CSE-673: Project-3

Karan Bali
Department of Computer Science
SUNY Buffalo
Buffalo, NY 14260-1660
kbali@buffalo.edu

December 2021

Abstract

This project was completed as a final project for the computer vision course CSE-673 at the SUNY, Buffalo. The project was mainly related to the Chart-OCR paper and repository, details of which can be found in references. It had five specific tasks. The first task was to read related papers like the original paper from Chart-OCR author, CornerNet paper, Hourglass network paper and finally present a presentation related to those papers. The second task was to establish a working repository and perform a feasibility study. The third task was to train the Chart-OCR model on the UB-PMC dataset. The fourth task was to train the same model on 5 disjoint datasets in an ensemble setting. The fifth task was to modularize the code and find suitable hyperparameters.

1 Introduction

Chart analysis is an important application area for the recent advances made in computer vision and deep learning. With an idea to explore new ideas related to chart analysis, some of the distinguished professors from SUNY, Buffalo, decided to conduct a competition related to it. The competition was termed as ChartInfographics. The first edition of the competition was held in 2019, followed by the second edition in 2020. A new Chart analysis dataset UB-PMC was released for the competition. The details related to the dataset and the competition can be found from the link provided in the references.

At the same time, a new paper related to chart analysis was released by some researchers from Microsoft, Penn State University, and USC. This was called "ChartOCR: Data Extraction from Charts Images via a Deep Hybrid Framework". You can find more details related to the paper at the link provided in the references. The Char-OCR paper promised good results for some of the similar tasks provided during the ChartInfo-2020 competition. Although, it was trained on a different ExcelChart400K dataset. Thus, one of the ideas was to

apply the approach from Chart-OCR on the UB-PMC dataset. The idea formed the basis of this project.

However, applying the Chart-OCR paper on the UB-PMC dataset looks easier than done. The original Chart-OCR repository was a huge, complex jigsaw puzzle. It had many open issues and practical challenges. Thus, the project-specific tasks were defined by the instructors of the course.

1.1 Project Details

Recently a team from Microsoft Research has also independently shown some traction on a new large-scale dataset. They focus only on three types of charts and propose a point-detector-based model having a backbone based on CornerNet with Hourglass Network. Your objective is to do a reproducibility study for Task-6 based on their method and provide an improvement on the same.

Papers to Read :

- ICPR 2020 - Competition on Harvesting Raw Tables from Infographics
- Chart Mining: A Survey of Methods for Automated Chart Analysis
- ChartOCR: Data Extraction from Charts Images via a Deep Hybrid Framework
- CornerNet: Detecting Objects as Paired Keypoints
- Stacked Hourglass Networks for Human Pose Estimation

Project Phases:

Phase 0: (10 Points)

- Read the 5 papers
- Meet with the instructor and confirm your understanding
- Discuss the dataset and benchmark
- Prepare for presentation in class

Phase 1: (20 Points)

- Setup open source Chart-OCR benchmark repo (Viability study - has multiple open issues)
- Benchmark pre-trained models(provided) on UB-PMC

Phase 2: (40 Points)Explore leveraging new data :

- Develop a training code from scratch based on the CornerNet repo.
- Train and Evaluate on UB-PMC(task6)

Phase 3: (20 Points)

- Can we train an ensemble of 5 models on disjoint splits of Excel400k to match/improve on results?

Phase 4: (10 Points)

- Hyperparameter search
- Single code repository with a modular structure to plug-n-play different detectors, on different datasets.

To summarize the above tasks, Phase-0 was related to the conceptual understanding of the Chart-OCR paper. Phase-1, Phase-2, and Phase-3 were related to actual experiments conducted in the course of this paper. Phase-4 was an add-on side task to Phase-1, Phase-2, and Phase-3.

