CHART2TABLE WITH MULTI-STAGE AND END-2-END MODEL

NGUYEN DAC HOANG PHU
VIETTEL DIGITAL TALENT PROGRAM 2023

ABSTRACT

Visual language data such as plots, charts, and infographics are ubiquitous in the human world. Visual language is also highly complex – besides texts, its structural units can include line, shape, color, orientation, scale, angle, space, etc. One needs to recognize patterns from these structural units, and perform spatial grouping and/or alignment to extract information for reasoning. However, state-of-the-art computer vision models do not perform well on these data. To extract raw information from this, there are many approach such as: rule-based system, use multi-stage model and end2end deep-learning model,.. So on, each type of chart has unique characteristics that are different from others, so using only one method to solve most of the charts will not yield optimal results. Therefore, this report proposes a combination of multiple methods to process each type of chart separately and consider how much it is reasonable to use an end-to-end model.

1 INTRODUCTION

Chart images can be easily found in news, web pages, company reports and scientific papers,... Automatic analysis of these data can bring us huge benefits, including scientific document processing, automatic risk assessment based on financial reports, and reading experience enhancement for visually impaired people. However, raw numerical tables are lost when charts are published as images. These underlying data of charts can be easily decoded. Extracting the raw data table from chart images is the key step for understanding the chart content, which would lead to better analysis of related documents. Recent studies about question answering focusing on querying. chart images would also benefit from it. Another example of the benefits of the problem according to [18] that is: "Millions of students have a learning, physical, or visual disability that prevents a person from reading conventional print. The majority of educational materials for science, technology, engineering, and math (STEM) fields are inaccessible to these students. Technology that makes the written word accessible exists. However, doing so for educational visuals like graphs remains complex and resource intensive. As a result, only a small fraction of educational materials are available for learners with this learning difference —unless machine learning could help bridge that gap. Therefore, extracting the chart to the table information is extremely important in this regard, it will help the students' learning somewhat easier.

To solve the problem, we first need to determine the important information such as the location of the data points, the position of the axes, the marker-label and value of the data points received, the summation steps. All common above step to extract raw data includes:

- **Chart Image Classification (e.g. bar, box, line):** classification task to help create more feature vectors for other tasks. Text Detection and Recognition: detect and recognize the text, bounding box related to that text.
- **Text Role Classification (e.g. title, x-axis label):** based on the above task, we need to determine which bounding box and corresponding label belongs to which areas, helping us to get more information in extracting points data.
- **Axis Analysis:** find labels for data, in addition can be information about point coordinates (for scatter plots) and information about units of measure, multiplier factors.
- **Legend Analysis:** find out how many instance_types are in the data set, whether the plot is in the form of stack, pie,...
- **Data Extraction:** With all the above information, we will find out what is the data point, what shape from the AI model or ad-hoc algorithms will come to the final goal of giving an extracted data table. outputs all the corresponding label and value points information.

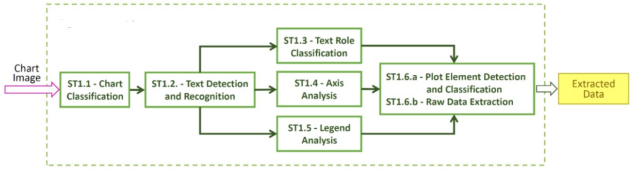


Figure 1: All above step for task data extraction

Note: For the received data set, since there is no legend, we will eliminate the legend analysis task and limit the number of instance charts, so the classification and recognition tasks can be easier, not requiring too much in the design. Set up rules-based for each type of chart.

Contribution Summary.

- Propose new approach uses both multistage model combine heuristic and end2end model to solve this problem.
- Fine-tuning Donut model to solve new task - raw data extraction from chart

2 BACKGROUND

2.1 Common Approach

2.1.1 ***Based On Rule-Based Handcraft.*** These models have in common that they give good results and are limited in extending or exposing samples that have not yet appeared. Famous for extracting data from charts based on intention, image processing algorithms can be mentioned such as Revision (processing for 2 tasks of classification for 5 types of histograms and extraction for only 2 types of pie and bar, model based on connected component detection in the image and color information, pixel frequency to identify instances in that chart), chart-sense (classification for

Supervised by Nguyen Quang Tuan, Pham Thai Hoang Tung.

Mini-Project in VDT, gen 3, (2023)
(C).

about 10 histogram types and experimental extraction for 3 pie chart types), bar, a line based on leveraging more information about people and ad-hoc algorithms based on that information and images), WebPlotDigitizer (result not as good as chart-sense),...

(a) Revision model: In the scope of the main topic, I only mention task data extraction. This process include of 3 step:

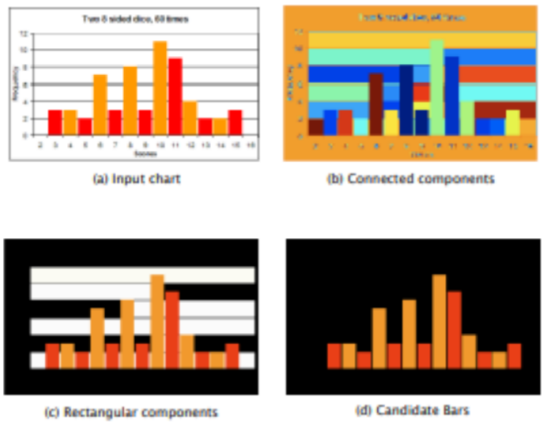


Figure 2: Step1: detect component

- This process includes: b) calculating and creating components that are distinguished by color. c) Remove non-rectangular elements and rectangles that have a different color than the surrounding colors (see the image inside this page), touch the x-axis of the baseline, and have a height greater than 2 pixels or roughly the same width as the average candidate bars. d) determine the candidate bar by determining the width, and determine whether the 4 vertices in the same component are the same color or not.

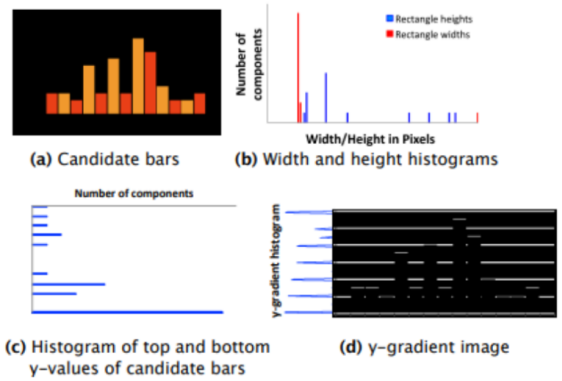


Figure 3: Step2: detect axis

- This process includes: with input from the previous step, (b) a Histogram of the width and height of the bars. Note that the width histogram mode (red) is stronger than the height histogram's mode (blue), indicating that the histogram is vertical. (c) To estimate the base axis, we construct a histogram of the y-values on the top and bottom of the candidate bars and consider the mode of this histogram to be the initial estimate of the baseline. (d) Many charts plot the base axis as a solid line. To fine-tune the axis position, we compute the y-gradient image of the original histogram and then sum the rows to create a y-gradient histogram. We consider the top of this histogram near our original estimate as the final base axis.

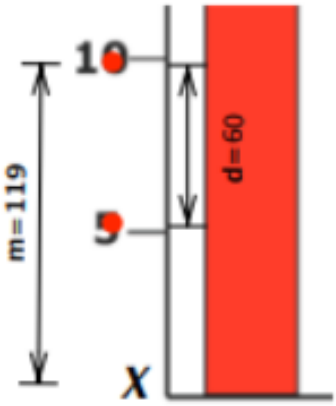


Figure 4: Step3: calculate scale

- After providing information about the axis, bar candidate, we proceed to give an estimate by calculating the scale of the true value and the pixel in the image.

