1 Introduction

Visualizing data in order to perform analytical tasks and get a better insight into the data has been an effective and common approach during the past centuries. Nonetheless, the type of visualization to choose can have a profound effect on the insights to be gleaned out of the data. If the data is readily available, reanalyzing the data and/or changing the format of visualizations and inspecting the data from a different angle is quite straightforward. However, the problem arises when there is nothing but a bitmap picture of the data at hand, which is usually the case.

One naive approach to get the data out of the image is to do it manually. Although it might be possible to manually extract the desired data from a couple of images with a high accuracy, it is a terribly cumbersome task, and doing it for many images and/or frequently is infeasible. In order to tackle the problem in an automated manner, image understanding methods need to be utilized to extract the data presented from the image. In addition to data re-analysis, image understanding has other motivations. Enriching knowledge discovery, as well as reformatting the data in an accessible manner for a visually impaired person are other notable examples.

Nevertheless, understanding the chart images by computers is by itself a hard problem. Given an image, the very first step is to determine what type of chart it is presenting. Based on the chart type, e.g., bar, line, or pie chart, the image understanding algorithm to be exploited varies a lot. So, a high level vision to the solution of the problem is to have different pipelines for different chart categories. But before we can direct each image to its corresponding pipeline, we need to classify to which type the image belongs. This is the sub-problem we are going to address in this work, i.e., classification of chart images, which can also be utilized for image indexing and retrieval.

A formal definition of the chart image classification problem is as follows:

Given we have a chart image \(I \in I\) where \(I\) is the set of all chart images of type \(c \in C\), where \(C\) is the disjoint set of all allowed chart types, we are to determine to which \(c\) the image \(i\) belongs. So, we need to devise the function

\[ f(i): I \rightarrow C \]

Delving into general image classification problem, we can see that engineering the features to be used is a necessary step. Feature engineering is the task of identifying/composing the features that represent images in a way to smooth the task of classification and improve accuracy. For the task of classification we’d like to have as few number of features as possible – to break the curse of dimensionality - while at the same time we’d like those features to represent sufficient information about images with respect to the target classification task. One naive approach is to take each pixel of an image as one feature: if we have 100x100-pixel images, we end up having 10,000 features. Although we can rest assured that we have any necessary information required for our classification task, we have captured too much unrelated information that degrades performance and classification accuracy. Therefore, we need to devise features with regard to what information presented in the images are important for the classification. For the chart image classification, we may try to capture basic graphical components of the charts as the features. Using image patches as features, although seems to be not very wise, is another approach; applying some tricks for patch-based classification can yield state-of-the-art performance, which will be discussed in this report as well. Having the appropriate features prepared, the next
The purpose of the classification step is to perform the classification. Standard classification algorithms, such as Support Vector Machines (SVMs), can be applied for this purpose.

The structure of the report is as follows. In section 2, related work is introduced. A taxonomy of the chart image classification approaches, from the perspective of feature engineering, and how to utilize the features to perform classification is introduced. Sample related work is also pointed out. For presenting a better insight to the range of the existing methods being used, two methods have been chosen to be presented in more detail in sections 3 and 4. Section 5 describes my own project, where I tried to run three different classification experiments using the existing methods presented, as well as a combination of them. There, I tried to fix a selected dataset of chart images, and run the classification experiments across three different approaches/settings in order to make a fair comparison of the approaches. Finally, section 6 summarizes and concludes the report.

2 Related Work

Classification of images from the real scenes and objects has been a well-studied problem in computer vision [1]. However, classification and analysis of document images has recently gained much attention in order to serve as tools to speed up the work of human dealing with those documents as the short-term goal, as well as building intelligent machines that can understand the documents and utilize the understanding to perform other tasks as the long-term goal.

Chart classification belongs to the intensive research area of recognition of special types of graphics. The majority of existing approaches to chart image classification were developed within the scenario of image features extraction followed by a feature-based comparison. The latter varies from comparison of a test image to abstract models, representing particular classes (Model-based approach, such as [2]) to comparison of a test image to training images (Learning-based approach, such as [3]).

