Automatic Chart Interpretation

Catarina Pires
catarinajrpires@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal
October 2021

***Abstract*—Nowadays, charts are a key form of representing data, used in all sorts of documents. In many cases, the data underlying the chart images is also crucial; however, it is not always available. An accurate method to perform chart data extraction would benefit several areas: it can be used to improve web search results for charts or help visually impaired users understand charts in documents. An additional application is the synthesis of programs that uses graphic representation of data. The tool that motivated this research work is UNCHARTIT, a program synthesizer to recover data transformations from chart images given an input table and a chart.**

This document analyses the tools already developed with the aim of extracting data from charts and, based on the most promising methods explored, proposes a new tool, BARXTRACTOR. Our tool extracts both numerical and textual data from images of simple, grouped and stacked bar charts. BARXTRACTOR does not require human interaction, using Convolutional neural networks (CNNs) for chart type classification and Faster R-CNNs for object detection within the chart image, such as bars and numbers. For textual extraction an optical character recognition (OCR) engine is applied. Moreover, BARXTRACTOR was integrated into the UNCHARTIT tool in order to improve its accuracy and to eliminate the need of user interaction.

Experimental results for BARXTRACTOR show that it is able to successfully classify simple, grouped and stacked bar charts. Moreover, results also show that BARXTRACTOR outperforms state of the art tools that rely on user input to correctly extract data from the chart image. Additionally, the integration of BARXTRACTOR in the UNCHARTIT tool allows to improve its accuracy in finding the correct program for several table transformations. Finally, BARXTRACTOR is also able to extract textual data from the charts. The textual data can be used to improve the interpretation of the extracted numerical data.

***Index Terms*—Bar chart images, Data extraction, Object detection, Convolutional Neural Networks, Optical character recognition.**

I. INTRODUCTION

Charts are widely used to visually represent various types of data: they can be found in reports, journal and research articles, presentations, among many other documents. This visual representation of data can compact and organize loads of information into a simple and accessible image. However, the numerical and textual data underlying a chart image are not always accessible and might be challenging to obtain. The chart's data extraction is an important and useful topic that has already been explored in several research works [1]–[10], including different methodologies or subjects of study. For instance, some focus on a chart type [1], [3], [4], [8], [10], others on textual extraction [11], [12]. Each one of the explored methods have their pros and cons and have been developed in order to be applied in different contexts. Machine learning algorithms, specially Convolutional Neural Networks (CNNs) [13], have provided good results when dealing with problems related to image processing. We consider that this concept should be further explored for chart textual and numerical data extraction from chart images.

A. Motivation and Objectives

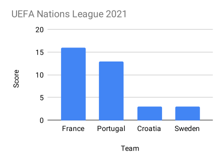
Data analysis processes have evolved over the last decade, resulting in an increasing need to employ data analysts, some of them without the ideal programming skills. For this reason, some tools for automating programming tasks emerged [14]–[19]. These tools work with examples where the user provides a series of input-output examples and the tool recovers the program that matches the input to the output, without requiring the user to program. However, until very recently, there were no tools to recover a program where the user is able to insert a visual element as the output goal.

The UNCHARTIT[1] tool [8] was proposed with the goal of recovering the program underlying a chart image. Consider an example where a user has (a) a table containing all customer complaints from 2011 to 2016, and (b) a bar chart with the number of complaints grouped annually. Suppose that a data analyst with limited programming skills needs to reproduce an updated chart with the annual consumer complaints, but he does not have access to the program underlying the generation of the chart. UNCHARTIT should be able to recover the program and data transformations applied to transform the table (a) into the bar chart (b), given only the table and the chart as input. With the program recovered by UNCHARTIT, the data analyst could easily produce an updated chart.

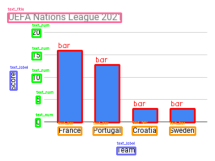
The program to be synthesized should include the chart generation, including the necessary data transformations, and it can be recovered given: (1) an input table, with the data used to create the chart, and (2) a desired output chart image. Since the desired chart is given through an image format, there is the need to extract the chart's numerical data into a tabular representation. Given the input table and the extracted table from the chart image, UNCHARTIT generates program candidates that are evaluated by a Program decider module. These candidates are ranked by the most probable correct query and the user is given the opportunity to answer some simple questions in order to disambiguate the top-n candidates.

UNCHARTIT proposes two ways of extracting data from bar charts. The first is using WEBPLOTDIGITIZER [20], a tool that requires user interaction and calibration. The other is a CNN

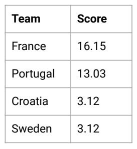
[1]http://sat.inesc-id.pt/unchartit/home/



(a) Simple Bar Chart (input).



(b) Object detection.



Team	Score
France	16.15
Portugal	13.03
Croatia	3.12
Sweden	3.12

(c) Csv file (output).

Fig. 1. In BARXTRACTOR, the input is a bar chart. In this example, a simple bar chart (a). Then, the object detection step is displayed (b) and lastly the output is represented in a csv file (c).

model that requires the user to provide two y-axis values, the minimum and maximum values of the axis.

Our proposed approach emerged by the need to get highly confident results in the chart image extraction step, in order to obtain more accurate results in UNCHARTIT. Our main goal is to develop a data extraction tool that does not require human interaction and that also includes the extraction of textual data (e.g. axis labels and titles) that might help in the numerical extraction process and in the candidate generation step from UNCHARTIT.

B. Contributions

In this document, BARXTRACTOR is proposed as a tool for extracting values from three different types of bar charts: simple, grouped, and stacked bar charts. The user simply needs to enter a chart image and the extraction process does not require any user interaction. It extracts both the numerical and textual values into a table that is saved as a csv file.

BARXTRACTOR first uses a CNN to classify the bar chart type into either 'simple, 'grouped' or 'stacked'. Then, it is composed of three Faster R-CNNs models, one for each chart type, which detects the relevant chart elements, such as bars. For each element a bounding box is detected and the correspondent label is assigned. The labels detected are extracted using optical character recognition and the numerical labels are used to automatically extract the value of each bar. The output is a csv file containing each bar value and the corresponding textual bar label.

BARXTRACTOR's most relevant steps are summarized in Figure 1. A bar chart is inserted into the system (a) and classified as 'simple'. The object detection phase detects the relevant objects and classifies them (b). And lastly, the output of the system is a csv file (c) with the extracted values of each bar and their corresponding labels.

On simple bar charts, the results obtained using BARXTRACTOR outperformed those obtained using the previous UNCHARTIT method and the WEBPLOTDIGITIZER tool. BARXTRACTOR additionally supports the extraction of grouped and stacked bar charts, obtaining similar results to the simple bar charts. BARXTRACTOR also extracts the textual labels automatically.

