CSE 673 Final Project

Project 5: ChartOCR

Parth Rajput
parthket@buffalo.edu
50368229

Tyler Perison
twperiso@buffalo.edu
50236786

Maruf Shaikh
marufbas@buffalo.edu
50388962

December 2021

Abstract

Charts are capable of conveying thousands of lines of information with just one picture. The capability of humans to understand things better when they have a visualization in front of their eyes is what makes charts so interesting. But many times, we would want to look at the actual figures and numbers, which, most often then not, is not easily available. ChartOCR is a way of extracting this information back from a given chart, by using Computer Vision techniques. In this project, which is a part of the CSE 673 course, we tried to implement this with the help of Box detector and Point Detectors, using CenterNet.

1 Introduction

Our main objective is to reproduce the methodology implemented by DeepBlue Technology[1], in their submission for the competition on Harvesting Raw Tables from Infographics hosted by Chart-Infograohics Organizers. Keeping in mind the complexity of the problem, the problem statement is subdivided into 6 main tasks, which are Chart Image Classification (Task 1), Text Detection and Recognition (Task 2), Text Role Classification (Task 3), Axis Analysis (Task 4), Legend Analysis (Task 5), Plot Element Detection and Classification (Task 6.a), Data Extraction (Task 6.b). Another task(Task 7) is pipelining the whole process. In [1], the Zhipeng Luo et. al. had explained the whole process from task 1 to 6 as shown in figure 1 in detail. Our main focus area is task 6 of the challenge, i.e. Plot element detections, classifications and data extractions. We have used two datasets for the same, Adobe Synth and UB PMC.

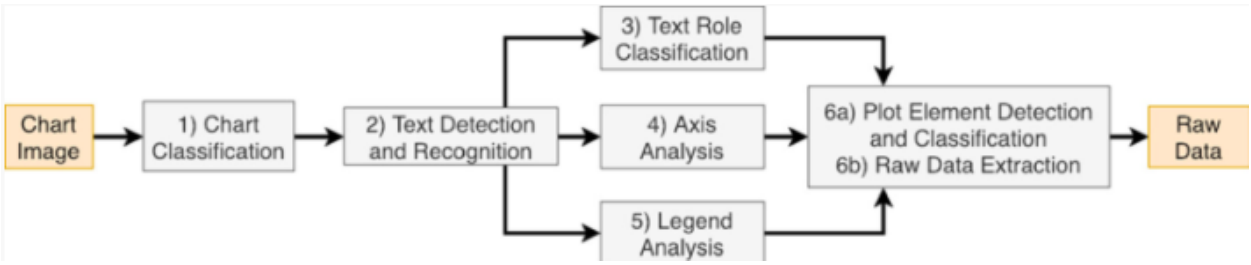

Figure 1. Pipeline of ChartOCR

Charts are of various types, and each type will need a different approach. In this project, we focused only on 4 types of charts, i.e. Bar Charts, Box Plots, Scatter Plots and Line Plots. Now, if we look at the objects in each individual graph, there is a lot of difference in the type of objects. For example, bar charts consist of rectangular bars for various types(mentioned in legend), whereas scatter plots consist of multiple keypoints. Line charts can be considered as a series of keypoints, where all of the key points are joined together to form a line. When it comes to box plots, it is a combination of boxes and keypoints. In order to detect these key points and boxes, the authors had used 3 different Centernet models for different types of charts. For bar charts, we have used an out of the box Centernet model which does anchor free object detections. For box, scatter and line charts, we have used Centernet with DLA-34 as a backbone. The details about these models will be discussed in the next section. After getting the keypoint or the bounding boxes, our next target was to generate meaningful information through the detections. It is this particular section that is heavily dependent on the outputs of previous tasks, since we will need text extractions and other data to gather information about the detected points. Firstly, we detect the labels of the points/bar using a heuristic method of Histogram of gradients, and then map the detections with the axes using distances.