Within the model-based approach, each $c \in C$ is defined by a model $m_c$ consisting a set of definitive features $F_c$. Presence or absence of each $f \in F_c$ in the chart image $i$ yields a matching score $s_{ic}$. The class $c$, to which $i$ belongs will be the one corresponding to the maximum $s_{ic}$. If, however, none of the scores are above a minimum threshold, $i$ may be classified as none of $c \in C$.

In the learning-based approach, the entire set of classes $C$ is represented by a set of constant number of features $F$. Each $c \in C$ is represented by a combination of values for all $f \in F$. Corresponding values of all $f$’s for each $c$ is trained using a training dataset. Proximity of the value combination of the corresponding features within $i$ to those trained values of $f$’s classifies the image $i$.

Another classification of approaches is by the type of extracted features. According to it, all methods can be divided into the following types: low-level, middle-level and high-level. The former tends to calculate low-level features and rely on vast representative training sets. A typical example is utilizing Hough transform for feature extraction and classification, such as [4]. Other approaches, utilizing higher-level features, rely on optimality of their feature set. An example of high-level model-based chart classification is discussed in [2]. Authors detect basic shapes in the image and check if they satisfy a certain set of constraints corresponding to a certain chart type. Another fascinating approach using low to mid-level features, that is, using image patches as features is presented in [3].

To sum up, image classification is a two-fold problem. The first fold is feature engineering, in which the features may lie anywhere along the spectrum of low-level to high-level ones. The second fold is the classification approach, which is categorized to model-based and learning-based approaches. In order to give the reader a better understanding of the methods used for each of the two folds, we chose to present the approaches in [2] and [3] in more detail. The former uses high-level features and a model-based classification approach while the latter uses low-mid-level features and a learning-based approach.
3 Model-Based Chart Image Classification

The proposed approach in [2] has focused on 4 common chart types as the authors claimed, that is, bar chart, pie chart, line chart, and high-low chart (Fig. 1).

![Fig. 1. Chart Types Used in the Model-Based Approach (left to right: bar, pie, high-low, line)](image)

The authors used a bottom-up approach for the model-based matching. In this application, the basic features, including the straight line segments and arcs, are extracted and used to match an existing model.

3.1 Feature Extraction

In the first step, some preprocessing, i.e., noise removal and conversion to grayscale is performed. After it, graphic/text separation, edge detection, and vectorization is performed on images to extract basic shapes and high-level relation among them.

3.1.1 Graphic/Text Separation

A typical chart image contains both textual information and graphical information. Textual information includes the chart title, the name of the data series, some descriptions of the chart, and the numerical labels. On the other hand, graphical information includes the actual drawings, such as reference lines, edges and arcs. The first step in the proposed system is to distinguish these two types of information and store them separately for further processing. For this purpose, 8-connected components are constructed, followed by a set of filters that perform thresholding based on various properties of the connected components, including height, size, black pixel density and height/width ratio. The threshold values used are determined through some training samples. Using the set of filters, most of the text and graphics can be separated (Fig. 2).

![Fig. 2. Example of Text/Graphics Separation](image)

After this phase, the authors only focused on the graphical part since their experiment of text recognition using off-the-shelf OCR packages reportedly yielded poor result (due to various font types and sizes, as well as different orientations of the text strings).
3.1.2 Edge Detection
In a chart image, the color or grayscale level within a graphical component is consistent. On the other hand, the color difference or grayscale level difference between neighboring graphical components is normally significant. Thus the most straightforward approach is to detect the high differential values in the pixel intensity. However by doing so, the two sides of a thick line will appear as independent lines in the edge map. So, a threshold to judge how far two edge points can be to be considered as lying on the two sides of the same line needs to be set. In other words, the maximum line thickness in the given image needs to be determined. An example of edge detection is depicted in Fig. 3.

