Automatic Chart Interpretation

Catarina Julião Relvas Pires

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Professor Vasco Miguel Gomes Nunes Manquinho
Professor Maria Inês Camarate de Campos Lynce de Faria

Examination Committee

Chairperson: Professor Alberto Manuel Rodrigues da Silva
Supervisor: Professor Vasco Miguel Gomes Nunes Manquinho
Member of the Committee: Professor Arlindo Manuel Limede de Oliveira

October 2021

Acknowledgments

These past few years have culminated on a beautiful path of ups and downs, academic life and lock-downs. They led me to meet new places, new friends and challenge myself with new experiences.

I would like to thank my supervisors Prof. Vasco and Prof. Inês for their insight and support on our weekly meetings.

I would like to thank my grandparents who taught me values and love: Carlos, Mimi, Armando and specially Silvia who is struggling and whom I miss so much. Thank you once again for everything.

The friends I made in Coimbra and Lisbon along the way. Specially Jessica and Gonçalo, you are for life!

To my parents who never once doubted me ”Of course I knew you could do it!”. They are always supporting my decisions, but also always ready to a warm and lovely welcome home. Love you. Couldn’t forget my pet Nami for the company during the long lockdown months.

To João who has always been there for me. You’ve been by my side since my first algebra class, which I missed due to the bus... and panicked. I love you.

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.

Resumo

Os gráficos são, atualmente, um formato de representação de dados indispensável, sendo utilizados em diversos tipos de documentos. Geralmente, os dados subjacentes às imagens dos gráficos são também considerados cruciais, no entanto, nem sempre estão disponíveis. Um método preciso de extração de dados de gráficos beneficiaria diversas áreas, tais como, melhorar os resultados de pesquisas web relativas a gráficos ou ajudar os utilizadores com incapacidades visuais a compreender gráficos em documentos. Uma outra aplicação é a síntese de programas que utiliza a representação gráfica dos dados. A ferramenta que motivou este trabalho de pesquisa é o UNCHARTIT, um sintetizador de programas que, dada uma tabela de entrada e uma imagem de um gráfico, possibilita a recuperação das transformações aplicadas aos dados.

Esta dissertação analisa as ferramentas já desenvolvidas com o objetivo de extrair dados de gráficos e, com base nos métodos mais promissores, propõe uma nova ferramenta, o BARXTRACTOR. A nossa ferramenta extrai dados numéricos e textuais a partir de imagens de gráficos de barras simples, agrupadas e empilhadas. O BARXTRACTOR não requer interação humana, utilizando redes neuronais convolucionais (CNNs) para classificação do tipo de gráfico e Faster R-CNNs na deteção de elementos existentes na imagem do gráfico, tais como barras e números. Para a extração textual é aplicada uma ferramenta de reconhecimento ótico de caracteres (OCR). Adicionalmente, o BARXTRACTOR foi integrado na ferramenta UNCHARTIT de modo a melhorar a sua precisão e eliminar a necessidade de interação com o utilizador.

Os resultados experimentais provaram que o BARXTRACTOR é capaz de classificar, com sucesso, gráficos de barras simples, agrupadas e empilhadas. Além disso, também foi possível verificar que o BARXTRACTOR supera as ferramentas do estado da arte que dependem da interação com o utilizador para a extração correta de dados da imagem do gráfico. Adicionalmente, a integração do BARXTRACTOR no UNCHARTIT permite melhorar a sua precisão na seleção do programa correto para as várias transformações da tabela original. Finalmente, o BARXTRACTOR também possibilita a extração dos dados textuais dos gráficos, que podem ser utilizados na melhoria da interpretação dos dados numéricos extraídos.

Palavras-chave: Imagens de gráficos de barras, Extração de dados, Deteção de objetos, Redes Neuronais Convolucionais, Reconhecimento ótico de caracteres.

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 dissertation 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.

Keywords: Bar chart images, Data extraction, Object detection, Convolutional Neural Networks, Optical character recognition.

Contents

List of Figures

List of Tables

Chapter 1

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 [2, 4, 7, 10, 24, 26, 33, 36, 43, 54], including different methodologies or subjects of study. For instance, some focus on a chart type [2, 7, 10, 36, 54], others on textual extraction [34, 51]. 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) [5], 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.

1.1 Motivation and Objectives

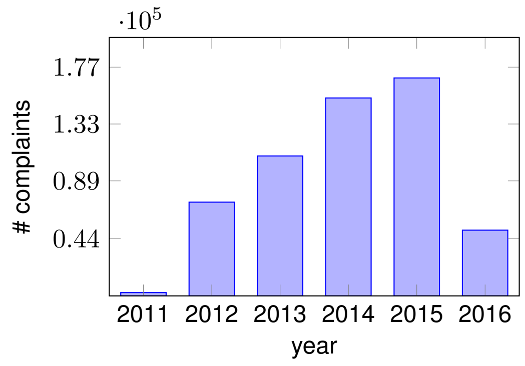
Data analysis processes have evolved over the last decade and now play a major role for companies in driving their informed decisions. This means that more data analysts have to be employed, some of them without the ideal programming skills. For this reason, some tools for automating programming tasks emerged [12, 13, 25, 27, 52, 53]. 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 [36] was proposed with the goal of recovering the program underlying a chart image. Consider the example in Figure 1.1, which contains (a) a sample of the table with all customer complaints from 2011 to 2016, and (b) the corresponding bar chart with the number of complaints grouped annually. Suppose that a data analyst with limited programming skills needs to reproduce

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

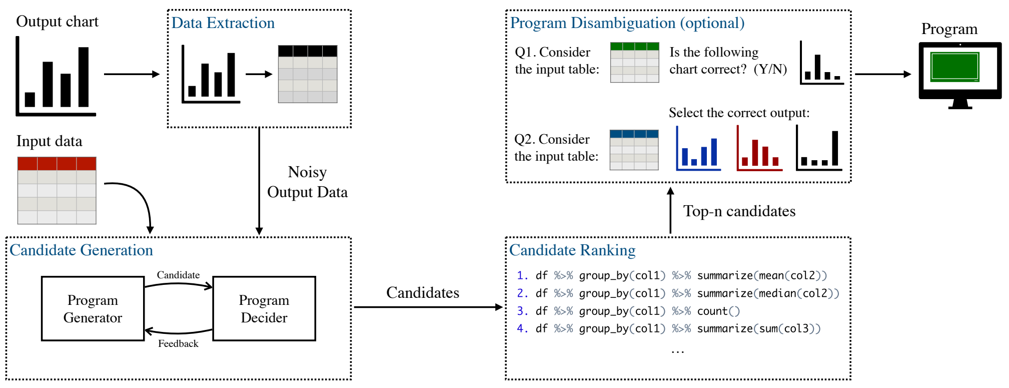
date_received	product	...
08/30/2013	Mortgage	...
08/30/2013	Mortgage	...
08/30/2013	Credit reporting	...
08/30/2013	Student loan	...
08/30/2013	Debt collection	...
08/30/2013	Credit card	...
08/30/2013	Credit card	...
08/30/2013	Debt collection	...

(a) Sample of the consumer complaints table (175.39MB).



(b) Bar chart with yearly number of consumer complaints.

Figure 1.1: Consumer complaints data from 2011 to 2016. [36]



```
1. df %>% group_by(col1) %>% summarize(mean(col2))
2. df %>% group_by(col1) %>% summarize(median(col2))
3. df %>% group_by(col1) %>% count()
4. df %>% group_by(col1) %>% summarize(sum(col3))
...
```

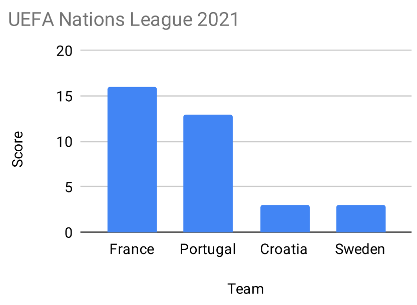
Figure 1.2: UNCHARTIT architecture.

an updated chart with the annual consumer complaints, but he does not have access to the program underlying the generation of the chart in Figure 1.1b. 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.

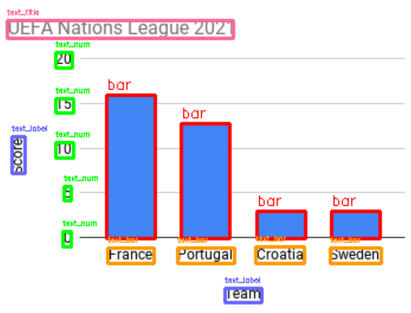
The program to be synthesized should include the chart generation, including the necessary data transformations, and it can be recovered given: (1) an input table, with the data used to create the chart, and (2) a desired output chart image, as it can be seen in Figure 1.2. Since the desired chart is 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 the Program decider. 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. This candidate disambiguation step is optional and it helps the user finding the final correct program.

UNCHARTIT proposes two ways of extracting data from bar charts. The first is using WEBPLOTDIGITIZER [1], a tool that requires user interaction and calibration. The other is a CNN 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



(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).

Figure 1.3: 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).

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.

1.2 Contributions

In this dissertation, after the related work revision, BARXTRACTOR is proposed as a tool for extracting values from three different types of bar charts: simple, grouped, and stacked bar charts. Its development was motivated by UNCHARTIT. The user simply needs to enter a chart image and the extraction process is automatic without any additional 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 1.3. 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.

1.3 Document Structure

This document is organized in six chapters. The introduction aims to present the problem, the motivation for the project and a brief overview on the proposed approach. Chapter 2 explains the concepts mentioned throughout the dissertation that are required to understand the following chapters. Chapter 3 provides an overview on the state-of-art of the tools and techniques to extract numerical and textual information from chart images. It includes chart identification, chart classification, and chart numerical and textual data extraction. The architecture and implementation of BARXTRACTOR are described in chapter 4, followed by chapter 5 which presents and discusses the experimental results. Finally, chapter 6 concludes the dissertation.

Chapter 2

Preliminaries

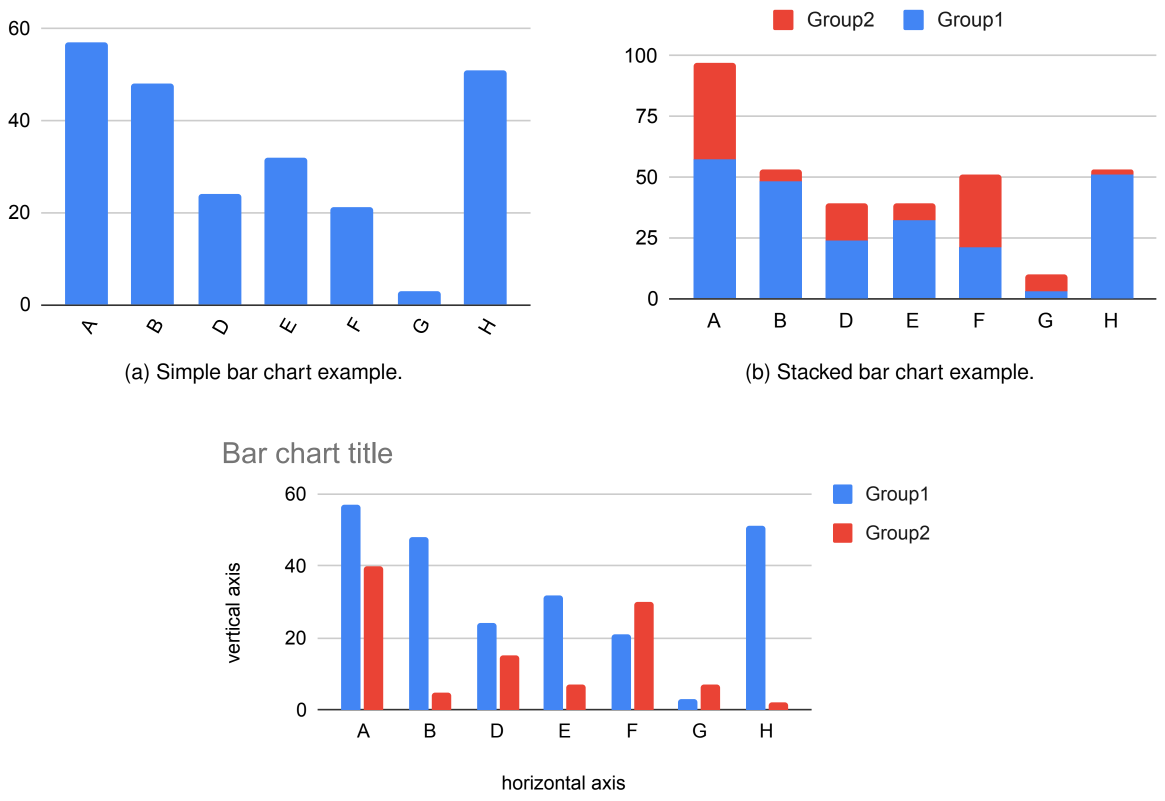
This chapter provides a brief review on the most relevant topics and methods that are mentioned throughout this dissertation and that are commonly employed in chart classification and data extraction. This includes the Connected Component Analysis (CCA), Convolutional Neural Networks (CNNs) and Region Based CNNs. It also provides an introduction to the chart types explored in the following sections.

