Computer Vision-BAsed Glyph and Label Extraction (CV-BAGLE)

Gregory Glatzer, Shivram Iyer, Ian Arakkal

***Abstract*—Data visualizations are commonly used in scientific literature, along with accompanying captions. The generation of these captions, whether by human or AI, involves the interpretation of shapes (glyphs) and text (labels) in the visualization. Using computer vision approaches, along with OCR tools, one can perform Computer Vision-BAsed Glyph and Label Extraction (CV-BAGLE) to improve AI-generated data visualization captions. Additionally, features from the glyphs and labels can be calculated to feed into a classifier to identify the visualization type, which may further aid in caption generation. This study provides techniques for CV-BAGLE to aid in eventual automated caption generation. In this paper, we show the extent at which features computed with CV-BAGLE can be used for chart classification and data extraction. Additionally, we explore the difficulties and limitations of OCR on charts. These results will aid computer vision researchers in best practices for computer vision-aided GLE. In the future, our pipeline could help build a broader system to analyze data visualizations and automate figure caption generation in scientific papers.**

***Index Terms*—Data Visualization, Computer Vision, Machine Learning, OCR, Glyph Extraction**

I. Introduction

In most data analysis, data visualizations play a crucial role. Data visualization typically takes the form of various graphs, charts, etc. They are an important part of scientific writing as they present data in a way that allows individuals to gain insights and relay information in an visual, organized manner. However, some individuals with visual impairments may not be able to gather insights from these data visualizations. These individuals in many cases require the use of screen-reading features which essentially read out text displayed on a screen. Since data visualizations include visual features beyond text, screen-readers are not able to completely gather everything included in the visualizations.

The development of a data science solution that can generate captions and interpret them would greatly benefit readers who use Screen Readers or other accessibility tools. In some settings, there are often limited or no descriptions for charts, and this tool will help people that cannot interpret the chart themselves gain insight from the visualization.

When humans visually interpret a data visualization, they identify visual elements in the visualization as well as text to come to a conclusion about the meaning of the chart. Example visual elements, known as ”glyphs” in the data visualization domain, include bars in bar charts, slices in pie charts, lines in line charts, and points in scatter plots. To aid in caption generation, it would be useful to be able to extract these glyphs from a visualization. Additionally, label (axis ticks and labels, titles, etc.) extraction would provide important data needed to generate captions.

Considering the past work in glyph analysis, glyph extraction, chart classification, and chart caption generation, we propose the contribution of our research. In this paper, we aim to contribute to the growing work of glyph and label extraction (GLE) techniques, as well as the classification of glyphs into types (small and large glyphs). Using extracted glyphs and labels, we also aim to contribute to demonstrating the importance of glyphs in the classification of data visualizations. Although we are not directly tackling caption generation, we can see from related work that chart classification and GLE is an important step to perform to aid caption generation pipelines.

II. Related Works

There are several approaches to the problem of GLE. A computer vision-based approach to extracting glyphs and labels would very much mimic the process humans go through when reading a chart - a human identifies visual elements on the chart and pairs them with the information gained from identifying and reading the labels to help gain meaning from the chart. In contrast, one may take a more abstract approach to extracting out information to generate captions, such as the neural network-based approach found in Balaji et al. (2018), which generates abstract features within the hidden layers of the network. The former approach is much more interpretable and familiar to the traditional way of interacting with charts (by analyzing them with biological systems, i.e. the eyes). With interpretability in mind, we propose a computer vision-based system to extracts out glyphs and labels from data visualizations in order to aid in the caption generation process.

As a precursor to caption generation, a useful step would be to classify a chart to know what information needs to be included in the caption. The combination of glyphs present in a chart greatly influences the type of chart. For instance, Cuadrado-Gallego et al. (2021) attempts to categorize data visualizations using the shape of glyphs as a defining attribute to differentiate between chart types. The shapes considered in this paper were axes, circles, maps, rectangles, networks, and drawings. Regarding the differentiation of different glyph types, there is some past work that has compared different types of glyphs and their attributes. One study by Fuchs et al. (2017) involved an analysis of various glyph types and their roles. This study aimed to standardize glyph design variations and the role of glyphs in the successful design of visualizations. Similarly, Siva et al. (2012) considered the various

visual aspects of glyphs, and how these aspects relate to their functionality in the data visualization. The authors of this paper also considered the importance of shape, color, aspect ratio, etc. in the successful analysis of data visualizations, and by extension their importance in the generation of captions.

Another interesting approach for glyph extraction can be found in DataQuilt - a novel technique that allows data visualization creators to design pictorial visualizations as collages (Zhang et al., 2020). DataQuilt uses computer vision techniques to aid in their feature extraction of important image elements. They use the Python module OpenCV to assist with various computer vision techniques for glyph extraction. They implement GrabCut which allows for distinction between the foreground and the background of an image. They also implement Canny Edge Detection which is a function in OpenCV for detecting edges in an image.

Regarding the broader application of our research towards caption generation, we can see that glyph extraction has been used as input to caption-generation pipelines (Balaji et al., 2018; Mittal et al., 1998). In Balaji et al. (2018), the authors developed a glyph-dependent ML pipeline to generate captions for visualizations using a combination of OCR, neural networks, and NLP-related processes. Additionally, Mittal et al. (1998) out of Pittsburgh and Carnegie Mellon used computer-vision centered approaches to generate captions.