2 Background

Extraction of raw data from chart images is a very complicated process, mainly because of the existence of a variety of charts and no uniformity between them. Thus, there was a need for a pipelined procedure for this. In [4], the authors have proposed a very novel framework which focuses on providing a pipeline for the same(figure 1). Now, to make things simpler, our focus was restricted to only four types of charts, namely bar, box, line and scatter plots. Again, in these, there was a different model for each of the charts.

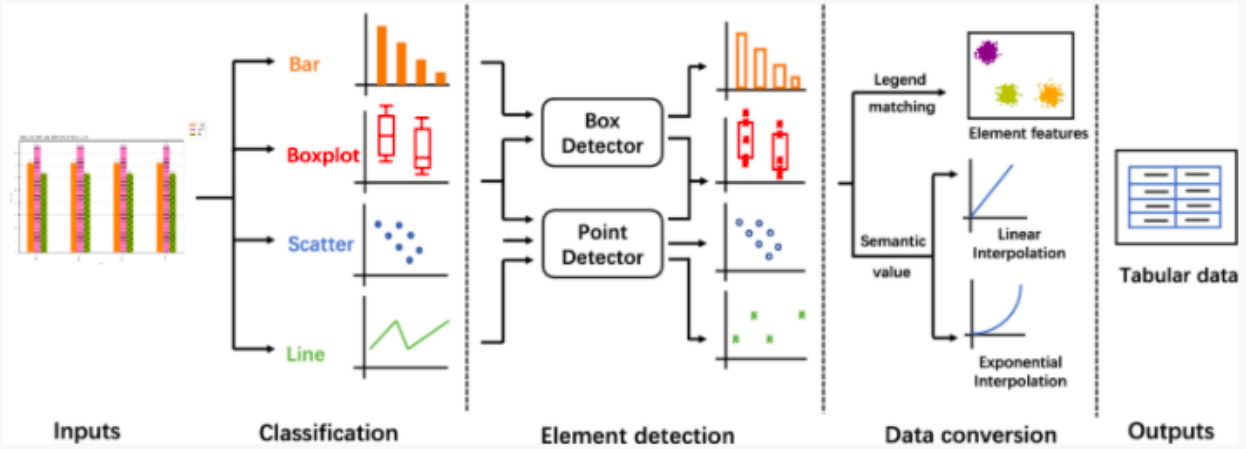

Figure 2. Detection Models for Different Charts[9]

In figure 2, we can see a concise pipeline for our target chart types. Our implementation has one difference from this, the box plots are fed only to the point detectors and we detect 5 points through that, rather than combining outputs of two different models. So, our project is mainly divided into two parts, the model training and then mapping the detections, which is more or less based on heuristic based approaches. Both of the approaches have been mentioned in [1], and we are making use of similar approaches, and if there are any changes in that, we are mentioning them in the next section.

Object detections are a classic problem in the field of Computer Vision. We have a variety of object detectors like RCNN, Fast RCNN, YOLO, etc. All of these were making use of either ROI pooling or the anchor boxes. Corner Net[5] gave us a solution to do the same thing without using anchor boxes, thus saving some computation time for us. CenterNet is a progress made up on CornerNet, and adds a center

pooling aspect to it. By using the center points, we can do object detection as keypoint triplets and that too with one stage detection.

3 Methodology

We tried to closely follow the methodologies mentioned in [1]. It can be listed down into several aspects which are as follows:

1. **Dataset**

The two datasets that we used were the Adobe synthetic dataset and the UB PMC dataset. The synthetic dataset consisted of many types of charts that fit under the types of charts that we used. These were labeled as hBox, hGroup, hStack, line, scatter, vBox, vGroup, and vStack. For the UB PMC dataset, they were labeled as horizontal_bar, line, scatter, vertical_box, and vertical_bar. These were all sorted into more basic types, bar, box, scatter, and line in our data loaders. Each basic type of chart has a list of visual elements that consists of points, boxes, or both. In the UB PMC dataset, there are many charts that are missing the task 6 outputs, so they were cleaned out. In the synthetic dataset, some of the points from the task 6 outputs extend past the chart, so they were clipped in the data loaders.