2.1 Chart types

This section provides a brief overview on the chart types that are explored in the following chapters regarding the classification and data extraction of chart images. It covers bar, pie, line, area, scatter, radar and high-low charts. Note that only a subset of these charts is being addressed in the BARXTRACTOR tool. However, to comprehend Chapter 3, these chart types should also be presented.

Simple bar charts typically follow the layout presented in Figure 2.1a, with bars representing the numerical data for each category. These charts have two axis, the x-axis contains each bar's nominal label and the y-axis represents the numerical values. A variation of the simple bar chart is the **stacked bar chart** (Figure 2.1b) where each bar is divided into sub-bars. For instance, in the first bar ”A” it can be observed that the 'Group1' (blue) has a value of 57 and the 'Group2' (red) has a value of 40, hence the bar ”A” has a total value of 97. Another relatively similar type is the **grouped bar chart** (Figure 2.1c) in which each group has multiple bars in the same nominal label. These charts usually contain a title and a label for each axis. Its color legends may or not be present and can be placed in different chart locations. Note that both charts can also be horizontal and that the bars' labels can be non-horizontal as in Figure 2.1a.

Pie charts are used to represent percentages and are divided into slices of different colors or patterns. Note that these charts can be represented with distinct visual features (Figure 2.2). It can be a 2D or 3D chart, the slices can have borders (a) or not (b), each slice's percentage can be visible inside or



(a) Simple bar chart example.

(b) Stacked bar chart example.

(c) Grouped bar chart example.

Figure 2.1: Bar charts.

outside (b) the slice, or not shown in the chart (a). The slice's labels can also be represented in various ways.

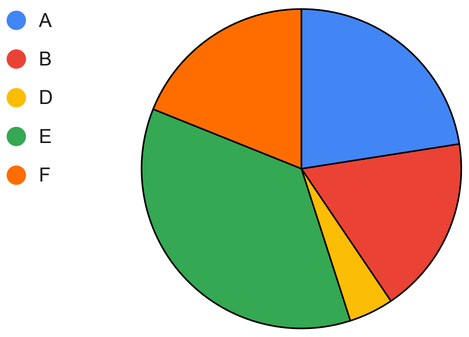
Line charts (Figure 2.3a) contain both a x- and y-axis and represent a series of data points linked in a continuous line.

Area Charts (Figure 2.3b) are similar to line charts but the area between the line and the bottom axis is filled by the line's color.

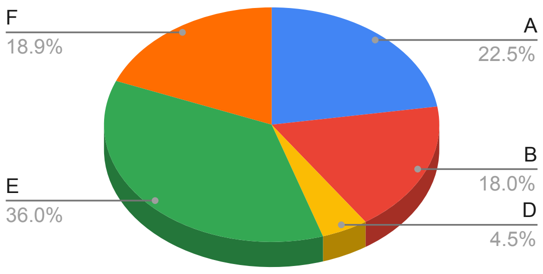
Scatter plots (Figure 2.4a) are composed by several points in which each point represents values for two distinct variables, one for each axis (x and y).

Radar charts (Figure 2.4b) represent multiple variables at the same time by having various equidistant axis with the same origin point. Each polygon represents a category and is distinguished by its unique color. Each polygon's vertex represent its value for that axis.

High-Low charts (Figure 2.4c) are composed by multiple vertical lines. Each line represents a range of values, where its topmost point represents the highest value observed, and its bottom point represents

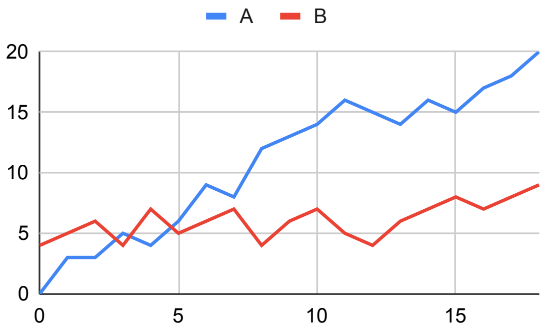


(a)

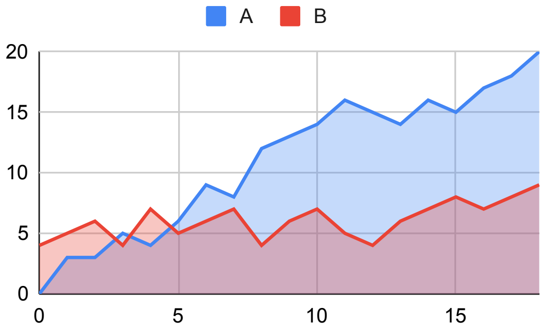


(b)

Figure 2.2: Pie Charts.



(a)



(b)

Figure 2.3: Line Chart (a) and Area Chart (b).

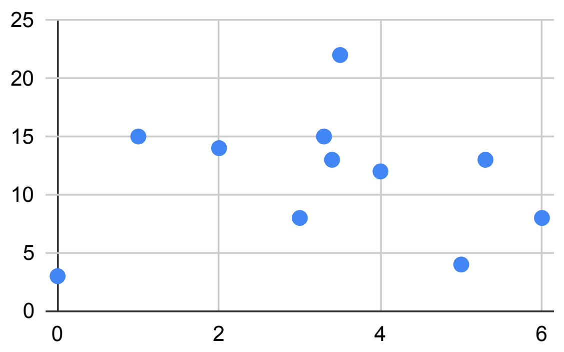
the lowest value observed. In high-low charts, the horizontal axis usually represents a measure of time. The horizontal tick in the line can represent any chosen value, such as the average value or, for stock market charts, a closing price for the day.

2.2 Connected Component Analysis

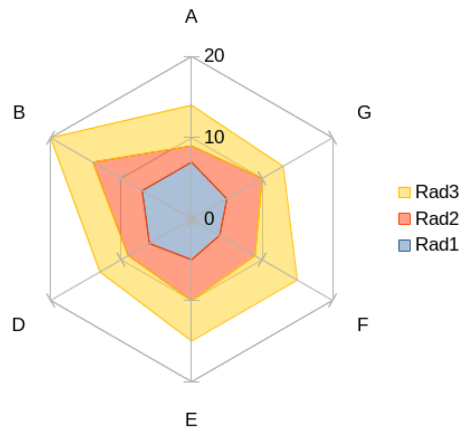
Connected Component Analysis (CCA) is an algorithm with the purpose of grouping neighboring pixels with the same color, in binary images, or of similar color, in colored images. The algorithm groups similar-color neighbor pixels, separating the image into groups, each group is a ”connected component”. The analysis can be either ”Four-connectedness”, if only the four adjacent pixels are considered neighbors, or ”Eight-connectedness”, if the pixels touching the corner are also considered neighboring pixels [21].

Figure 2.5 presents the application of CCA on an image with two bars. By analyzing the neighbor pixels in the image, the algorithm generates three connected components (CC), namely: the background, since all the white pixels are connected, and the two bars represent two distinct CC, since even though they have the same color, their pixels are not neighbors.

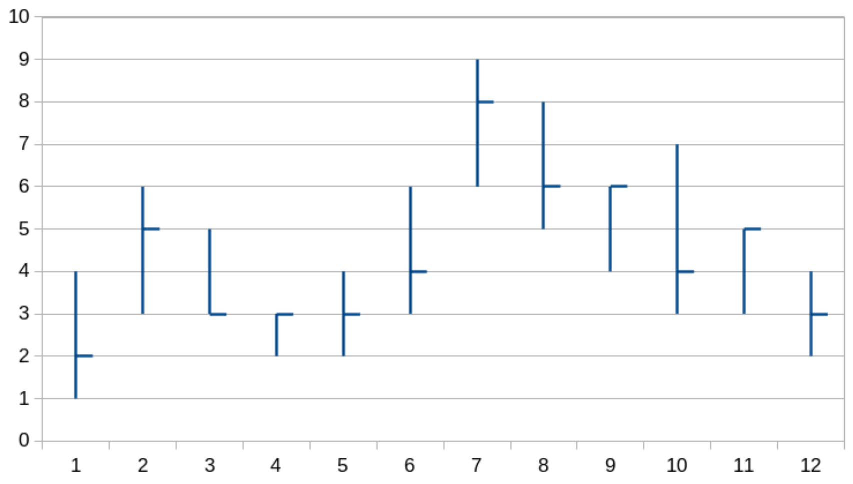
The CCA technique was previously applied to detect both graphical and textual elements on chart images [2, 10, 24, 43]. Detecting the elements in a chart is useful for its classification or data extraction.



(a) Scatter Plot



(b) Radar Chart



(c) High-Low chart

Figure 2.4: Scatter Plot (a), Radar chart (b) and High-Low chart (c).

2.3 Convolutional Neural Networks

In this section we assume the reader is familiar with Neural Networks [5, 15].

Convolutional Neural Networks (CNNs) are a type of neural network inspired by the visual cortex of the human brain [37]. These networks achieve good results in image recognition and classification problems [5], which are both common applications.

Thus, CNNs are frequently used in chart classification [4, 26, 32, 33] and, although it has been little explored, it can be a good basis for chart images data extraction [33, 54].

In order to further understand CNNs, the typical architecture of these networks is represented in Figure 2.6. CNNs are typically composed by several layers, namely: input layer, convolutional layers, pooling layers, fully connected layers and output layer.

2.3.1 Convolutional Layers

The convolutional layers have the goal of extracting image features by performing an operation named ”convolution”. It consists on applying filters (kernels) to the input, in order to extract feature maps. This

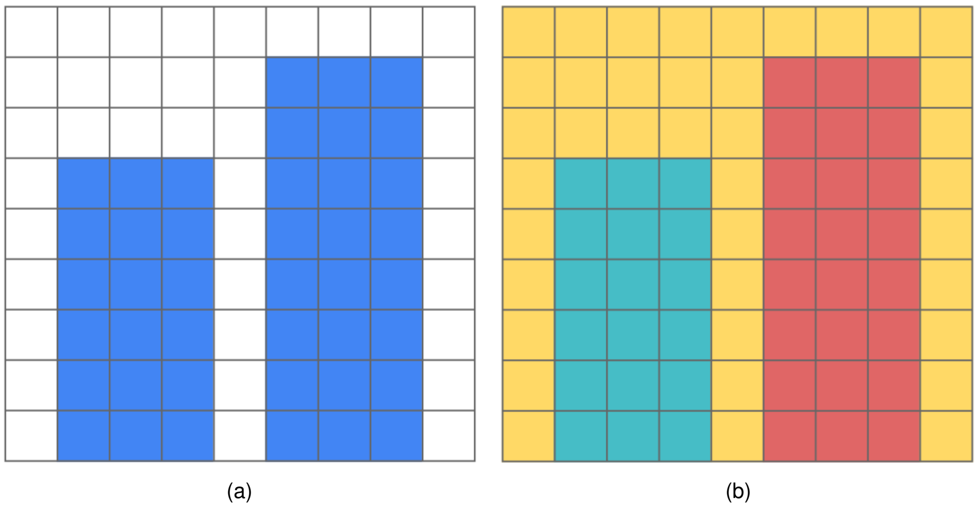


Figure 2.5: Connected Component Analysis Example. (a) Original image. (b) Correspondent CCA.

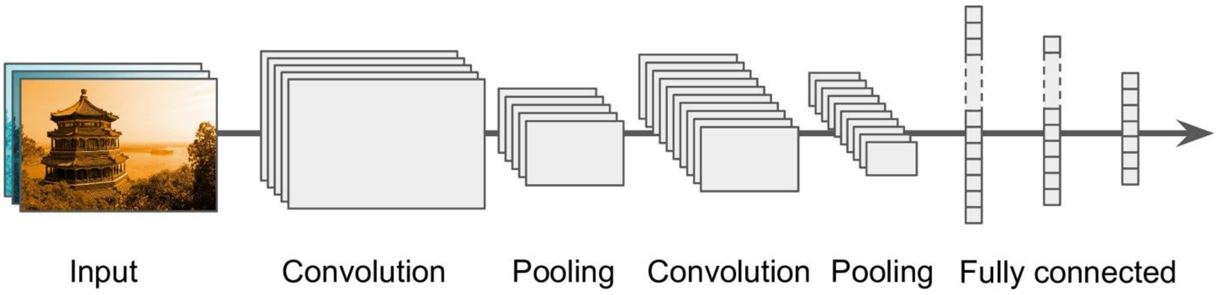


Figure 2.6: Typical CNN architecture. Image extracted from [15].

input might be the inserted image, for the first convolutional layer, but it can be the output of other layers.