![Figure 3. Example of Edge Detection.](image)

After the edge detection step, only edge points are kept and passed to the next step

3.1.3 Vectorization
In order to vectorize the detected edges, first of all, a straight line segment is found as a seed, and it is tried to be extended to find a complete line or arc by examining the neighboring segments. To construct a straight line, the x-y coordinates of the line segments should change monotonically along the line.

To construct an arc, we need to find three consecutive line segments and check if the bisectors of these line segments converge to a common point. If so, the point becomes the estimated center for the arc and is used to find out the remaining part of the arc.

After the lines and arcs are vectorized, there is a need to refine them. The reason is that the image quality may be poor and as a result there may be broken lines and arcs. If two lines have the same orientation and the distance between their end points is smaller than certain threshold, they are merged to form a new line segment. If two arcs share the same center and similar radius, they will merge to become a new arc. Furthermore, the noise in the image becomes extra segments. So a length filter is used to remove the line segments whose length is smaller than a threshold value.

3.2 Chart Type Recognition
After the vectorization, a set of vectors representing the straight lines and Arcs are available. We can check the following relationships among the straight lines: parallelism, perpendicular and convergence. These relationships are important to us because they are the indications of the existence of certain chart components, such as bars or pies etc. Besides these components, we are also interested in the existence of the x-y axes that is an important component for some chart types.
Given the information about the existence of the basic components and the x-y axes, we can calculate the likelihood of a given image $i$ to match a chart model $m$ as:

\[
(2) \quad \text{likelihood}_{mi} = \max(0, \sum_{k=1}^{n} W_{mk} C_{mik} - \sum_{l=1}^{m} W_{il} O_{il})
\]

where $W_{mk}$ is the weight of $k^{th}$ component in the chart model $m$, $W_{il}$ is the weight of $l^{th}$ unexpected object in the image $i$. $C_{mik}$ is 1 if the $k^{th}$ chart component from model $m$ exists in the image $i$, and it is 0 otherwise. $O_{il}$ indicates the existence of the $l^{th}$ object that is unexpected to appear in model $m$.

Different chart models have different set of basic components, resulting in different likelihood values when the formula (2) is applied to the given image. If all the likelihood values are small, then the given image is unlikely to be a chart image. If some likelihood values are sufficiently large, then we can treat the given image as a chart and pick up the model with the maximum likelihood as the type of the chart.

Each chart type has some individual features that help to calculate the likelihood value. For instance bar chart has two parallel lines touching the horizontal axis for each bar. The top of each bar is closed with a horizontal line. High-low charts have a number of vertical line segments above the x axis without touching it. Pie charts have a number of concentric pie segments, which totals to a full circle. Line charts consist of one or more polylines above the horizontal axis, and so on.

Since the number of features for each model and an image is not constant, setting the weights is a little tricky. For instance, a bar chart with 10 bars should not be very much likely to be a bar chart than a bar chart with only 4 bars. One approach in order to address this issue is to order all the features according to their importance within a model. The most important feature will get the weight of one half, the second one quarter, the third one eighth, ..., and the last will fill the remaining gap to one (the sum of all weights should be one).

3.3 Experimental Result

The images to be tested were collected from the internet or some scanned document pages. The entire test collection contained 8 bar chart images, 8 line chart images, 8 pie chart images, 3 high-low chart images and 10 engineering drawings. The charts were generated using various packages, with the Microsoft Excel as the most popular one. The chart images include most of the features that are commonly used. The resolution of the images is reasonably controlled.

Each testing image is passed into the system. After preprocessing and vectorization, the system checks for the existence of chart components, namely the x-y axes and the data components. These components are then used to calculate the likelihood values for all the chart models based on the formula (2). If the likelihood is smaller than 0.5, then the input image is not treated as a chart image. Otherwise the image is treated as a chart and the chart model returning the maximum likelihood value will be chosen as the chart type. The results of likelihood calculation and type determination are shown in Table 1.