2. **Data Loaders**

In our data loaders, we read through each image and its task 6 expected outputs for the visual elements. Since we are working with two different datasets, and we had observed that the annotations for both of them were quite similar, we merged them both into one, although train loaders will be different for them. We are taking the name of the dataset, the input image size and the output heatmap size as other arguments. Accordingly, we are reading the images, calculating the downsample ratio and calculating the required heatmaps depending on the dimensions given. Since annotations are of original size, we are multiplying everything with the respective downsample ratios, so that we get the perfect mapping of the images. After calculating the heatmaps, we are doing a Gaussian blur on the heatmaps, so as to create a highlighted effect on the heatmaps. This will propagate better learning for our models. Also, we are calculating the width/height indices of objects(in a flattened space), regression mask, etc as per the requirements of our implementation. We also added annotation outputs of previous tasks, so as to use them for HoG and mapping with axes.

3. **Box detection on Bar Charts**

Since bar charts consists of various boxes, we have used an object detection based model of Centernet. Centernet is an anchorless object detection method, which encompasses the theory from CornerNet. After using an Hourglass neural network for extracting the feature embeddings, Centernet uses a series of convolutions along with center pooling and cascade corner cooling to extract more information.Center pooling helps in focusing more on the center pixels, thus giving us centers with more recognizable patterns in the object. Cascade corner pooling is a methodology used in CornerNets, which helps in getting the corner points. We have used the model from [2] and had to make some changes to suit our use case. Also, the data loaders had to be adjusted in order to fit the loss calculations for this, which will be explained in a later section. The output of Centernet is a heatmap, which is basically a probabilistic value of a detection(center point) being present at a given point.

4. **Key point detections for Box Plots**

For box plots, as mentioned earlier, we had to detect multiple points(5 to be precise) for each of the box plot components present in the model. The model that we have used for this instance is CenterNet with DLA-34 as a backbone. Deep Layer Aggregation is a combination of Iterative Deep Aggregation and Hierarchical Deep Aggregation. It is preferred over ResNets for this use case because of ResNets being considered as shallow, i.e. the skip connections happen only between two subsequent blocks. And keypoints are very small points present in the images, so we cannot afford to lose information learned in the initial layers. DLA performs hierarchical aggregation, which means that initial layers contribute equally

to the final layer, also called the head. This serves as an advantage when we want to detect keypoints. In order to implement this model, we had to use the official DLA implementation, and used the feature embeddings(after chopping the last dense layers). These embedings were fed to the CenterNet module, and since there are 5 different classes in box plots(min, max, median, 1st quartile and third quartile), the output channels are also 5. So, we get 5 different heatmaps as output, each corresponding to a different point. Rest of the method is similar to that of the Box detector, except for the fact that we did not consider width and height for loss calculations since we were more concerned about center points, or the key points. Some other methodologies had indicated the use of an ensemble approach as well, i.e. using box point as well as key point detectors for box plots.

5. Key point detections for Line/Scatter Plots

Since line and scatter plots are a combination of points in a chart, we can directly use the same model mentioned above for box plots, but this time, the output heatmap will be only 1 channel. The thought process for using DLA as a backbone remains the same. Since all of the points can be considered as a single channel, we just need to predict one class instead of multi-class as we have in box plots.

6. Loss Calculations

Calculation of loss is the tricky part in this project. For calculations of detection heatmaps, we compare it with the heatmaps generated using the ground truth. We use Focal Loss for this comparison, which is a weighted Cross Entropy Loss. The reason we use focal loss here is that we wanted to give more weight to the predictions at the place where a detection is expected, since the detections here are just small points and a heatmap which is just black will still give lower loss.

For calculation of regression points(width and height), we are passing the width and height of each detection from the data loader. We are also keeping a track of indices of these detections(when the heatmap is flattened out), as well as a regression mask, which indicates the presence of an object. We combine all of this and calculate the loss using L1 Regression Loss.

