Bar X-Tract

A Pipeline for the Extraction of Data from Images of Bar Charts

15.S04: Hands on Deep Learning
Massachusetts Institute of Technology
Spring 2023

Ayush Shukla, Gavin Findlay, Rachit Jain, Dafne Badilla

Contents

1 Problem

1.1 Motivation

Millions of students have a learning, physical, or visual disability that prevents a person from reading conventional print, making the majority of educational materials for STEM fields inaccessible to them. As part of the **Benetech - Making Graphs Accessible** Kaggle competition, we would like to democratize the access to data visualizations, leveraging deep learning tools.

The objective is to predict the data series represented by four kinds of scientific figures (or charts); but given the different topics covered over the semester, we would like to take a step further and scrape the information of the data visualizations using ML. As an extension of this project, we would also like to develop an AI generated text summarizing the content of the plots, which could be easily used as a voice note, successfully helping us to reduce the accessibility gap to be addressed.

1.2 Data

The dataset consists of 60,587 charts, which have been annotated and categorized into five types: horizontal and vertical bar graphs, dot plots, line graphs, and scatter plots. The majority of the figures have been artificially generated, but the dataset also includes several thousand extracted from reliable, professionally-produced sources.

The data contains JPG images of the graphs, alongside JSON files containing detailed annotations describing source, chart type, text boxes in the plot, the locations of the tick marks along both axes as well as the data point values. The JSON file is essentially the data that we wish to build models to extract from the images of the graphs. The data was split into training and test sets, with a 80/20 ratio. We applied stratified sampling to ensure both sets were balanced with the amounts of each chart type.

1.3 Problem Rescoping

Due to the timeline of the project and the scale of the Kaggle competition, the scope of this project for this course was updated. Since, the course is on Deep Learning, we focused our efforts on developing strong models for the chart classifier, and the origin detector/bounding box models. Due to this, in the stages after the deep learning models we only focus on charts that are classified as vertical bar. We will expand efforts to the other chart types for the Kaggle competition. Additionally, the stretch goal of creating an AI model to text summarize the data was removed from the scope.

2 Approach

2.1 Bounding Boxer & Origin Detector

One of the challenges of extracting data from graphs is identifying the important information, such as the data region where the bars or lines are located. By locating this region, we can remove any unnecessary text and titles to better detect the height of the bars. Additionally, determining the exact location of the axis lines is crucial for accurately determining the scale and location of the tick marks.
To address these challenges, we use two CNN models in the first stage of our pipeline. The first model is trained to predict the bounding box of the data region, which includes the x and y coordinates of one vertex, as well as the width and height of the box.
The second model is used specifically to detect the origin of the graph, which is necessary for determining the graph scale and tick marks. While the bounding box model may not require precise origin coordinates, we found that training a separate CNN model to predict the x and y coordinates of the origin resulted in more accurate predictions.
Before passing an image to the CNN, we applied some pre-processing steps. Firstly, we re-size the image to be of size 256x256. To preserve the aspect ratio, the image was re-sized until either the width or the height

was 256 pixels, and then zero padding was added to the other dimension. The next step was to de-noise the image. To this end, we applied some traditional image processing techniques, such as Canny Edge Detection. The first stage is to convolve the image with a Gaussian filter, i.e. apply a Gaussian blur. This smooths the image and reduces noise. Edges are then detected via calculating the gradient of the image intensity. Edges are then thinned to a single pixel width by suppressing all the gradient values except the local maximum. Finally, two thresholds are specified. "Weak" edges with gradient intensity less than the lower threshold are eliminated, and edges with gradient intensity higher than the upper threshold are classified as "strong" edges and retained. Edges with gradient intensity between the two thresholds are retained if they are connected to a strong edge.
The image was then passed to the CNN models. The architecture of the CNNs can be found in 2. The Swish activation function was used for hidden layers, and these are both regression tasks, a linear activation function was used in the output layers. We used a Mean Squared Error loss metric, and trained for 30 epochs.
Figure 1 shows a sample image with the predicted bounding box on top of it. Since, the bounding box model also predicts the origin, one may question the need for the origin detector model. The origin prediction from the bounding box model, possibly due to it trying to predict 4 values instead of 2, is not very accurate. The tick detection algorithm is highly dependent on a strong origin prediction and thus we trained a separate model only on the origin, which as see in Figure 2 performs well.