C. Document Structure

This document is organized in five sections. The introduction aims to present the problem, the motivation for the project and a brief overview on the proposed approach. Section II includes the preliminaries and an overview on the state of the art techniques to extract numerical and textual information from chart images. The architecture and implementation of BARXTRACTOR are described in section III, followed by section IV which presents and discusses the experimental results. Finally, section V concludes the document.

II. BACKGROUND

A. Preliminaries

In this subsection we assume the reader is familiar with Neural Networks [13], [21] and Convolutional Neural Networks (CNNs) [13], [22], [23].

1) ResNet: The residual neural network (ResNet) [24] is a CNN architecture, which was selected to perform several tasks throughout this project. It was chosen for BARXTRACTOR since it won the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [25] classification task. ResNet was designed to address the issue of adding more layers causing the network's performance to decrease [24]. It introduces residual blocks which contain a skip connection between the input and the output of the block. ResNet has numerous variants obtained by changing the number of layers. In BARXTRACTOR the ResNet-50 was chosen for classification tasks since it is more accurate than ResNet-34 [24] on ImageNet and it is unnecessary to use the networks with 101 or 152 layers, which would be slower to train.

2) Faster R-CNNs: To address object detection, given an input image, the goal is to output the various detected objects' bounding box and classification. Simple CNN architectures have a fixed output layer which is ineffective for these type of problems. Region based CNNs are object detection networks that, given an image, return for each detected element: a bounding box and a classification. Faster R-CNNs [26] are used in this project for object detection tasks. There are other object detection systems, however Faster R-CNN achieves a better accuracy for small elements [27] compared to other systems such as YOLO [28], SSD [29] and R-FCN [30].

The Faster R-CNN architecture is composed of a CNN model, to extract the image features and a Region Proposal Network (RPN), that proposes bounding boxes that may contain relevant elements. Then, it uses a Region of Interest (RoI) pooling layer to obtain a RoI feature vector. Using this vector the network outputs, for each RoI, the softmax probabilities for the object classification, and the bounding box offsets.

B. Related Work

CNNs are commonly used in chart classification [2], [6], [7], [31] and, although it has been little explored, it can be a good basis for chart images data extraction [7], [10].

The first approaches to tackle chart data extraction worked by requiring user interaction [6], [9]. Others, worked automatically [4], without deep learning, but using fixed rules, which are less generalized and more prone to errors. More recent tools use object detection networks to detect automatically elements in charts [3], [7], for instance Scatteract [3] detects the points location from scatter plots automatically. Similarly, using deep learning there is a tool [10] that uses an encoder–decoder framework that integrates CNNs and recurrent neural networks (RNNs) to extract numerical information.

For textual data detection some tools require the user to select the textual elements [9], others can be automatic by using fixed rules, such as pixels' analysis [1], [11], [12]. Recent tools can detect textual elements using object detection [2], [3], [7], [10], including the Faster R-CNN approach [2]. To extract the text, most tools apply an OCR engine [1]–[3], [9]–[11]. Finally, in order to obtain the numerical values of a chart usually the textual information of the vertical axis is extracted to obtain a ratio that represents the number of pixels utilized in the image to compose a data unit.

Recently, the UNCHARTIT tool [8], introduced before, also explored chart numerical data extraction. Since the desired chart is given through an image format, there is the need to extract the chart's numerical data into a table. The article focuses on bar charts and proposes two ways of performing the extraction. The first is through WEBPLOTDIGITIZER [20] version 4.2, a semi-automatic tool that is able to extract numerical data from simple 2D bar charts. However, the tool requires user calibration and does not extract bar labels. The other proposed extraction method uses CNNs, more specifically the EfficientNet-B7 [32] with an alternative output layer. The first n nodes of the output layer contain the probability that the chart has $i \in \{1, 2, ..., n\}$ bars, and the next n nodes represent a relative height from 0 to 1, defining how full each bar is. Then, using the maximum and lowest value of the bar chart (which are values provided by the user) the real numerical value of each bar is obtained using linear interpolation. In this approach the labels' extraction was also not explored.

III. AUTOMATIC BAR CHART EXTRACTION

The software presented in this document was named BARXTRACTOR and its goal is to extract the visual information represented on a bar chart image into a data table.

A. Architecture

To get a better understanding of the overall system we should take a look at Figure 2. The input of BARXTRACTOR is a bar chart image. That image is fed to a classifier which will label it in either a simple, grouped or stacked bar chart. After being successfully classified, the image goes through one of the three Faster R-CNN models and outputs several bounding boxes with a corresponding label. For instance, it finds several bounding boxes that translate to the 'bar' label.

Then, a couple of post-processing techniques are added after the elements' detections in order to tackle common issues, such as, understand and correct if there are missing bars.

The labels that correspond to a textual or numerical value are subjected to an OCR procedure to obtain its correspondent information. The detections that are believed to represent a "number" in the vertical axis, go through an outlier detection (RANSAC). Those numbers and their correspondent coordinates in the image are used to calculate a ratio between pixels and data which is referred to as $pixel\_per\_data$ ratio. To obtain the real value of each bar, simple calculations are performed using the ratio to transform the height in pixels of each bar in the real height in the units of the chart.

The title, labels and legends however are not used in the calculations, but are part of the chart's csv output file, which aggregates all the extracted information.

Each of these steps will be detailed in the next subsections.

B. Chart image Classifier

The chart image classifier is the first step of the BARXTRACTOR system. It was integrated in BARXTRACTOR since there are three disparate object detection models for the three bar chart types. It is necessary to identify which of the three object detection networks is going to be employed in the data extraction process. Given a chart image, the classifier returns a prediction of the chart type, which can be either 'simple', 'stacked' or 'grouped' bar chart.

1) Model and training: To implement an image classifier, a model pre-trained on ImageNet dataset was chosen, more specifically, the ResNet-50 [24]. Before the training, all the images are resized to the same dimensions and transformed into a tensor. The dataset is divided into 80% for training and 20% for validation. During the training phase, the pre-trained layers are not updated unlike the fully connected layer which needs to be trained for our bar chart images. The chosen optimizer was Adam [33] and the loss function was the negative log likelihood loss[2], which is useful for classification problems. In order to determine the parameters that would lead to the best classification results, the model was trained with different parameters, learning rate and number of epochs. By analyzing the training and test loss values, the best results were observed with a learning rate of 0.002 and 300 epochs of training. After the model was trained, if given a chart image it will return a prediction for one of the three classes.