The filter goes through the input data starting on the top left corner. In the first convolutional layer, the filters applied to the input image will extract low-level features such as lines. In the following convolutional layers this process continues to be performed and more complex features can be extracted. Note that, during the training phase, these filter's weights are updated by the backpropagation algorithm [40].

Figure 2.7 illustrates the effect of applying a filter to the input image, for instance, the vertical filter extracts vertical lines.

Two important concepts to understand the convolutional step are: stride and padding. **Stride** is the number of pixels that the filter shifts when going through the input. **Padding** is a technique that consists in filling the input's edge pixels in order to make it wider. If this padding is applied with zeros it is called Zero Padding. Padding is used to overcome some issues, such as: the image size shrinking over the layers, or to avoid losing information from the image corners, since the kernel only takes them into account once, contrary to the middle pixels that are accounted for multiple times.

To illustrate stride and padding, Figure 2.8 illustrates a convolution operation with an 3×3 input, a 2×2 filter kernel, a stride of 1 and zero padding. The padding is represented in the input by the cells with dashed borders. In order to simplify the understanding of the convolutional operation two blocks are highlighted in the input and their respective output is highlighted in the output matrix. For instance, the highlighted red cell in the output matrix (10) consists in the dot product of the red block in the input by

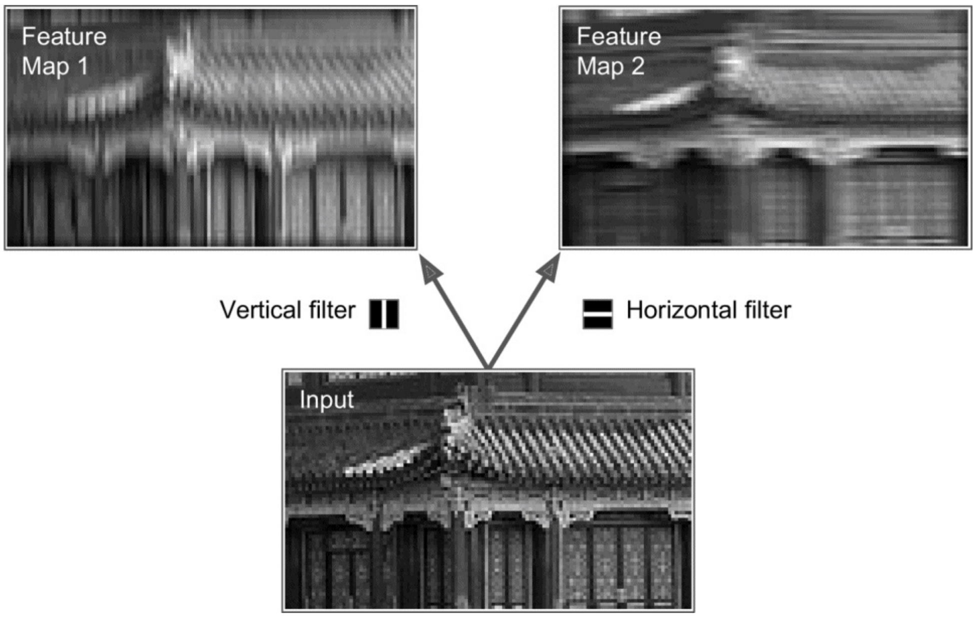


Figure 2.7: Application of two different filters. Image extracted from [15].

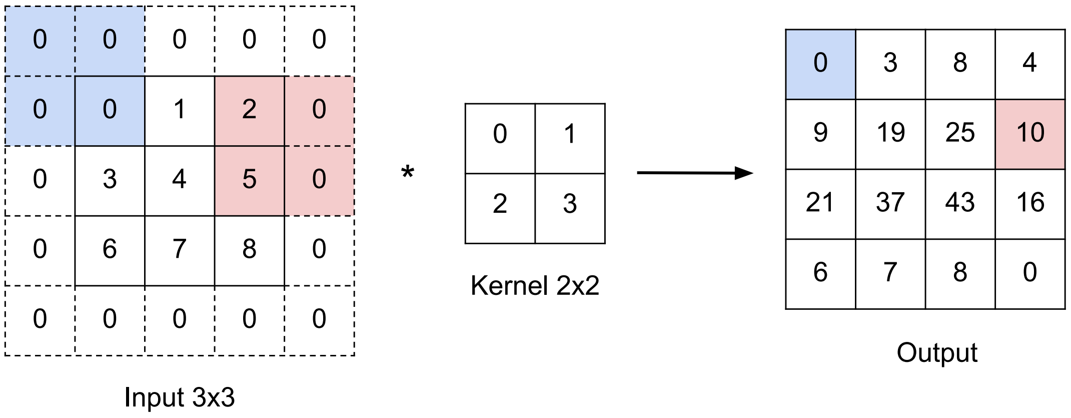


Figure 2.8: Example of Convolution operation.

the kernel ($2 \times 0 + 0 \times 1 + 5 \times 2 + 0 \times 3$), this operation consists in the sum of products. The convolution is performed for every block obtaining sequentially each entry in the output matrix.

2.3.2 Pooling Layers

The pooling layers are subsampling layers, with the aim of reducing the spatial size of the input. These layers operate on each feature map. The most common pooling approach is max pooling [15]. In Figure 2.9 we can observe an example of max pooling, where only the maximum value in each kernel is stored.

2.3.3 Fully Connected Layers

The fully connected layers on a CNN receive the image features extracted by the convolutional and pooling layers and outputs the class label, performing the classification. The output of the previous layers is reshaped into a flat vector. The output represents the probabilities of the image corresponding

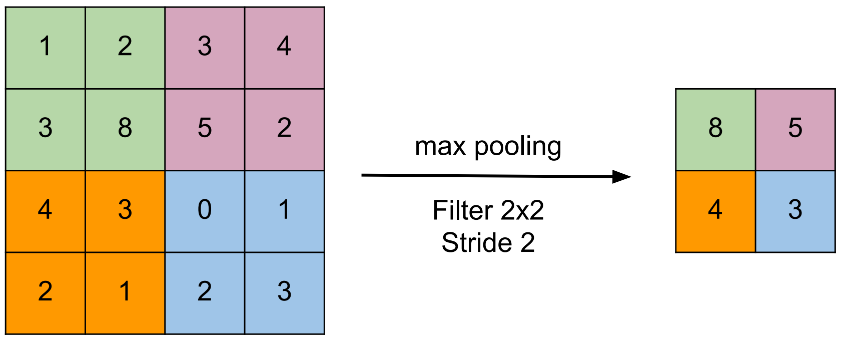


Figure 2.9: Example of Max pooling layer.

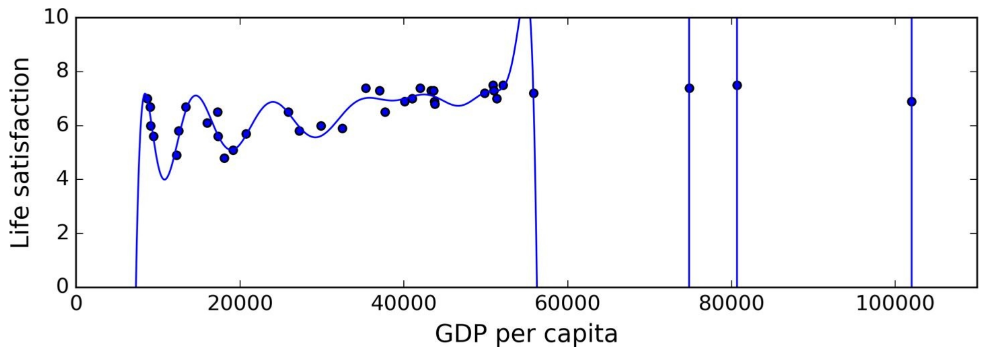


Figure 2.10: Example of overfitting of the training data. Image extracted from [15].

to a certain class. In order to aim for a better classification accuracy, the fully connected layers weights are also updated during the training phase, through backpropagation.

2.3.4 Regularization

Overfitting is a common problem in machine learning, it occurs when the model perfectly fits to the training data but the results do not generalize [15]. Figure 2.10 shows an example of an overfitted model. As we can observe, the model is attempting to fit to all the data points rather than having a generalized form. In this situation a polynomial function with less degrees would be a better fit. This phenomenon can happen when the model trains for too long or when it is a very complex model. Overfitting causes high performance on training data and low on testing data, not generalizing to new instances.

One way of overcoming overfitting is to collect more training data, but that may not be possible or may not solve the issue. A different technique to avoid overfitting is the use of regularization that introduces a way of penalizing very complex models. Regularization can be considered any modification applied to a model with the goal of reducing its generalization error (how accurately it predicts unseen data) [18].

Some common regularization methods will be described next. The Lasso regression (L1) and the Ridge Regression (L2) are some common regularization techniques [15]. These add a penalty term to the cost function that keep the model's weights simple. To clarify, the cost function, also defined as the loss function, represents the distance between the current output and the desired output [5, 15]. The optimizer is responsible for minimizing this loss function by updating the weights.

Another common type of regularization is the dropout technique [15, 18], which essentially consists in having a probability of temporarily ”dropping out” some neurons and making them temporarily inactive. This works by making other neurons perform the predictions to compensate the missing ones, leading to a more generalized model.

Early stopping is another regularization technique that consists in monitoring the training and validation errors and stopping the training process when the validation error starts increasing, or the accuracy starts decreasing. Early stopping is usually implemented by saving the model whenever the best validation errors are observed, if it does not improve after a defined number of iterations, then we return to the saved model.

Data augmentation is another technique that simply consists on generating more training data based on the existing data, for instance, by resizing and rotating the images. This leads to a bigger training dataset, which results in a more robust model.

2.3.5 ResNet

The residual neural network (ResNet) [19] 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) [41] classification task. This subsection’s goal is to share ResNet’s main characteristics.

With the evolution of convolutional neural networks, deeper models were being created by adding more layers to try and learn more complex features. However, there is a point where adding more layers causes the network’s performance to decrease. One of the reasons for the network’s decreased performance is the vanishing gradient. When the network has too many layers, the gradient might shrink tremendously while being propagated, influencing the learning process, by not allowing all the network’s weights to be updated [19].

Residual Block

In order to solve the vanishing gradient problem and allow the training of deeper networks, ResNet [19] proposes the introduction of residual blocks. Figure 2.11 presents a residual block which contains a skip connection between the input x and the output $F(x)$ of the block. These connections allow to skip layers, thus the name. With this connection, the output of the layers is now $H(x) = F(x) + x$. By introducing these connections the gradient does not tend towards zero when backpropagating. As a result, the networks can be trained with a large number of layers without increasing the training error.

Architectures

ResNet has numerous variants mostly obtained by changing the number of layers. The first architecture presented was ResNet-34 which is 34 layers deep and each residual block is two layers deep. Other architectures include ResNet-50, ResNet-101 and ResNet-152 which have three layers deep blocks. The ResNets with 50, 101 and 152 layers are more accurate than ResNet-34 [19] on ImageNet. In

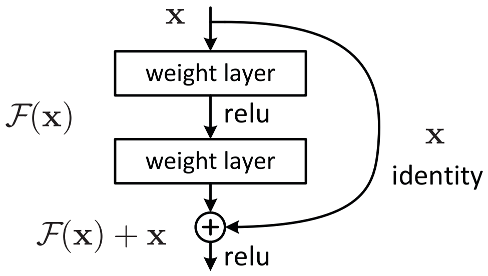

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.11: Residual Block. Image extracted from [19].

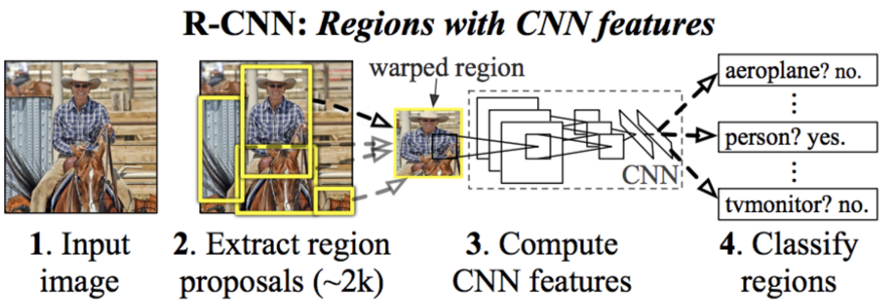


Figure 2.12: R-CNN. Image extracted from [17].

BARXTRACTOR the ResNet-50 was chosen for classification tasks since it is unnecessary to use 101 or 152 layers, which would be slower to train.

