, apply a linear regression, and classify the graph using statistical techniques. In this pipeline we used Python to manipulate the images using both PIL and $OpenCV$.

3.1.1 Isolating Chart Area

We first identify the perimeter of the actual chart and crop the image accordingly. This is a crucial step in our pipeline as we must re-size all graphs to the same dimensions in order to have consistency within our labels. We take advantage of the grid lines present in most line graphs to define the minimum and maximum width and height of a graph. In the case of a grid-less graph, the axis suffices to capture the bounding corners of the graphs. Because the axis and grid-lines are, in general, perfectly straight, we first apply a Canny edge detector [3] with Gaussian filter of 3, max value of 150, and min value of 50, to intensify the edges on each line, followed by the Probabilistic Hough Transforms to determine the location of these lines and extract their coordinates [11]. Since some grid-lines are very subtle and blend in with the background very well, the line detection method is applied to a version of the image that has been filtered through a min filter of size 3, and whose sharpness and color are both boosted by a value of 25 in order to accentuate color changes. The boundary lines identified are then used to crop the original image.

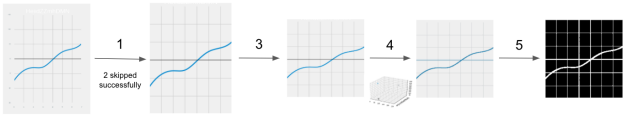

Figure 4: Start of the traditional methods pipeline. An image is first cropped to contain only the graph by cropping to the grid-lines. The image is then resized to 224×244, color quantized, and the background and foreground are separated. In this instance, OCR is not necessary as the title is fully removed using the grid-line removal method.

3.1.2 Removing the title

The Hough Line transform can be fooled by a title full of 'T's. To avoid this pitfall, we used the out-of-the-box Tesseract OCR system to detect text in the cropped images. We then take the bounding box of the text closest to the top but not farther than 10 pixels from it, and crop the image to the bottom of said box (cropping out the title). If the graph contains no title, OCR doesn't detect any boxes that meet this criterion and the process continues.

3.1.3 Re-sizing

We uniformly resized all graphs to 224×224 pixels. The size proves to be large enough for our system to not misclassify the majority of the boundary cases that are still easily classifiable by the human eye. Simultaneously, we keep the image small enough for the algorithm to be speedy while maintaining the signal.

3.1.4 Color Quantization

We implemented a color quantization procedure using the k-means algorithm initialized with random centers and $k = 4$. Because selecting the parameter k is not trivial, we chose 4 for single-trend graphs to represent the colors for the background, the grid, the trend-line, and potentially a differently colored axis. Color quantization is also the first step towards a multi-line segmentation procedure, which is an area for future work.

3.1.5 Foreground Background Separation

To extract the background, we set the most dominant pixel color to zero. We thus produce a binary map to represent the foreground.

3.1.6 Removing Grid-lines

To separate the data from the grid lines, we analyze the rows and columns of the binary map and set to $False$ all the rows or columns that have more than 90% $True$ values. We assume the consequences of misidentifying a flat line as a grid-line, which is a mistake that can potentially be made by humans as well if color selection is not carefully done.

3.1.7 Line Segmentation Algorithm

Line segmentation is done to further eliminate noise stemming from random pixels being colored as foreground yet not quite aligned with any grid-lines. The algorithm goes from top to bottom, left to right, assigning pixels, represented by their (x, y) location, to segment groupings. The algorithm works as follows: we first define a separation threshold, d, to represent the largest gap we will permit between pixels that belong to the same segment. We analyze pixels sorted first by x and then by y. For each unique value of x, we create a dynamic list of groups, each group consisting of y values separated by distance at most d from any of the other pixel in the group. A new group is added if the pixel at hand is d away from any pixel in any group. We proceed by iterating through each group of pixels, comparing it to a list of segments, represented by a dictionary of cached pixels and all pixels. We loop through each segment and assess if any of the values in the group is at least d away from any of the cached pixels. If so, we update the cached value to be the group of pixels and we also add the group of pixels to the list of pixels within the segment. This way, a segment is a list of pixels that are separated by a value of d.

This algorithm works under two assumptions; namely, that the lines within the graph go from left to right, and that they don't cross over. Relaxing the latter assumption is critical, and almost the only improvement necessary for processing multi-line graphs. This presents an exciting area for further research.

3.1.8 Line Graph Reconstruction and Analysis

We now reconstruct the line graph by picking the segments with a total amount of pixels greater than half the width of the graph. Because regression analysis depends on the number of data points forming a line, the number of distinct points we identify in a line is important. We reduce the number of points in the trend-line by taking the average of the y values for each unique value of x. Lastly, we treat these (x,y) values as coordinates in a plot and perform a linear regression to identify a slope and p-value of the graph. A p-value less than 0.05 or a slope of less than 2.5 pixels classifies a graph as having 'no trend', otherwise a negative slope and positive slope are classified as 'decrease' or 'increase,' respectively.