2) Classifier's dataset: The dataset used to train the classifier is a compilation of images that were used in the Faster R-CNN training and evaluation process (dataset explained in the next subsection), in addiction to 125 more images for each chart type. These 375 images were collected from web search engines to try and generalize the classifier to real charts used in different contexts and tools. These images were obtained using search queries such as "bar charts" in web search engines.

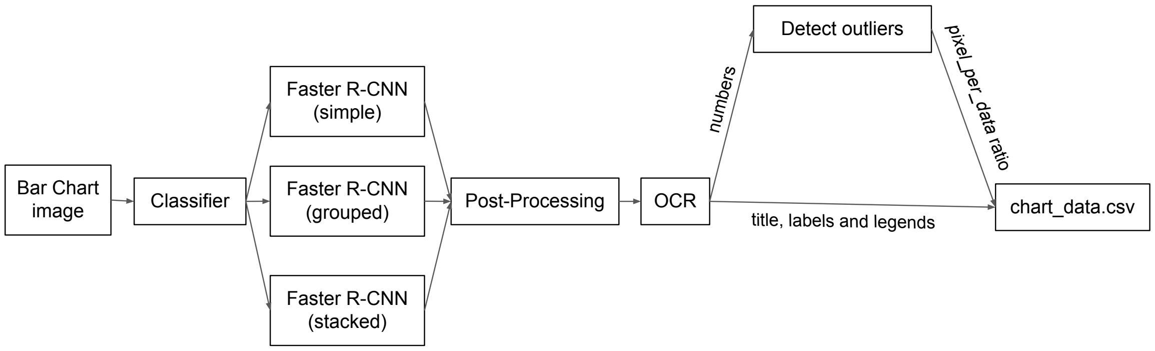
Then, a random 10 percent of each class was removed from the training dataset so that it could be used to evaluate the classifier after the training phase. Neither the training nor the testing phases make use of this fraction of images. After this refinement, the classifier's training dataset is composed of a total of 5122 images.

C. Faster R-CNN

The Faster R-CNN, in the context of this work, is used to predict the bounding box and class of each element of interest in a given bar chart image.

[2]https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html

Fig. 2. BARXTRACTOR's architecture.



1) Models and training: A different model was trained for each of the three classes: simple, grouped and stacked bar charts. After the classification step, the classifiers' result defines which model will be used to extract the bar chart information. The detection step could have been performed using only one general model although it would not be so precise for each chart type.

The chosen model was the Faster R-CNN, which uses the ResNet-50 as the model's backbone CNN. The model is pre-trained on the diverse COCO dataset[3]. The decision of finetuning from a pre-trained model was made since training a CNN algorithm from scratch requires much more time and data. Since the classes we want to train are relatively distinct from one another, and typically found in the same spots, a pre-trained model satisfies our needs and converges faster to the solution.

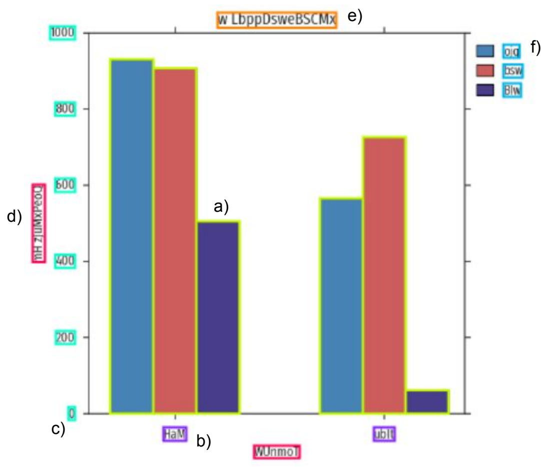
The first step is to configure the dataset and the model's attributes. First, the images are transformed into pytorch tensors. The dataset is split into 85% for training and 15% for testing. Then, the parameters are defined, the chosen optimization algorithm was the stochastic gradient descent (SGD) with a learning rate of 0.0005. A learning rate scheduler is also defined, with parameters that determine how the learning rate value decays. The models for simple, grouped and stacked bar charts were trained 60, 65 and 60 epochs respectively. The number of epochs for each model was determined based on the outcomes of each epoch, namely the current AP values. The models do not need a huge amount of training since they are pre-trained.

When training, the model provides the user some important information to analyze, such as the average precision and average recall for different IoU values. This helps comprehend the model's current stage on learning to recognize the desired elements of an image. Unlike many object detection problems, for BARXTRACTOR it is crucial that the bounding boxes are tight to the exact space of the bars and numbers for the computations to be accurate, which implies we must prioritize a high IoU.

After the training process the Faster R-CNN model is saved and given an input image, it outputs a bounding box, a label and a score, related to the confidence in the prediction.

[3]https://cocodataset.org/

Fig. 3. Labels contained in an image from the grouped bar charts dataset.



2) Faster R-CNN datasets: Three datasets were created, for training the simple, grouped and stacked bar charts models. The datasets are composed by several images and a correspondent $annotations.csv$ file containing several bounding boxes with a correspondent label. The bounding box is represented through four values: $xmin$, $ymin$, $xmax$, $ymax$. The images are generated using $matplotlib$ and the coordinates of each region of interest are stored along with the labels on the annotations file. The procedure used to generate the bar chart images with each object's corresponding coordinates was initially adapted[4] from Fangfang Zhou et al's work [10]. The procedure was adjusted to include the random generation of grouped and stacked bar charts, and it was also modified to include additional parameter variation for the chart images.

Each dataset entry is composed by an image and several bounding boxes with its corresponding labels. Figure 3 shows an example of the bounding boxes required for the information contained in a grouped bar chart image. The $annotations.csv$ file is composed of eight columns, being: filename, width, height, class, xmin, ymin, xmax and ymax.

There are six labels in all of these three datasets, which are:

- bar (the bar rectangle) (Fig. 3 a))
- text_bar (the labels below each bar) (Fig. 3 b))

[4]https://github.com/csuvis/BarchartReverseEngineering/blob/master/generate_random_bar_chart.py

- text_num (the vertical numerical values) (Fig. 3 c))
- text_label (the axis label x and y) (Fig. 3 d))
- text_title (the title of the chart) (Fig. 3 e))
- text_legend (the textual legend of each color (Fig. 3 f))

Some labels are not mandatory in a bar chart image, for instance, in a simple bar chart it is uncommon to find charts with a color legend since there is usually only one bar color.

In all datasets several data parameters were changed in order to generate a diverse dataset. All include changes in the number of bars, number of groups, the spacing between bars or groups, the charts theme colors and style. Also the textual parameters are generated randomly and vary in size and length: the titles, axis labels and bar labels. The figure quality and size, in height and width, is also varied. The file format is also randomly assigned between .jpg and .png.