2.3.6 Region Based CNNs

To address object detection, given an input image, the goal is to output the various detected objects' bounding box and their 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 are used in this project for object detection tasks. There are other object detection systems, however Faster R-CNN was chosen since it achieves a better accuracy for small elements [23] compared to other systems such as YOLO [38], SSD [31] and R-FCN [9].

R-CNN

The first method proposed for object detection was R-CNN [17]. Figure 2.12 presents the architecture of the system. Given an image as input, the system runs a selective search algorithm and extracts roughly 2000 region proposals. These regions, are warped into a square and fed to a CNN that extracts the features for each region. The features are then fed to a Support Vector Machine (SVM) [8] that classifies each region. The algorithm also outputs four values (xmin, ymin, xmax, ymax) corresponding to the adjusted bounding box of the detected object.

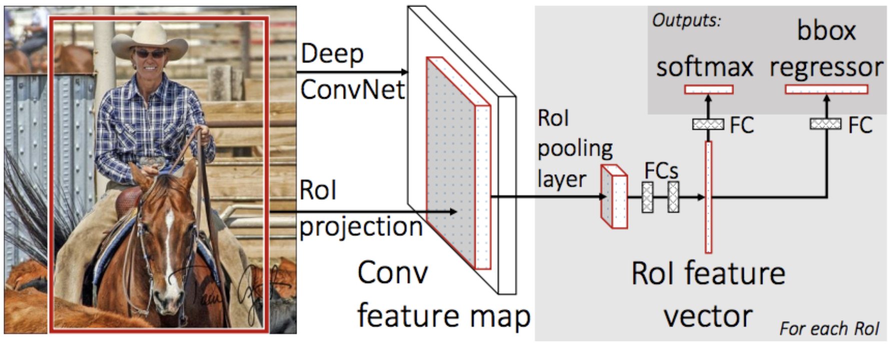


Figure 2.13: Fast R-CNN. Image extracted from [16].

Fast R-CNN

Despite the good results presented by the R-CNN there were still issues mostly related to the training and testing times. Ross Girshick proposed the Fast R-CNN [16] which achieves a better accuracy in less time. Figure 2.13 presents the Fast R-CNN architecture.

In this proposal, the entire input image and a set of regions of interest (RoIs), obtained through selective search, are fed to the CNN in order to generate a convolutional feature map. The feature map is reshaped by using a RoI pooling layer which is fed to a fully connected network to generate a RoI feature vector. The network outputs, for each RoI, the softmax probabilities for the object classification, and the bounding box offsets.

The Fast R-CNN solves some of the R-CNN drawbacks including that the system can now be trained end-to-end, since each step is not an independent component of the system as in R-CNN. Another advantage is that instead of performing the convolution 2000 times, once for each region proposal like in R-CNN, it is done once per image.

Faster R-CNN

The third network to arise was Faster R-CNN [39] whose architecture is presented in Figure 2.14, its purpose was to improve the Fast R-CNN performance. Using selective search on the image to find the region proposals was still slowing the Fast R-CNN. The Faster R-CNN, instead employs a Region Proposal Network (RPN) which receives as input the images' convolutional feature maps and returns region proposals. Now, since it is using a network, it can be trained and customized to each detection problem.

Important concepts related to object detection

There are some important concepts related to object detection that should be clarified since they are commonly used as metrics in these types of problems.

The first concept is IoU (Intersection over union), illustrated on Figure 2.15, which consists on dividing the amount of overlap by the area of union between the predicted bounding box and the ground truth

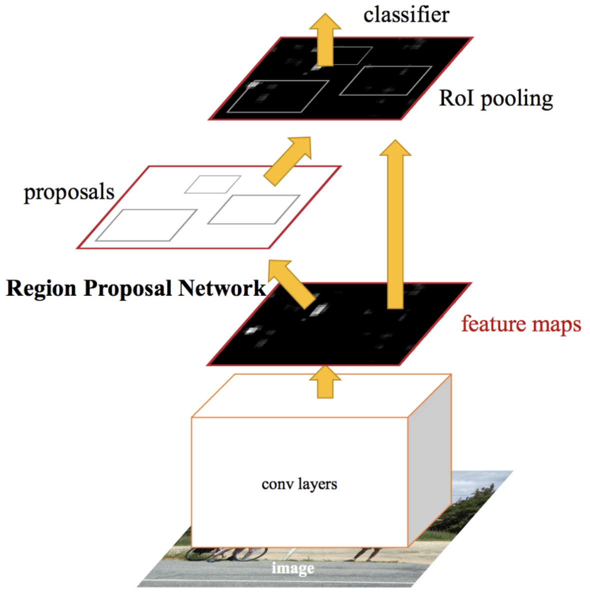


Figure 2.14: Faster R-CNN. Image extracted from [39].

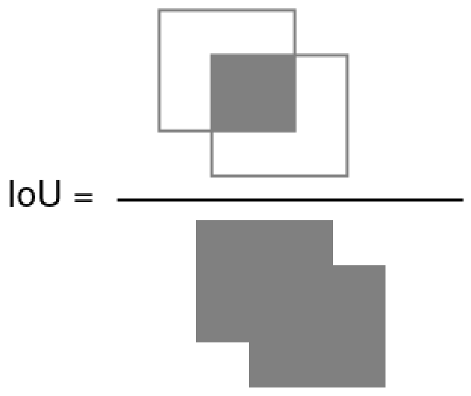


Figure 2.15: IoU (Intersection over union) equals to the area of overlap divided by the area of union.

bounding box. The IoU values vary from 0 to 1 and the higher the better, since it means the predicted area is almost overlapped with the ground truth area.

Another concept is the mAP (mean average precision) where we use the precison and the recall values to calculate the average precision of each class. Precision is the ratio of relevant instances among the detected instances and recall is related to the number of detected relevant instances among all the relevant instances. The average precision (AP) is the result of the area below the precision-recall curve which represents the trade-off between the precision and the recall. High results on AP mean more accurate results. The mAP is the mean of the calculated AP for each class.

Chapter 3

Related Work

Automatic interpretation of chart images has been previously explored throughout the years, and different approaches have been used to solve each of the following challenges. The first topic is related to the identification of a chart image, either identify its location within a document or, given an image, categorize it as a chart or not. The second challenge concerns chart type classification (e.g. line chart, bar chart, pie chart), as well as chart dimension classification, since it might be necessary to differentiate between 2D and 3D charts before the extraction. Chart numerical data extraction is usually the main goal of these projects and several technologies can be applied in order to obtain the chart values. The extraction of text and labels from chart images is another topic, it can be useful to improve the accuracy of the classification and the extraction of the numerical chart data, or it can be extracted just to provide more information to the user. All these topics are detailed in the next sections.

3.1 Chart images' identification

Before proceeding to chart data extraction, there may be other processes to take into account, depending on the project. For instance, it may emerge the need to categorize a given input image as a chart or a non-chart [34]. Another example is when the input is a document image, as a scanned paper, and the chart images need to be detected within the document [10, 28]. Paramita De [10] proposed the use of a Region-based Convolutional Neural Network (R-CNN) model in order to identify pie charts and provide its bounding region in the input document image.

In BarXtractor, neither the classification of an image as a chart, nor the detection of charts within a document is relevant since the chart image will be given as input by the user.

3.2 Chart type classification

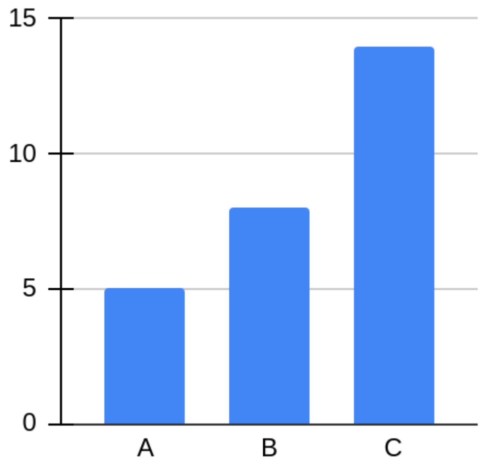
Techniques to perform chart type classification have been the subject of several research works, each using distinct means to address this problem. Nevertheless, machine learning is one of the most successful approaches, using algorithms such as Convolutional Neural Networks (CNNs).

For instance, the CHARTSENSE tool [26] builds their chart type classification model based on CNNs, more specifically the GoogLeNet model [49] was trained to perform the classification. GoogLeNet is a CNN variation that won the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [41], showing great results in image classification. Xiaoyi Liu et al. [33] also base their chart type classification on CNNs, more specifically on VGG-16 [46], which was the 1st runner-up of the ILSVRC 2014 in the classification task. The VGG-16 model was applied to identify the chart type, modifying only the output layer into two output categories: ”bar chart” and ”pie chart”. Following the same approach, the CHART-TEXT system [4] uses the MobileNet [22], which obtains a similar accuracy to VGG-16 using fewer parameters on ImageNet. The images are resized to a fixed size and normalized before being fed to the network. The network was previously trained with the ImageNet dataset and the weights were used as a starting point to this classification problem. This technique is named transfer learning.

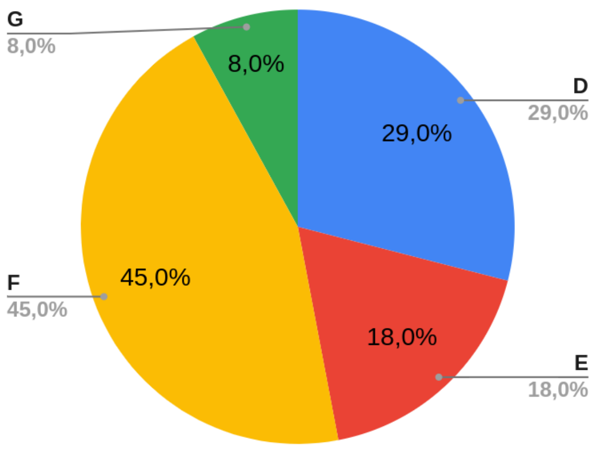
A different application of CNNs in chart classification was proposed by Xiao Liu et al. [32]. It consists of a framework that combines CNNs and deep belief networks [15]. The chart image is given as input to the CNNs and the deep hidden features of the chart are extracted from its fully-connected layers. Then, based on these features, the deep belief network is employed to predict the category of the chart, between 5 categories: pie chart, scatter chart, line chart, bar chart, and flow chart.

Other machine learning algorithms besides CNNs can be used to perform the classification. For instance, the REVISION tool [43] uses both image and text feature vectors to perform classification using Support Vector Machines (SVMs) [8], which is a supervised learning model. The image feature vectors are obtained by extracting random patches from the image and applying K-means to them. This way, the most common information on these patches can be found by obtaining each cluster’s centroid. This information represents chart’s graphical marks, such as lines and arcs. The combination of the textual feature vector, that contains information such as position and size, with the image feature vector proved to increase the classification accuracy with SVMs. Karthikeyani et al. [28] also proposed a method that uses three different classifiers for chart type recognition, namely Support Vector Machines (SVMs), a multilayer perceptron (MLP) neural network and the K-Nearest Neighbor (KNNs) algorithm. The method with the most promising results was the KNN classifier.

There are also more traditional solutions to tackle the classification problem, without machine learning algorithms. Huang et al. [24] proposed a model-based approach for chart classification, more specifically for bar, pie, line and high-low 2D charts. First, feature extraction in performed using Connected Component Analysis (CCA) to detect the features. The detected elements are categorized into textual or graphical elements simply by analyzing the properties of each connected component. Secondly, for the graphical elements, edge detection and vectorization is performed. These steps are made by analyzing each image pixels and generates vectors with the detected elements, such as lines and arcs. Next, the algorithm analyses relationships between these vectors, such as, parallelism and perpendicularity. Finally, for the classification part, the authors defined a set of constraints that match each chart type model. These constraints are based on the features and relationships analyzed in the previous steps. Using these set of constraints, the input chart is matched to the model whose constraints are satisfied by its properties. A similar approach using a model-based classification method was proposed



(a) Bar Chart



(b) Pie Chart

Figure 3.1: Examples of a bar chart (a) and a pie chart (b).

by Mishchenko and Vassilieva [34]. The features are extracted and matched with one of the models that satisfies the constraints.

The classification problems previously presented cover the chart type categorization. However, a different classification problem may arise when working with both 2D and 3D charts. The need to discern between these two chart types is mandatory before proceeding to the data extraction, since the extraction methods cannot be applied identically on both. Paramita De [10] proposed a method to find this characteristic in pie charts. The program preprocesses the image in order to get a binary image representing only the pie space in black, and removing the background and text, this process is further explained in the next section. With this obtained image, the centroid of the black pixels is computed. Then, the shortest Euclidean distance between the centroid and all the components in the image is calculated. If there are differences between the computed normalized shortest Euclidean distances, it can be inferred that the chart is 3D, since these charts have a group of pixels outside the circle to give the idea of perspective. Otherwise, if there are no big differences, the chart is a perfect circle and is labeled as 2D. Although this solution works for pie charts, it is not scalable for other types of charts.