Strength:

- Get more information from the image (color, layout, background in the background).
- The model gives the heuristic to give the value to the bar charts.

Weakness:

- Unable to handle cases like stack-bar, small-bar (undetectable), and marker not in landscape.
- Scalability when dealing with pie-chart, bar-chart only.
- There is no possibility to share knowledge between the 2 tasks.
- The information extraction is not model-based at all, so the results are quite low in the prediction at just over 60

(a) ChartSense model: some symbols [CS] - computer set, [US] -> user set

With bar-chart extraction:

CS Detect baseline (x-axis) and overlay detected baseline on an input image

US Adjust baseline if needed
US Specify two y positions and corresponding data values
CS Detect bars, convert them to values, generate bar graph from extracted values, and overlay bar chart over an input image
US Correct incorrect identification bars if necessary

With line-chart extraction:

Figure 5: The user specifies the key features: two position/y-value pairs (blue line and text) and the time period of the data points (red line).

US Specifies a bounding rectangle around the chart
US Specify two y positions and the corresponding data values for the distance.
US Specifies the interval of the data points.
CS Detect lines within the rectangle specified in Step 1
US Select the correct line from the detected lines.
CS Convert selected lines to values, regenerate line graph from values, and overlay line chart with an input image
US Correction of incorrectly recognized data points

Strength:

- The model increases in accuracy by taking advantage of more human feedback.
- Using backbone classification is less complicated than handling
- Revision when relying on SOTA CNN like VGG, or ResNet.
- Support automation and manual mode

Weakness:

- Because more information about people is required to give this more accurate result, it is not possible to output a large number of outputs, just one at a time.
- Inconvenient when the user usually does not need to edit too much. Algorithms for line charts, radar charts, bar charts, pie-chart

2.1.2 *Based On Multi-Stage Deep-Learning Model*.

One framework propose: based on CNN and also image processing algorithms, heuristics for value estimation.

This model mainly takes raster image, this model will combine separate networks for each task such as classification, detection (Textual Region Detection based on CRNN, Text Role Classification based on SVM), recognition (Decoding Visual Encodings of Charts, Decoding Bar Charts, ..) and information extraction will rely on heuristics and ad-hoc algorithms and on information such as axis, marker. In particular, the framework also proposes to add a plu-

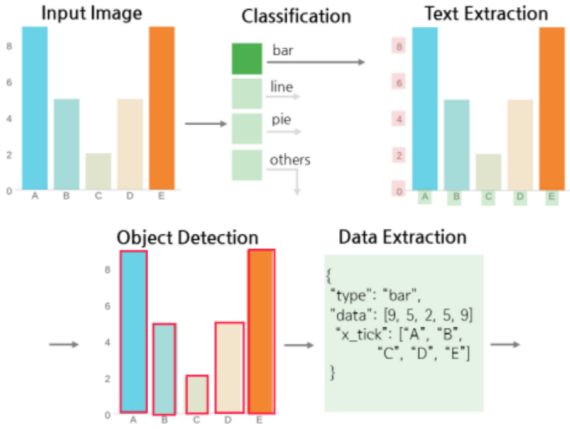


Figure 6: pipeline propose

gin to static charts by getting the extracted information into the chart with the corresponding marker and label to make it easier for users to visualize.

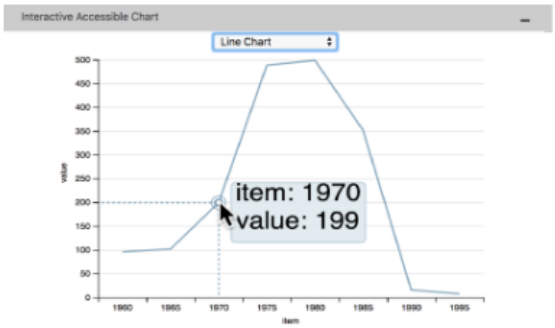


Figure 7: interactive with chart

- **Strength:** Using heuristic algorithms with additional color information can help solve multiple-line, stack-bar problems. Introducing interactive mode.
- **Weekness:** Error propagation, local optimization, complexity in model building, and limited number of charts learned.

ChartOCR Model: This model also relies on sub-model matching, but the special feature is that instead of the heuristic of estimating the value, an estimator will be built immediately inside the model and using this value applied to the loss function of the model. model. Each chart will have its own evaluation metric that matches this estimate. Besides, this model also proposes to use a key-point instead of a bounding-box to make the extracted value more accurate.

After getting the information from the image, we will use a heuristic to extract the data range in the axes according to the following pseudocode:

Each chart will have its own loss information, and these loss functions are designed according to the paper.

- **Strength:** The model uses key-points instead of bounding-boxes to help detect more complex shapes (For column charts, the key points are the upper-left and lower-right corners of individual columns. For line charts, the key points are pivot points on the line). The model can help the estimated value to be learned and self-corrected in the model. The model can be used for both stack-bar and multiple-line.
- **Weakness:** Error propagation, local optimization. Building loss functions for estimated values is quite difficult, and sometimes the search space is different for each chart, it may be necessary to build a separate loss for each data type. Complexity in model building and training time.

2.1.3 ***Based On End2End Deep-Learning Model:*** Some popular models in this task can be mentioned as Donut (based on transformer architecture using both encoder and decoder), DePlot, Matcha (also based on backbone architecture like Donut), and Pix2Struct. The strength is that we have built-in weight sets that can be used for other downstream tasks. We will focus on 2 main models, Donut, and Deplot, while the remaining models are mainly designed to learn tasks such as visual to code, and math reasoning,...

(a) Donut Model: The encoder relies on a swin-transformer architecture where the input is split into patches and encodes to pass through the swin blocks and the decoder uses Bart with the hidden vector from the encoder to output the output. Donut's main task is learning how to read a document (from top-left to bottom-right) by predicting new words based on the next image and previous words (training the encoder). After the results are good, we can use this encoder for other downstream tasks like document extraction, and QA through the fine-tuning decoder and providing input fields needed to distinguish tasks from each other like: <classification >, <parsing>, <vqa>,...

Strength:

- Do not use engines like OCR or recognition and rely mainly on transformers and attention mechanisms, this can increase model scalability and increase accuracy if trained long enough.
- This is an end-2-end architecture, the number of params is not too large compared to other SOTAs. Multi-task learning can be achieved through fine-tuning the decoder.
- The higher the quality of the input image, the better the prediction result.
- Due to the attention mechanism, it will pay attention to all components in the image, solving many types of histograms.

Weakness:

- Annotation for data is very demanding and must provide the fields corresponding to the required task.
- Long training time for encoder (200K steps with 64 A100 GPUs and mini-batch size of 196).
- If there are not enough resources, the result may not be good for the desired task. The inference time is very slow

(b) Deplot Model: This is a model with the main task being plot-to-table, building on the backbone of pix2struct with matcha train, then fine-tuning to perform few-short learning tasks, to perform other downstream tasks like QA, often combined with other networks such as LLM, Classification network.

Strength:

- Invariant to transpose, as well as permutations of columns and rows (so charts like horizontal bar, and vertical bar are well learned).
- Allow but penalize minor errors in numbers or text values up to a certain threshold. Clearly reflect a loss in precision and recall.
- Focus more on the plots in the chart, so training can be faster. Gives better results on labeled data than on data synthesis or generated, dealing with many types of graphs.