<table>
<thead>
<tr>
<th>Image Type</th>
<th>Average Likelihood value</th>
<th>Type correctly Determined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar chart</td>
<td>0.9374</td>
<td>100%</td>
</tr>
<tr>
<td>Pie chart</td>
<td>0</td>
<td>0.8473</td>
</tr>
<tr>
<td>Line chart</td>
<td>0.45</td>
<td>0</td>
</tr>
<tr>
<td>H-L chart</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>Not chart</td>
<td>0</td>
<td>0.0288</td>
</tr>
</tbody>
</table>

**Table 1.** Result of Likelihood Calculation and Type Determination.
4 Revision: Learning-Based Chart Image Classification

In the Revision project [3], the focus of classification is on 10 chart types (Fig. 4).

![Chart Types for the Revision Project](image)

**Fig. 4.** Chart Types for the Revision Project (left to right, top to bottom: Venn Diagram, Map, Area Graph, Bar Graph, Table, Curve Graph, Pareto Chart, Pie Chart, Radar Plot, Scatter Plot)

The authors performed a number of interesting steps in order to identify the effective set of features, and then used those features to perform classification using SVMs with a quadratic kernel function.

4.1 Feature Engineering

1. **Image Normalization.** Images are normalized to a constant size of $D \times D$ pixels to ensure homogeneous sampling and equivalence of each exemplar in the training corpus. The aspect ratio of the original image is preserved by padding with the background color. Color images are converted to grayscale, as color is rarely indicative of visualization category. Image sizes from $D = 32$ up to $D = 512$ are considered, and $D = 128$ is found to achieve the best classification accuracy.

2. **Patch Extraction.** Square patches from the images are extracted by uniform random sampling for 100 locations per image. Patch sizes over the range from $2x2$ up to $20x20$ pixels were tested, and a patch size of $6x6$ pixels was found to give the best classification performance. The optimal patch size was consistently about 5% of the normalized image dimensions. To filter out frequently occurring constant color regions, sample patches with variance less than 10% of the maximum pixel value were rejected.

3. **Patch Standardization.** For classification, absolute pixel values are not as important as variation within a patch. Thus, the contrast of each patch is normalized by subtracting the mean and dividing by the standard deviation of the pixel values in that patch. This normalization ensures that a single appropriately weighted patch can represent patches with different absolute intensities but similar variations. A “whitening” procedure is performed on the entire patch set which reduces the cross-correlation between patches and improves classification performance.

4. **Patch Clustering.** Given the extracted patch set, K-means clustering is performed to obtain a set of “centroid” patches that correspond to the most frequently occurring patch types. A Euclidean $L_2$ distance metric is used over the pixel values. In practice the number of centroids $K$ is set to 200 as the authors found that larger numbers of centroids did not achieve better performance. The centroid patches constitute a feature set, or
“codebook”, which can be used to describe the image corpus. These codebook patches capture frequently occurring graphical marks such as lines, points, corners, arcs and gradients.

*Using Low-Level Image Features for Classification.*

To classify an input image, it is first normalized to a 128 x 128 grayscale image and 6x6 sized patches centered on each pixel are extracted.

5. **Codebook Patch Response.** For each extracted patch, the nearest codebook patch is determined by Euclidean distance. Thus, a D2 centroid response map is retrieved over the entire image, where \( D = 128 \) is the normalized image dimension.

6. **Feature Vector Formulation.** The dimensionality of the codebook patch response map is reduced by dividing the image into quadrants and counting the frequency of the activations for each codebook patch in a given quadrant, similar to constructing a histogram of activated patches. Thus, a 4K-length feature vector for each input image is obtained. Since \( K = 200 \) is set, we obtain an 800 element image feature vector which is much smaller than the set of all 128 x 128 patch responses.