III. Data

In order to train and evaluate a GLE system, an ideal dataset would have visualizations along with the corresponding underlying data to act as a ground truth of the exact positioning, measurements, and text of the glyphs and labels. A dataset comprised of real-world visualizations from many sources, while providing more realistic, heterogeneous visualizations, would most likely not provide the corresponding underlying data.

Some existing datasets of notable size are SciCap (416k visualizations from 290,000 papers) and ReVision (3201 URLs in 16 classes) (Hsu et al., 2021; Savva et al., 2011). SciCap contains visualizations from research papers, along with extracted captions. The ReVision corpus only contains URLs to images of visualizations, as well as the classification of the chart (bar, pie, scatter, etc.). Both datasets, however, do not have the underlying data for the visualizations, which would be needed to evaluate the accuracy of the extracted glyphs and labels.

After considering our options for available datasets, we decided to generate synthetic data visualizations using the Python seaborn library (Waskom, 2021), while retaining the underlying data for each chart, giving us access to the ground truths for glyphs and labels.

A. Synthetic Data Classes

We define 12 chart classes $c \in C$. These classes, along with how many instances we generated, are shown in Table I.

With so many possible charts to choose from, we choose these charts as they appear to be what most of the literature regarding computer vision and charts focuses on, as well as these charts being fairly common charts that one would see often. For each chart class, we generated 6000 samples.

Chart Class	Size
Scatter	6000
Scatter with trend line	6000
Horizontal Bar	5994
Vertical Bar	5990
Stacked Bar	3023
Normalized Stacked Bar	2977
Pie	6000
Histogram	6000
Histogram with KDE	6000
Vertical Box and Whisker	6000
Horizontal Box and Whisker	6000
Line	6000

TABLE I
CHART CLASSES WITH SIZE

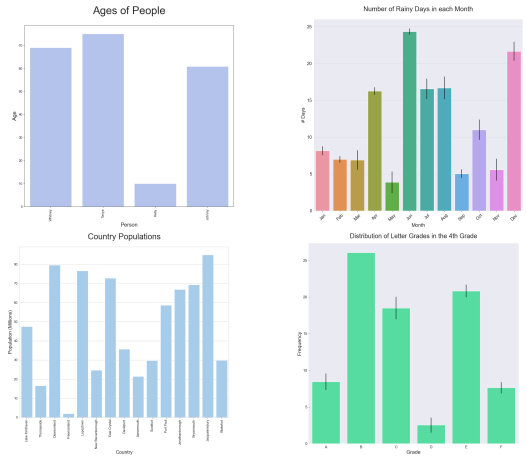


Fig. 1. Variation in synthetic bar charts, including varying label size and rotation, color, and presence of error bars.

B. Data Generation Procedure

In order to generate our synthetic data, we used a combination of NumPy, the Python random library, seaborn, matplotlib, pandas, Faker, and pickle. We developed a set of function that generate random ranges of numbers, which were then used to generate aspects of the charts, such as number of bars in a bar chart, percentages in a pie chart, domain and ranges of the data, etc. To generate labels, we used the Faker library to generate random labels such as addresses, city names, human names, etc. where necessary, such as the categorical labels in a bar chart. Finally, we randomized the aesthetics of the charts, including font size, rotation, and padding, as well as color and other visual aspects of the charts that would allow for a more realistic, heterogeneous synthetic dataset. Figure 1 illustrates the range of variation achieved in the synthetic data visualizations.

While generating synthetic visualizations, the size of the figures needed to be carefully considered in order to ensure that the various tick, title, and label sizes could all fit within the figure. This problem had to be solved on a case-by-case basis, as the matplotlib/seaborn rendering engine results in various chart types taking up different amounts of the figure, which makes choosing a single figure size to generate all charts

on infeasible. Instead, we chose to generate some charts on different figure sizes, and then later scale the images to a common resolution of 1028x1028px.

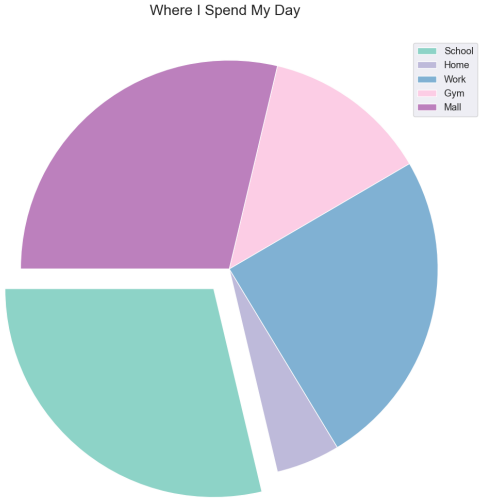


Fig. 2. Synthetic pie chart with ground truth data and labels

Data	
Category	Percentage
School	29%
Home	5%
Work	25%
Gym	13%
Mall	29%
Labels	
Title: Where I Spend My Day	

TABLE II
UNDERLYING DATA AND LABELS FOR FIGURE 2

With synthetic visualizations, we have access to the ground truth labels, as well as the underlying data which contributes to the generation of the glyphs. Figure 2 illustrates a synthetic pie chart, along with the underlying data in Table II. We can use this data to evaluate the accuracy of GLE.