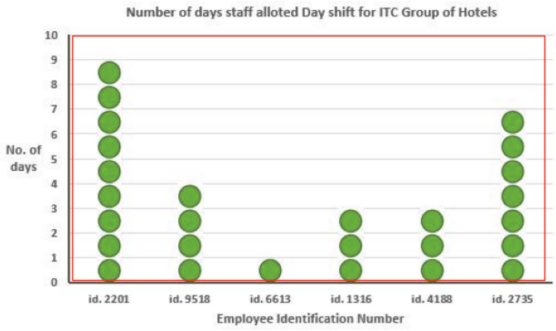

Figure 1: Predicted Bounding Box on a Bar Chart

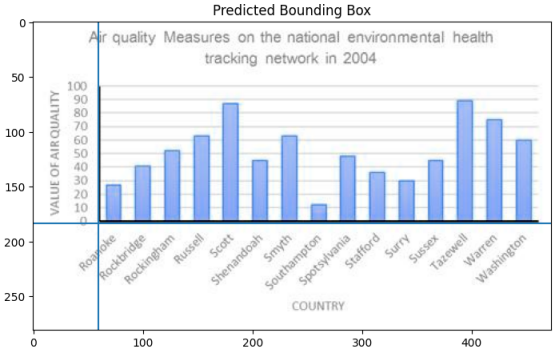

Figure 2: Lines Intersecting at the Predicted Origin

2.2 Chart Classifier

This is the other deep learning model in the pipeline. The role of this model is to take in a chart and classify it into one of the five categories (vertical bar, horizontal bar, dot plot, scatter plot, line graph). The inputs to the model are an image that is cropped to the bounding box around the chart. i.e. all the text, axes information and labels are removed. The cropping in training is done using the ground truth bounding box, but for test it uses the generated bounding box outputted by the model described in Section 2.1.

For the Kaggle competition, to score any points the chart needs to be classified correctly thus several models were experimented with to find the best performing one. The architecture of the three models is specified in 2 along with the training and validation accuracies. Softmax was used for the output layer activation function.

From the results we can see that the performance of all three models was superb, but also similar to each other. We picked the wider model since it had the best performance. The output of the model is a label indicating which one of the five chart types it is.

2.3 Text Recognition

This phase of the pipeline, referred to as "text recognition", involves identifying and interpreting the various text boxes on the chart image.

Optical Character Recognition (OCR), is a well-studied field concerned with the conversion of images of typed, printed or handwritten text into machine-encoded text. Traditionally, OCR involved complex algorithms and manual feature engineering which did not generalise well to characters in varying conditions, such as different fonts, sizes, orientations, and backgrounds. However, the application of deep learning techniques has significantly improved the state-of-the-art performance, learning feature representations directly from the raw image.
In our pipeline, we have employed two famous pre-trained models. The first model, known as DBNet[2], performs image segmentation, detecting text within an image and placing a bounding box around it.