3) Simple bar charts dataset: The simple bar charts dataset is composed of 1445 images. The initial number of images was 1000 but the first results showed that some image parameters were missing, for instance, initially there was no 'legend' label and when the image had a label present, the Faster R-CNN recognized it as an extra bar. Another problem was that only the small titles were recognized. This proved that the dataset had to be more diverse so new images with certain characteristics were added: images with more bars, with lower distance between bars, larger titles and the color legend label.

4) Grouped and Stacked bar charts datasets: Both the grouped bar charts and stacked bar charts' dataset are composed of 1300 images similarly with diverse parameters. As before, the initial number of images was 1000 and the remaining 300 were added based on what was revealed to be missing on the dataset. Besides the simple bar charts' variations, for these we must also consider changing the number of groups and bars per group. The spacing between bars of the same group and between groups. The colors of the bars within a group and its correspondent labels.

For stacked bar charts, the label 'bar' is considered the rectangle represented by one color on each group. Meaning, each color in a group of stacked colors is considered a 'bar' when composing the dataset. The reason for not recognizing the entire bar is because then we would have to check manually where the colors change within each bar.

D. Post-Processing

There are situations in which the Faster R-CNN can not solve the entire problem alone. Sometimes a bar has a zero value and it is not represented for the object detection network to detect but there is still the bar label underneath. Hence, some Faster R-CNN post-processing methods are applied in order to correctly find bars in those situations.

1) Simple bar charts: For simple bar charts, the bars detected by the Faster R-CNN with a confidence higher than 70% are added to the bars' list and the others are stored if needed later. Between experiments, the value 70% proved to be the right threshold between a correctly found bar and a misdetected one.

First, the mean width of the bars is computed. One of the conditions that is examined is if each label has a corresponding bar. If a bar is not found but there is a label below the expected spot, we can assume a bar is missing. The first approach is to search for a bar, located on top of the label, among the recognized bars with confidence score lower than 70%. If it is found, it is added to the correctly recognized bars.

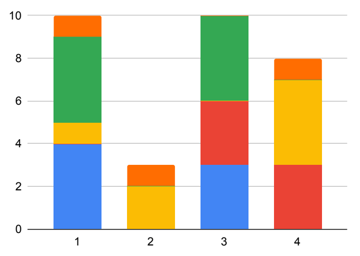
The bars and their corresponding labels are ordered by their horizontal positions. After, the distance between consecutive bars is analyzed. If there is a distance higher than the minimum distance between consecutive bars plus half the bar width, we assume a bar is missing in that spot. If there is a bar label in that spot, without a corresponding bar, then a bar was not recognized by the Faster R-CNN and is added to that place with only 2 pixels height. However, this method does not solve the case where a bar is missing in the first or last spot of the chart, so if there is a label with no bar in the beginning or in the end, and at the same distance as the others, it is similarly added a bar in that spot.

Other verifications are made for simple bar charts, related to check if a label was misdetected for another, for instance, if a short x-axis label is detected at the same height as the bar labels, it was misdetected for a bar label. Another verification that is made is if there is a detected number outside the expected x coordinate, since charts with scientific notation were not considered in this project. The numerical values can still be extracted without the multiplication for the scientific notation value.

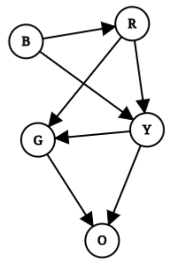
2) Grouped bar charts: For grouped bar charts, the bars detected by the Faster R-CNN are considered if the confidence of the detection is higher than 70%. For this chart type, the first step is to divide the bars in groups. This is performed by ordering the bars by their x coordinate, calculating the distance between consecutive bars and joining the closer ones into a group. Since this is a simple aggregation method, an inconsistency check is performed that verifies if the number of bars in a group is lower than the number of color legends detected or if the number of bars is different between groups.

If an inconsistency is found, a different approach is taken by checking the distance between each bar and the bar labels. There is only one bar label per group and it is the closest to each group's bar. If there are missing bars in a group, the distance between the group bars is checked and the places with missing bars get, as before, a bar added to that spot. After, if there are still missing bars in the group, it is because they are the first or last bars, and the same approach is used as with simple bar charts. Other verifications are made, as before, related to check if a label was misdetected for another.

3) Stacked bar charts: The stacked bar charts have a different approach to check for missing bars. Colors are stacked on top of each other and we may not know the missing color's location. The first step, similar to grouped bar charts, is to divide the bars into groups. For this chart format, the process of separating into groups is simpler, since same group bars are all stacked in the same x cardinal position. Then, if the number of bars differ between groups or it differs from the number of



(a) Stacked Bar Chart.



(b) Colors' directed graph.

Fig. 4. Example of a stacked bar chart (a) and the corresponding colors' directed acyclic graph (b).

color legends, we can assume that there are missing bar colors. In order to find which colors are missing in a group and extract them in the same order as they are represented in the chart, we need to understand the color disposal. A procedure based on directed acyclic graphs was used to accomplish this. It works by going through every bar of the chart and checking the colors upwards. Every time a color is found next to each other, a directed edge between those colors is added to the graph. Figure 4 (b) shows the resulting directed color graph corresponding to the stacked bar chart in Figure 4 (a).

When the graph is completed, we apply a topological sort algorithm in order to find the color disposal. For Figure 4 it would be [B,R,Y,G,O].

E. Obtain the real chart values

The first BARXTRACTOR's method to acquire the real values of a bar chart was inspired by the previous UNCHARTIT approach. It consisted in asking the user to provide the lowest and highest values depicted on the vertical axis, which correspond to the minimum and maximum values of the chart. However, our goal included the automation of the system, which means the tool has to work without user interaction. With the purpose of automating the system, OCR is applied on BARXTRACTOR both to find the numerical values of the vertical axis and to find the textual labels.

1) Optical Character Recognition: The chosen tool used to output the text, given a cropped image, was EASYOCR from PYTORCH. Every detected element in the original image, that needs to go though the OCR process, is cropped into a new image containing only that element. Before executing the OCR, the image is cropped by the detected element's bounding box plus an extra number of pixels. These extra pixels are added to ensure the element's area is fully contained inside the cropped image, otherwise it can affect the OCR process. Another transformation applied that has shown to improve the results is to add padding to the cropped image. Because the OCR tool is not designed to handle a very zoomed-in image of an element, this transformation is useful. The padding acts as a "zoom out" in relation to the image's center. The figure is then saved, and the EASYOCR tool is applied in order to output the characters found.