One more important factor that we had to consider was the offset loss. Since heatmaps given as output by CenterNet are of downsampled size, calculation of center point will result in loss of position(since indices are supposed to be integers). Thus, there is a small offset created, which might be a problem as the training progresses. Thus, we save this offset and calculate offset loss using the same Regression L1 loss.

7. Mapping detections with Legends

After extracting the key points and bars, the next step is to map the detections to the legends. For this, we use the outputs from task 5 and task 2. This is done with the help of histogram of gradients. We first calculate the HoG features of the labels, and then calculate the same for the detected points. This is done using OpenCV's HoG compute function, and we crop the image according to the output of the heatmaps. But first, the heatmap has the effect of gaussian blur, which can be solved using a normal maxpool. After this, we get the coordinates, which are then scaled up as per the original image dimensions. After this, for every object, we crop that part out and detect the histogram of gradients. The features obtained by both of these are compared to determine which object belongs to which legend.

8. Mapping detections with Axes

Now, once we have the mapped legends and detections, the next step is to match the axes of the detections. The paper describes various divisions among the axes, depending on the type of chart/axes. We tried a plain simple Euclidean for this, since we were not able to generate proper heatmaps due to training issues, which will be explained later in experiments sections. We matched the closest x and y tick points from the output of task 4.

9. ICP and GM Improvements

As a part of improvements, we were suggested two intuitions. They are Iterative Closest Point and Graph Matching, which will serve loss functions to replace the loss functions used by the authors. Iterative Closest Point tries to match two point clouds by a series of comparison and rotations. Here, all the detected key points in a cloud can be considered as one cloud point and heatmap's key points can be considered as the other one. So, we just plug this in the training. One challenge that we thought of was that the initial cloud points for the model would be very sparse, and we needed a way to combat that, maybe by plugging it to an already trained model. For GM, we can use any of the tessellation models and compare it with the heatmap's tessellation model. We borrowed code from Richardos' ICP github repository to run ICP using the output coordinates from the predicted heatmaps as the source, and the actual expected coordinates from the data loaders as the target. The ICP algorithm returns a transformation matrix that maps the source to the target, and a list of distances from each point in the source to its new nearest point in the target. For GM, we used bipartite minimum weight full matching. This finds the best matching that matches all of the points in the source with unique points in the target if there are fewer points in the source, and matches all of the points in the target with unique points in the source if there are fewer points in the target. Using this matching, we gave larger weights to points that were close to their matched points, smaller weights to those farther away, and negative weights to points that were not matched to the target.

4 Experiments

1. Data Loader

Since we are using the PyTorch framework for this project, we make use of the template for the custom image datasets for the data loaders. For bars, the heatmaps generated are for the center points. For box plots, 5 heatmaps are generated, one for each type of classification that we need. For scatter and line plots, the heatmaps consist of all the points in the graphs. We saw that the data loaders were taking a lot of time, in order to generate the heatmaps and passing all the values. After some experimentation, we noticed that a majority of the time is taken due to the fact that we were loading the images from our drive, in order to save us some time in unzipping it every now and then. After making the changes, we observed that the time had reduced, but still considerable enough to slow down our training progress. Figure 3 shows how the heatmaps are generated. Please note that we are casting the heatmap on top of the original image, to give you a better idea of it. And different labels are colored differently, which is not the case for our actual data loader.