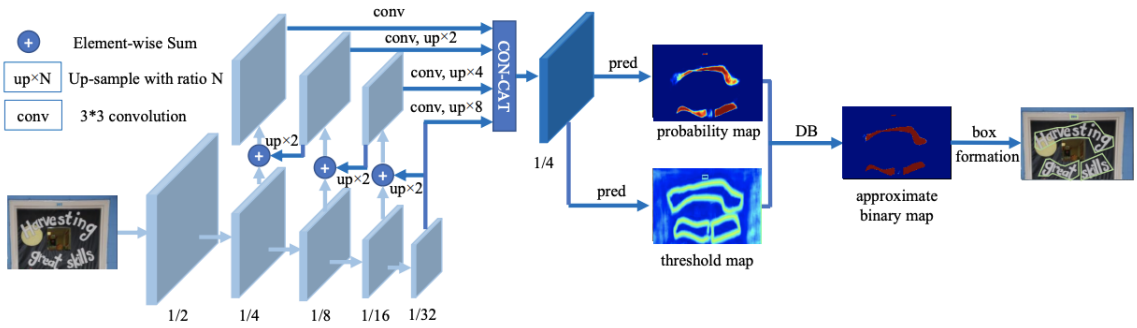

Figure 3: The architecture of the DBNet model. Figure taken from [2].

DBNet is a modified version of the ResNet architecture, which takes an input image and generates a segment "heatmap" which indicates areas of the image that may contain text. Traditionally, this heatmap would then be binarized to characterize the image segments. The novel proposal of DBNet is a differentiable binarization technique which allows the binarization section of the network to be trained simultaneously with the feature extraction for optimal performance.
The sections of the image contained within the boxes output by DBNet can can then be passed to any text recognition algorithm such as the famous Tesseract engine. For our purpose, we utilised another deep learning model called ABINet[1].
At a high level, the ABINet model works by first passing an input image through a CNN to extract a set of feature maps. The feature maps are then processed by a bidirectional RNN that iteratively refines the text recognition results through multiple rounds of attention-based decoding.
Isolating the text using DBNet before passing to the text recognition can help with performance. However, since the the text on the chart images can be slanted, or even vertical, correcting the orientation of the boxes output by DBNet before passing to ABINet will allow for even more reliable text-recognition performance.

2.4 Tick Extractor

At this stage, we successfully obtained bounding boxes encompassing all text items within the image. However, our objective was to identify only the elements that are pertinent to creating ticks on the graph, specifically those in close proximity to the axes. Defining "closeness" posed a challenge as we sought to avoid arbitrary thresholds that may yield unreliable results. We aimed to develop a robust heuristic that effectively eliminated bounding boxes associated with irrelevant items, focusing exclusively on the x-axis and y-axis labels.

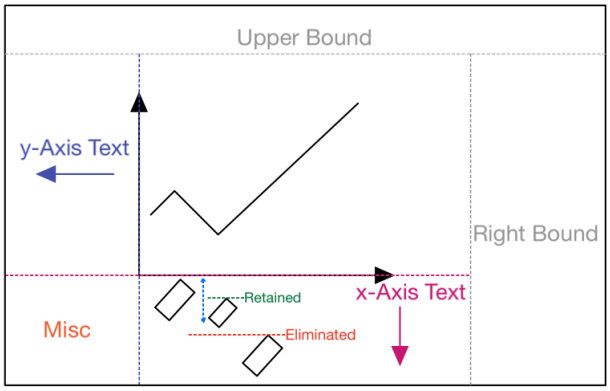

Figure 4: A diagram showing the procedure for attributing text boxes to the different axes.

Firstly, we leveraged the bounding box of the graph to place an "upper bound" which eliminated the graph title, and "right" bound which eliminated extraneous text on the right of the graph (here we made the assumption that the graphs only had left-axis information, though this bound could be removed if two y-axes were detected). Next, we used the predicted origin coordinates to denote text with x coordinates less than (to the left of) the origin x coordinate as "y-axis text", and text with y coordinates more than (below the) the origin y coordinate as "x-axis text". Note that for images, x coordinates are measured as increasing from left to right, while y coordinates are measured as increasing from top to bottom. Text which fell into the "miscellaneous" section was attributed to the y-axis if "started" above the origin, and to the x-axis if it "started" to the right of the origin", to handle slanted text which went beyond the origin.

Given that axis titles and legends also obeyed these constraints, and under the assumption that tick labels are placed at an equal distance from the axis, we located the closest text box to the axis, and eliminated text boxes whose closest vertex did not fall within the range of the closest textbox.