IV. METHODS

Unlike past methods that take a chart-class by chart-class approach to extracting glyphs, such as that seen in Balaji et al. (2018), we take a generalized, computer vision-based approach to the extraction of glyphs and labels. Given a chart, we extract out all of the possible glyphs, including polygons, lines, points, etc. We also extract out all labels, including titles, axis labels, etc. Then, using the properties (number of glyphs extracted, size of glyphs, shape of glyph, etc.) of these extracted glyphs as features to represent the chart, we feed these features into a classification model to determine the chart class. We then output the extracted glyphs and labels, along with the chart type classification. The overall procedure for extracting glyphs and labels from a chart is defined in Algorithm 1.

Our pipeline for extracting out relevant labels and glyphs is illustrated in Figure 3. The rest of this section will elaborate on the implementation of these steps.

Algorithm 1 Chart Classification from Glyph and Label Features

Input: chart image
Output: glyphs, labels, and classification

$L \leftarrow$ Extract labels from chart using OpenCV + pytesseract.
$G \leftarrow$ Extract all possible glyphs from chart using OpenCV.
$F_L \leftarrow$ Calculate label features from L.
$F_G \leftarrow$ Calculate glyph features from G.
$F \leftarrow F_L \cup F_G$
$c \leftarrow classifyChart(F)$
return $\{G, L, c\}$

A. Label Extraction

The extraction of labels from data visualizations was done using the Python wrapper for the Optical Character Recognition (OCR) tool developed by Google, Tesseract (https://github.com/madmaze/pytesseract), called Pytesseract. Pytesseract was originally written by Samuel Hoffstaetter. In order to optimize the performance of Pytesseract, we experimented with adjusting the *psm* parameter, which controls the OCR mode used. We also used multiple preprocessing methods to improve OCR performance. In this section, we will elaborate on the parameter tuning and preprocessing steps used in order to optimize label extraction.

1) Pytesseract Parameter Tuning (psm): When using (Py)Tesseract, one of the most influential hyper-parameters to performance is the *"psm"* (Page Segmentation Mode) option. There are 14 different modes for *psm*, with each mode making different assumptions about the image being fed to the model. Additionally, some options perform "OSD" (Orientation and Script Detection), which help identify text at various rotations.[1]

After testing all of the modes, we found that mode 12 worked best for extracting labels from data visualizations. This makes sense, as mode 12 tries to find sparse text while using OSD. In data visualizations, text is often in small chunks throughout the image, such as axis labels, ticks, titles, and legends. The *psm* mode 12 would not be an appropriate option for a more traditional OCR use case, such as extracting text from a typed document, however the sparse nature of labels in a data visualization lends itself to this option.

2) Preprocessing for Label Extraction: Tuning the *psm* parameter in Pytesseract is often not in itself enough to optimize label extraction. We used a combination of image preprocessing techniques from the popular Python library OpenCV (Bradski, 2000) to remove noise and make the text more apparent for the OCR model . These steps include:

1) Increasing the contrast of the image.
2) Converting the image to grayscale.
3) Applying Otsu's Thresholding method (thresh=0, maxval=255)
4) Removal of large objects based on contour area.

[1] All of the available psm modes, along with their descriptions, can be found at https://stackoverflow.com/a/44632770/12616695