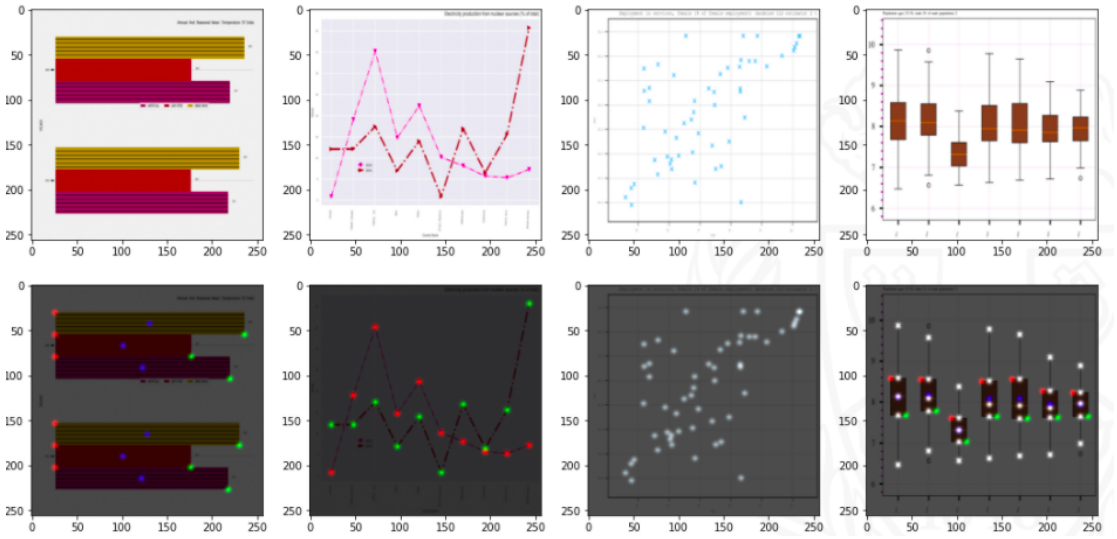

Figure 3. Data Loader Heatmaps

2. **Model Training**

Training the models is where we face most of the problems. The training was taking unusually longer time than we had expected, but we were able to see the results. Also, for bar charts, we are facing some issues, mostly due to some discrepancies in the data loader, which happened to due to some updations in the end. For models with DLA backbone(keypoints), it is running fine. Initially, when we used a larger learning rate, we saw that the model was giving black heatmaps in the initial epochs only, and after decreasing the learning rate by a considerable factor(0.0001), we saw learning(figure 4) but still the training time was painfully slow. Also, due to computational limitations, we were not able to use a larger batch size, otherwise we were getting out of cuda memory error. This was another blocker and we had to reduce the batch size to 3. This also meant that training process will be slower.