2 Background

2.1 Stacked Hourglass Networks for Human Pose Estimation

This excerpt from the Hourglass paper explains the concept beautifully:

This conventional pipeline, however, has been greatly reshaped by convolutional neural networks (ConvNets), a main driver behind an explosive rise in performance across many computer vision tasks. Recent pose estimation systems have universally adopted ConvNets as their main building block, largely replacing hand-crafted features and graphical models; this strategy has yielded drastic improvements on standard benchmarks [1, 21, 22]. We continue along this trajectory and introduce a novel "stacked hourglass" network design for predicting human pose. The network captures and consoli- dates information across all scales of the image. We refer to the design as an hourglass based on our visualization of the steps of pooling and subsequent up- sampling used to get the final output of the network. Like many convolutional approaches that produce pixel-wise outputs, the hourglass network pools down to a very low resolution, then upsamples and combines features across multiple resolutions [15, 23]. On the other hand, the hourglass differs from prior designs primarily in its more symmetric topology. We expand on a single hourglass by consecutively placing multiple hourglass modules together end-to-end. This allows for repeated bottom-up, top-down in- ference across scales. In conjunction with the use of intermediate supervision, repeated bidirectional inference is critical to the network's final performance. The finnal network architecture achieves a significant improvement on the state- of-the-art for two standard pose estimation benchmarks (FLIC [1] and MPII Human Pose [21]). On MPII there is over a 2 percent average accuracy improvement across all joints, with as much as a 4-5 percent improvement on more difficult joints like the knees and ankles.

2.2 CornerNet: Detecting Objects as Paired Keypoints

This excerpt from the CornerNet paper explains the concept beautifully:

But the use of anchor boxes has two drawbacks. First, we typically need a very large set of anchor boxes, e.g. more than 40k in DSSD (Fu et al., 2017) and more than 100k in RetinaNet (Lin et al., 2017). This is because the detector is trained to classify whether each anchor box sufficiently overlaps with a ground truth box, and a large number of anchor boxes is needed to ensure sufficient overlap with most ground truth boxes. As a result, only a tiny fraction of anchor boxes will overlap with ground truth; this creates a huge imbalance between positive and negative anchor boxes and slows down training (Lin et al., 2017).

Second, the use of anchor boxes introduces many hyperparameters and design choices. These include how many boxes, what sizes, and what aspect ratios. Such choices have largely been made via ad-hoc heuristics, and can become even more complicated when combined with multiscale architectures where a single network makes separate predictions at multiple resolutions, with each scale using different features and its own set of anchor boxes (Liu et al., 2016; Fu et al., 2017; Lin et al., 2017). In this paper we introduce CornerNet, a new onestage approach to object detection that does away with anchor boxes. We detect an object as a pair of keypoints— the top-left corner and bottom-right corner of the bounding box. We use a single convolutional network to predict a heatmap for the top-left corners of all instances of the same object category, a heatmap for all bottomright corners, and an embedding vector for each detected corner. The embeddings serve to group a pair of corners that belong to the same object—the network is trained to predict similar embeddings for them. Our approach greatly simplifies the output of the network and eliminates the need for designing anchor boxes

Another novel component of CornerNet is corner pooling, a new type of pooling layer that helps a convolutional network better localize corners of bounding boxes. A corner of a bounding box is often outside the object—consider the case of a circle as well as the examples in Fig. 2. In such cases a corner cannot be localized based on local evidence. Instead, to determine whether there is a top-left corner at a pixel location, we need to look horizontally towards the right for the topmost boundary of the object, and look vertically towards the bottom for the leftmost boundary. This motivates our corner pooling layer: it takes in two feature maps; at each pixel location it max-pools all feature vectors to the right from the first feature map, maxpools all feature vectors directly below from the second feature map, and then adds the two pooled results together. An example is shown in Fig. 3. We hypothesize two reasons why detecting corners would work better than bounding box centers or proposals. First, the center of a box can be harder to localize because it depends on all 4 sides of the object, whereas locating a corner depends on 2 sides and is thus easier, and even more so with corner pooling, which encodes some explicit prior knowledge about the definition of corners. Second, corners provide a more efficient way of densely

discretizing the space of boxes: we just need O(wh) corners to represent O(w 2h 2) possible anchor boxes.