To be clear, the EASYOCR tool supports the recognition of the text location, which was tested on bar charts. When compared to the developed Faster R-CNN model, EASYOCR was less accurate on recognizing the chart image text. As a result, the text is cropped into a new image before being fed to the OCR.

The extracted numerical values are the most important information that will lead us to the real chart values. For instance, in Figure 4(a) the numerical labels are: "0", "2", "4", "6", "8" and "10". Sometimes a "0" is recognized as an "O" and a "0," as a "Q". Hence, after the OCR, an extra step is applied for numerical labels in order to minimize some inaccuracies detected in the results. These specific errors were corrected simply by replacing these characters with the corresponding numbers. Another common error is when the OCR outputs, for instance, "060" instead of "0.60". When this happens, a "." is added after the first zero.

2) Outliers' detection: By detecting the bounding box of each element, through the Faster R-CNN model, we get the y-coordinates in pixels of each numerical label. Before proceeding to the actual calculation of each bar value, we need to perform an outlier detection on the numerical values extracted. Assume we have the following numerical labels extracted from a chart image and their correspondent coordinates:

$$labels = [0, 50, 100, 15, 200, 25, 300]$$
$$coordinates = [100, 200, 300, 400, 500, 600, 700]$$

The labels "15" and "25" were returned by the OCR with a missing "0" and should not be considered in the following calculations. For this reason Random sample consensus (RANSAC) method [34] is applied to detect these outliers. RANSAC returns a list of the inliers and outliers based on the labels and coordinates of this problem. For the example above this would return the following inliers.

$$X = [0, 50, 100, 200, 300](labels)$$
$$y = [100, 200, 300, 500, 700](coordinates)$$

On SCATTERACT [3] the RANSAC regression solution was proven to work well on outlier detection comparing with other robust solutions. It also worked well with BARXTRACTOR, thus it was integrated into the system. After the outliers' detection, only the inliers are considered for the calculations.

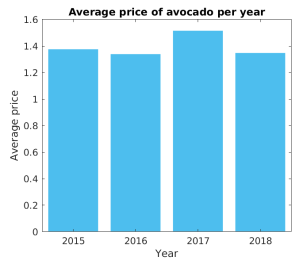
Each bar's height in pixels has now been determined. In order to get the real value of each bar, a ratio of pixels per data is calculated as following.

$$pixel\_data\_ratio = \frac{\sum_{i-1} \dfrac{abs(X[i+1] - X[i])}{abs(y[i+1] - y[i])}}{len(X) - 1} \quad (1)$$

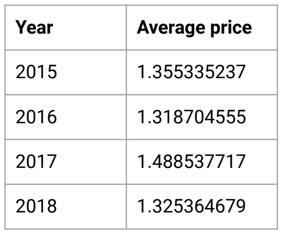
In Equation 1, X represents the numerical label values and y the corresponding y coordinates.

Computing the $pixel\_per\_data$ ratio for the previous example would get a ratio of 0.5.

The height in pixels of each bar is then multiplied by the $pixel\_per\_data$ ratio to obtain the real value of each bar. The height in pixels can be obtained by subtracting the known $ymax$ by the $ymin$ in pixels of the bounding box.



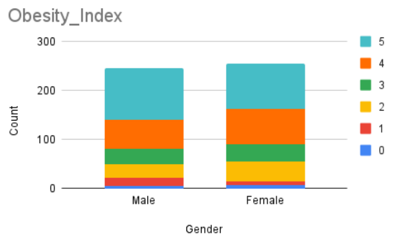
(a) Simple Bar Chart image.



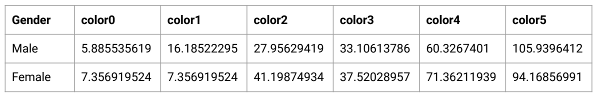
Year	Average price
2015	1.355335237
2016	1.318704555
2017	1.488537717
2018	1.325364679

(b) Bar chart corresponding csv output.

Fig. 5. Example of a simple bar chart and BARXTRACTOR's csv output.



(a) Stacked bar Chart image



Gender	color0	color1	color2	color3	color4	color5
Male	5.885535619	16.18522295	27.95629419	33.10613786	60.3267401	105.9396412
Female	7.356919524	7.356919524	41.19874934	37.52028957	71.36211939	94.16856991

(b) Stacked bar chart csv output

Fig. 6. Example of a stacked bar chart and the systems' csv output.

F. Output of the system

BARXTRACTOR's last step is to aggregate the extracted information into a csv file. The file is used to store the extracted bar values and label information. For simple bar charts, the csv output file is quite simple. It consists of two columns which represent for each bar: the bar label and the corresponding bar value. Figure 5 presents a simple bar chart example (a) and its corresponding output csv (b).

For grouped and stacked bar charts the csv representation is slightly different since there is more than one bar to represent per group. Figure 6 displays an example of the input and the corresponding output of a stacked bar chart image. In the file, the lines represent each group while the columns correspond to each of the color legends.

IV. RESULTS

In order to evaluate the developed method, several images were tested and two tools were used to compare the results obtained by the Faster R-CNN method. One is the previous UNCHARTIT approach, the EfficientNet-B7 adaptation, and the other is WEBPLOTDIGITIZER, used in the UNCHARTIT article [8] to compare with its results.

To analyze the following simple bar chart results there are some aspects we should keep in mind. BARXTRACTOR, the Faster R-CNN approach, does not require user interaction, it is fully automatic. Its dataset, described in detail in Section III, is composed of almost 1500 *matplotlib* generated images and a file containing each elements' label and bounding box. The previous UNCHARTIT approach consisted in training the EfficientNet-B7 network with 90000 latex bar charts. The training dataset has each image and the correspondent value of each bar. This approach requires the user to insert the minimum and maximum value of the bar chart image. The WEBPLOTDIGITIZER tool requires the user to manually calibrate the y-axis by clicking on two points of the axis and inserting its correspondent values. After this, it requests the user to select the bars' color, the width of the bars in pixels (Δx) and the height of the highest bar in pixels (Δval).

A. Numerical extraction evaluation

In order to evaluate the effectiveness of the numerical extraction of a bar chart, the number of detected bars and the height of each bar are the two main points that have to be considered. We consider the number of detected bars to be correct if the output has the same shape as the data. To evaluate the extracted height of the bars, for each chart the mean absolute error (MAE) is calculated. The MAE (Equation 2) indicates the error between the real (x) and the observed values (y), being n the number of values.

$$mae = (\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i| \qquad (2)$$

The MAE is calculated using the output csv of each tool. In order to output an error between 0 and 1, the data is normalized.