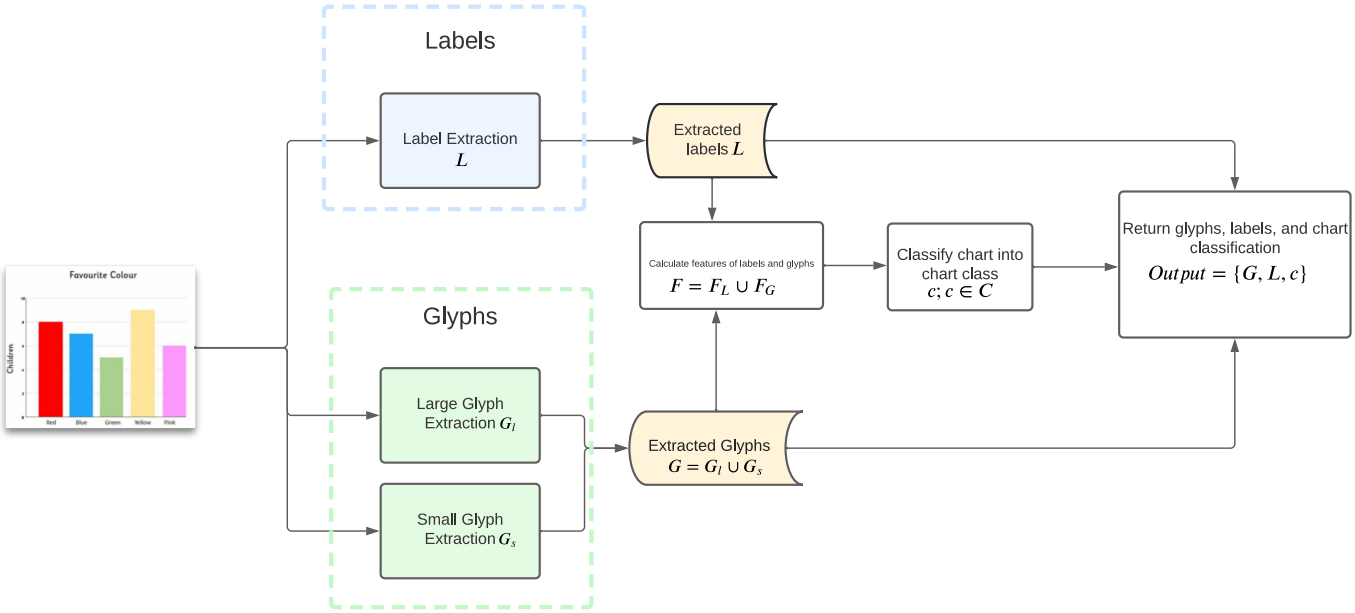


Fig. 3. The CV-BAGLE pipeline. First, the labels and glyphs are extracted. Using the extracted labels L we approximate the mapping from pixels to the chart coordinate system. The sets of small and large glyphs are extracted, notated G_s and G_l, respectively, to form the glyph set G. We then engineer the feature set $F = F_L \cup F_G$, where F_L and F_G are feature sets derived from L and G, respectively. Using a classifier with F as the input, we classify the chart into a chart class $c; c \in C$, where C is the set of chart classes defined in Table I. Finally, we output a set containing the extracted glyphs G, labels L, and chart classification c.

5) Removal of horizontal and vertical lines.

Steps (1)-(3) are self-explanatory. Step 4 attempts to remove large objects in the data visualization. When we first attempted to use Pytesseract with only steps (1)-(3), we noticed that the OCR model would incorrectly identify large objects in the visualization. By removing large objects from the visualization, we aim to prevent this incorrect text identification.

We define ”large objects” for Step (4) as objects that have an area greater than the average area of all objects in the visualization. This is implemented by calculating all the contours in the processed image from Steps (1)-(3), and then calculating the area of each contour. Contours with areas above the average are removed from the processed image. Figure 4 shows an example of how Step (4) removes incorrect OCR identification. In Figure 4(a), the large blue pie slice is incorrectly identified as text (notated by the large green rectangle). Figure 4(b)-(c) shows the result after applying Step 4, and the associated mask, respectively.

The final step, Step (5), aims to remove horizontal and vertical lines from the processed visualization. Step (5) consists of the following sub-steps:

1) Create a kernel (either horizontal or vertical) with size (1,40).
2) Call a morphological opening transform with the kernel over 2 iterations.
3) Find contours and mask them out of the processed image.

Step (5) helps in the removal of any extraneous lines (whiskers on box and whisker plots, axis spines, grid lines, etc.), which can be incorrectly identified as text, similar to large objects removed in Step (4).

B. Glyph Extraction

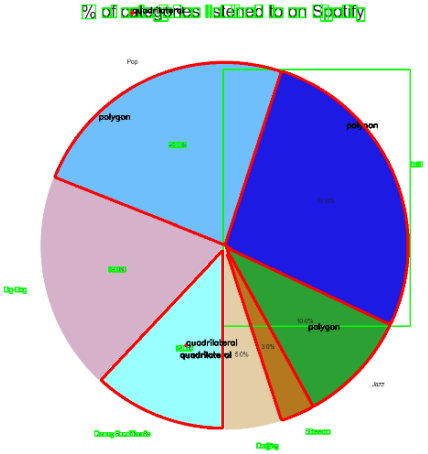
When considering the visual characteristics of glyphs from the lens of a computer vision problem, glyphs can be separated into two subsets: large glyphs and small glyphs. Examples of large glyphs are bars in a bar chart, slices in a pie chart, and bars in a histogram. Examples of small glyphs are points in a scatter chart or regression plot, lines in a regression plot, and error bars in a bar chart. Large glyphs, by definition, have a large area, and take up a significant portion of the chart. Small glyphs, however, are often numerous (points on a scatter plot) or thin (line on regression plot). When considering these two subsets of glyphs, it could be easy to imagine the difficulty of developing one technique that successfully identifies glyphs of both types. Motivated by this notion, we take a divided approach to identify each type of subset separately, and then combine the results.

1) Extraction of Large Glyphs: In order to extract large glyphs, we first preprocess the image to make the target glyphs more apparent. The preprocessing steps include:

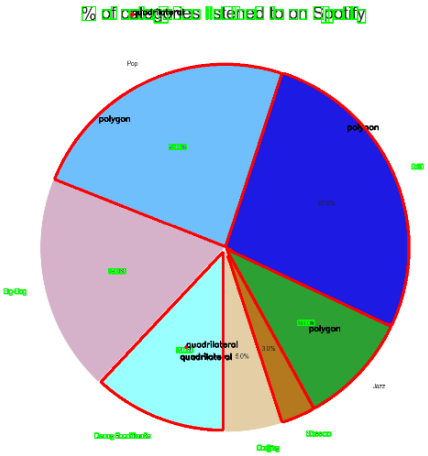
1) Apply a Gaussian Blur (kernel = (15x15), sigma = 0)
2) Increase the contrast of the image.
3) Apply Canny edge detection (hysteresis threshold 1 = 100, hysteresis threshold 2 = 200)

After applying these preprocessing techniques, we can more clearly see the target large glyphs. Figure 5 illustrates the results of these preprocessings.