4.2 **Image Classification**

After the feature engineering step, each sample image is reduced to an 800-element vector of positive integer values. So, the problem is now a standard classification problem. The authors use the feature vectors to perform classification using SVMs with a quadratic kernel function.

4.3 **Experimental Result**

The authors introduced a corpus with 2500 annotated images of 16 different chart types, which has been made publicly available. As mentioned earlier, they performed the classification across 10 types. The classification result is summarized in Table 2. As SVM classification is inherently binary, for multi-class tasks, an array of binary SVM classifiers is used, e.g., class 1 or not, class 2 or not, etc. Table 2 shows the result of binary and multi-class classification for each of the 10 chart categories.

<table>
<thead>
<tr>
<th></th>
<th>Multi-Class</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area Graphs</td>
<td>88%</td>
<td>98%</td>
</tr>
<tr>
<td>Bar Graphs</td>
<td>78%</td>
<td>95%</td>
</tr>
<tr>
<td>Curve Plots</td>
<td>73%</td>
<td>91%</td>
</tr>
<tr>
<td>Maps</td>
<td>84%</td>
<td>97%</td>
</tr>
<tr>
<td>Pareto Charts</td>
<td>85%</td>
<td>97%</td>
</tr>
<tr>
<td>Pie Charts</td>
<td>79%</td>
<td>97%</td>
</tr>
<tr>
<td>Radar Plots</td>
<td>88%</td>
<td>93%</td>
</tr>
<tr>
<td>Scatter Plots</td>
<td>79%</td>
<td>93%</td>
</tr>
<tr>
<td>Tables</td>
<td>86%</td>
<td>97%</td>
</tr>
<tr>
<td>Venn</td>
<td>75%</td>
<td>97%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>80%</strong></td>
<td><strong>96%</strong></td>
</tr>
</tbody>
</table>

Table 2. Patch-Based SVMs Classification Result for 10 Chart Categories
5 Current Project
The model-based approach introduced in section 3 used high-level features, such as arcs and straight lines, as the features to do classification. The learning-based approach in section 4, on the other hand, tried to identify an optimal set of low-level features, that is, the frequencies of codebook patch responses, to perform classification. For this project, I am going to experiment high-level features along with a learning-based method.

First of all, I take 3 most common chart types, that is, bar, pie, and line, under focus. Secondly, I am going to run a classification experiment over these three types using the Revision’s learning based approach. Thirdly, I devise high-level features out of the three chart types (similar to what was done in section 3), and experiment an SVMs and a decision tree classification using the features. Finally, I will try to do a fair comparison of the result of the three experiments.

5.1 Dataset
As mentioned above the focus of the current project would be on three most common chart types (Fig. 5). The Revision corpus has images for 194 bar charts, 210 pie charts, and 318 line graphs, which has been collected from various web resources. This ensures that the dataset is a reasonable sample of the charts appearing everywhere. Therefore, the current dataset would have a collection of the three chart types (fig. 5).

![Fig. 5. Chart Types to Experiment (bar, pie, and line)](image)

5.2 Low-Level Features for Classification
In order to engineer the features for the current dataset, as instructed in the Revision’s original paper, I converted all the images to 128x128 pixel (in order to preserve the aspect ratio, I padded each image with its background color to make it a square beforehand). After it, I converted them to grayscale. At this time, I started sampling 100 random 6x6 patches per image. As instructed, if the average differential grayscale values from the maximum value of the pixels within each patch was less than 10%, that patch was discarded. This was done to cancel out the patches with constant background color. After gathering all the sample patches, I ran the K-means clustering using Euclidean distance to extract K centroids. For getting to the appropriate value of K, I tried different values from 20 to 300 with the step size of 20 (Fig. 6 left).