With the refined selection of relevant text boxes, our next objective was to create a tick mark on the axes for each box. Text boxes were identified as either flat or slanted using the vertex coordinates. Then, a tick was placed according to the following table:

Orientation	**Axis**	**Tick Coordinates**
Flat	x-axis	$(\bar{x}, O_y)$
Flat	y-axis	$(O_x, \bar{y})$
Slanted	x-axis	$(x[\text{argmin } y], O_y)$
Slanted	y-axis	$(O_x, y[\text{argmax } x])$

Table 1: Tick coordinates for axes with different orientations.

Where O_x denotes the x coordinate of the origin, $\bar{x}$ is the mean x coordinate of the textbox, and $x[\text{argmin } y]$ is the x coordinate corresponding to the vertex with the minimum y coordinate.
This extensive manual processing underscores the importance of identifying relevant information from the model's outputs. This section exemplifies the intricate steps of post-processing and refining the model's output to derive meaningful insights. Extracting pertinent information contributes to the overall success and value of the project, emphasizing the need for thoughtful analysis, meticulous data manipulation, and informed decision-making beyond the initial modeling phase.

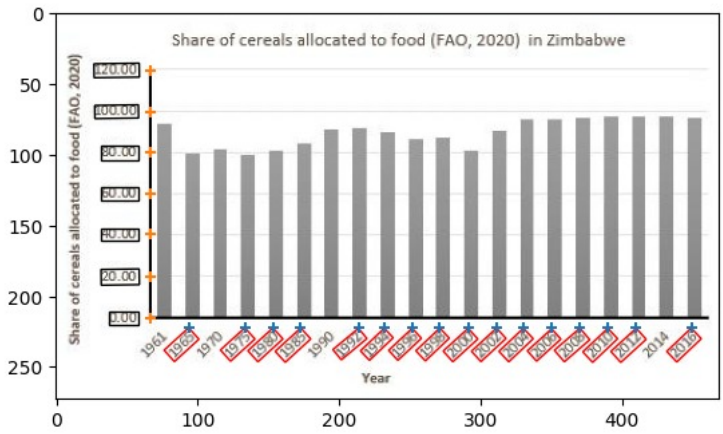

Figure 5: Ticks detected by our algorithm plotted on the original image. Note that the text recognition model does not recognise all text on the x-axis, and some inaccuracy in our origin prediction has led to x-axis ticks being predicted slightly below the axis.

2.5 Edge Detector

The goal of this model is to finally extract the numeric data behind the graphs. For this case, and our pursuit of using manual methods, we chose to focus only on vertical bar charts. The high level idea is to utilized edge detection to find the locations of the tops of each of the bars. From there we can use the coordinates of those points and the scale of the y-axis to estimate the numeric value. The necessary inputs for this model to run are the locations of all the tick marks on the x-axis and at least 2 ticks on the y-axis , the bounding box for the chart and of course the image. The algorithm for this is as follows.

- **Preprocessing:** Unlike CNN's a few preprocessing steps are required. The first is in addition to cropping the original image to the bounding box area, clipping the top and bottom of the box to ensure to axes are not in the image. In order to conduct the next steps effectively the image is grayscaled and binarized into black and white pixels only. Otsu's method is used to find the threshold for this. Finally, the image pixels are inverted. The output after preprocessing can be visualized in Figure 6.
- **Gradient:** A gradient is then taken vertically across the whole image, starting from the bottom of the chart and moving toward the top. This will result in most pixels having a 0 value, but anywhere the pixels transition from white to black there will be a 1 - these represent the edges.
- **Edge Detector:** To find the edges, first we take the coordinates of the x-axis ticks. In those columns +/- 2, we find the argmax, which will return the y-coordinates for the locations of the 1's, or edges, in the chart array.
- **Coordinate to Value:** The final step is to convert the y-coordinates of the edges to their respective numeric values. To do this we use the extracted y-axis tick values and their associated labels to create the scale of the chart. The coordinate is converted on the scale to get the value. Details on this calculation are found in Subsection 2.5