3.3 Data extraction

This section provides an overview on the current state of the art methods to extract numerical data from charts. As seen before, it is common to expect machine learning applications for chart type classification. Despite not being the most usual approach for chart data extraction among the analyzed research works, some recent studies explore machine learning methods to tackle this problem.

In 2003, Huang et al. [24] explored the extraction of numerical data from charts by using the detected objects in the classification step to recover the data from bar, pie and high-low charts. For bar charts, the height in pixels of each detected bar is calculated and the bars are sorted by size. However, the real numerical values of each bar are not computed since the tool does not support numerical labels' extraction. For pie charts, the pie slices are used, namely the angle of each slice is obtained and converted to a percentage by knowing that the total angles' sum is equivalent to 360º. The high-low charts' data is obtained by computing the difference between the highest and lowest points of a vertical "bar" and its distance to the x-axis. However, these are not the real values but relative values. For line charts, a sample rate is chosen and used to go through the image and recover some data points of the line, again, not recovering the real values.

However, in 2011 the REVISION tool [43] was already able to extract real values from charts, focusing on extracting data from 2D bar and pie charts. Since REVISION uses an image dataset from the web, the first step is to apply a filter that reduces the noise of the images. The next step is to perform mark extraction. Regarding bar charts this includes bars and axes. The bars are identified by applying Connected Component Analysis (CCA), grouping adjacent pixels of similar color and detecting the connected components that form a rectangular shape. Some background rectangles, formed by grid-lines, may also be identified but their removal is performed considering the fact that the bars and the background have distinct colors. For example, in Figure 3.1a the bars are represented in blue and the background is represented in white. Next, the chart orientation is determined by identifying the side that differs between bars. For instance, in vertical bar charts the height may be different, while the width remains the same for every bar. Then, the baseline axis is detected, the x-axis usually touches the bottom of each bar, for positive bar charts, but it can be touching the top, for charts representing negative values. The text labels were previously extracted in the classification step and will now be useful for the numerical data extraction. The y-axis labels are identified by finding the ones that are vertically equidistant. For instance, in Figure 3.1a the vertically equidistant labels would be "0","5","10" and "15". These can be used to find the scaling factor, or pixel_per_data ratio, by obtaining the number of pixels between two pair of labels. For instance, if between label "10" and "15" there are 50 pixels, this means each data unit corresponds to 50/(15-10)=10 pixels. Having this information and the minimum value in the y-axis, the numerical values of each bar can be recovered by analyzing its height in pixels and multiplying it by the pixel_per_data ratio obtained.

For pie charts, the REVISION tool extraction is divided in two steps. The first consists in fitting an ellipse in the pixels that represent the pie edge. For this, the pixels where the color changes sharply are identified. The second step consists in identifying each sector's edges. The pie is "unrolled" into a rectangle by uniformly sampling points from a circle with the same center as the pie and smaller radius. In the obtained rectangle, the points where the color changes are identified, since they represent the transitions between pie slices. As before, to find the percentage of each pie slice, the value of the angle between each sector edge is used. In the REVISION tool, the extracted textual labels are assigned to its elements in a simple manner. For bar charts, the label below the x-axis that is closer to the bar is the one assigned to it. For pie charts, it is assigned the closest label to the slice's arc. Note that this approach is prone to errors, for instance in pie charts where the label is represented at a certain distance, or for bar charts with rotated labels.

A similar method to the REVISION tool extraction of 2D bar charts was later proposed by Al-Zaidy and Giles [2]. The input image is pre-processed and the textual and graphical components are extracted, more specifically the bars and axes. As previously, the algorithm used to recover the bars is based on the Connected Component Analysis (CCA) and the pixel_per_data ratio is obtained using the extracted labels. With this information the chart data is inferred.

Another tool for chart data extraction is CHARTSENSE [26]. It is considered a semi-automatic system because it requires multiple user interactions to perform the extraction. For line and area charts, CHARTSENSE asks the user to specify some features such as the x-axis intervals of data. Since the tool does not support text detection or extraction, it also asks the user to specify some y-labels' positions and their respective numerical values. The detection of lines is a set of processes based on the dominant colors of the chart, where this automatic detection can be corrected by the users. For radar charts, the extraction process also relies on various user interactions such as the identification of the center point, the axes and the colors of the polygons. Bar chart numerical extraction uses a similar approach to REVISION [43], applying CCA to identify the bars, but asking the user to specify some y-labels' positions and its correspondent numerical values. Pie charts numerical extraction is also similar to REVISION, where 1000 pixels are uniformly sampled along a circle with the pie center and with a smaller radius. For every chart, in each step of the extraction, the user has the possibility to adjust the automatic detected parts. For example, the user is able to adjust the pie chart's center if needed.

Later, with the goal of recovering data from charts within a document image, Paramita De [10] proposed a different algorithm to extract the numerical data of pie charts. For 2D charts this extraction is performed in 4 steps. Considering that in this work the chart images are obtained from scanned documents, the first step consists in removing image noise, and after, converting the image to gray scale. The second step, "Gradient Analysis", is based on the assumption that different chart slices are represented by different colors (Figure 3.1b). These local gradients are analyzed in order to isolate only the image parts where the color changes between adjacent pixels and ignoring the rest. The result is then binarized, forming an image containing, in black, the chart slices' boundaries, the textual information and possibly some noise. In order to get an image containing only the boundaries of the pie chart, the Connected Component Analysis (CCA) algorithm is performed and only the largest connected component is kept, which corresponds to the chart boundary, removing this way the text. For instance, in Figure 3.1b the percentages inside or outside the pie would be removed. The third step inverts the image's colors and removes the background color, obtaining an image with each slice filled with black pixels and not touching the others. Lastly, the chart data extraction step is very simple. The total number of black pixels in the image is obtained and CCA is performed. At this point, each chart slice is a connected component (CC), so the number of pixels in each CC is computed. The percentage of each slice is computed by applying the following formula:

$$slicePercentage = (pixelsInsideSlice * 100)/totalPixelsInImage \tag{3.1}$$

where *pixelsInsideSlice* represents the number of black pixels in the slice, and *totalPixelsInImage* the total number of black pixels in the image. For 3D pie charts, the data extraction starts with a similar preprocessing step. Then the 3D structures are removed and a perspective correction is applied to the

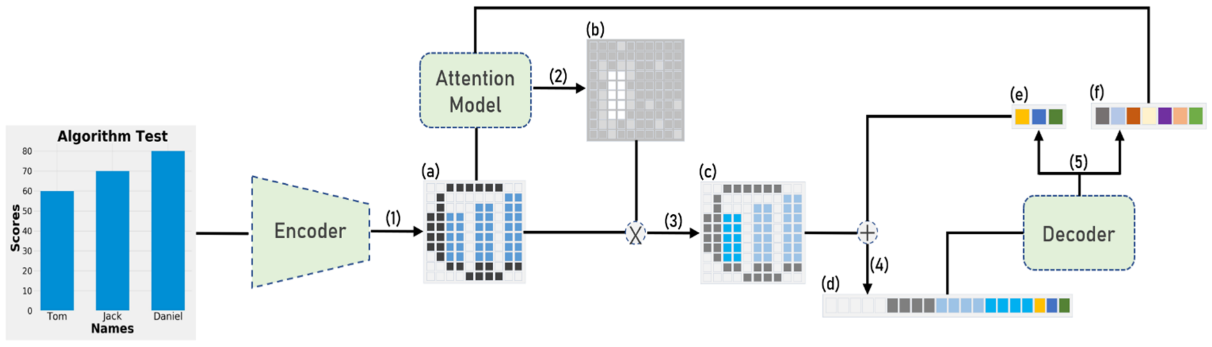


Figure 3.2: Proposed pipeline for extracting numerical data. Image extracted from [54]. Step (1) produces the feature vector. Steps (2) to (5) iteratively generate the numeric information, that is, bar vector.

image, in order to obtain a 2D chart. Now, the same data extraction methods can be applied as the ones described before for a 2D pie chart.

CHART-TEXT [4] is another tool created with the purpose of generating a textual description for a chart image. The object detection model chosen was Faster R-CNN [39]. Firstly, the detected text is extracted using optical character recognition (OCR). Then, for bar charts, CCA is used to identify and extract each bar. This algorithm also supports stacked bar charts, by identifying sudden changes between pixel colors in a bar. Using the detected textual labels, a pixel_per_data ratio is obtained and the numerical values are computed. For pie charts, a similar method to CHARTSENSE [26] is used, sampling a number of pixels uniformly along the circle. The last step of this system is to generate a description of the chart, given the extracted textual and numerical data.

SCATTERACT [7] is a tool that aims to extract data specifically from scatter plots, a chart type whose extraction is less explored. It starts with object detection, using the REINSPECT tool [48], in order to localize points, axis marks and values. Then, the axis values' textual elements are extracted and associated with its correspondent tick mark. After this, regression is used to find the mapping between pixel and chart coordinates. The mapping is performed by using the ticks' values and detecting possible outliers caused in the text extraction. Having the previously extracted points and the data per pixels ratio, it is now possible to generate the table with the scatter plot's numerical values.

A different approach for bar chart data extraction was presented by Fangfang Zhou et al. [54]. Neural networks are used to extract the numerical data, more specifically, with the encoder-decoder framework presented in Figure 3.2. The encoder is implemented using a CNN, more specifically the Xception CNN [6] with two additional layers. It receives the chart images as input and extracts their key features, creating a feature vector (Figure 3.2 (a)). Then, an attention mechanism [3] is included, consisting of a two layer neural network. It generates an attention vector (Figure 3.2 (b)) for the current iteration by taking as input the feature vectors (a) and the hidden state of the decoder from the previous iteration (f). This attention vector gives information to the decoder on what portion of the image it should analyze next. Next, a context vector (d) is generated by combining the attention and the feature vector (c) with the bar vector, produced in the previous iteration (e). The decoder interprets the context vector (d) in order to iteratively generate the correspondent numerical data, namely the bar vector, that contains the center coordinates and normalized height of each bar. This is implemented through a recurrent neural network (RNN), more specifically a long short-term memory (LSTM) network is used, which is a type of RNN capable of memorizing long-term dependencies. The LSTM network outputs the hidden state vector (f) and is followed by a fully connected layer that transforms its output into the bar vector (e) for the current iteration. Since the obtained heights of the bars by the encoder-decoder framework are normalized, traditional algorithms are applied to obtain the actual heights of the bars. The pixel_per_data ratio is obtained using the extracted y-labels, similar to previous research work [43]. The outliers in the extracted y-labels are filtered since the labels' extraction is prone to errors, such as, a label "90" might be recognized as "9". Although this tool extracts both the numerical and the textual information, including x-axis labels and y-axis labels, it does not match the x-axis label to its corresponding bar.

Also using a machine learning approach, Xiaoyi Liu et al. [33] developed a deep learning model to address bar and pie chart data extraction. There are two models, one for each chart type, and in order to decide which data extraction model should be applied, the chart is classified first. The extraction starts with object detection, which uses a model based on Faster R-CNN [39]. After the feature map generated by the Faster R-CNN is obtained, it is applied a branch that aims to find relationships between the objects detected (an object matching branch), based on relation networks (RN) [42] with some modifications. This branch finds relationships between elements and matches, for instance, bars or pie slices to their respective legend. For bar charts, the values are inferred by employing linear interpolation using pixel locations. Since the pie charts' slices are more difficult to detect using a rectangular bounding box, it is predicted the whole pie chart location. Then, based on the feature maps, a two-layer LSTM is used to obtain the angles of each slice's boundary, recurrently outputting each sector's angle. Each slice's feature map is obtained by rotating the whole pie by the predicted angle of its boundary. As a result, the rotated feature map has each slice in a specific region. Using the slices' and legends' feature maps, the object matching is performed as previously, by obtaining the relationships between legends and slices. The data can be inferred by converting the extracted angles into percentages.

Recently, the UNCHARTIT tool [36], introduced in Chapter 1, 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 [1] 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 [50] with an alternative output layer. As the first required information is the number of bars in the chart, the first n nodes of the output layer contain the probability that the chart has $i \in \{1, 2, ..., n\}$ bars. The other necessary information is the height of each bar, thus more n nodes were added to the output layer, each node representing a relative height from 0 to 1, defining how full the respective 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.