Weakness:

- Focus more on the plots in the chart, so training can be faster. Gives better results on labeled data than on data synthesis or generated, dealing with many types of graphs.
- Often omitted information about color, axis_name, and axis_layout should be like that for dot plots, scatter plots are difficult to give accurate results because the axes often contain information about measurement units, and multiplier weights.
- Untested against out-of-context data sets. For good results must depend on the input.

2.2 Metric Evaluation:

The data series for a single figure comprises two instances for evaluation: the series of values along the x-axis and the corresponding series of values along the y-axis. Each data series will be either of numerical type or of categorical type as determined by the type of chart. Predicted data series are evaluated by some common metric such as: **Levenshtein distance** for categorical (that is, string) data types and **RMSE** for numerical data types, with an initial exact-match criterion for the number of values in the series.

2.2.1 ***Leveinstein Score:*** Levenshtein score, also known as Levenshtein distance or edit distance, is a measure of the difference between two strings. Specifically, it calculates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. To calculate the Levenshtein score, we start by defining a matrix to hold the scores for each possible pair of substrings. The matrix is initialized with the scores for the empty strings, which are simply the lengths of the non-empty strings. We then fill in the matrix row by row and column by column, considering each pair of substrings in turn and calculating the score for each position in the matrix based on the scores for the adjacent positions. Formula of this is:

```
L(i, j) = j, if i = 0
          i, if j = 0
          L(i-1, j-1), if s1[i] = s2[j]
          1 + min(L(i, j-1), L(i-1, j), L(i-1, j-1)), if
               s1[i] != s2[j]
```

```
where: L(i, j) is the Levenshtein distance between the
     prefixes s1[1..i] and s2[1..j]. s1 and s2 are the
     input strings and i and j are indices of the
     characters in s1 and s2, respectively.
```

In this problem, to measure matching between ground-truth and predict categorical series value, first we compute leveinshtein between them. Then, we normalized this result distance by use sigmoid function follow the formula to gain a value in range [0,1] (1 is the best matching value best and vice versa):

```
NLev = Sigmoid(Sum(Lev_(y_pred[i],y_gt[i])/len(y_gt) for i
     in range(len(y_pred))
```

2.2.2 RMSE. RMSE stands for Root Mean Square Error, which is a commonly used metric for evaluating the accuracy of a predictive model. It measures the square root of the average of the squared differences between the predicted and actual values. RMSE is generally prefered than MSE because it provides a more intuitive measure of the average magnitude of the errors in the predicted values and is more sensitive to large errors.
In this problem, similar to Leveshtein score, we use sigmoid to normalize this, this formula is:

```
RMSE(y_pred,y_actual) = sqrt(MSE(y_pred,y_actual))
#formula of metric in our problem:
NRMSE=Sigmoid(RMSE(y_pred,y_actual)/RMSE(y_pred,mean(y_actual)))
```

2.2.3 Others. both of metric are used in some specific smaller task than task data extraction. We use this in our-problem because some chart such as: line, scatter,.. are hard to "fit" with end2end model, to solve this, we break data extraction task to some smaller task include of object-detection, text recognition and heuristic we propose,..

- **Precision, Recall:** they are two commonly used metrics for evaluating the performance of a predictive model. Precision is a measure of the accuracy of positive predictions made by the model. It is defined as the ratio of true positives (TP) to the total number of positive predictions (TP + FP). In other words, it measures the percentage of predicted positives that are actually true positives. High precision indicates that the model makes few false positive predictions. Recall, on the other hand, is a measure of the completeness of positive predictions made by the model. It is defined as the ratio of true positives (TP) to the total number of actual positives (TP + FN). In other words, it measures the percentage of true positives that are actually identified by the model. High recall indicates that the model makes few false negative predictions.
- **mAP-50, mAP-95 (Object detection score):** It measures the accuracy of a model in localizing objects in an image and assigning a class label to them. mAP 50 and mAP 50-95 are two variations of the mAP metric that differ in the way they compute the average precision. mAP 50 and mAP 50-95 are two variations of the mAP metric that differ in the way they compute the average precision. The precision-recall curve is then plotted for each class by varying the confidence threshold of the model's predictions. The area under the curve (AUC) is then computed for each class, and the average precision (AP) is calculated as the mean of all the AUCs. mAP 50 is the mean average precision calculated at a threshold of 0.5 for the Intersection over Union (IoU) metric. It measures the accuracy of the model in detecting objects that have an IoU of 0.5 or higher with the ground truth bounding boxes. mAP 50-95 is the mean average precision calculated at a range of IoU thresholds from 0.5 to 0.95. It measures the accuracy of the model in detecting objects with a range of IoU values, from moderate to high overlap with the ground truth bounding boxes.

3 DATA OVERVIEW

3.1 EDA Process

The dataset includes 60,578 instances which are divided into 4 main types:

- **Bar chart:** has 3 varients include of: vertical bar chart has 19183 instance, horizontal bar chart has 73 instance and histogram (Special form of a vertical bar chart - the number of x-axis markers will be equal to the number of bar + 1) and 6 instance - but this chart can be grouped together with the vertical bar chart
- **Dot chart:** has 5131 instance
- **Line graph:** has 24942 instance
- **Scatter chart:** has 11243 instance

The input will be a .jpg image and includes 3 channels, there is no fixed size and the corresponding annotation will be in JSON format. The data set has a source field with 2 main values: extracted (can be taken from a book, scan, excel) and generated (generated from the code).

Horizontal_bar contributes a relatively small amount of only

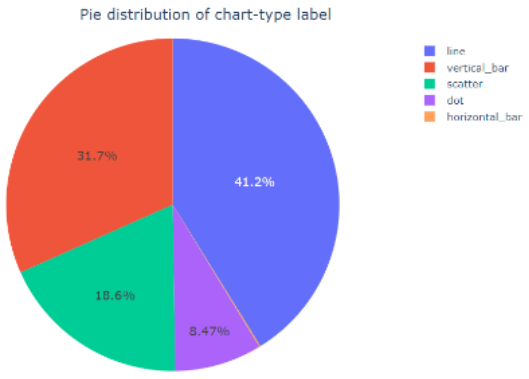
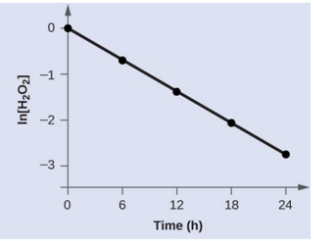


Figure 8: percent how much each type of chart contributes to the overall data set

about 73 instances, which makes the dataset imbalance and makes it difficult for the model to correctly identify tasks related to this type of chart.

In addition, we will delve a bit deeper into data annotation:

- **source:** Whether generated or extracted.
- **chart-type:** One of dot, horizontal_bar, vertical_bar, line, scatter.
- **plot-bb:** Bounding box of the plot within the figure, given by height, width, x0, and y0.
- **text/id:** Identifier for a text item within the figure.
- **text/polygon:** Region bounding the text item in the image.
- **text/text:** The text itself.
- **text/role:** The function of the text in the image, whether chart_title, axis_title, tick_label, etc.
- **axes/x|y-axis/ticks/id:** Identifier matching the tick to the associated text element id.
- **axes/x|y-axis/ticks/tick_pt:** Coordinates of each tick in the figure.
- **axes/x|y-axis/tick-type:** The graphical depiction of the tick element.
- **axes/x|y-axis/values-type:** The data type of the values represented by the tick element, whether categorical or numerical. This field determines how the predicted data series are scored. See the Evaluation page for more information.
- **visual-elements:** Indicates part of figure representing the data series. Depends on the chart type. Visual-element related chart corresponding is visualized with red color (in the form of outline or point) in figure 10, 13, 14 and 15
- **data-series/x|y:** The x and y coordinates of values depicted in the figure. For the test set images, this is the target to be predicted. So the main task is to extract data series values along this of the corresponding image from the input image (figure 9 show this).