![Fig. 6. Sum of all intra-cluster distances for different values of K](image)
Since the curve of the total sum distance tends to have an elbow between $K= 80$ and $K = 100$, I made the step size 5 over this range to get a more precise value for the appropriate $K$ (Fig. 6 right). Finally, I ended up choosing $K = 85$, which is very different from what the authors reported in their paper, i.e., $K = 200$. However, since the original paper’s work included 10 chart categories, and I am including only 3, it looks normal to end up having less number of clusters. Therefore, the codebook of my work has 85 centroid patches.

In order to build a feature vector for each image, I take each quadrant of the image (64x64 pixel), and extract all possible 6x6 patches. For each extracted patch, I retrieved the closest centroid from the patch codebook, and increment the count associated with that centroid. So, for each quadrant of the image, I ended up having 85 positive integers resulting in having $4 * 85 = 340$ features representing each image.

In order to perform classification, I used SVMs implementation provided by the WEKA [5] using a quadratic kernel function (the same library used by the Revision project). I experimented a 4-fold cross-validation on the dataset with three types, and I got 88% classification accuracy.

5.3 High-Level Features for Classification

The next experiment would be learning-based classification using high-level features. Thus, I tried to extract the basic shapes through a similar approach introduced by the model-based classification in section 3. I converted the images to grayscale. However, I did not do graphic/text separation due to the time constraints. In the next step, I used the Matlab “edge” function with the “canny” edge detection algorithm to retrieve the edges. The canny edge detection algorithm is reportedly stronger in cancelling out the noise (Fig. 7).

The edge detection, as depicted in Fig. 7, returns two sides of a line as two edges. So, those double edges need to be reduced to one average edge. However, since I am going to count some basic features from the edges, those counts would probably be doubled, which does not affect the comparison and classification of images. Thus, I decided to leave the double edges as is. After extracting the edges, I tried to vectorize straight lines and arcs from the edges, I used Matlab “houghlines” function to vectorize all straight lines, and I used Hough circular shape detector [6] to victories the arcs. Before going further, I tried to refine the two vectors. Any two straight line segments with same orientation and close endpoints (less than 3% of the image height) were merged to one longer line segment. Any two arc segments with same center and radius were also merged if the endpoints were close enough (less than 4% of the image height). After it, any little line or arc segment that was shorter than 4% of the image height were omitted (this phase removed most of the remaining text strings that are not useful for our classification).
Having the vectors of straight line and arc segments, I end up using the following features after some trial and error experiments:

1. Existence of x-y axes (two perpendicular line segments with the length more than half of the shorter image dimension); an integer number is saved, that is, the count of pairs of x-y axes
2. Count of the horizontal line segments (an indicator of top of a bar)
3. Count of the vertical line segments (an indicator of the sides of a bar)
4. Count of the two intersecting line segments that were approximately perpendicular (bars, x-y axes, rectangular legend boxes)
5. Count of the full circles (arcs with the degree of about 360) (indicator of a pie)

After formulating each image with the 5 features above, I used the same SVMs library as above, as well as C4.5 decision tree to perform 4-fold cross validation experiments (decision tree is appropriate here because of the few number of features). The SVM classification achieved an accuracy of 90% while the decision tree achieved 93%. Table 3 summarizes the result of the experiments. Below are the extracted rules from the decision tree:

Rule 1:
   circles > 0
   -> class pie [96.8%]

Rule 6:
   Xy > 0
   Vseg > 4
   -> class bar [90.7%]

Default class: line

5.4 Results and Comparison

Table 3 presents the average accuracy over 4-fold cross-validation of the 3 classification experiments I conducted for the sub-corpus of 722 images for 3 chart categories.

<table>
<thead>
<tr>
<th></th>
<th>Patch-Based - SVM</th>
<th>Shape-Based - SVM</th>
<th>Shape-Based Decision Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>88.02</td>
<td>89.88</td>
<td>92.80</td>
</tr>
</tbody>
</table>

Table 3. Average Classification Accuracy Measured over 4-Fold Cross-Validation

Using the high-level features has produced better result when classifying using SVMs. Since there are only 5 representative features when using high-level features, decision tree Classification is appropriate, and produced a better accuracy. Moreover, apart from the feature engineering procedure, which is a problem-specific and a quite demanding task for both of the high and low-level features, high-level features has resulted in a much lighter and faster training and testing procedures because of less dimensionality.