3.4 Chart text extraction

This section aims to present the current state of art in chart images text recognition and extraction. These textual elements, depending on the chart type, may be composed of the title, the x-axis and y-axis title, the x-axis labels, which are usually composed by characters, and the y-axis labels, usually constituted by numerical characters. For pie charts, it also includes each sector's label. This task requires rigorous text detection, image preprocessing and text extraction steps, since it is not easy to achieve a good accuracy and not all the previously mentioned research works extract text and labels. Chart textual elements have some common characteristics that might hinder the extraction process. For instance, the text is usually presented in a small font size and may be rotated to a non-horizontal position.

The REVISION tool [43] implemented a tagging interface that allows the user to manually specify and annotate the textual regions' location. After identifying these regions, optical character recognition (OCR) is performed, more specifically the TESSERACT engine [47] is employed, which is the Google's open source OCR engine. After the text recognition, the user can then correct the results, if needed.

In the same year, Vassilieva and Fomina [51] proposed a fully automatic text detection algorithm specific for chart images and their corresponding problems. The algorithm consists in a set of preprocessing steps that are employed before the OCR. It starts by performing CCA and separating the results with character characteristics. Then, the Hough Transform [11] is applied in order to detect lines and their orientation, among the obtained characters. Now, the text regions can be found based on the detected lines, since usually all characters in a label have the same direction, and based on the distance between detected characters. Each detected textual region in the chart image is cropped and rotated to a horizontal position before being fed to the OCR engine. In this case the TESSERACT engine [47] was used to test the algorithm's effectiveness. Mishchenko and Vassilieva [34] applied this preprocessing algorithm in a chart data extraction framework combined with the TESSERACT engine.

Al-Zaidy and Giles [2] extract the textual data by first identifying the text regions in the image and then performing OCR to each of them. Using the image previously binarized, the components whose area occupy more space than a typical character are removed. In order to identify words, each character is enlarged and CCA is performed which results in each connected component representing a word. Then, the TESSERACT OCR engine [47] is employed to each word in its normal format, before the character dilatation. Based on the text region characteristics, a class is inferred for each extracted text, for instance, if its position is on the left of the y-axis, then the text is classified as an y-axis label or value.

The previously presented SCATTERACT [7] and CHART-TEXT [4] tools, first identify the bounding boxes for each element. Secondly, they make use of the TESSERACT OCR engine [47] to extract the text from the detected images. Some preprocessing steps are applied to the image before employing OCR since it was shown to improve the accuracy results. This includes, for SCATTERACT, converting the image to gray scale, and for CHART-TEXT, converting the image to a binary color representation. Both tools rescale the images to a fixed height and rotate the image in order to be in a horizontal position before the OCR application.

As explained in Section 3.3, Xiaoyi Liu et al. [33] employ Faster R-CNN [39] for object detection, namely proposing a class and a bounding box for each detected object. Since charts commonly contain non-horizontal text and the Faster R-CNN only generates horizontal bounding boxes, an additional layer for orientation detection is added before the text recognition phase, so the non-horizontal text can be rotated to a horizontal position. Furthermore, a text recognition model is employed, more specifically, Convolutional Recurrent Neural Network (CRNN) [44] which combines Deep Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

Fangfang Zhou et al. [54] propose the use of Faster R-CNN [39], an object detection network, to extract the charts' textual information. This extraction is made under the assumption that the chart only contains horizontally and vertically oriented text elements. First, a chart image is given as input to the Faster R-CNN, the network concurrently locates and classifies the textual elements. These can be classified as title, legend, axis title or axis label (differentiating x-axis and y-axis). Now, having the text elements' bounding boxes, an OCR engine can be individually applied to each of these images, in order to get the correspondent text. These images are binarized and scaled before the process, the OCR recognition is applied three times in the same image using different rotation angles. The axis labels and legends are sorted in accordance to their bounding box's coordinates in order to facilitate the combination of numerical and textual information when obtaining the bars' heights using the y-axis labels.

Chapter 4

Automatic Bar Chart Extraction

The software developed for this dissertation was named BarXtractor and its goal is to extract the visual information represented on a bar chart image into a data table with the values of each bar.

The detailed description of BarXtractor's architecture and methods used will be further explained throughout this chapter specifying in each part of the system, which is mostly based on a classifier, an object detection network and a few other procedures that help it get to the desired output.

4.1 Architecture

In order to further understand the whole concept of the system we should take a look at Figure 4.1. The input of BarXtractor is a bar chart image. That image is fed to the trained image 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 in order to tackle common issues, for instance, detecting a bar label without detecting a corresponding bar often means the bar has a zero or almost zero value. These techniques added after the visual elements' detections attempt to understand and correct if there are missing bars and other typical issues.

The labels that correspond to a textual or numerical value are cropped into a new image containing only the textual/numerical information found. The resulting image is then 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 and are utilized to create the chart csv output file which may be represented in a different way for the three chart types. After all the real bar values and textual labels are obtained, the output of the system is a csv file with all the extracted

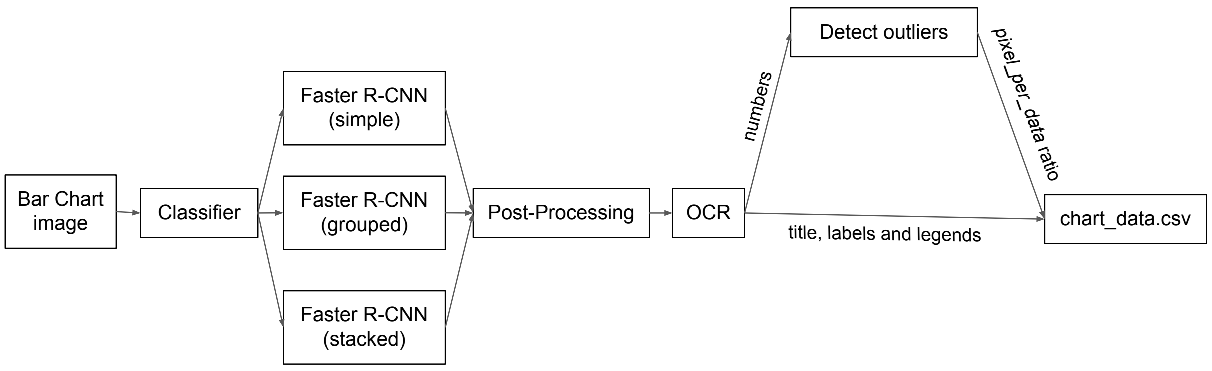


Figure 4.1: BARXTRACTOR's architecture.

information.

Each of these steps will be detailed in the next sections. The first and second sections focus on the classifier and the three Faster R-CNN models respectively. The models, training and respective datasets used are explained for both. The final sections of this chapter discuss the post-processing techniques applied and the methods enforced to obtain the real chart values.

4.2 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.

4.2.1 Model and training

To implement an image classifier, a model pre-trained on the ImageNet dataset was chosen, more specifically, the ResNet-50 convolutional neural network. 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 [29] and the loss function was the negative log likelihood loss[1], 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, it can simply be loaded to resume the training or to perform a classification. If given a chart image it will return a prediction for one of the three classes.

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

4.2.2 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 section), 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 by using search queries such as ”bar charts” in the web search engines.

All the collected images were separated into three folders, one for each class: ’simple’, ’grouped’ and ’stacked’. Then, a random 10 percent of each class was removed from the training dataset to a separate folder 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.

4.3 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. This section will address BARXTRACTOR's Faster R-CNN models, datasets and training.

4.3.1 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 element detection step could have been performed using only one general model although it would not be so precise for each chart type.

To train each model a custom pytorch dataset class had to be defined in order to return the images, their ground truth labels and their bounding boxes.

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[2]. 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

[2]https://cocodataset.org/

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 all the parameters are defined, the model can be trained for a pre-defined number of epochs. After each test, the training provides the user some important information to analyze, such as the average precision and average recall for different IoU values. This helps comprehending 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. A sample of the output values follows.

```
IoU metric:  bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.823
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.997
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.956
```

The AP values for an IoU between 0.5 and 0.95 must be maximized in order to achieve high accuracy in the chart elements' detection.

After the training process the model is saved, and all that is required to utilize it or resume the training is to load it. Given an image as input, the Faster R-CNN outputs a bounding box, a label and a score related to the confidence in the prediction.

4.3.2 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[3] from Fangfang Zhou et al's work [54]. 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 simple bar chart images.

Since for this object detection problem it is crucial that the bounding boxes are tight to the exact space of the elements, the `matplotlib` method `get_window_extent()` is used to get the exact bounding box of the elements within the image.

Each dataset entry is composed by an image and several bounding boxes with its corresponding labels. Figure 4.2 shows an example of the bounding boxes required for the information contained in a grouped bar chart image. Table 4.1 shows a sample of the `annotations.csv` file that contains the annotations of the elements contained in Figure 4.2.

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

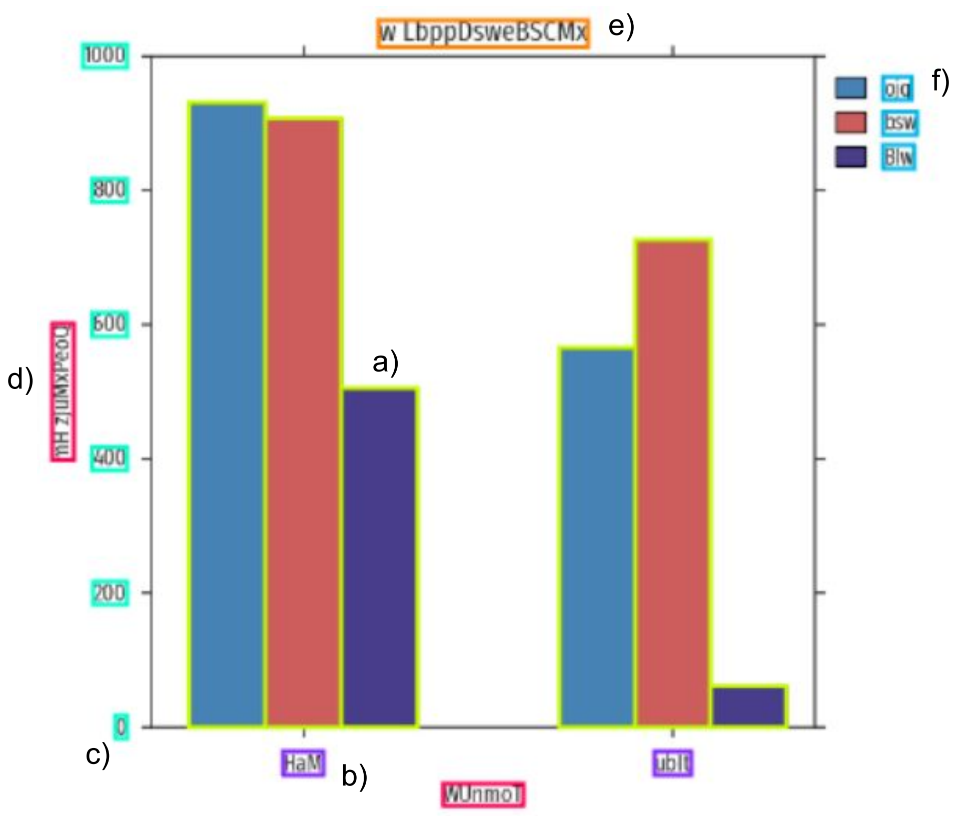


Figure 4.2: Labels contained in an image from the grouped bar charts dataset. Corresponding to the `annotations.csv` on Table 4.1.

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

- bar (the bar rectangle) (Figure 4.2 a))
- text_bar (the labels below each bar) (Figure 4.2 b))
- text_num (the vertical numerical values) (Figure 4.2 c))
- text_label (the axis label x and y) (Figure 4.2 d))
- text_title (the title of the chart) (Figure 4.2 e))
- text_legend (the textual legend of each color (Figure 4.2 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 color present.

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.

filename	width	height	class	xmin	ymin	xmax	ymax
...	...	...	...	...	...	...	...
test_660.jpg	728	637	bar	120	78	179	562
test_660.jpg	728	637	bar	179	90	238	562
test_660.jpg	728	637	bar	238	299	297	562
test_660.jpg	728	637	bar	407	268	466	562
test_660.jpg	728	637	bar	466	184	524	562
test_660.jpg	728	637	bar	524	530	583	562
test_660.jpg	728	637	text_bar	193	581	224	599
test_660.jpg	728	637	text_bar	481	581	509	599
test_660.jpg	728	637	text_num	63	553	72	571
test_660.jpg	728	637	text_num	46	449	72	467
test_660.jpg	728	637	text_num	45	345	72	363
test_660.jpg	728	637	text_num	45	241	72	259
test_660.jpg	728	637	text_num	45	137	72	155
test_660.jpg	728	637	text_num	38	33	72	51
test_660.jpg	728	637	text_label	317	606	379	623
test_660.jpg	728	637	text_label	14	249	31	355
test_660.jpg	728	637	text_title	267	14	429	35
test_660.jpg	728	637	text_legend	658	58	680	76
test_660.jpg	728	637	text_legend	658	85	684	103
test_660.jpg	728	637	text_legend	658	111	682	129
...	...	...	...	...	...	...	...