Time	ln[H202]
0	0
6	-0,8
12	-1,4
18	-2,1
24	-2,8

Figure 9: Input is image chart and output is table

Beside, based on the fields and values when annotation, I have drawn a few conclusions such as:

- This dataset has no legend so each type_instance in the chart is unique (for plot lines - infer only a single line, scatter - dots of the same color, same size, dot), all chart have only 2D shapes.
- Bar chart will not stack the bars up, nor will there be negative values like the two images below.
- With annotation, there is a source field that only receives 2 extracted values - ie the doer and generated - the machine makes the bounding box results of some of these instances wrong (bounding-box coordinates have negative values, do not match with the bounding box coordinates). figure), erroneous instances are gathered in:
- Dot chart only appear in 'generate source' (can create dot chart from: link) and horizontal bar chart only appear in 'extracted source', so sometimes dot chart annotation gives wrong results.
- The Y-axis can sometimes carry important information (such as units of measure, multipliers, or information that has been normalized). Some instances sometimes don't have markers on the x-axis and if there are markers the markers can also be tilted.
- Each data point may or may not have a representative value, requiring us to rely on the axis and bounding box to find the rescales that bring about the appropriate coverage ratio.

3.2 Chart Feature And Limitation

3.2.1 ***Bar-chart:*** There are independent values along the x-axis and their dependent values along the y-axis, while the horizontal bar chart has the role of transformed axes. The independent values are categorical and can be defined as a series of tick marks below each bar. The dependent values will be numeric and determined by the height (or length) of bar.

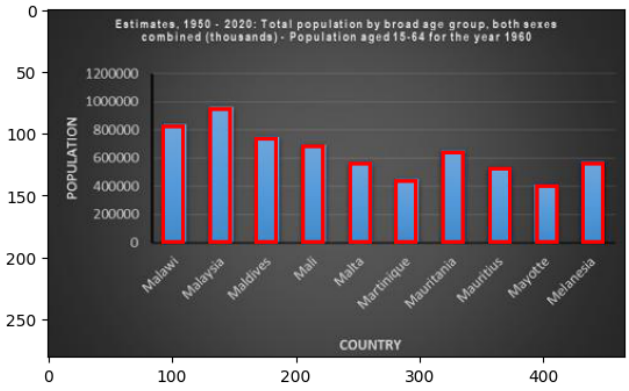


Figure 10: Vertical Bar

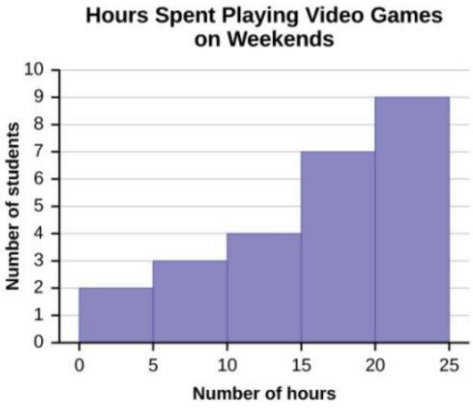


Figure 11: Vertical Bar with Historgram Type

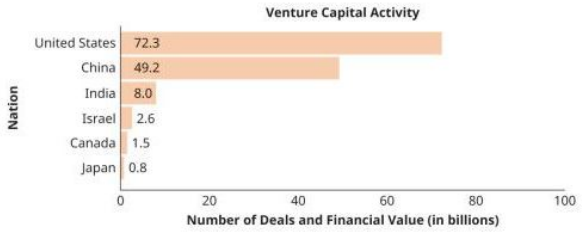


Figure 12: Horizontal Bar Type

Advantage:

- The bars are connected components.
- Bounding boxes of rectangular bars are easy to detect.
- More information about colors can be used

Disadvantage:

- The background has rectangular bars, the distance between the bars is also rectangular, so it can cause disadvantages in the algorithms related to the connected component.
- Markers can be slanted
- Must consider more cases related to histogram (x_axis = n_bar + 1)

3.2.2 ***Dot-chart:*** The dot plots have independent values along the x-axis and their dependent values on the y-axis. The x-axis values will be numeric if the tick labels can be parsed as Python floats; on the other hand, they are categorical. The y-axis values are the number of dots in each column. The y-axis itself may or may not be present in the figure.

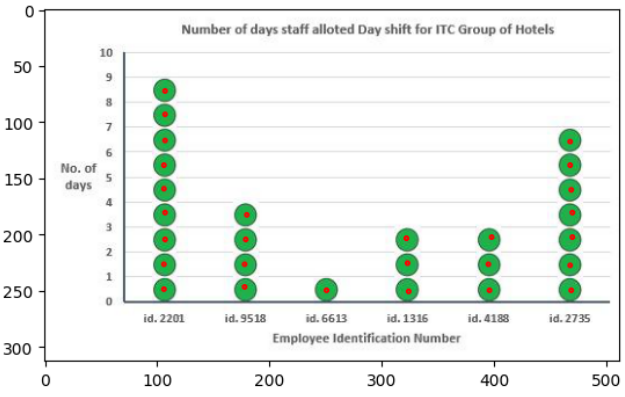


Figure 13: Line Chart

Advantage:

- The predicted numeric values do not need to be evaluated by the scale value, it is possible to count the number of objects belonging to the same marker.
- The number of objects is not too much in each marker.
- Usually contains the values of the unit of measure, the weight multiplied in the y_axis.
- More information about colors can be used

Disadvantage:

- The dot-chart is generated so the bounding box information may be wrong.
- There exists a case where there is no y_axis or x_axis in the annotation.

3.2.3 ***Line-chart:*** Line charts always have categorical x-axis values and numeric y-axis values. The values for the x-axis are the labels that mark the x-axis below certain parts of the line chart. Markers that are not below some parts of the chart are not included in the series. The values for the y-axis are the respective dependent values represented graphically.

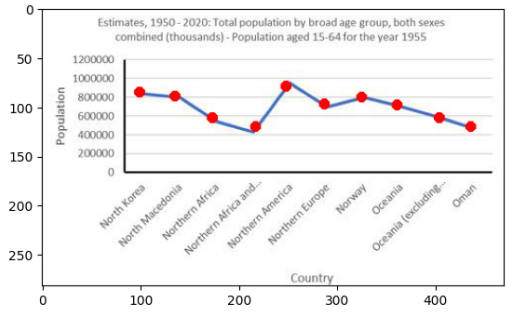


Figure 14: Dot-plot

Advantage:

- There is a marker to mark the vertices on the chart (or can be determined by the coordinates of the vertices in the bend).
- The number of markers on the chart equals the number of markers on the x-axis

Disadvantage:

- Bounding box along the winding road makes it difficult.
- The value must be estimated based on information from the 2 axes (data range with the y-axis and pixel distance with the x-axis) to find the appropriate scale value.