The bar chart images used in the numerical extraction evaluation were generated using three different tools: latex, matlab and excel. The data used to generate the images is real data extracted from Kaggle[5]. For simple bar charts these are the same 50 instances that were used to evaluate and compare the numerical extractor approach in the UNCHARTIT paper. For grouped and stacked bar charts 40 instances of real data were extracted from Kaggle and different aggregations were applied in order to generate groups. In total two images were generated per instance resulting on 80 charts per tool to evaluate the grouped charts and 80 more per tool to evaluate the stacked bar charts.

1) Simple bar charts: In the UNCHARTIT tool article, 50 instances were extracted from Kaggle in order to evaluate the tool. These instances were transformed into latex charts in order to analyze if the program could correctly return the charts' number of bars. Then, for the charts where the number of bars were correctly estimated, the MAE is computed. The ones with scientific notation were not considered in this calculation since this was not explored. In order to extend this study and have a more diverse evaluation, the same 50 instances were used to generate matlab and excel charts with the same data but different visual representations.

Table I presents the results obtained on the 50 images related to the **number of charts where the correct number of bars**

[5]https://www.kaggle.com/

	Latex (50)	Matlab (50)	Excel (50)
WebPlotDigitizer	**48**	45	46
EfficientNet-B7	46	28	26
Faster R-CNN	**48**	**49**	**48**

TABLE I
NUMBER OF CHARTS WHERE THE NUMBER OF BARS WAS CORRECTLY PREDICTED AMONG THE 50 SIMPLE BAR CHART IMAGES.

	Latex	Matlab	Excel
WebPlotDigitizer	**0.002201**	0.006543	0.026541
EfficientNet-B7	0.037356	0.220274	0.212933
Faster R-CNN (Without OCR)	0.019915	0.005507	0.0187
Faster R-CNN (With OCR)	0.010964	**0.00542**	**0.011812**

TABLE II
MAE OF THE BAR VALUES ON THE SIMPLE BAR CHART IMAGES WITH CORRECTLY PREDICTED NUMBER OF BARS.

was detected. The EfficientNet-B7, as expected, performs better on latex images and similarly worse for matlab and excel images. This is because the training dataset is composed of only latex images and the tool is not able to generalize.

The WEBPLOTDIGITIZER tool performs better on latex images which vary less visually. For instance, the colors are predefined to be the same and the distance between bars is also fixed. On matlab and excel it performs slightly worse since the images are more diverse. For instance, if a chart has different colored bars, the WEBPLOTDIGITIZER tool can not recognize them since it asks the user to select only one color, which the tool assumes it is the same for every bar. Another situation WEBPLOTDIGITIZER fails to recognize the bars is when a bar is very small, overlapping with the horizontal axis. BARXTRACTOR solves this problem by also detecting the bar labels and checking for each label if there is a corresponding bar.

The Faster R-CNN solution performs equally good on the data generated by the three tools, but better on matlab, since the training dataset was generated through *matplotlib.pyplot* module. The one chart that all tools fail to recognize has the first bar really small, overlapping with the axis, and without a bar label, which is a difficult case to solve. In general BARXTRACTOR performs better on detecting the correct number of bars between the data generated on the three tools.

In terms of **mean absolute error** four situations were considered as we can observe on Table II. These are the WEBPLOTDIGITIZER tool, the EfficientNet-B7 approach, and the Faster R-CNN both without and with OCR. The Faster R-CNN without OCR was tested before the tool was completely finished and uses the input of the user to indicate the minimum and maximum value of the y axis, meaning, it uses the same input as EfficientNet-B7 approach. In the Faster R-CNN with OCR tests, the tool was already automatic using OCR to extract the various numbers of the y-axis and does not require any user interaction. The MAE of each tool is related to the images in which the number of bars was correctly detected.

EfficientNet-B7, as expected, has a lower error for latex images, since it was trained on them, and performs similarly worst on matlab and excel images which are, as said before, visually more complex.

WEBPLOTDIGITIZER performs better on latex images, then on matlab and lastly on excel. This is explained by the fact that they are in this order more visually complex while WEBPLOTDIGITIZER performs really well on simple images but does not support some features. For instance, a common problem that may cause a higher error is when the chart title or axis have a similar color as the bars. Since this tool works on recognizing the bars by their color, and if the title has a similar color to the bars, it will recognize the bar height higher than it was supposed (in the title).

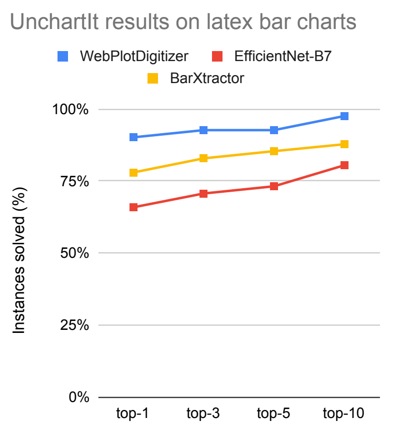
On the first tests to check the accuracy of the new implementation, the OCR was not yet developed. The tests without OCR were performed using the highest and lowest detected 'number' labels found, and were given as input the minimum and maximum values in the chart to calculate the $pixel\_per\_data$ ratio. The Faster R-CNN with OCR performed better overall because it is more accurate to check all the numerical values and their locations than only the detected highest and lowest label, which might not be the correct ones. As expected it has the lowest error on matlab images because of the dataset. It performs equally good on latex and excel despite not having had contact with these images during the training. However it does not have the lowest MAE on latex, WEBPLOTDIGITIZER performs better on simple cases. From the results obtained on Faster R-CNN, it can be observed that the detected rectangle is not always as tight to the bar as it could be, this could be solved by adding latex and excel images to the dataset although it could be complex to get the pixel location of each bar on these tools. In general, besides being the only fully automatic tool, BARXTRACTOR performs well on the chart images generated by all different tools.

To further evaluate the performance of these numerical extraction methods, the **UNCHARTIT tool** was used to compare the performance on finding the correct query for each problem. These results were obtained using the extracted numerical values from the images where the number of bars was correctly identified. The extracted numerical values were provided to the UNCHARTIT together with the original table of data. Using a timeout of three minutes, Figure 7 show the success rate on finding the correct query on the top-1, top-3, top-5 and top-10 candidates.