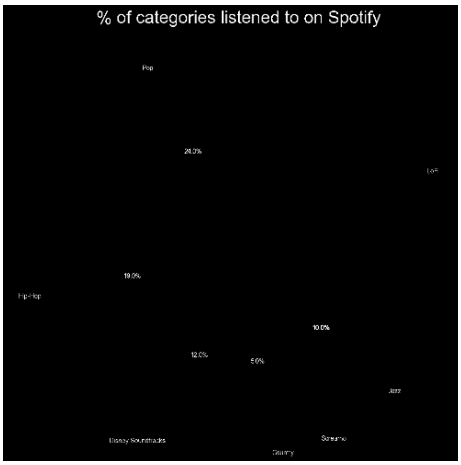
We then find the contours of the processed image using OpenCV. For each identified contour, we calculate the area and approximate the number of sides using the Douglas-Peucker algorithm, which is later used for feature engineering. We remove all contours that have a bounding box with a width



(a) Incorrect OCR results (large green rectangle) without large object removal.

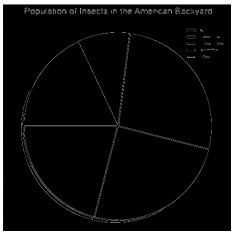


(b) Improved OCR with large object removal.

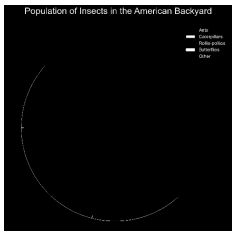


(c) Processed image after large object removal in Step 4, which has greatly improved the OCR results.

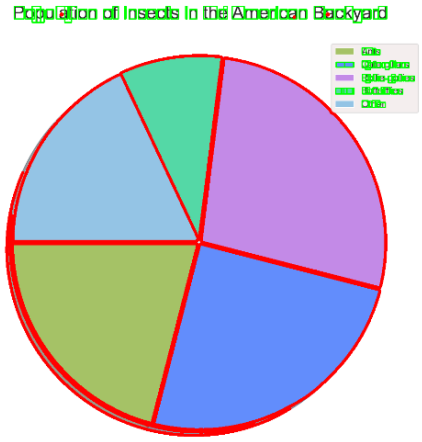
Fig. 4. The effect of removing large objects on the performance of Pytesseract OCR. Green boxes represent identified characters from Pytesseract. In (a), we see the large dark blue pie slice incorrectly recognized as a character. After removing large objects in Step 4, we eliminate this incorrectly identifies character as shown in (b). (c) is the resulting processed image.



(a) Preprocessed to expose large glyphs.



(b) Preprocessed to expose labels.



(c) Combined results highlighting extracted glyphs (in red) and labels (in green).

Fig. 5. Effect of preprocessing for GLE. The results of preprocessing for large glyph extraction is shown in (a), and for labels in (b). The combined results are shown in (c), with detected large glyphs outlined in red and OCR results in green boxes.

and height less than 100 pixels. The remaining contours are considered the extracted large glyphs.

2) Extraction of Small Glyphs: Similar to large glyph extraction, we first preprocess the image to make the target glyphs more apparent. However, we add an additional step of masking after the initial preprocessing. The preprocessing steps include:

1) Apply a Gaussian Blur (kernel = (15x15), sigma = 0)
2) Increase the contrast of the image.
3) Apply Canny edge detection (hysteresis threshold 1 = 100, hysteresis threshold 2 = 200)
4) Mask the bounding boxes calculated from label extraction onto the processed image.

Step (4) of the preprocessing involves taking the bounding boxes calculated from label extraction and masking these pixels onto the processed image for small glyph extraction. This effectively prevents labels from being incorrectly identified as small glyphs. Of course, this is dependent on the success of label detection by the OCR.

C. Feature Engineering

After extracting glyphs and labels, we then must engineer a feature set calculated from various characteristics of the extracted glyphs and labels in order to input into a classifier. Since individual chart instances in one chart class will have

Large Glyphs
Glyphs
Mean size
Standard Deviation of size
Standard Deviation of glyph x center
Standard Deviation of glyph y center
Mean # of sides
Standard Deviation of # of sides
Mean aspect ratio
Standard Deviation of aspect ratio
Small Glyphs
Glyphs
Standard Deviation of glyph x center
Standard Deviation of glyph y center
Mean aspect ratio
Standard Deviation of aspect ratio
Labels
Labels
numeric labels
Mean of numeric label x center
Mean of numeric label y center
Standard Deviation of numeric label x center
Standard Deviation of numeric label y center

TABLE III
ENGINEERED FEATURES FOR LARGE AND SMALL GLYPHS, AS WELL AS LABELS.

varying numbers of glyphs and labels, and even more variation across chart classes, we decided to represent various glyph and label characteristics as the distribution of the characteristic across all extracted glyphs/labels in the chart. Thus, for each characteristic calculated (glyph size, glyph, position, etc.), we include the mean and standard deviation of the characteristic for all glyphs/labels in our engineered feature set. These characteristics are calculated based on the contours and bounding boxes of the glyphs and labels, as well as the values of the text in the labels. The entire set of engineered features can be found in Table III.