3.2.4 ***Scatter-chart:*** Scatter charts always have numeric x-axis and y-axis values. Predict both x and y values for each point included in the figure. There can be multiple points with the same x or y coordinates. This is the most difficult chart type to deal with

Disadvantage:

- The x and y values of a point must be quantified in both axes.
- The points may overlap or overlap, making detection difficult.

4 IMPLEMENTATION

4.1 Proposed Architecture

With the goal of the given problem and the desire to be different from other approach, my main idea in this problem is summarized in upon figure. Initially with the input chart image, first I pass it

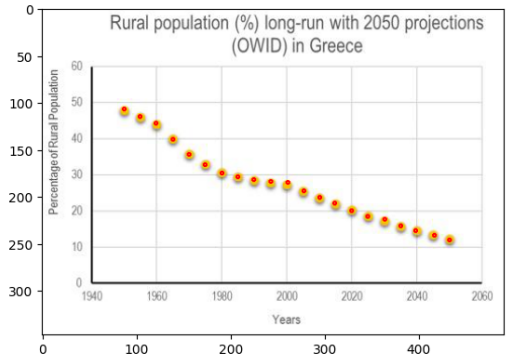


Figure 15: Scatter-Chart

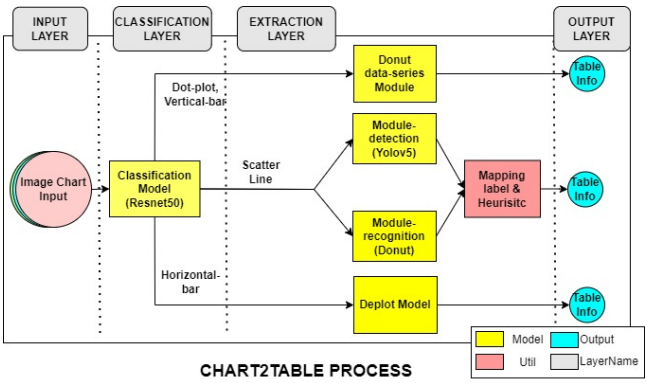


Figure 16: My architecture approach

through a classification label to get chart-type. Then with chart-type result, if chart-type result is horizontal_bar, I pass this image to pretrain deplot model. Else if chart-type is vertical_bar or dot plot, we pass it to my custom donut model to extract information. Finally if chart-type is scatter or line, we pass it into 2 submodule - one is module detection and other is module recognition (implement by donut follow document-parsing task). With output of 2 submodule, we mapping it through classification type (one box mapping one axis label if they have the same class-type), then I use heuristic to get raw data extraction can be listed including the steps: Inference empty marker in axis, inference pixel distance between 2 marker in x_axis and y_axis, inference real value distance between 2 marker, then inference height or width of visual-point (the points marked on the graph) in comparison to the landmark points,... Finally, we reconstruct result convert to data table format.

4.2 Classification Stage

This is the first stage of my approach, all input image will be pass through it to get chart-type. I use ResNet-50 as backbone architecture with output has 5 class corresponding to the 5 types of charts listed in the graph. There is an issue encountered during training, which is that there are too few instances (around 73) for the horizontal_bar data, while the other classes have 5000 instances

Source	Chart-type	Time infer
extracted	horizontal-bar	73
	line	423
	scatter	165
	vertical-bar	457
generated	horizontal-bar	18000
	line	24519
	scatter	11078
	vertical-bar	18732
	dot	5131

or more. Therefore, I have proposed some approaches below to address this severe imbalance problem.

- Add data from external sources, and this data volume is not too different from the other classes.
- By rotating the vertical_bar 90 degrees counterclockwise, we will have a corresponding horizontal_bar image, so we can create an additional horizontal_bar set from the vertical_bar set.
- Discard the horizontal_bar image altogether and do not use it in subsequent stages.

After considering ease and effectiveness, I chose the first option with a data source from Kaggle, and this dataset consists entirely of horizontal_bar images. I took a number of instances equal to the number of instances in the vertical_bar dataset.

Regarding the model, the input will be processed in batches, and each data point will be normalized according to the mean and variance of that entire batch before being fed into the model (which will be taken from torchvision and trained from scratch with 10 epoch)

4.3 Data Extraction Stage

As mention above, with chart-type result from previous step, each type of chart will follow a specific model, and there is no common model to handle the 5 types of charts because each type of chart has its own unique and very different characteristics, and sometimes these characteristics are contradictory to each other. The details of each execution thread will be listed below.

4.3.1 ***Propose Approach For Horizontal-Bar:*** Because the imbalance in horizontal_bar class and there is no external source which has annotation, we don't train for this class. Instead of, we use one model is trained and has been verified to be suitable for this type of chart, which is deplot model. This model and processor are processed and fetched from Hugging Face. Then, we will transform the input to fit the model format (through the processor). The processing steps include.

```
#get model from hugging-face
model = Pix2StructForConditionalGeneration.from_pretrained
        ('google/deplot').to(device)
processor = Pix2StructProcessor.from_pretrained('google/deplot')

#transform input to match format model input
inputs = processor(images=image, text="genData", return_tensors="pt")
inputs = {key: value.to(device) for key, value in inputs.items()}
```

```
#prediction and decode information
predictions = model.generate(**inputs, max_new_tokens=512)
data = processor.decode(predictions[0], skip_special_tokens=True)
```

4.3.2 ***Propose Approach For Vertical-Bar, Dot-Plot:*** Because the characteristics of these two types of charts are similar, they have a number of data points that need to be extracted by the number of markers on the x-axis and the data points contain a lot of color information - quite large and strongly connected. I have group them together and use donut End2End model to extract it. Because to specify a specific task for the Donut task, we need to use a pair of tag values that can be represented as follows: <|PROMPT|> and <|PROMPT_END|> (which can be considered as specifying the start and end points of the generated string). This is also the tag pair specified in defining tasks for my model. The output string of this Donut model will have the following format: <|PROMPT|><chart-type><x_start>x_data_series<x_end><y_start>y_data_series<y_end><|PROMPT_END|>, I add field chart-type in this because I want the model to learn which type of chart will have its own way of reading, for bar-charts, we only need to read the length of each bar, but for dot-plots, we need to count each element on the same x column, and this label will also serve to double-check the classification results above.

The challenge of training the Donut model is that we have to set up the input to fit the model's input_format. This input will include many types, such as the ground_truth in the form of <|PROMPT|>..<|PROMPT_END|> as mentioned before and must map this ground_truth according to the dictionary of the processor, images in pixel_value format that have been padded

Finally, we apply pytorch_lighting to train this model in multi-gpu environment. The process of predicting data is similar to the way Deplot does above, so it will not be mentioned here.

4.3.3 ***Proposed Approach For Scatter, Line Chart***. These are the two most difficult types of charts to process because their characteristics are completely different. For line-charts, the x-marker axis labels are tilted and the visual-points of this chart type are equal to the number of x-markers and lie on a straight line. The number of points is also not too many. On the other hand, scatter-charts can have a large number of visual-points and they are not in any predetermined order. Both the y and x values of scatter-charts need to be estimated. That's why I separated these two types of charts and brought them to multi-stage processing according to the steps listed above. I will not merge the object detection and text recognition models to process them together for both types. Instead, I separate them to achieve the best corresponding efficiency and help optimize the extraction design according to specific heuristics. Combining object detection and text recognition for both types of charts could be a potential direction for the future work of this project.

(a) Module detection At this stage, I use the YOLOv5m model as the general baseline. It is also relatively fast and easy to develop, thanks to its user-friendly interface and pre-trained models that can be fine-tuned on specific datasets. However, the accuracy of the model may vary depending on the complexity of the object detection task and the quality of the dataset used for training. Therefore, it is important to carefully select and preprocess the data to achieve optimal results.