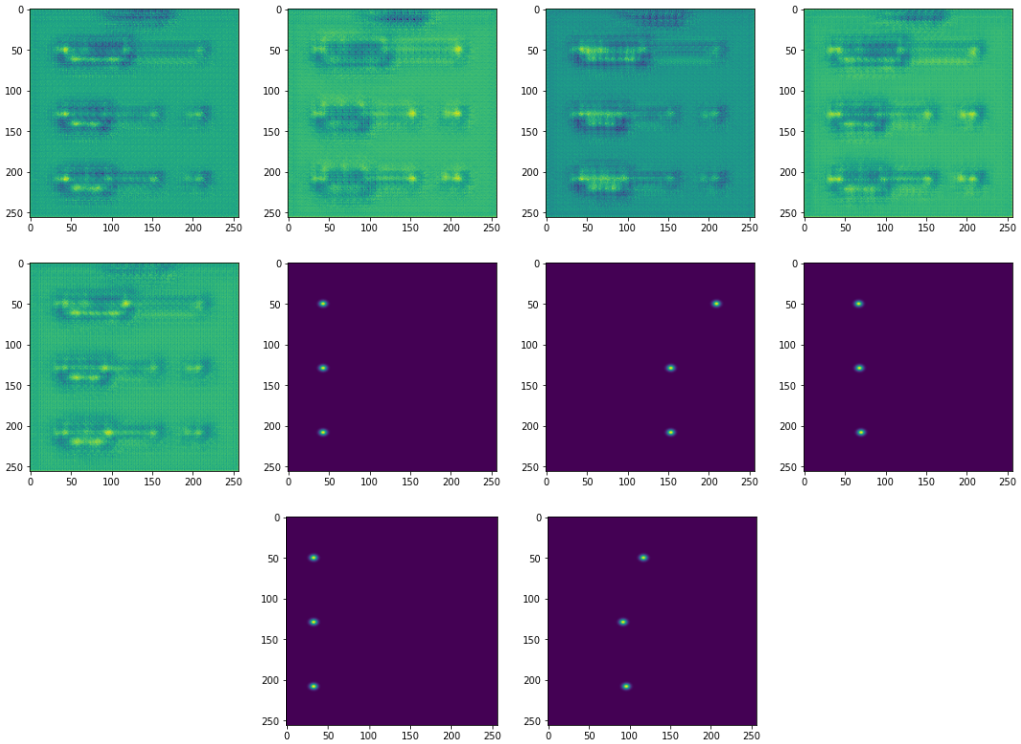

Figure 4. Output Heatmaps from our model(partially trained)

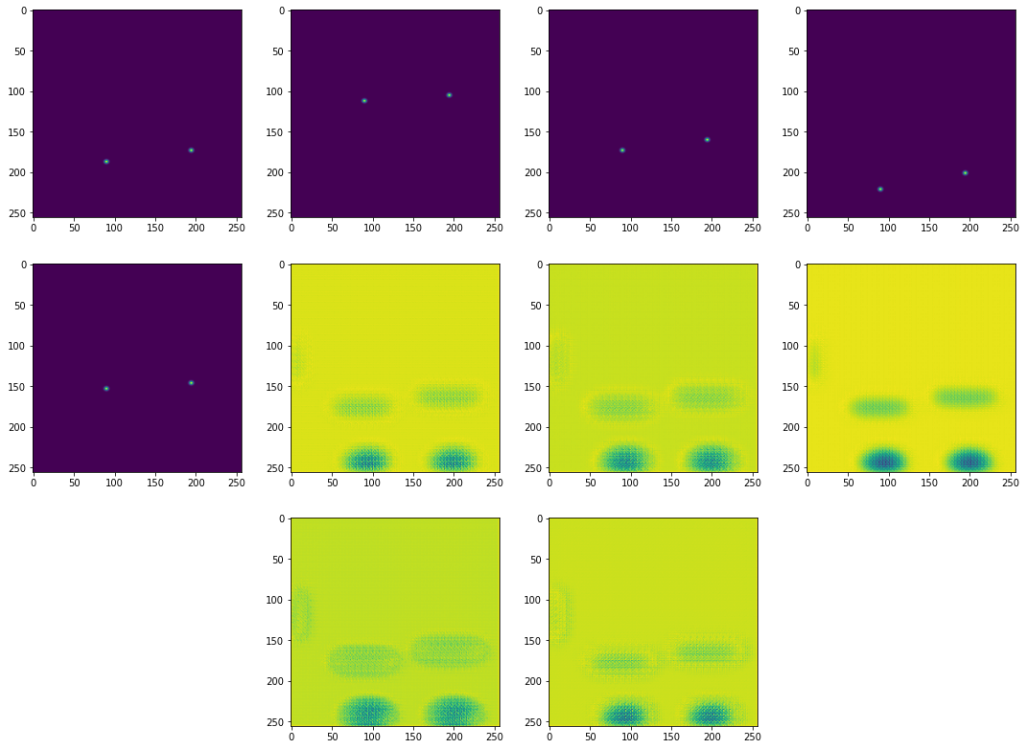

Figure : Training on UB PMC dataset

3. Mapping Legends and Axes

For this, we have rescaled the key points to actual image coordinates and then added some padding around it to accommodate the whole keypoint. Since our model was not trained, this was done using the data loaders heatmap. So, if we do get results from the training, this code should work with some minor tweaks. In order to remove Gaussian Blur, we use a normal MaxPool and maintain the dimension of the heatmap. This gives us just one single point per detection, which is what we need to map. Figure 5 shows how the detections would look like when cropped from the actual image. The labels are also cropped as per the image's annotations. After this, we take the HoG features of both key points and the labels, and map them accordingly. Even though the paper used K Means clustering, due to time constraints, we had to resort to normal distance based feature matching.