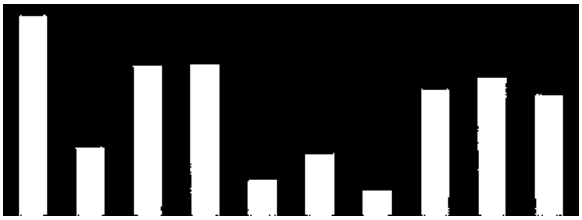

Figure 6: Ouput after preprocessing for edge detection. Otsu's method automatically calculates the threshold of an image by maximizing the between-class variance of an image's grayscale intensities.

Creating Y Scale

Inconsistency in the text recognition and tick extraction led to being creative for making the y-axis scale. The necessary information to convert the predicted y pixel coordinates of the bars into data values are: the y pixel coordinates for two ticks on the y-axis and their respective data value (text from the label next to the tick). To get these, we sort the extracted y ticks on location order, then take the lowest and highest pixel value, where the associated text can be converted into a float data type (sometimes it reads numbers as letters, if so we skip these). Now, we have the four values:

- High: the greater of the two data values associated from the ticks
- Low: the lesser of the two data values associated from the ticks
- high_y_coord: the y coordinate corresponding to High
- low_y_coord: the y coordinate corresponding to Low

The following equation is then used to convert the predicted y-coordinate to a numeric value:

$$pred\_y\_data\_value = ((High - Low) * (1 - \frac{(pred\_y\_coord - high\_y\_coord)}{(low\_y\_coord - high\_y\_coord)})) + Low \quad (1)$$

The red lines on Figure 7 show the predicted locations of the tops of the bars after this model is run.

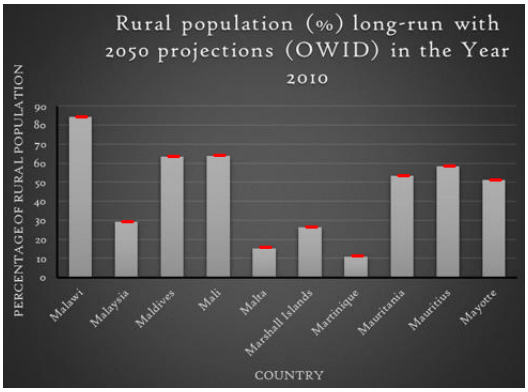

Figure 7: Predicted location of the top of bars

This method works, well but could be improved. In particular low contrast images, for example where the bars are colored yellow and the background is white, after thresholding is applied the graph vanishes because the RGB values are too similar. The same happens for dark bars on a dark background. To work around this, the edge detector would normally predict all 0's. Since, this is unlikely to be reality, if this happens we set the predicted values to be in the middle of the chart to minimize expected error.
The other method that was tested was using a Harris corner detector to find all the corners in the chart. This resulted in noisy results such as multiple corners in the same locations, or corners at locations where

gridlines intersected the sides of bars. Even though, when this method worked, the NRMSE was extremely low, the inconsistency in results from removing the extra corners deemed it an unrealistic method to pursue for now.

2.6 Connected Pipeline

Figure 8 is a good illustration of the fully connected pipeline that takes in a image and outputs the predictions of the data values for the bars in the chart. We implemented checks along the way in the pipeline to ensure data consistency. Since the text recognition and thus tick extraction did not always perform with perfection (i.e. it did not always find the locations of all the x axis ticks), we only ran the edge detector module of the pipeline with proper data (charts where we confirmed the identification of all x-axis tick marks). For the Kaggle competition, we would of course not be able to do this and instead would have to create a check to make sure the number of extracted ticks matches the expected count. To do this we could work with the spacing of known ticks or vertical edges of bars.