1) Glyph Features: The most basic glyph feature that was calculated was the *number of glyphs* extracted from each chart. The logic behind this feature is that some charts consistently have more glyphs than others. For example, a bar chart or histogram may have a smaller set of glyphs than a scatter plot. This feature also is used as a flag for when feature extraction fails (i.e. $num\_glyphs = 0$). In the case of glyph extraction failure, this feature acts as a trigger to help the model ignore other features related to glyphs.

Glyph Size is represented as the area of the glyph contour. The logic behind this feature calculation is that some chart glyphs, such as charts with bar features (bar charts or histogram), typically have a relatively larger glyph area than other chart glyphs (scatter plot points). We represent the distribution of this feature across all extracted glyphs in a chart with 2 features: the mean and standard deviation. Charts such as scatter plots would display a smaller standard deviation in glyph size than a bar chart or a pie chart. This is because scatter plots represent data as an individual positional glyphs rather than by their glyph size. In a bar chart or pie chart, the data represented in those chart are reflected by the respective individual glyph sizes, rather than position. With this in mind, there would be less differences in scatter plot glyph sizes than those found in bar or pie charts.

Glyph Position is represented as the x and y coordinate of the center of the glyph's corresponding contour. Each x and y coordinate of all the glyphs in a chart was then extracted and the standard deviation of these coordinates were calculated. Essentially the standard deviation of a charts glyph x coordinates and y coordinates were calculated separately. The logic behind this is that certain chart types have specific positional arrangements in their glyphs. For example, horizontal bar chart glyphs would typically have a smaller standard deviation in their x coordinates than glyphs found in a vertical bar chart. Also, horizontal bar chart glyphs would typically have a larger standard deviation in their y coordinates than glyphs found in a vertical bar chart.

The *number of sides* for each extracted glyph was another calculated feature. Each contour extracted by OpenCV is classified as a polygon with a specific number of sides. According to the OpenCV documentation, the OpenCV implementation of contour shape approximation uses the Douglas-Peucker algorithm. We classify each glyph's contour as either a triangle (3-sided polygon), quadrilateral (4-sided polygon), or a polygon (5+ sided polygon). For bar charts or box-and-whisker plots, most of the contours are classified as four-sided polygons, whereas for pie charts, the extracted glyphs (pie slices) are classified as either 3-sided or 5+ sided polygons. This is most likely because of the shape approximation algorithm has difficulty dealing with the curved edges of a pie chart. The standard deviation of number of glyph sides helps determine whether the chart has multiple types of glyphs such as those found in a scatter plot with a regression line included.

The *aspect ratio* of the bounding box for each glyph extracted within a chart was also calculated as a relevant feature for classification. A bounding box is a the smallest vertical rectangle that can fit around the glyph's contour. The aspect ratio of all the bounding boxes for each glyph contour found in a chart was extracted. The mean and standard deviation of these extracted aspect ratios were then calculated. In a vertical bar chart, the bounding box aspect ratio of a glyph would typically be smaller than that found in a horizontal car chart. This same logic can be extended to differentiating between a vertical and horizontal box-and-whisker plot. Another example is that in scatter plots the standard deviation of the bounding boxes for their collected glyphs would be close to 0.

2) Label Feature Calculation: If *numeric labels* (percents, floats, integers, etc.) are found within a chart, the mean and standard deviation of the position of the extracted numeric labels are calculated as features. This helps determine whether either one or both of the axes in a chart are quantitative. Specifically, low standard deviation of the x coordinate, along with a low x mean for numeric labels, may infer a y axis. Similarly, we can infer an x axis from the associated features for y. This helps us differentiate between chart types that have varying types of axes. For example, a scatter plot or line plot has both x and y quantitative axes, compared to a bar chart which has only one quantitative axis. Finally, a pie chart has no axes, which may be revealed by very few numeric labels. With this in mind, we include the *number of numeric labels* in addition to the distribution of their position.

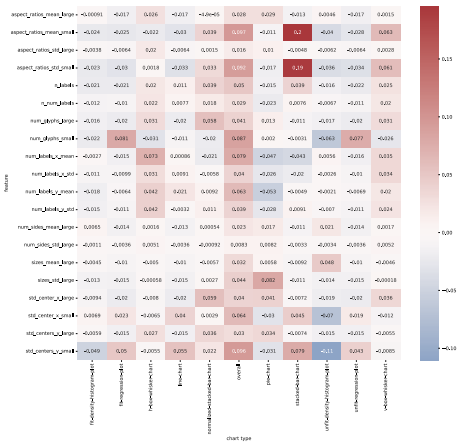


Fig. 6. Feature importances for each chart type, as well as overall, for chart type classification. Notably, the importance of std_center_x, std_center_y, $num\_labels\_x\_mean$, and num_glyphs.

V. Results

To evaluate the ability of our extracted glyph and label features to differentiate between chart classes, we first ran steps (1)-(5) of Algorithm 1 on all of the charts in the training data. With 65,984 images in the training data over 12 classes, processing the images took over 6 hours on a Lenovo Yoga 9i. We then trained a baseline Random Forest Classifier from the *sklearn* Python library to achieve a training accuracy of 98.15% and testing accuracy of 81.60%. In order to better understand the performance of the model, we present the feature importances for the engineered glyph and label features, F_G and F_L, respectively, described at the end of Section IV.