The bounding box coordinates are typically represented as a tuple of four values: (x, y, w, h), where x and y are the coordinates of the top-left corner of the bounding box, and w and h are the width and height of the bounding box, respectively. Can be represented as normalized coordinates, where the values are scaled to be between 0 and 1, relative to the size of the input image.

The bounding box of a ground truth annotation in the format ((x1,y1), (x2,y2), (x3,y3), (x4,y4)) represents the 4 corners of a rectangle in order from left to right and top to bottom. Therefore, it is necessary to fine-tune the annotation set to select the center point and estimate the width and height ratio.

After training the model, we can save the weights and use torch.hub to facilitate the inference process using the trained weights. The image appears below to be an example of an object detection result produced by a trained model. The model has detected and drawn bounding boxes around various objects in the image, indicating the location and size of each object. Several concepts need

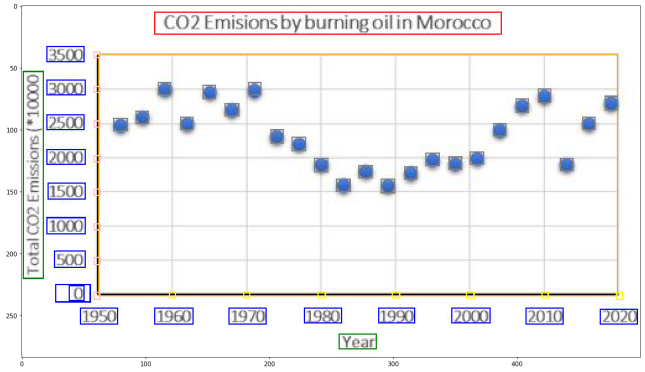


Figure 17: The example image has been detected by a trained model

to be explained based on the detection of bounding boxes, and we will use this image to explain those concepts.

- The **red** bounding box is a **chart-tile**
- The **pink** bounding box is a **y-axis-marker**
- The **yellow** bounding box is a **x-axis-marker**
- The **gray** bounding box is a **visual-point**
- The **blue** bounding box is a **label-marker**
- The **green** bounding box is a **chart axis-title**
- The **orange** bounding box is a **plot - the graphing part**

(b) Module Recognition This stage has many options, such as using the latest OCR models, including easyOCR, paddleOCR, kerasOCR, etc. However, these models may depend on the result of the bounding box, but the received data has tilted and vertical markers. Therefore, if this method is applied, it requires additional preprocessing and relabeling steps. Besides, I am also using Donut, so I tried to fine-tune Donut for this task (which is similar to the document parsing, one of the three main tasks of Donut) and succeeded in obtaining a fairly good result regardless of the marker cases. The only drawback of this method is the relatively long inference time because the inference of the transformer model involves high computational complexity. However, to trade-off for the high accuracy

in extraction, I have chosen this approach. Fine-tuning the Donut model and adjusting the data format exactly the same as the image reading task in Donut, as presented above, resulted in the output of reading a chart. This is text extract from figure 17

```
{'x_tick_label_mark': ['1950','1960','1970','1980','1990','2000',\
'2010','2020'],
 'y_tick_label_mark': ['3500','3000','2500','2000','1500','1000',\
 '500','0'],
 'chart_title_mark': ['CO2 Emisions by burning oil in Morocco'],
 'x_axis_title_mark': ['Year'],
 'y_axis_title_mark': ['Total CO2 Emissions (*10000']}
```

(c) Heuristic For Extraction Data Each chart will have its own heuristic for processing, due to the characteristics of scatter charts having many points and requiring estimation of both x and y values. The number of points depends entirely on the bounding box results, so if some visual points are missing, we cannot deduce anything. Therefore, we have to accept this limitation in scatter charts. In contrast, with line charts, we can deduce the missing points based on color and marker-x (since the number of marker-x equals the number of visual points in the majority of cases). That's why line charts require more processing to identify missing data points.

Heuristic For Line chart. Some characteristics of the bounding box obtained from the model could include:

- Due to the tilted x-axis label, the related bounding box may not be detected.
- Some markers may not be detected, either inside or outside the bounding box.
- On the line points, sometimes the bounding box is missing (as in this image) or there are extra bounding boxes nearby."

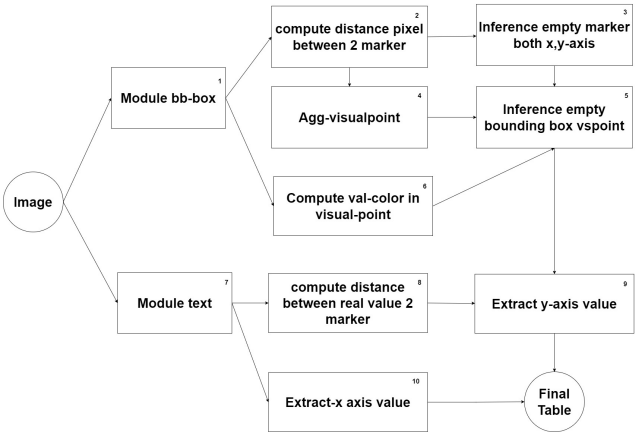


Figure 18: An overview diagram of the heuristic

This is an explain each step in my heuristic.

- **Two module bb and module text - block 1 and 7 in figure 17** are based on yolo-v5 and donut as mentioned before.
- **With bounding-box coordinate marker - block 2** in axis, I will compute distance pixel between 2 marker - the calculation is performed for both the x and y axes.
- **Inference empty marker - block 3**: This step mainly infers the missing x-axis markers to utilize the constraint (the number of visual points equals the number of markers), and the inference is straightforward. First we compute distance pixel between 2 marker consecutive to form a distance sequence, then I use IQR method (IQR stands for Interquartile Range, which is the range between the first and third quartile of a dataset) with this list to determine the allowable range of values for pixel distance.

```
#coords (coordinate of bounding box)
distance_pixel = []
for i in range(1,len(coords)):
    distance_pixel.append(coords[i] - coords[i-1])

def IQR_method(input_lst):
    Q1 = np.percentile(input_lst, 25, method='midpoint')
    Q3 = np.percentile(input_lst, 75, method='midpoint')
    IQR = Q3 - Q1
    upper = Q3 + 1.5*IQR
    lower = Q1 - 1.5*IQR

    return lower, upper

lower, upper = IQR_method(distance_pixel)
for ele in distance_pixel:
    if ele >= lower and ele <= upper: ...
    else: ...
```

With every element in distance sequence, we check whether the element falls within the allowable range or not to detect outliers (and outliers are the approach to infer missing markers). If it does, we skip it; otherwise, we know that between these two points, there will be k missing markers, where k is the ratio between the distances to the reference points of those two points. After interpolating the missing markers between two marker, we check if there are any missing markers at the beginning or at the end of axis by comparing the number of markers were detected with the x-axis label-marker (label-text has been detected by the text recognition module). If not, we continue to infer these missing markers using another constraint, where the first missing marker must have a greater x-coordinate value than the x-axis, and the last sequence of missing markers must appear at the end of the x-axis.