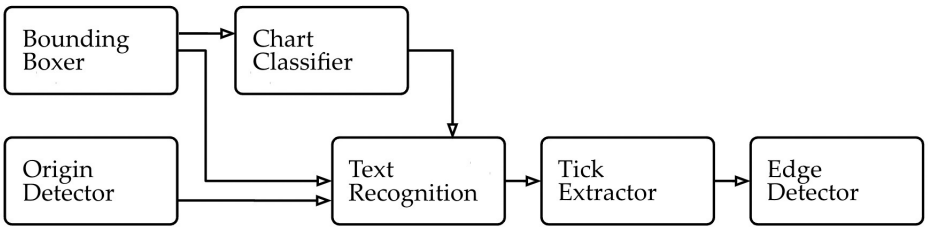

Figure 8: Connected pipeline representing the interactions between the defined modules

3 Results

The results of the individual modules as well as the complete pipeline are summarized below. Some relevant remarks regarding how results were evaluated:

- The bounding box, origin detection and chart classifier are the deep learning models we built. These were trained on the complete training test with a 20% validation split. Test accuracies were also measured.
- The text extraction model was pretrained so it was only evaluated on the test set.
- The tick mark extractor and edge detection model are manual models that do not learn. These models were developed on the training data but only run with the test set. The edge detector was only run on vertical bar charts.
- Here we define Misprediction Error for ticks as the total number of ticks we did not detect. For text the Misprediction Error uses the Levenshtein distance to compare how similar two strings are.
- Due to the inconsistencies of the training JSON data, it was hard to guarantee data was available and fully automate the evaluation of the edge detector. The scale for each graph is unique and extracting that efficiently was complex. Thus, the individual training and test results for the edge detector are from a random sample of 100 images in each segment. However, the results from the overall pipeline use the extracted scale from the tick extraction model.
- Normalized RMSE (NRMSE) is used for the Edge Detector because the scale of each chart is independent from another and the ranges can be broad. Normalizing the RMSE with the range of potential y values for each chart eliminates this.
- Since each individual module was trained independently, only the test set was applied and evaluated on the final pipeline, the outputs of each module were properly connected to the input of the next.

Model	Name	Metric	Training Value	Validation Value	Test Value
1	Bounding Boxer	MSE	2.9795	3.0323	3.0069
2	Origin Detector	MSE	1.9345	2.0498	1.9993
3	Chart Classifier	Accuracy	0.9997	0.9917	0.9626
4	Text Recognition	Misprediction Rate	N/A	N/A	50.06%
5	Tick Extractor	Misprediction Rate	N/A	N/A	20.9%
6	Edge Detector	NRMSE	0.1390/0.0163*	N/A	0.1842/0.01743*
Overall	Full Pipeline	NRMSE	N/A	N/A	**31.2875/0.2190**

Table 2: Performance of individual submodules and overall pipeline

* The preprocessing for the Edge Detector model was relatively robust, but there are a substantial amount of images that lead to poor performance. When incorporating the images where we guess the middle of the chart after preprocessing eliminates the bars, the NRMSE is worse than when the heights are estimated well. Thus, two NRMSE's are reported, the first on the full random sample of 100 charts, and the second on all charts where the NMRSE is less than 0.1. Clearly, the performance is much better when preprocessing keeps the bars. This is a challenge we are currently working on solving.

The MSE of the origin detector and bounding box model support the need to having both models. We can see the MSE for origin detector is consistently smaller, however the bounding box is still needed to properly crop images. Further, the other deep learning model has extremely high performance. The chart classifier having a 96% accuracy on the test set sets the rest of the pipeline up to work on the correct images. For text recognition, the results are quite poor - about half the strings are incorrectly extracted, however this can be explained. Firstly, the primary goal was to use the text boxes to find the ticks, not only extract text, thus often text boxes that were still relevant were eliminated because they created noise for the tick marks. Secondly, we employed a pre-trained model for this stage, but the chart images are likely from a specific domain of images that the original model is not trained on. The resolution of the text can be quite poor and have small font size explaining the poor results.