Figure 6 illustrates the feature importances for classifying charts. When considering the feature importances in the $overall$ column (i.e., the feature importances for classifying any chart type) we consider the importance of the aspect ratio of small glyphs, number of small glyphs, and standard deviation of the y position of small glyphs. It is interesting that the model put so much importance on small glyphs, especially when many chart classes (bar charts, pie charts) should have little to none small glyphs extracted. It is worth noting that the top 3 features for classifying charts are $aspect\_ratios\_mean\_small$, $std\_centers\_y\_small$, and $aspect\_ratios\_std\_small$, with importances 0.097, 0.096, and 0.092, respectively.

Narrowing the analysis to features that are important to classifying a particular chart type, we consider the importance of $aspect\_ratios\_mean\_small$ (importance=0.2) and $aspect\_ratios\_std\_small$ (importance=0.19) for classifying a stacked bar chart. Similarly, we consider the importance of $std\_centers\_y\_small$ (importance= -0.11) to guide the model away from classifying a chart as an unfit density histogram. Once again, both these chart types have few to none small glyphs. However, when visualizing the small glyph extraction, we see that many of the horizontal lines between the bars in a stacked bar chart are identified as small glyphs.

A. Glyph and Label Extraction Metrics

In order to quantify the performance of glyph and label extraction, we crafted metrics for each chart class. This is necessary since each chart class brings with it specific metrics. For example, in a bar chart we measure the number of bars extracted and the accuracy of the height of the bars. However, in a scatter chart this is not an appropriate metric. Tables IV and V show the various GLE metrics calculated for each chart type. These metrics help explain the poor performance of classification on real world data, as the accuracy of the calculated features will suffer due to the inconsistent extraction of glyphs and labels. For instance, in a horizontal bar chart, only 23.5% of bars are found, on average.

The varying performance of label extraction seen in Table V may be impacted by the complexity and visual crowdedness of various chart types. Additionally, many lines on the chart often result in lower label extraction performance. For instance, line charts and fit regression plots, which each have a lot going on in the image (compared to a pie chart, for instance), have a lower label extraction performance due to more noise in the image for the label extraction to avoid.

In order to calculate glyph metrics in Table IV, we needed to map the coordinates of extracted glyph contours in the image to the coordinates of the data. Since we did not know this exact mapping for every chart, we attempted to map the extracted glyphs to the ground truth by mapping each glyph to the closest ground truth value after normalizing both the ground truth data and extracted glyph coordinates to a range of [0, 1]. This allows us to disregard the range of the data and treat both the extracted glyph and contours and ground truth data on the same scale. Thus, all of the distance metrics are independent on the scale of the data, but rather on a normalized scale of 0 to 1, where 1 is the maximum of the ground truth data. For example, if the average distance calculated is .25, and the data has a min of 0 and max of 10, than the true distance of the glyph to the ground truth data point is 2.5. We were unable to calculate a glyph metric due to the lack of coordinate system to map to for pie charts.

B. Real World Data Classification Performance

Although we had moderate performance on the test data (test accuracy = 81.60%), we unfortunately did not have acceptable performance on classification of real world data. Our classifier achieved an accuracy of 18% on the real world data. Given that there are 12 classes, random guessing would expect to yield a performance of $100/12 = 8.33\%$, so the model has learned something, but it is far below the desired performance. In Figure 7, we present the multi-class confusion matrix for the real world data. Besides validating the poor performance on real world data, the confusion matrix also reveals that many charts, regardless of their true chart type, are being misclassified as a pie chart. When running Principal Component Analysis on the training data with 2 principal components, we observed that, although there may be some somewhat clear decision boundaries between chart classes, the

Chart type	Metric	Average value
fit density histogram	$\lvert predicted\ bar\ height - true\ bar\ height \rvert$	0.0072
fit regression chart	$\lvert\lvert glyph\ coordinate - true\ coordinate \rvert\rvert_2$	0.030446
horizontal box & whisker chart	$\lvert predicted\ box\ width - true\ box\ width \rvert$	0.0183
horizontal bar chart	% of bars found	23.54%
horizontal bar chart	$\lvert predicted\ bar\ width - true\ bar\ width \rvert$	0.0086
line chart	$\lvert\lvert glyph\ coordinate - true\ coordinate \rvert\rvert_2$	0.0761
normalized stacked bar chart	% of bars found	26.25%
normalized stacked bar chart	$\lvert predicted\ bar\ height - true\ bar\ height \rvert$	0.0134
stacked bar chart	% of bars found	4.78%
stacked bar chart	$\lvert predicted\ bar\ height - true\ bar\ height \rvert$	0.0160
unfit density histogram	$\lvert predicted\ bar\ height - true\ bar\ height \rvert$	0.0057
unfit regression chart	$\lvert\lvert glyph\ coordinate - true\ coordinate \rvert\rvert_2$	0.0294
vertical box & whisker chart	$\lvert predicted\ box\ height - true\ box\ height \rvert$	0.0185
vertical bar chart	% of bars found	23.93%
vertical bar chart	$\lvert predicted\ bar\ height - true\ bar\ height \rvert$	0.0094

TABLE IV
GLYPH METRICS

Chart Type	Average % labels extracted
fit density histogram	52.34
fit regression chart	48.04
horizontal bar chart	60.97
line chart	49.17
normalized stacked bar chart	62.51
pie chart	73.50
unfit density histogram	75.40
unfit regression chart	0.461431
v box whisker chart	73.27
vertical bar chart	42.45