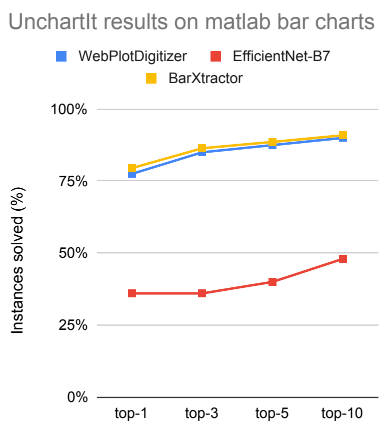
For latex images, it is possible to observe that the WEBPLOTDIGITIZER tool has a better performance, which is expected since it has the lowest MAE for latex images. For matlab images, the Faster R-CNN approach performs best on UNCHARTIT followed by WEBPLOTDIGITIZER as expected by the MAE results. The excel images results on UNCHARTIT are also related to the MAE, since the Faster R-CNN has a lower error, it helps UNCHARTIT perform better on finding the correct query.

As expected, the success rate of synthesizing the correct query in UNCHARTIT is correlated with finding a good approximation of the chart values.

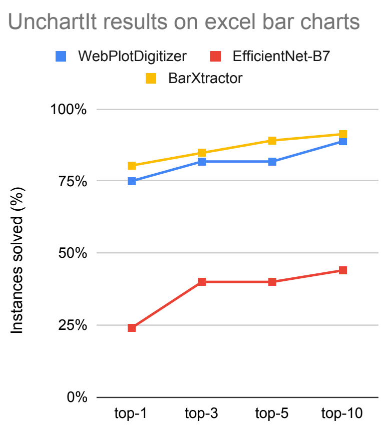
2) Grouped and Stacked bar charts: To evaluate the grouped and stacked bar charts 40 instances were extracted



(a) Results on latex.



(b) Results on matlab.



(c) Results on excel.

Fig. 7. UNCHARTIT results on the number of queries solved both for latex (a), matlab (b) and excel (c) bar charts.

from Kaggle and grouped in specific ways. For instance, Figure 6 is a result of grouping the obesity index (1-5) in a school and by genders. With these 40 instances two visually different plots were generated per instance, varying parameters such as the colors and the spacing between bars. Generating a total of 160 plots to evaluate the grouped bar charts and another 160 to evaluate the stacked charts with the same data, 80 were generated on matlab and 80 were generated on excel.

The results of this evaluation on **grouped bar charts** shown that in 95% (76/80) of the matlab charts the number of bars and groups were correctly identified with a mean absolute error of 0.053231 and a standard deviation of 0.115919. On excel images 91.3% (73/80) were correctly identified with a mean absolute error of 0.006493 and a standard deviation of 0.006037. Although matlab has a higher MAE it gets correct on more difficult images and the error is more spread as we can observe by the standard deviation. The network did not have contact with excel images during the training phase unlike with matlab images, thus getting correct more complicated cases.

Using the same data, 80 **stacked bar charts** were generated in matlab and 80 on excel. The number of images with correctly guessed number of bars is 98.8% (79/80) and 92.5% (74/80) respectively for matlab and excel. For matlab images, the MAE corresponds to 0.036683 with a standard deviation of 0.077973. For excel images the MAE corresponds to 0.014248 with a standard deviation of 0.029962. The same situation happens here on matlab, as it gets more correct complex images, the MAE is slightly higher but the standard deviation is also higher. As for grouped bar charts this is also due to the fact that the training was performed on *matplotlib* generated images. On average the MAE is low for both tools.

The UNCHARTIT tool does not support multi-column output tables, since its internal metrics would need to be adapted. However, the MAE analysis for grouped and stacked bar charts demonstrated to be similar to the MAE for simple bar charts. The WEBPLOTDIGITIZER tool and the previous UNCHARTIT approach are not able to extract data from grouped nor stacked bar charts.

	latex	**matlab**	**excel**
100% correct labels (no character differences)	70.6522%	79.3814%	65.7143%
Labels <=1 character difference	72.8261%	80.756%	77.1429%
Labels <=2 character differences	98.913%	99.6564%	97.8571%
Labels >2 character differences	1.087%	0.3436%	2.1429%

TABLE III
BAR LABELS SUCCESS RATE (NUMBER OF TOTALLY SUCCESSFUL READS)

B. OCR

In order to evaluate the precision of the the textual labels extraction, more precisely on the bar labels, the same 50 instances on latex, matlab and excel were used. We evaluate both the success rate on extracting the label fully correct and the precision of the bar labels in these extraction, by checking the number of differences found between the real bar label characters and the extracted characters.

The metric considered for this evaluation is the **medium number of differences**, being, the number of differences needed to transform the characters recognized in the actual characters of the label. A difference is considered when there is the need to add a character (1) and to remove a character (1). So when there is the situation of replacing a character, that counts as 2 differences.

The results proved that each label need a medium number of 0.59 character changes for latex, 0.40 character changes for matlab and 0.63 for excel. Excel needs slightly more changes because it uses different fonts for each theme, and some might be more complex. Matlab has a simple font in all charts and needs slightly less character changes. Overall these results are promising since the medium number of character changes per label is less than 1.

The **success rate** in this context is considered the number of totally successful reads, meaning, the labels which characters were 100% correctly recognized. By analyzing Table III we can conclude that for at least 98% of all labels and tools the number of differences is less or equal to 2, which means a character replacement.

The results have shown that only two changes to the textual label are enough. It could be useful to use these labels in the

UNCHARTIT synthesis tool by using an approximate string matching with a maximum of two operations to correspond the extracted labels from the chart to the column names of the original table. To implement this, every column name of the original table has to be compared with the extracted bar label and be considered a match if the number of differences is lower than two operations.

C. Classifier

In order to evaluate the classifier in an unbiased form, a random 10% of each of the three dataset classes was separated from the training set into a different folder for evaluation. These images were not seen by the classifier during the training phase. The evaluation images contain the charts used in the training and evaluation of the numerical extraction module and some real images from web. A total of 1130 images were used to perform this evaluation. The results show that 93.5% of the simple bar charts, 92.0% of the grouped bar charts and 92.4% of the stacked charts were correctly labeled.

The simple bar charts have two common situations where they are mislabeled: when the bars are too close to each other and when there is more than one bar color they may be classified as a grouped bar chart. The same problem occurs with stacked bar charts which may also be incorrectly labeled as 'grouped' when its bars have little distance between each other. Grouped bar charts might also be classified as 'simple' when there are two colors and one color's bars are always small. Simple and stacked bar charts may also be confused when the colors are similar.

We could try to solve the problems mentioned above with a bigger and more diverse dataset, and by applying a post processing technique that solves some of these issues. For instance, differentiating between simple and stacked bar charts could be solved by checking if there is only one color in each bar. The miss identification between grouped and simple could be solved by checking if each bar has a label or more than one bar per label as we already check with the Faster R-CNN.