Table 4.1: Example of the `annotations.csv` file.

4.3.3 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 color legend 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.

4.3.4 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. For instance, Figure 4.3 highlights in yellow the 'bar' labels, one for each group color. The reason for not recognizing the entire bar is because then we would have to check manually where the colors change within each bar.

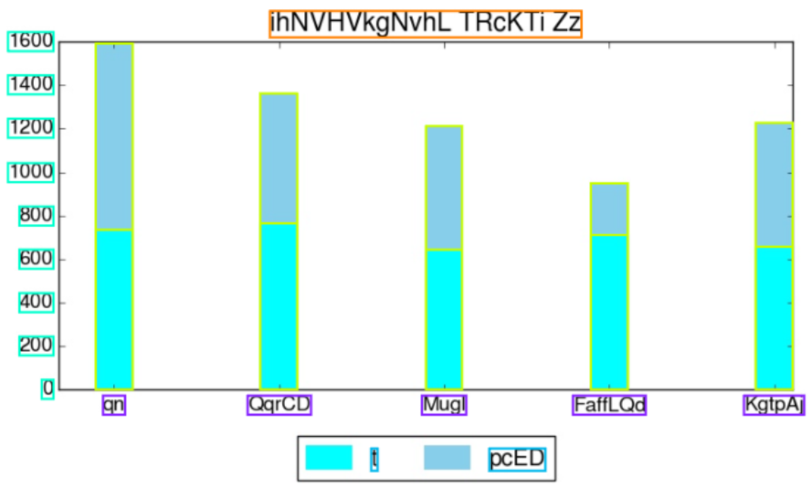


Figure 4.3: Labeled stacked bar chart example.

4.4 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.

4.4.1 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.

4.4.2 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. It is considered an inconsistency 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.

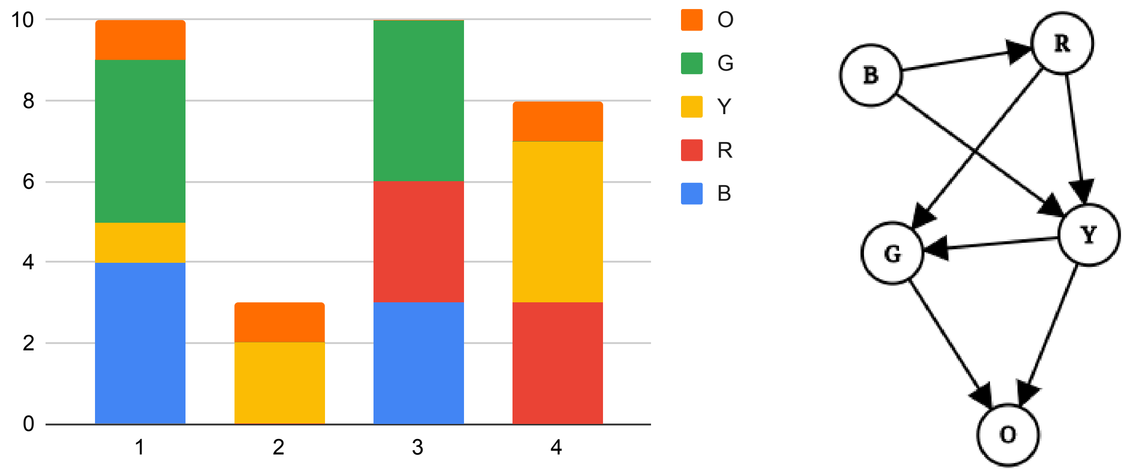
4.4.3 Stacked bar charts

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 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 4.4 (b) shows the resulting directed color graph corresponding to the stacked bar chart in Figure 4.4 (a).

When the graph is completed, we apply a topological sort algorithm in order to find the color disposal. For Figure 4.4 it would be [B,R,Y,G,O].

In order to perform the calculations that check if the color is the same or not, the RGB value of each color is used. However, there may be slight differences between the same color in different positions of the image, so a method to calculate the distance between two colors was developed. The method examines if, for each value (R,G,B), the values differs less than 10% between the two colors. If they do not, it is considered a different color. Then, the method returns true or false according to whether the colors can be considered the same or not.



(a) Stacked Bar Chart.

(b) Colors' directed graph.

Figure 4.4: Example of a stacked bar chart (a) and the corresponding colors' directed acyclic graph (b).

4.5 Obtain the real chart values

The first BARXTRACTOR's approach 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. Applying OCR to the vertical axis numbers is a solution for this problem. OCR could also be applied in other chart elements to provide extra information to the user, such as the bar labels and titles. 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 text labels.

4.5.1 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.

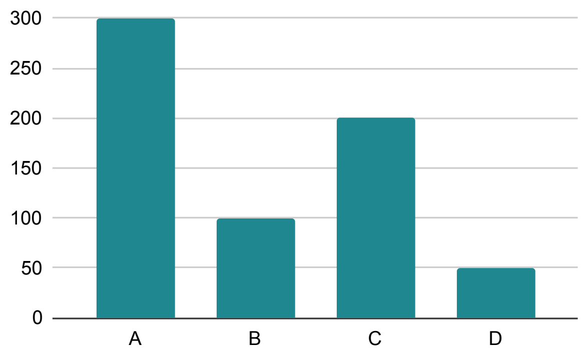


Figure 4.5: Bar chart used as an example to explain the vertical axis numbers extraction.

The extracted numerical values are the most important information that will lead us to the real chart values. For instance, in Figure 4.5 the numerical labels are: ”0”, ”50”, ”100”, ”150”, ”200”, ”250” and ”300”. Sometimes a ”0” is recognized as an ”O” and a ”0,” as a ”Q”. Hence, an extra step is needed after the OCR 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.

4.5.2 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 labels extracted from Figure 4.5 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 [14] is applied to detect these outliers. RANSAC returns a list of the inliers and outliers based on the labels and coordinates of each problem, as shown in Figure 4.6. 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 [7] 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 following 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.

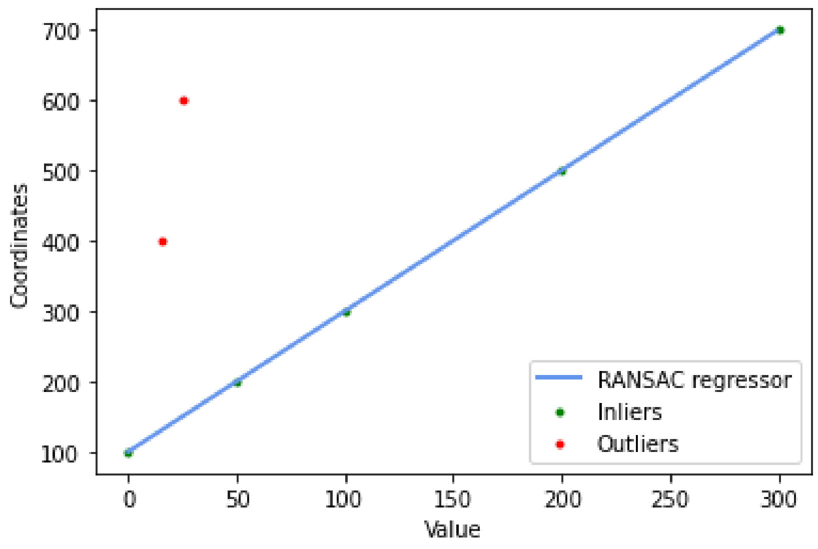
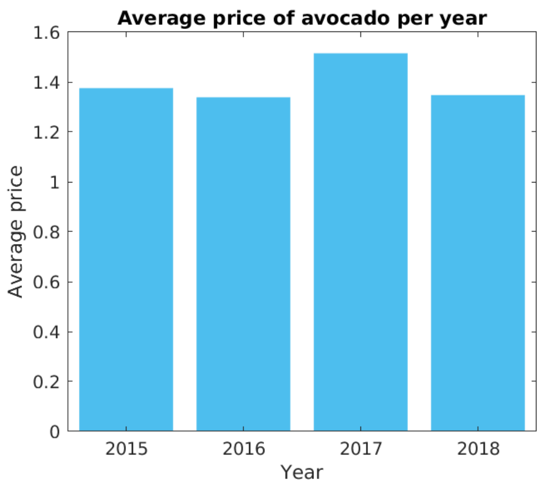


Figure 4.6: Outlier detection with Random sample consensus (RANSAC).



(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.

Figure 4.7: Example of a simple bar chart and BARXTRACTOR's csv output.

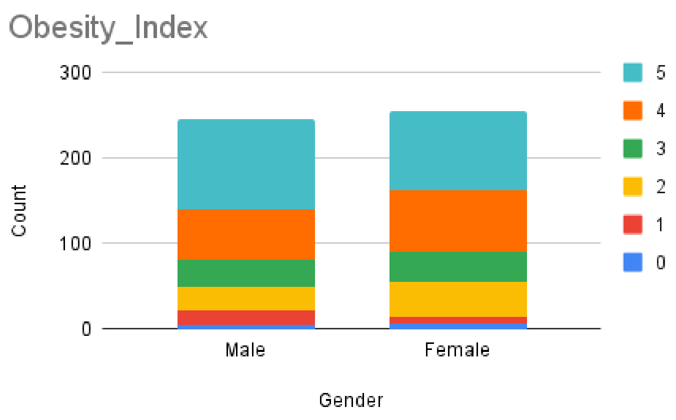
$$pixel\_data\_ratio = \frac{\sum_{i-1} \frac{abs(X[i+1]-X[i])}{abs(y[i+1]-y[i])}}{len(X)-1} \tag{4.1}$$

In Equation 4.1, 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.

4.6 Output of the system

BARXTRACTOR's last step is to aggregate the extracted information into a csv file. The file is used to



(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

Figure 4.8: Example of a stacked bar chart and BARXTRACTOR's csv output.

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 4.7 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 4.8 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.

Chapter 5

Results and Discussion

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 [36] 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 Chapter 4, 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 bar color and an acceptable distance from that color. Two more values need to be inserted: the width of the bars in pixels (Δx) and the height of the highest bar in pixels (Δval).

5.1 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 5.1) indicates the error between the real (x) and the observed values (y), being n the number of values.

$$mae = (\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i| \tag{5.1}$$

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

Table 5.1: Number of charts where the number of bars was correctly predicted among the 50 simple bar chart images.

The error calculation was implemented using a method that receives the real and the predicted data, which is contained in the output csv of BARXTRACTOR. 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[1]. 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.

5.1.1 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.

Number of bars detection

Table 5.1 presents the results obtained on the 50 images related to the number of charts where the correct 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, which are visually very close to each other, unlike the charts generated by the other tools.

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

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

	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 5.2: MAE of the bar values on the simple bar chart images with correctly predicted number of bars, excluding charts with scientific notation.

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 the Faster R-CNN performs better on detecting the correct number of bars between the data generated on the three tools.

Mean absolute error

In terms of mean absolute error four situations were considered as we can observe on Table 5.2. 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, so a test without OCR was performed. 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

	top-1	top-3	top-5	top-10
WebPlotDigitizer	90.2%	92.7%	92.7%	**97.6%**
EfficientNet-B7	65.9%	70.7%	73.2%	80.5%
Faster R-CNN (with OCR)	78.0%	82.9%	85.4%	87.8%

Table 5.3: UNCHARTIT results on the latex bar charts.

	top-1	top-3	top-5	top-10
WebPlotDigitizer	77.5%	85.0%	87.5%	90.0%
EfficientNet-B7	36.0%	36.0%	40.0%	48.0%
Faster R-CNN (with OCR)	79.5%	86.4%	88.6%	**90.9%**

Table 5.4: UNCHARTIT results on the matlab bar charts.

highest and lowest label, which might not be the correct ones. As expected it has the lowest error on matlab images since the training dataset was developed on `matplotlib.pyplot` module. 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.

UnchartIt results

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, Table 5.3, Table 5.4 and Table 5.5 show the success rate on finding the correct query on the top-1, top-3, top-5 and top-10 candidates for the three tools.

For latex images, Table 5.3 shows that the WEBPLOTDIGITIZER tool has a better performance, which is expected since it has the lowest MAE for latex images. For matlab images, Table 5.4 shows that, the Faster R-CNN performs best on UNCHARTIT followed by WEBPLOTDIGITIZER as expected by the MAE results. The excel images results on UNCHARTIT are on Table 5.5 and 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.

	top-1	top-3	top-5	top-10
WebPlotDigitizer	75.0%	81.8%	81.8%	88.8%
EfficientNet-B7	24.0%	40.0%	40.0%	44.0%
Faster R-CNN (with OCR)	80.4%	84.8%	89.1%	**91.3%**

Table 5.5: UNCHARTIT results on the excel bar charts.