However, this is not all of the story. Unlike the low-level features, devising high-level features is very task specific and requires substantial human expertise. For instance, when defining the high-level features, I already knew that the classification is going to be on 3 chart types, and I already had a good priori knowledge in my mind about the potential shapes and patterns of the charts that can help to define well-distinguishing features between those three specific chart types. I, for example, knew that among bar, pie, and line charts, the only chart type that is very highly likely to have a full circle is pie chart. That’s why I decided to define a feature to
count the number of full circles. If, however, another chart type including circles, e.g., radar plots, was among the possible classes, I was required to tweak the feature set a little by introducing new features or modifying the existing ones in order to end up with a relatively good accuracy. All of these are the indication of human expertise that will get more complicated as the problem size increases; if there are only 2 chart types, I need to devise distinguishing features between 2 classes, but if there are 10 classes, I need to devise distinguishing features for about \( \binom{10}{2} = 45 \) pairs of chart types. For the case of low-level features, I ran a pilot experiment over all the 16 chart types available in the initial corpus without much effort and tweaking. I extracted a codebook, which tended to have 220 centroids for a good performance, which is a little more than the original paper’s codebook for a 10-type classification. The rest of the steps were quite similar. The 4-fold cross-validation produced an accuracy of 73.7%, which is comparable to the result for smaller versions of the problem, i.e., 80% for the 10-class version as reported in the original paper, and 88.1% for the 3-type version as I reached in my experiment. On the other hand repeating the experiment of basic shape classification for a 16-type version of the problem is not straightforward at all, and I can’t think of a way to do that unless by closely considering all the 16 types and doing a lot of trial and error and tweaking, which is complicated and very time-consuming.

In short, the high-level features can yield better performance and accuracy if they are devised sufficiently optimal, which is not an easy task as the problem size increases. However, low or mid-level features, if devised wisely enough, can yield a comparable result without much human expertise as the problem size increases. Using low-level features is, nevertheless, slower and requires more computational resources. That’s because the data to be extracted and passed to the classification methods bears more (probably unrelated) information in order to hopefully the training procedure - and not a human expert - captures required information.

### 6 Take Away and Conclusion

The problem of chart image classification as a sub-problem of document image understanding is introduced in this work. A taxonomy of image classification approaches was described. From the perspective of feature engineering, image classification methods may utilize a range of low to high-level features. Model-based classification and learning-based are the two main approaches to perform the classification based on the features. A specific model-based approach that uses high-level shapes was described. The Revision chart image classification, which uses low-level features (image patches) to perform a learning-based classification was also described.

For my own part of work, I, firstly, fixed a corpus of 722 bar, pie, and line charts selected out of the Revision big corpus of 16 types. In the first experiment, I tried to repeat the Revision approach, where I achieved a reasonable accuracy (88%). In the second experiment, I tried to extract basic shapes from the image through a relatively similar approach to the model-based classification introduced in section 3. After devising the high-level features through some trial and error and tweaking, I ended up with 5 integer-valued features. The SVM classification reached the accuracy of 90% over those features. As the third experiment, I ran decision tree classification over the same features, which generated an accuracy of 93%.

Although model-based classification tends to yield better accuracy through lighter computations, it is not scalable to the large problems. That’s because of the intellectual effort needed to devise suitable models for each class. On the other hand, Learning-based classification is much more scalable, even though it is more computational intensive and may produce not as good accuracy. A similar trade-off exists between high-level and low-level features; although high-level features tend to produce better result in a faster way, devising the suitable high-level features is by itself another problem requiring human expertise. Low-level features, are on
the other hand, heavier to work with and may produce not as good result, even though there are easier to be utilized.

References