- **Agg-visual point - block 4:** This step occurs because many bounding boxes detect overlapping visual points. Therefore, I propose a simple way to handle this, based on the bounding box coordinates. We check if the distance between two sorted bounding boxes is greater than 0.5 times the pixel distance between two markers. If it is, they do not overlap; otherwise, they overlap, and we combine the overlapping points into a single point based on the mean.
- **Determine color value in visual-point - block 5:** I will map the visual point coordinates with the x-axis markers to determine which visual points are missing. For the existing visual points, We will take the bounding box of that point and iterate through each pixel within it and add them to a list, which we will temporarily refer to as color_list. We will also use the IQR method with color_list to determine

an acceptable range of colors that can be considered as the color of the visual_point.

- **Inference empty bounding box visual point - block 6:** With the current set of markers and visual-points that have been calculated based on their respective abscissa values, we will map each visual-point with its corresponding marker (if their abscissa values are similar). For markers that do not have a corresponding visual-point, we will iterate downwards along the ordinate axis starting from those markers (i.e., moving upwards towards the y = 0 coordinate) by keeping the marker's abscissa value constant and subtracting 1 from the ordinate value. We will continue iterating in this manner until we encounter a point that falls within the acceptable color range of the calculated visual-point located directly above it, at which point we will stop and designate this point as the new visual-point. From there, we obtain the complete set of visual points.
- **Compute distance between real value 2 marker - block 8:** This step is similar to the pixel distance calculation, but instead, we calculate the real value on the label to estimate the actual distance - this time we will calculate along the y-axis. Based on this value, we can divide it by the distance_pixel value along the y-axis to obtain the scale of the height.
- **Extract y-axis, x-axis value - block 9**: For each visual point, we will calculate its distance to the x-axis of the chart. Then, we can obtain the y_value corresponding to the height by multiplying the height with the previously calculated height scale. Finally, we add or subtract the anchor value (depending on whether y_value is increasing or decreasing). For the x-axis value, we can simply extract the value from the text module.

Heuristic For Scatter chart: The inference process for data points on a scatter plot is similar to the line chart. We will calculate the scale_x and scale_y based on the corresponding distance_pixel and real distance values. Then, we will perform inference by multiplying the distance of each visual point to each axis with its corresponding scale and adding or subtracting the anchor value. However, one difference between a scatter plot and a line chart is that a scatter plot lacks constraints on the number of markers or visual points, so it may be difficult to automatically infer the corresponding visual points that are missing.

In the explanation below about the execution flow of this heuristic, it will rely on blocks as shown in Figure 18. For the scatter chart heuristic, it will use the block 1 to retrieve the bounding box coordinates, then pass through the block 2 by calculating for both the x and y axes. With the text module, the heuristic will also calculate the block 8 to retrieve the actual values of x and y, then pass through the block 9 to estimate the values and produce the final result.

5 RESULTS AND DISCUSSION

5.1 Classification Stage

Model/Res	Test_score (acc) ↑	Time inference ↓
Res-Net50	0.995800	0.1s
Donut classification	0.98	1.9s

ResNet takes around 40 minutes to complete one epoch of training, while the corresponding Donut model takes around 1 hour and 40 minutes. However, ResNet achieves high accuracy, even up to 100% on the validation set. Therefore, you have chosen to continue using ResNet due to its high accuracy and faster inference time.

5.2 Text Recognition Stage

Model/Res	NED ↓	Time infer ↓
Line	0.0121818	25.5s
Scatter	0.0026136	28s

Both of these models were fine-tuned using Donut, with each epoch taking approximately 2 hours to complete. The NED result of Scatter is much lower than that of Line because the label markers of Line sometimes tilt and overlap each other, while Scatter does not have such cases. However, after only 2 epochs, the results of both models have almost converged.

5.3 Object detection stage:

Model/Res	MAP-50 ↑	MAP-50-95 ↑	Time infer ↓
Yolo-v5m (scatter)	0.711	0.462	10.8ms
Yolo-v5m (line)	0.71	0.468	10.8ms

We can see the MAP50 of both models, but these are results of detecting total instances. Only the instances related to the title are prone to be incorrectly detected (as the data sometimes contains elements and sometimes not), while the detection rates of important components such as markers and visual-points are still very high. In addition, heuristics can infer missing markers, while for visual-points, Line has been able to deduce the missing points, so this weakness can be overcome

5.4 Data extraction stage

result /model	Val score↑	Hor score↑	Ver score ↑	Scatter score ↑	Line score↑	Dot score↑
(1)	0.33831	0.0	0.4783	0.0152	0.4275	0.9809
(2)	0.46141	0.54204	0.5405	0.06213	0.447	0.99076
(3)	0.64447	0.54204	0.57031	0.85818	0.43776	0.94915
(4)	0.6996	0.54204	0.53674	0.85818	0.69673	0.98957

- **Experiment 1** - donut dataseries model as present in (1) but for 5 chart: line chart, scatter chart, dot plot, vertical bar, horizontal bar.
- **Experiment 2** - donut dataseries model as present in (2) but for 4 chart: line chart, scatter chart, dot plot, vertical bar, horizontal bar, the horinzontal bar use deplot to extract data information.
- **Experiment 3** - donut dataseries model as present in (3) but for 3 chart: line chart, dot plot, vertical bar, the horinzontal bar use deplot to extract data information, scater use multi-stage model.
- **Experiment 4** - proposed approach.

The val_score will be calculated as the average instance with either Levenshtein or RMSE metric, depending on the type of the corresponding series (numeric or categorical value). The number

of instances in each chart is as follows: Horizontal_bar: 73, Vertical_bar: 114, Scatter: 100, Line: 100, Dot: 50. The general formula for calculating the val_score is:

```
Val=(73*Hor+114*Ve+100*Scatter+100*Line+50*Dot)/(73+114+100+100+50)
```

The scores for each chart will be calculated similarly to the val_score, value in range [0,1] but the instances will be based on the corresponding chart type. All of these sets will be calculated on the pre-prepared test set, which was not used in the training process.

The green cells represent the method with the best score for the corresponding chart type.

5.5 Time Inference

Model/Res	mean-Time infer ↓
Line	21.8s
Scatter	25s
Deplot for hor	4s
Donut bar	20s
Donut dot-plot	21s

5.6 Limitation

We observe that inference takes a considerable amount of time due to the models mainly using transformers, but in return, it will provide us with a fairly accurate result.

- In reality, there are some charts that use multiple pie charts, radar charts,..., but they are not included in the dataset. Therefore, we need to collect and process additional data to be able to apply them in practice.
- Not performing task axis analysis can affect our ability to apply the technique to real-scenario charts, which often have multiple instance types. This requires collecting additional data in this format and labeling additional fields, such as the legend and legend mark, to enable recognition algorithms to detect this additional useful information. This is particularly important for stacked bar charts and multiple line charts. Special charts such as 3D shapes, composite plots of multiple different charts, and charts that are outside the scope of the training data require additional consideration for how to approach them when encountered.
- Rule-based algorithms that are hand-crafted by humans may fail in some cases when run automatically, as they are heuristic algorithms with fixed hyperparameters that may perform well in some cases but may be sensitive to other special cases (no free lunch).
- Deployment has not yet been performed so that others can use and test the model. Transformer models take a relatively long time to inference results, while other models provide results much faster. Therefore, some adjustments or additional methods may be needed to help these models produce results that are nearly as accurate as the original ones but faster.
- End-to-end models require a sufficiently long training time and a large dataset to produce good results. Within the scope of the project, there were not enough resources or data to train Pix2Struct or Donut to meet initial expectations.