5.1.2 Grouped and Stacked bar charts

To evaluate the grouped and stacked bar charts 40 instances were extracted from Kaggle and grouped in specific ways. For instance, there is one table with the obesity index (1-5) in a school and we grouped it by genders as in Figure 4.8. With these 40 instances two visually different plots were generated per instance, varying several 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.

Grouped bar charts

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.

Stacked bar charts

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.

	latex	matlab	excel
Faster R-CNN	0.586957	0.402062	0.62500

Table 5.6: Bar labels medium number of differences

	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 5.7: Bar labels success rate (number of totally successful reads)

5.2 Textual extraction

In order to evaluate the precision of the the textual labels extraction, more precisely on the bar labels, the same 50 instances generated 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.

Medium number of differences

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.

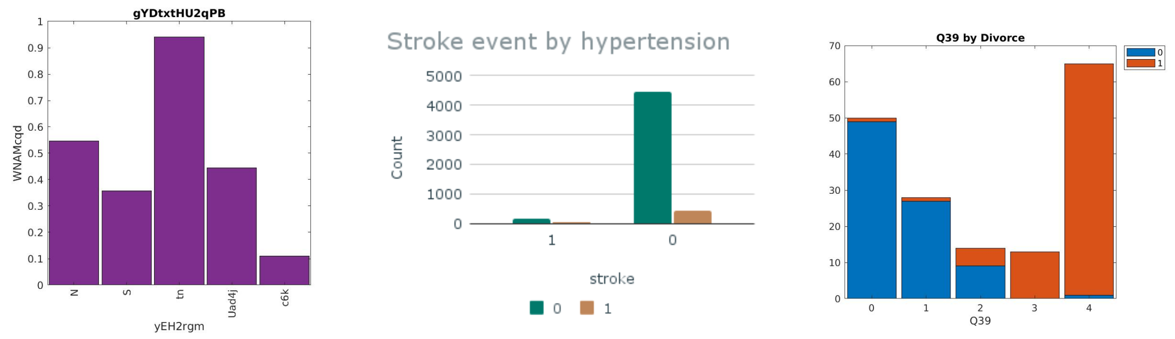
A difference is considered when there is the need to add a character (1) and to remove a character (1). So when there is the situation of replacing a character, that counts as 2 differences.

The results on Table 5.6 show that each label need a medium number of 0.59 character changes for latex labels, 0.4 character changes for matlab labels and 0.63 for excel labels. 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.

Success rate

The success rate in this context is considered the number of totally successful reads, meaning, the labels which characters were 100% correctly recognized. Table 5.7 shows those values and 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



(a) Simple bar chart mislabed as 'grouped'. (b) Grouped bar chart mislabed as 'simple'. (c) Stacked bar chart mislabed as 'grouped'.

Figure 5.1: Mislabeled charts on the classification process.

with the extracted bar label and be considered a match if the number of differences is lower than two operations.

5.3 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 some of the images 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 were correctly labeled, 92.0% of the grouped bar charts were correctly labeled and 92.4% were correctly labeled as stacked.

In order to further understand the classifier results, the images that are incorrectly labeled by the classifier were analyzed. 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. Figure 5.1 (a) shows an example of a mislabed simple bar chart classified as 'grouped' because the bars are too close to each other. 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 as in Figure 5.1 (c). Grouped bar charts might also be classified as 'simple' when one of the bars is small, as in Figure 5.1 (b). 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.

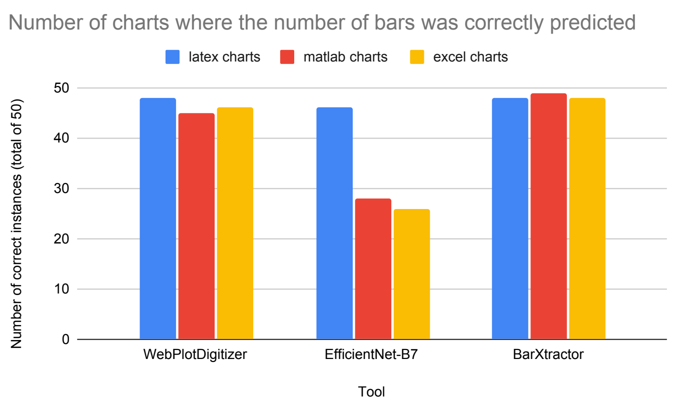


Figure 5.2: Number of charts where the number of bars was correctly predicted.

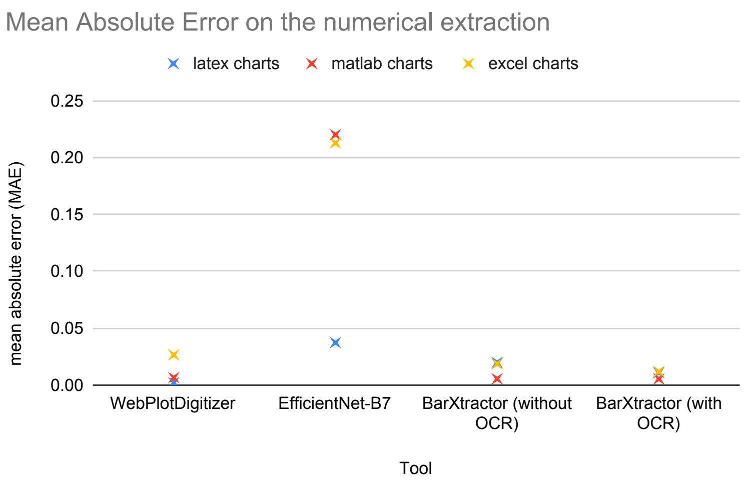
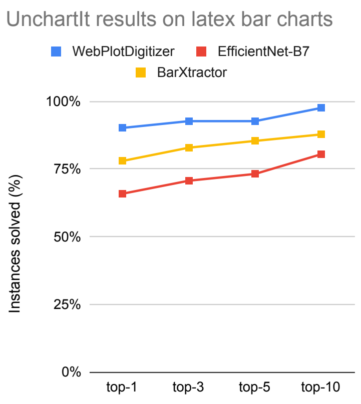


Figure 5.3: MAE between the simple bar charts where the number of bars was correctly predicted.

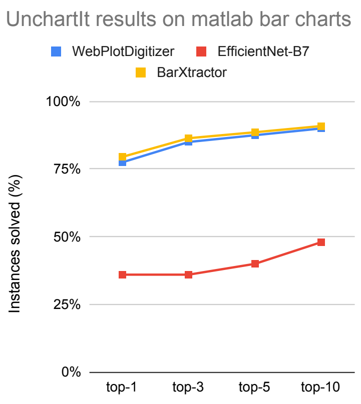
5.4 Results' summary

This section contains a concise summary of the results obtained. First, 50 simple bar charts were generated on three different tools (latex, matlab and excel) using instances of real data. Figure 5.2 presents BARXTRACTOR's results on detecting the correct number of bars in a chart and compares it with the EfficientNet-B7's and with the WEBPLOTDIGITIZER's results. BARXTRACTOR outperforms the other tools on detecting the correct number of bars within the charts generated on latex, matlab and excel. It is important to note that BARXTRACTOR is fully automated, whereas the other two tools require input from the user.

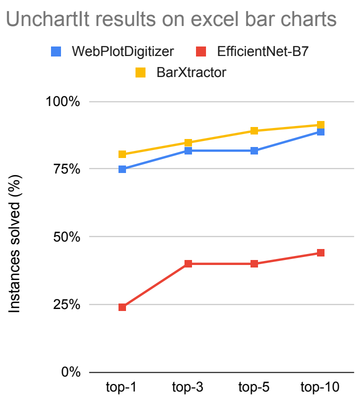
After, in order to evaluate the precision of the numerical extraction on each bar, the MAE of each chart was computed. Figure 5.3 plots the results. BARXTRACTOR has the lowest MAE for matlab and excel images and WEBPLOTDIGITIZER has the lowest error for latex images, although not far from BARXTRACTOR. The EfficientNet approach performs significantly worse on matlab and excel generated charts since it was only trained with latex generated images. BARXTRACTOR performed the best on matlab generated images, which was expected since it was trained solely on matlab images. However,



(a) Results on latex.



(b) Results on matlab.



(c) Results on excel.

Figure 5.4: UNCHARTIT results on the number of queries solved both for latex (a), matlab (b) and excel (c) bar charts.

it is able to generalize these results to images generated on latex and excel.

The obtained extraction results were submitted to UNCHARTIT and it was concluded that there is a correlation between a low MAE on the chart extraction values and the success on synthesizing the correct query. Figure 5.4 presents the results on finding the correct query on the top-1, top-3, top-5 and top-10 candidate queries within 3 minutes.

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.

The OCR evaluation proved that a maximum of two character differences (one character replacement) between the ground truth label and the detected label are enough to obtain the correct label characters. The mean number of differences needed to correct each label is 0.538.

Last but not least, the classifier results have shown in average an accuracy of 93% detecting the correct type of bar chart, between simple, grouped and stacked.

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.

Chapter 6

Conclusions

This dissertation 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. It recognizes bars, labels, and titles, for example.
4. Post processing techniques are applied to resolve common problems in the extraction. For instance, since BARXTRACTOR detects both the labels and the bars, it 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. For instance, in the query synthesis tool UNCHARTIT, the user had to input the maximum and minimum value displayed in the bar chart's vertical axis. 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.

6.1 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. For instance, the bar labels and axis labels can correspond to a column or line in the original data table.

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 [20] to find the mask of each slice. Or detecting the line limit of each slice, the line that connects to the center of the pie, using key point detection [45], Line-CNN [30] or LS-Net [35]. 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.

Bibliography

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

[2] R. Al-Zaidy and C. Giles. Automatic extraction of data from bar charts. pages 1–4, 10 2015. doi: 10.1145/2815833.2816956.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] A. Balaji, T. Ramanathan, and V. Sonathi. Chart-text: A fully automated chart image descriptor. *arXiv:1812.10636*, 2018.

[5] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[7] M. Cliche, D. Rosenberg, D. Madeka, and C. 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.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[9] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

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

[11] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[12] Y. Feng, R. Martins, J. Van Geffen, I. Dillig, and S. Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

[13] Y. Feng, R. Martins, O. Bastani, and I. Dillig. Program synthesis using conflict-driven learning. *ACM SIGPLAN Notices*, 53(4):420–435, 2018.

[14] M. A. Fischler and R. 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.

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

[16] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[18] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[21] B. Horn, B. Klaus, and P. Horn. *Robot vision*. MIT Electrical Engineering and Computer Science. MIT Press, 1986. ISBN 0262081598,9780262081597.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[23] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. 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. doi: 10.1109/CVPR.2017.351.

[24] W. Huang, C. L. Tan, and W. K. Leow. Model-based chart image recognition. In *International workshop on graphics recognition*, pages 87–99. Springer, 2003.

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

[26] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo. Chartsense: Interactive data extraction from chart images. ACM, May 2017.

[27] D. V. Kalashnikov, L. V. Lakshmanan, and D. Srivastava. Fastqre: Fast query reverse engineering. In *Proceedings of the 2018 International Conference on Management of Data*, pages 337–350, 2018.

[28] V. Karthikeyani and S. Nagarajan. Machine learning classification algorithms to recognize chart types in portable document format (pdf) files. *International Journal of Computer Applications*, 39 (2):1–5, 2012.

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

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

[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[32] 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. doi: 10.1109/ICDAR.2015. 7333872.

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

[34] 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.

[35] V. N. Nguyen, R. Jenssen, and D. Roverso. Ls-net: Fast single-shot line-segment detector. *arXiv preprint arXiv:1912.09532*, 2019.

[36] D. Ramos, J. Pereira, I. Lynce, V. Manquinho, and R. 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.

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

[38] J. Redmon, S. Divvala, R. Girshick, and A. 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.

[39] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[42] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

[43] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. 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.

[44] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[45] T. Simon, H. Joo, I. Matthews, and Y. 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.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[47] R. Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[48] R. Stewart, M. Andriluka, and A. Y. Ng. End-to-end people detection in crowded scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2325–2333, 2016.

[49] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[50] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

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

[52] C. Wang, A. Cheung, and R. 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.

[53] S. Zhang and Y. 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.

[54] F. Zhou, Y. Zhao, W. Chen, Y. Tan, Y. Xu, Y. Chen, C. Liu, and Y. Zhao. Reverse-engineering bar charts using neural networks. *Journal of Visualization*, 2020.