TABLE V
LABEL METRICS

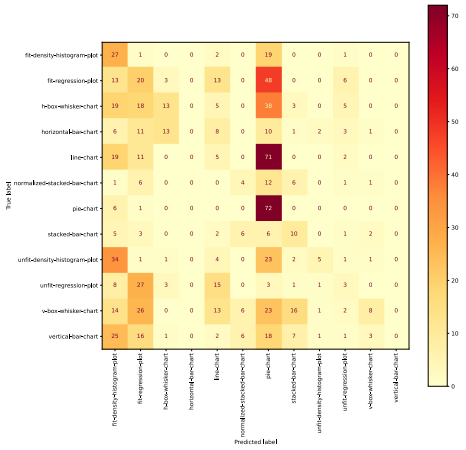


Fig. 7. Multi-class confusion matrix for real world data. Many charts, regardless of true label, are misclassified as pie charts.

points labeled pie chart spread out across almost all other chart classes, making classification near impossible. This implies that the features engineered by us may not be enough to successfully distinguish between our 12 chosen chart types, let alone more types as seen in the real world.

VI. DISCUSSION

The extraction of glyphs and labels using computer vision-driven approaches came with varying success. For both large and small glyphs, a major component of image preprocessing involved filtering the calculated contours based on width and height (with 100 pixels as the threshold). This method had varying success, as varying glyph size within one chart type can result in incorrectly identified glyphs. For example, in a bar chart, one bar may have a height of less than 100 pixels, while another may be much taller than 100 pixels. This would put both (large) glyphs in different categories of small and large, respectively. This could be problematic based on the use case of the extracted glyphs, especially if a chart correctly has both small and large glyphs that each need to be identified accurately (an example of a chart with both large and small glyphs is a box and whisker chart with outliers).

When calculating small glyphs, we mask out any labels calculated by the OCR. This is due to labels being incorrectly identified as small glyphs. If the contours of a character are found, they will most likely have a width and height smaller than 100 pixels, which will lead to the characters begin identified as a small glyph. With this in mind, it becomes clear that the performance of small glyph extraction is directly dependent on the success of OCR in order to generate a proper label mask.

We now bring attention the importances of numeric label positioning for overall chart classification. Referring to Figure 6, we can see that$num_labels_y_mean$ (importance = 0.063) and $num\_labels\_x\_mean$ (importance = 0.079) have moderate feature importances. As mentioned before, our motivation behind engineering these features was to help the model determine the type of variable (quantitative versus qualitative) on each axis. Observing the moderate feature importances of these features implies that these features are helping differentiate between charts with different axis types. With better OCR performance, we can more accurately extract labels and identify the numeric labels present in axes, which may in turn increase the importance of these features. In particular, OCR struggles to identify x axis ticks that are rotated 90 degrees.

For large glyph extraction, there is often difficulties identifying glyphs that have low saturation against a white background. Conversely, large glyph extraction does especially well when there is a dark outline around the glyph. With modification to the preprocessing steps for large glyph extraction, these low saturation glyphs may be more successfully extracted. The difficulty, of course, comes with creating a

single preprocessing method that successfully captures all large glyphs with varying aesthetics.

VII. Future Works

Although our approach generalizes to multiple types of data visualizations, there are still many additional types of visualizations that have not yet been tested. Additionally, data visualizations on the web come in many shapes in forms. With the growth of easy to use vector graphics libraries such as D3, interactive, animated charts are growing in popularitySiva et al. (2012) elaborates on the growing trend of animated visualizations in several fields, including the medical field and the military. An extension of our work could be the identification of glyphs over time. Finally, we envision our CV-BAGLE used as a Glyph and Label Extraction (GLE) tool to aid in broader pipelines for automated figure caption generation in scientific papers.

References

Balaji, A., Ramanathan, T., and Sonathi, V. (2018). Chart-text: A fully automated chart image descriptor. *CoRR*, abs/1812.10636.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Cuadrado-Gallego, J. J., Demchenko, Y., Losada, M. A., and Ormandjieva, O. (2021). Classification and analysis of techniques and tools for data visualization teaching. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 1593–1599.

Fuchs, J., Isenberg, P., Bezerianos, A., and Keim, D. (2017). A systematic review of experimental studies on data glyphs. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1863–1879.

Hsu, T.-Y., Giles, C. L., and Huang, T.-H. (2021). SciCap: Generating captions for scientific figures. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3258–3264, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mittal, V., Moore, J., Carenini, G., and Roth, S. (1998). Describing complex charts in natural language: A caption generation system. *Computational Linguistics*, 24:431–467.

Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., and Heer, J. (2011). Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 393–402, New York, NY, USA. Association for Computing Machinery.

Siva, N., Chaparro, A., and Palmer, E. (2012). Human factors principles underlying glyph design: A review of the literature and an agenda for future research. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 56(1):1659–1663.

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Zhang, J. E., Sultanum, N., Bezerianos, A., and Chevalier, F. (2020). *DataQuilt: Extracting Visual Elements from Images to Craft Pictorial Visualizations*, page 1–13. Association for Computing Machinery, New York, NY, USA.