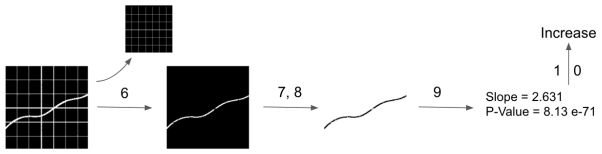

Figure 5: The pipeline concludes by removing the grid-lines and recreating the line using a line segmentation algorithm made from scratch. The pixels are then treated as the values in a coordinate system and are subsequently analyzed using a linear regression. The values indicate that the line is classified as an 'increase.'

3.2. Deep Learning Method

The deep learning model involves a ResNet-18 network [4] with standard pre-trained weights. This model was chosen as it has a quick training time yet is powerful enough to identify the relatively simple features we want to detect.

The biggest challenge encountered was overfitting to the synthetic data. This is problematic because the main advantage of a neural net is its ability to generalize to a wide variety of graphs (see Table 1).

Model results without transforms			
Train Images	Epochs	Val Acc.	Test Acc.
100k	13	95.28%	66.416%

Table 1: Validation and test accuracy for an early version of the model, trained on 100k training images and evaluated on 10k validation images, both generated using only linear functions. Resizing to $(224, 224)$ was the only transformation applied during this round of training. Performance on real-world (test) data is a lot worse than on synthetic data, indicating that the model generalizes poorly.

We mitigated this problem by defining a set of graph-specific transforms designed to mimic stylistic variations encountered in the line chart wilderness. Many data-augmentation techniques that are typically applied to natural images are not applicable to this domain. For instance, images cannot be trivially cropped because that could exclude critical parts of the series in question. Similarly, a horizontal flip of a graph may change its label. However, we do want to teach our network to ignore peripheral elements, such as text labels and legends, that often appear in charts. We also want the model to be resilient to deviations from the clean lines and unpatterned backgrounds that dominate in our train data. Thus, we define our own custom transforms, shown in Figure 6, to mimic these situations. These transforms include: randomly adding text and small polygons to the chart area, randomly adding a large box to the image to mimic legends or partial occlusions, and introducing noise to the image.

To further prevent overfitting, we also decided to reduce

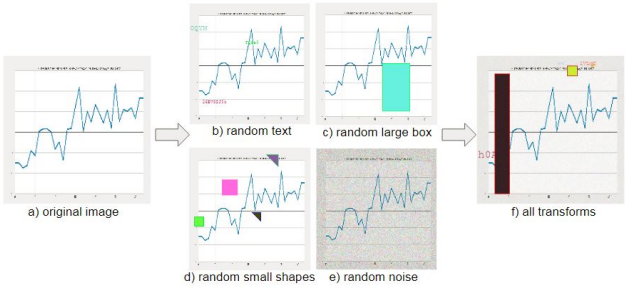

Figure 6: Custom graph-specific transforms applied during training to prevent stylistic overfitting.

the size of our data set, train for fewer epochs, and expand our data set to include more complex curves (as discussed in Section 2.1).

4. Experimental Results

4.1. Traditional Vision Method

Because this model does not use any explicit machine learning and all parameters are hand tuned via human observation, the accuracy of the model is tested on two sets: the validation and the test set. We test on the validation set as it is a representation of how well the model performs under the constraints used to tune it. Similarly, we test on the test set to see how well the model generalizes. The model reaches an accuracy of 76% on the validation set, which, upon inspection, only misses cases that could potentially be ambiguous for humans as well. On the other hand, on the test set it achieved an accuracy of 59%, which is almost double the probability of randomly picking a classification. This demonstrates that graph characteristics are quite general, and carefully crafted features are somewhat able to generalize.

4.2. Deep Learning Method

Our best model produced the following results:

Final Model Results			
Train Images	Epochs	Val Acc.	Test Acc.
50k	8	94.66%	84.061%

Table 2: Pretrained ResNet-18 trained on 50,000 train images composed of noisy lines, sinusoids, and exponential functions with different stylings and a variety of graph-specific transforms.

Adding the extra transforms improved our model by 16.696%. Including more diverse curves improved our model by an additional 0.949%.

Examining our data, we notice some interesting patterns about the successes and shortcomings of our network. Our network generalizes successfully to many graph styles, as

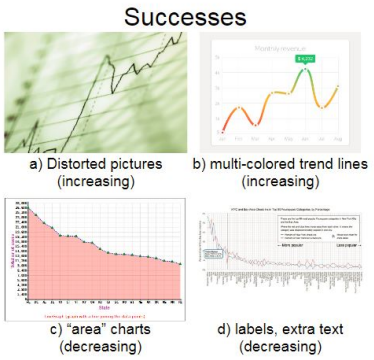

Figure 7: Our network successfully generalizes to a wide variety of styles.