D. Results' summary

In summary, BARXTRACTOR is: 1) fully automatic; 2) able to detect and support several bar chart types; 3) has better or similar performance than previous tools that need input from the user; 4) the incorporation of BARXTRACTOR in UNCHARTIT allows to synthesize more SQL queries due to a low MAE on the data extracted from charts.

V. CONCLUSIONS

This document presents the BARXTRACTOR tool, which extracts the numerical and textual information from simple, grouped and stacked bar charts. The tool can be summarized in the following steps: 1) The user inputs the chart image. It can be either a simple, grouped or stacked bar chart; 2) The image is automatically classified into one of the three chart types; 3) The corresponding Faster R-CNN model is employed on the chart image for object detection. 4) Post processing techniques are applied to resolve common problems in the extraction. For instance, BARXTRACTOR can detect missing bars by analyzing whether there is a label without a corresponding bar; 5) Each bar's numerical information is automatically computed using the y-axis extracted values, through OCR; 6) The results are exported to a table containing each bar's numerical values and the corresponding textual labels.

One of the main BARXTRACTOR's contributions is that the extraction process is fully automatic. Previous tools always needed input from the user. In BARXTRACTOR, the Faster R-CNN detects the vertical axis' numerical labels automatically and extracts them using OCR.

Another contribution is that BARXTRACTOR can detect missing bars, or bars with a value of zero, since it also extracts the bar labels. In addition, our tool obtains the bar labels and axis labels, which can be useful for the user, and for the UNCHARTIT program synthesis stage.

The results obtained by BARXTRACTOR are promising when compared to the previous UNCHARTIT approach and to the WEBPLOTDIGITIZER tool. BARXTRACTOR proved to be more flexible in terms of extracting values from chart types generated by different tools. It can achieve low errors when extracting from latex simple bar charts. However, even without requiring human interaction, its results stand out on bar charts generated through matlab and excel tools. Unlike the other two tools, BARXTRACTOR is additionally able to extract values from grouped and stacked bar charts, which have similar extraction errors to the simple bar charts' results.

A. Future Work

There are numerous ideas that should be further explored in the future. Related to the UNCHARTIT tool, with BARXTRACTOR one could minimize the number of generated candidates queries by the synthesizer, or help in the candidate disambiguation, by using the textual labels extracted.

UNCHARTIT should also be generalized in order to be able to synthesize extracted data from other table shapes. For instance, BARXTRACTOR extracts data from grouped and stacked bar charts which generate a table with more than one column. However, at the moment, UNCHARTIT is only able to synthesize tables with one column of data. If this is generalized to other table shapes, UNCHARTIT will be able to receive data from lots of other chart types.

Related to BARXTRACTOR, one relevant idea would be to add chart images from the real wold to the Faster R-CNN training dataset. Although these images are easy to obtain, it is very time consuming to manually annotate the bounding box and labels of every element detected in the chart.

By using similar approaches, BARXTRACTOR can also be adapted to other chart types. For pie charts, detecting a bounding box would not be precise on detecting each slice. Possible solutions are using Mask R-CNN [35] to find the mask of each slice. Or detecting the line limit of each slice [36]–[38], the line that connects to the center of the pie. BARXTRACTOR's approach to detect the vertical axis' numerical values and convert the pixels into real data, can be useful for all the chart types containing an axis.

Acknowledgment

This work was partially supported by Portuguese national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020, DSAIPA/AI/0044/2018, and project ANI 045917 funded by FEDER and FCT.

References

[1] Rabah Al-Zaidy and C. Giles. Automatic extraction of data from bar charts. pages 1–4, 10 2015.

[2] Abhijit Balaji, Thuvaarakkesh Ramanathan, and Venkateshwarlu Sonathi. Chart-text: A fully automated chart image descriptor. *arXiv:1812.10636*, 2018.

[3] Mathieu Cliche, David Rosenberg, Dhruv Madeka, and Connie Yee. Scatteract: Automated extraction of data from scatter plots. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer, 2017.

[4] Paramita De. Automatic data extraction from 2d and 3d pie chart images. In *2018 IEEE 8th International Advance Computing Conference (IACC)*, pages 20–25, 2018.

[5] Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. Model-based chart image recognition. In *International workshop on graphics recognition*, pages 87–99. Springer, 2003.

[6] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bong-shin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. ACM, May 2017.

[7] Xiaoyi Liu, D. Klabjan, and P. Bless. Data extraction from charts via single deep neural network. *ArXiv*, abs/1906.11906, 2019.

[8] Daniel Ramos, Jorge Pereira, Inês Lynce, Vasco Manquinho, and Ruben Martins. UNCHARTIT: an interactive framework for program recovery from charts. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, pages 175–186. IEEE, 2020.

[9] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. 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)*, pages 393–402. ACM, 2011.

[10] Fangfang Zhou, Yong Zhao, Wenjiang Chen, Yijing Tan, Yaqi Xu, Yi Chen, Chao Liu, and Ying Zhao. Reverse-engineering bar charts using neural networks. *Journal of Visualization*, 2020.

[11] A. Mishchenko and N. Vassilieva. Chart image understanding and numerical data extraction. In *2011 Sixth International Conference on Digital Information Management*, pages 115–120, 2011.

[12] N Vassilieva and Y Fomina. Text detection in chart images. *Pattern recognition and image analysis*, 23(1):139–144, 2013.

[13] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[14] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

[15] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. Program synthesis using conflict-driven learning. *ACM SIGPLAN Notices*, 53(4):420–435, 2018.

[16] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 683–698, 2017.

[17] Dmitri V Kalashnikov, Laks VS Lakshmanan, and Divesh Srivastava. Fastqre: Fast query reverse engineering. In *Proceedings of the 2018 International Conference on Management of Data*, pages 337–350, 2018.

[18] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466, 2017.

[19] Sai Zhang and Yuyin Sun. Automatically synthesizing sql queries from input-output examples. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 224–234. IEEE, 2013.

[20] WebPlotDigitizer Version 4.4. https://automeris.io/WebPlotDigitizer, 2020. Accessed: September 30, 2021.

[21] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[22] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[27] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2017.

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[30] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[31] X. Liu, B. Tang, Z. Wang, X. Xu, S. Pu, D. Tao, and M. Song. Chart classification by combining deep convolutional networks and deep belief networks. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 801–805, 2015.

[32] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[35] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[36] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1145–1153, 2017.

[37] Xiang Li, Jun Li, Xiaolin Hu, and Jian Yang. Line-cnn: End-to-end traffic line detection with line proposal unit. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):248–258, 2019.

[38] Van Nhan Nguyen, Robert Jenssen, and Davide Roverso. Ls-net: Fast single-shot line-segment detector. *arXiv preprint arXiv:1912.09532*, 2019.