6 CONCLUSION

Within the time constraints of the project, I have been able to implement a complete approach for this task to process and provide good results, although there is still room for improvement. The current method has performed well on the majority of the dataset with line and scatter charts, but there are still some special cases that require more error analysis to improve the heuristics. The method has worked well on dot plots and horizontal bars, and with enough training on vertical bars and donut charts, it may produce good results. Finally, I proposed a multi-stage algorithm to extract table information through object detection, text recognition, and a heuristic approach to tackle line-chart and scatter-chart problems. However, there is still much room for improvement in this highly applicable problem.

7 FUTURE WORK:

- Developing and improving the proposed heuristic algorithms to automatically adjust fixed hyperparameters can help to address more special cases.
- Collecting additional data to predict more types of charts, such as pie charts, radar charts, or charts with legends, can help to adapt to more real-world contexts.
- Expanding the system to allow user interaction, such as specifying markers, providing estimated ranges, adjusting bounding boxes to better fit entities, limiting the chart range, and providing feedback, can be very helpful in real-world systems. To be more visual, for each chart (now static), when running the algorithm, the annotation results can be provided directly to help users visualize more easily.
- The system can be used for the next steps of building a Question Answering (QA) module on data extracted from charts or generating comments to describe the semantics of the corresponding charts. Furthermore, it is possible to create a model based on the information in the chart to infer the causes of value decrease or increase, detect anomalies, and even predict future values that the chart may bring.
- Improvements can be made to the inference of transformer models through methods such as quantization, pruning, or deep intervention in the model, such as utilizing long-context, distance-enhanced attention score, efficient and low-rank attention, adaptive modeling,...

8 ACKNOWLEDGEMENTS

Throught this mini-project, I have gained a deeper understanding of how to fine-tune transformer models in the OCR field, such as Donut, Pix2Struct, and training YOLOv5 on custom datasets. I have also gained more experience in training time-series models on platforms like Kaggle, Colab, and Colab Pro to achieve faster results. Additionally, I have become more familiar with using PyTorch and related libraries to save models, such as PyTorch Lightning (for saving models on Hugging Face and training on multiple GPUs)

and WanDB for real-time performance tracking and training results. Although some report before, I have familiar and working with NLP, time-series field but this project help me learn a lot of skills about Computer Vision and image processing, so I think with my experience from this, I can do better at problems related to this image field.

I would like to thank the very enthusiastic and dedicated support in imparting knowledge and experience of two mentors, Nguyen Quang Tuan and Pham Thai Hoang Tung. Thanks to the Viettel digital talent program, there are many useful experiences and knowledge that will be extremely important baggage in my future career path with the goal of developing Vietnam's science and technology.

9 REFERENCES

[1]: **Revision: automated classification, analysis and redesign of chart image**
[2]: **ChartSense: Interactive Data Extraction from Chart Images (6706 - 6717)**
[3]: **Visualizing for the non-visual: Enabling the visually impaired to use visualization.**
[4]: **Data extraction from charts via single deep neural network.**
[5]: **Data extraction from charts images via a deep hybrid framework. 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1916–1924.**
[6]: **DEPLOT: One-shot visual language reasoning by plot-to-table translation**
[7]: **MATCHA : Enhancing Visual Language Pretraining with Math Reasoning and Chart Derendering**
[8]: **OCR-free Document Understanding Transformer**
[9]: **Donut Hugging-Face**
[10]: **Deplot Hugging-Face**
[11]: **Chart-QA**
[12]: **Plot-QA**
[13]: **VisionTapas-OCR, VL-T5-OCR, T5-OCR**
[14]: **Useful notebook**
[15]: **Metric implementation**
[16]: **Deep-visualize dataset**
[17]: **Donut fine-tuning**
[18]: **One Specific Usecase**

A APPENDIX

A.1 Detail Fine-tune The Other End2End Model

I have tried to fine-tune another end-to-end model on task extraction data, which is Pix2Struct with pre-trained weights from Matcha. Currently, I only trained on 2 types of charts, which are bar chart and dot chart. The combined dataset of these 2 chart types is about 23,000 instances with a batch size of 2. I trained for about 4 hours for 1 epoch, and with 2 epochs, this is the current result that I have achieved when i observe current loss:

The current model is overfitting when the train-loss has a decreasing signal to 0 but the val-loss fluctuates and doesn't converge and the loss is concentrated around 20-30, a very large range. When I test the model with any random image from the test set, the output always has the format <|BOE|><|BOE|>...<|BOE|> (similar to the DONUT indicator for custom tasks to be read). This further

Figure 19: Train loss

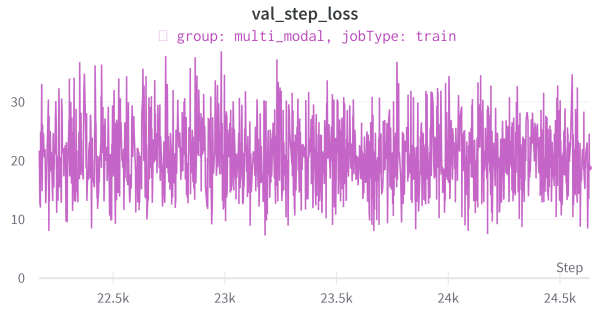


Figure 20: Val loss

confirms that the model is overfitting.

From this, we can infer that the given model, even though using pretraining, needs to be trained again on the entire encoder in order to fit the custom dataset. This is a challenge because we need to find suitable parameters for the model to learn well, and we have to train the model for a long time with a large amount of data before it can perform well.

A.2 Experiment Other Object Detection Model

I chose YOLOv5 for both scatter and line chart detection in order to quickly establish a baseline and focus on the main task of data extraction. After completing the data extraction task, despite time constraints, I was able to try other object detection models such as EfficientDet and Faster R-CNN, which also showed promising results. However, due to the time limit, I have only applied these models to scatter charts and have not tested them on a general approach to see if they can further improve performance. These are the current results I have achieved.

Model/Res	MAP-50 ↑	MAP-50-95 ↑	Time infer ↓
Faster-RCNN	0.770	0.556	5ms

I am currently having difficulty benchmarking EfficientDet with the two metrics mAP50 and mAP50-95, as I can only obtain the train_loss and val_loss results for each epoch to demonstrate that the model is performing well (with loss lower than both Faster R-CNN and YOLOv5).

Name/Loss	Train_loss	Val_loss
epoch 1	42.93	0.7661
epoch 2	0.4152	0.2665
epoch 3	0.2211	0.199
epoch 4	0.1801	0.1697

In general, these detection models hold promise for achieving even better results if trained for a longer period of time (as the two models you trained were only trained for 4 epochs, each taking about 30 minutes). This could potentially alleviate the burden of designing heuristics to handle missing bounding box information. However, this also depends on various factors such as dataset size, model architecture, training speed, and hardware capabilities. In addition to increasing training epochs, you may also consider using pre-processing techniques or other methods to address the issue of missing bounding box information.

A.3 Experiment Text Recognition Model

I have also tried several text recognition models but at this stage, the results are still not good. First, I researched and built a custom CRNN model using fine-tuned Resnet18 by removing the last 3 layers, adding a CNN layer, and then 2 LSTM layers for this task. When training, I noticed that the issue lies in:

Figure 21: train and val loss function get bad result

This happens because the CNN layer after Resnet18 needs to fix its kernel to identify the region where the text instance is concentrated. In addition, the number of characters in an image is not fixed, while this model usually performs well with a fixed number of characters (which is also a constraint for fixing the CNN kernel.

Additionally, I am also exploring the PaddleOCR and EasyOCR OCR engines, but I am encountering an issue where these models are difficult to use on Colab due to the requirement of libcudart library related to GPU configuration.