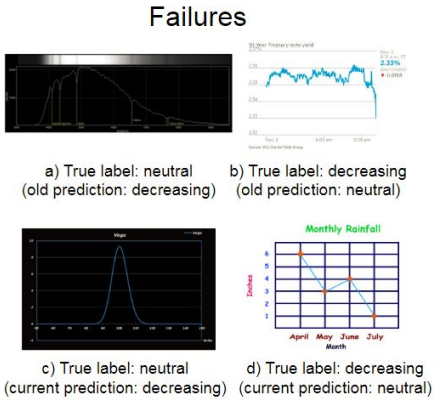

Figure 8: The top row shows examples of graphs that were incorrectly labeled before we expanded our data beyond lines, but are now correctly classified. The bottom row shows images that our best model incorrectly classifies.

shown in Figure 7. This indicates the success of our system for generating varied graphs and introducing relevant transforms. However, it still struggles to correctly classify certain line shapes (Figure 8. For example, it still has trouble with sharp peaks, and it is overly inclined to label jagged graphs as neutral. This reflects the limitations of our method of synthetic data generation using only a limited set of functions.

Future work involves iterating on our data generation and transformation and our model to attain better accuracy on jagged or highly non-linear graphs.

5. Application: Automatic Captioning

Automatic detection of trends in charts, especially when combined with textual elements, can lead to more effective communication and retrieval of rich graphical data.

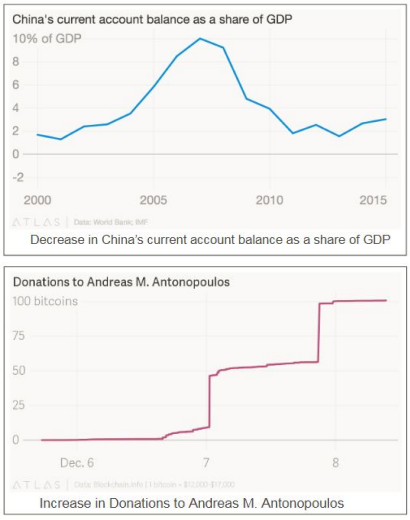

Figure 9: Examples of our automatic captioning application using only the images themselves.

In particular, we demonstrate a simple automatic captioning system that outputs trend-based descriptions. We detect the title of the chart using the out-of-the-box OCR system Tesseract [10] and detect the graph trend using our neural net. We output a relevant caption that takes into account both the subject material and the data presented (Figure 9).

As found in [2], the title of a graph is the most frequently recalled element of a visual, and titles that echo the main message of the graph are more memorable than a bland statement of the subject material. This application can automatically create descriptions that get to the heart of the underlying data, leading to more effective and memorable exposition of data.

6. Conclusion

This paper contains several contributions. We curated a synthetic dataset containing 125,000 train and 12,500 validation images, representing lines, sinusoidal, and exponential functions with randomized parameters, noise, aesthetic style, and titles. Additionally, we scraped and labeled a test set containing 527 real images from the internet from a variety of sources to assess generalization and manually labeled them for assessment.

To process the graph and classify the trends in a graph, we developed two models, one using traditional vision techniques and another using deep learning models. The traditional methods model achieved an accuracy of 59% and the deep learning model an accuracy of 84%, both on the test set. We showed that it is possible to use synthetic data with limited stylistic variants to train a model that can successfully generalize to the universe of graphs.

Lastly, we present a direct application of our model to automatically caption charts, which we hope will improve searchability, recall, and transmission of graphical information.

7. Acknowledgements

We would like to thank the 6.869 staff for inspiring us to tackle machine vision challenges, and equipping us with the toolkit to do so.

References

[1] J. Am, P. Kaur, M. Owonibi, and B. Bouaziz. Convolutional neural network based chart image classification.

[2] M. A. Borkin, Z. Bylinskii, N. W. Kim, C. M. Bainbridge, C. S. Yeh, D. Borkin, H. Pfister, and A. Oliva. Beyond memorability: Visualization recognition and recall. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):519–528, Jan 2016.

[3] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 679–698, 1986.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[6] Q. M. LLC. The atlas. 2017.

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

[8] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, Jan 2016.

[9] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. Revision: Automated classification, analysis and redesign of chart images. In *ACM User Interface Software & Technology (UIST)*, 2011.

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

[11] R. Stephens. Probabilistic approach to the hough transform. *Image and Vision Computing*, pages 66–71, Feb 1991.