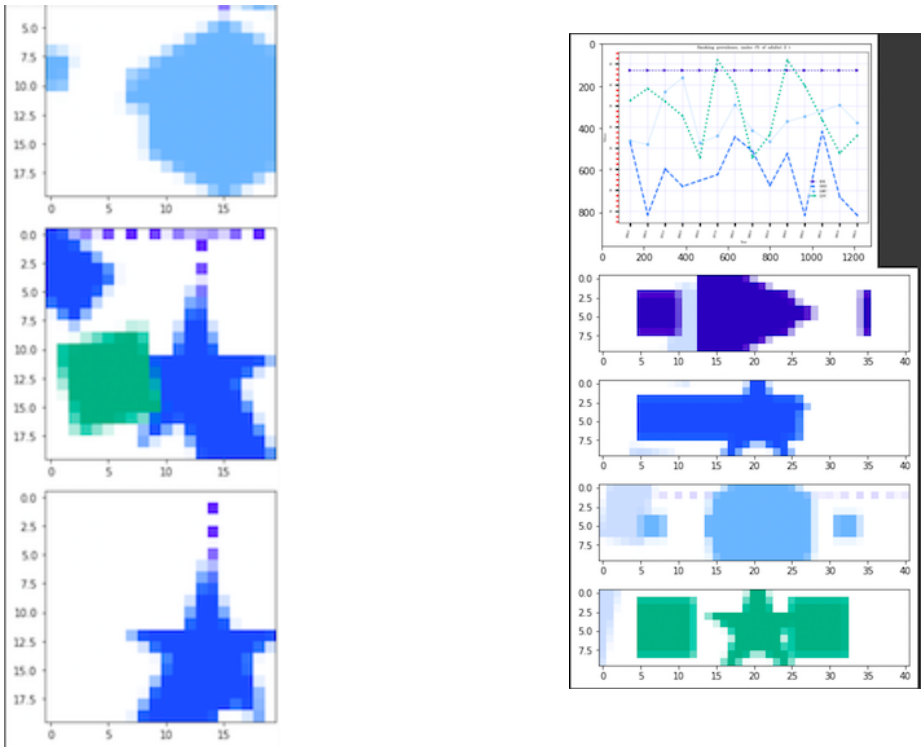

Figure 5: Legend and Key points cropped from Chart

5 Conclusion

In this paper, we discussed our project which is extraction of raw data from Chart images. We also discussed the various steps involved in this, the computer vision techniques that we had used, the issues that we have encountered, etc. Since training the data is taking a lot of time, we were not able to verify or achieve the final results with what we have set as the benchmark[1]. Also, figuring out the losses took a lot of time for us, since we had to run the code first on the ships dataset and figure out what all needs to be passed to the loss functions. Lately, we have noticed that we are facing some issues with the bar charts model, mostly due to some changes done in the data loader to accommodate some other task. We will try to rewind back and check where the issue is. For improvements, we will try to train the network for a longer period of time(or with better computational resources) and try to achieve the results that we had expected, i.e. heatmaps with distinct keypoints. From that, we can use the code that we have for legend matching and axes matching, which might need some more work, and generate some results. For ICP and GM, we already have our code ready, so if we get our model to train perfectly, we can plug those as loss functions and try to improve the performance. We need to check if training can become any faster than what we have.

6 References

[1]https://drive.google.com/file/d/1zkVhUzENo2M_ebpZ3B9SxMfTvjCsn1Ix/view
[2] https://github.com/JavisPeng/CenterNet-pytorch-detection-simple-tutorial
[3] richardos/icp: A Python implementation of the Iterative Closest Point algorithm (github.com)
[4] https://link.springer.com/chapter/10.1007/978-3-030-86549-8_37
[5] https://arxiv.org/abs/1904.08189
[6] https://github.com/Duankaiwen/CenterNet
[7] https://arxiv.org/pdf/1707.06484.pdf
[8] https://arxiv.org/pdf/1808.01244.pdf
[9] https://link.springer.com/chapter/10.1007/978-3-030-86549-8_37