The edge detector overall has moderate performance. The NRMSE is quite large when the full pipeline was run on the whole test set. However, when we remove outliers the results are quite promising. The NRMSE without outliers is 0.2190 (compared to 31.2875 with outliers). This again falls into needing to improve the preprocessing and text extraction. Sometimes decimal points are not read in correctly so it assumes the value "80.00" is "8000", which when calculating NRMSE is quite detrimental. Even standalone the edge detector works well as indicated by the low NRMSE for training and test. Some more refining on the images that was mentioned before should lead to results of this consistency.

4 Lessons Learned

The first important lesson learned from working on this project was that the scope of the project was way broader than we expected: we first started wanting to build a model that could turn all types of charts into tabular data, but once we started working with the dataset, we realized there were several details that were specific to each one of these different plots, and there was a lot of data processing that was not directly transferable to all of them. That ended up reducing our scope only to vertical bar charts.

That did not turn out to be our most significant challenge though, as once we started structuring the pipeline for our model, we did not anticipate how intensive the memory requirements would be. Trying to train CNN models on thousands of images pushed our RAM resources to the limit. Indeed, passing processed images between different stages of the pipeline necessitated deleting old images from memory once they had been utilised. Furthermore, it was helpful to compress the image sizes through a variety of techniques. Starting the pipeline with our **Bounding Boxer** model, significantly reduced the amount of data going through the pipeline, as it converted all images to gray scale, reducing the channels of the model from 3 to 1, and setting

important reference points (such as the origin of the plot) that where leveraged later on in subsequent models.

Additionally, there were specific challenges to overcome on each one of our separate models, which will be covered in the following subsections.

4.1 Bounding Boxer

In order to extract only the relevant information from the graph, a crucial step involved **eliminating unnecessary elements** that do not contribute to the process. Initially, we attempted to train a CNN model using the raw images and available annotations; however, the model performed poorly when tested on new images. One of the primary challenges arose from the inadequate color scheme of the graph, which made it arduous to discern between strong and weak edges within the image. Additionally, the bounding box predictions occasionally initiated from the first vertical bar or lacked sufficient height to encompass the entire graph. Furthermore, the presence of minor grids within the graph, mistakenly identified as horizontal and vertical lines, posed difficulties for the model in accurately bounding the image.

Consequently, to address these issues, it became necessary to incorporate a **denoising filter** for the image. This approach enabled us to reduce the number of channels from 3 to 1, utilizing only the black and white version of the image. This adjustment significantly contributed to the more **efficient utilization of memory resources**, which we elaborated upon in the broader lessons learned.

To ensure suitability for the CNN model, the images required **padding** to attain the appropriate dimensions. This process entailed **resizing** both the image itself and the accompanying annotations to align them appropriately. The challenge was then to take the model output and re-calculate the coordinates so as to refer to the original image dimensions when moving forward.

4.2 Chart Classifier

This was one of the first models we covered, and we quickly understood how challenging it was to handle different pieces of data together. The most important lesson here was to **leverage the use of dictionaries based on the name of the image and annotation files,** and not to rely on the order and availability of the files within our folder structure. When we first started training this model we were obtaining 49% accuracy, and the low performance of the model was mainly due a mislabeling process of the plots on our side. We were able to fix it quickly by referencing the unique identifies for each one of the images.

4.3 Tick Extractor

For this model it was key to be able to **properly understand the geometry within each one of the images,** as we wanted to be very precise on what text we wanted to focus on: axis labels. Based on the clear image set by our Bounding Boxer model, we could easily understand what text was closer to the axis, and based on the orientation of the text, we could understand the proper position of the ticks.

4.4 Edge Detector

Our final model also faced some technical challenges, specially with the plots with low contrast, as when we converted these images to black and white, some hue combinations did not allow us to detect the bars' edges, as white and yellow. We solved this by using gray scale, and detecting average changes on the pixel values.

5 Future Work

In the short term, we would like to replicate this process to be able to address all chart types, starting by horizontal bar charts, as we have a lot of transferable tools we can leverage with minor changes based on the axis we use as reference. After that, we could use similar approaches for line plots, dot plots and scatter

plots, using some of the geometry-based guidelines we already developed using the origin as a reference point.

Another relevant next step would include optimal prompt design based on the tabular data obtained from our model. this way, we would be able to feed into an LLM, to finally obtain a more detailed description of the plot, which together with other tools can finally give access to educational materials for STEM fields to millions of students dealing with disabilities, democratizing the access to data visualizations.

Bibliography

[1] Shancheng Fang, Hongtao Xie, Yuxin Wang, Zhendong Mao, and Yongdong Zhang. Read like humans: Autonomous, bidirectional and iterative language modeling for scene text recognition, 2021.

[2] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, and Xiang Bai. Real-time scene text detection with differentiable binarization, 2019.

Appendices

1 Links to Colab

- Complete Pipeline: https://colab.research.google.com/drive/1ksyikWhTxe4lJZTYEYnmy_WGtikKRVNQ?usp=sharing
- Bounding Box Model: https://colab.research.google.com/drive/1EjzcqxqtwLay1tINly30dRgv1D5HFcQN?usp=share_link
- Origin Detector: https://colab.research.google.com/drive/1mtotBh_OPo2QKGnZV-qUdj31Ghf9W3TV?usp=share_link
- Chart Classifier: https://colab.research.google.com/drive/1ZkhRluYgrubo8pc7gdFth1oCU-k8uAbu?usp=sharing
- Text Recognition & Tick Extraction: https://colab.research.google.com/drive/1HDY-jRqEXwaOmolDTF5c5trQx
usp=share_link
- Edge Detection: https://colab.research.google.com/drive/1arSGVJ9NS6RjSRWpO8GfJbrO94w650qD?usp=share_link
- Project Folder (contains data needed to run): https://drive.google.com/drive/folders/1OAJu0yvk264luMfEMCM2
usp=sharing

2 Model Architectures

Bounding Boxer and Origin Detector Model Architectures
Both models ran for 30 epochs.

- **Bounding Boxer**:
  - Convolutional Layer with 32 Filters
  - Max Pooling
  - Convolutional Layer with 64 Filters
  - Max Pooling
  - Convolutional Layer with 128 Filters
  - Max Pooling
  - 25% Dropout Layer
  - Dense Layer with 256 Neurons
  - 50% Dropout Layer
  - Output Dense Layer with 4 Neurons (Origin x, Origin y, width, height)
- **Origin Detector**:
  - Convolutional Layer with 32 Filters
  - Max Pooling
  - Convolutional Layer with 64 Filters
  - Max Pooling
  - Convolutional Layer with 128 Filters
  - Max Pooling
  - 25% Dropout Layer
  - Dense Layer with 256 Neurons
  - 50% Dropout Layer

  - Output Dense Layer with 2 Neurons (Origin x, Origin y)

Chart Type Classifier Model Architectures
All models were run for 30 epochs.

- **Base Model**:
  - 3 Convolution Layers with 16 Filters
  - 1 Dense Layer with 256 Neurons
  - Training Accuracy: 0.9992
  - Validation Accuracy: 0.9950
- **Wider Model**:
  - 3 Convolution Layers with 32 Filters
  - 1 Dense Layer with 512 Neurons
  - Training Accuracy: 0.9994
  - Validation Accuracy: 0.9860
- **Deeper Model**
  - 3 Convolution Layers with 16 Filters
  - 3 Dense Layer with 256 Neurons
  - Training Accuracy: 0.9998
  - Validation Accuracy: 0.9947

Transfer learning models using both ResNet and VCG-19 architectures were also tested but not found to improve performance.