2.3 ChartOCR: Data Extraction from Charts Images via a Deep Hybrid Framework

This excerpt from the ChartOCR paper explains the concept beautifully:

Some methods [1, 2, 20, 21] have been proposed for chart data extraction. These previous work heavily rely on manually crafted features. The diversity of chart designs and styles makes rule-based chart component extraction approaches difficult to scale. End-to-end solutions based on deep neural networks are also employed to tackle this problem because of their better accuracy [17, 6, 3], but these methods can not generalize well on all the chart types. For example, a framework designed for the pie chart cannot be applied to the line chart. Moreover, comparing to the heuristic rule-based methods, deep end-to-end approaches usually have no control of the intermediate results. Hence, a more general and flexible approach is desired to comprehend various chart images to further enhance the document analysis. In this paper, we propose an approach that tackles the chart components detection problem with key point detection methods [15, 7, 16]. In this way, the chart data extraction can be simplified as a uniform task regardless of the styles of the chart images. Afterwards, an unified network is used for underlying data extraction. We design a deep hybrid framework that combines the advantages of both deep and rule-based methods. As shown in Figure 2, our method first run common information extraction to obtain key points and chart type. Then, we apply type-specific rules to construct the data components (e.g. bar components, sector components) and data range. Finally, we transform these components into structured data format (e.g. tables). It not only exploits the generalization ability of deep methods, but also generates semantic rich intermediate results as in rule-based methods.

There were mainly 3 parts of ChartOCR:

2.3.1 Common Information Extraction

- Key Point Detection: In this step, we extract key points of chart components independent of the chart style. With the universal key point detection model, we no longer need to train separate object detection modules for different charts. For chart images with unseen style, we only need to finetune the existing key point detection model by adding more samples that reflect the new chart style. The key points are defined slightly differently depending on the chart type. For the bar chart, the key points are the top-left and bottom-right corner of each separate bar. For the line chart, the key points are the pivot points on the line. For the pie chart,

the key points are the center points plus the intersection points on the arc that segment the chart into multiple sectors. As shown in Figure 3, we adopt a modified version of CornerNet[15] with Hourglass Net[19] backbone for key point proposal. The output for key point detection network is a probability map that highlights the pixels in key point locations. The probability map has 3 channels to predict the locations of top-left, bottom-right and background. The size of the output probability map is the same as input image. The penultimate layer of key point detection network is a corner pooling layer adopted from the CornerNet[15]. Corner pooling layer performs max-pool on the horizontal and vertical direction respectively, which helps the convolutional layers to better localize key point locations. We follow the same setting of CornerNet[15] and define the loss functions as the summation of probability map loss and the smooth L1 loss for keypoint coordinates.

- Chart Type Classification: We add an additional convolutional layer to the direct output of Hourglass Net and convolve the key point feature map into a smaller size e.g. (32×32). Then we apply max-pooling on it to obtain a one-dim vector. We then feed the intermediate feature vector to fully connected (FC) layers to predict the chart type of the input image. The last FC layer of this branch has softmax activation and this branch is trained with crossentropy loss.

2.3.2 Data Range Extraction

Data range extraction helps us to convert the detected key points from image pixel space to the numerical readings. The data range extraction applies to line and bar charts. For pie chart, the summation of all the sectors should be 100 percent by default, thus the data range extraction can be skipped. We use Microsoft OCR API 1 to extract the text from the image. The extracted text comes from legend, title and axis-labels. For data range extraction, we need to identify the numbers that are associated with y-axis only. To separate out those y-axis labels, we assume that those numbers are always on the left-hand side of the plot area. Thus we only need to locate the plot area, then based on its position, the y-axis labels can be filtered out easily. The plot area is also defined by the top-left and bottom right corners, so we could follow the similar routine as keypoint detection described in 3.1 to locate the plot area. Once we have the plot area location and the OCR result, we design the Data Range Estimation algorithm 1 to get the data range of the chart. In this algorithm, we first use the detected corner points to identify the plot area, then find the recognized numbers that are on the left-hand side of plot area, and finally use the top and the bottom numbers to calculate the data range and pixel range to map the points to the actual data value.

2.3.3 Type-Specific Chart Object Detection

- Bar charts:
  In Section 3.1, we have extracted the top-left and bottomright key points from bar images using the key point detection network. In this step, we need to match all the top-left key points to the corresponding bottom-right key points to construct the bar objects. We binarize the key point probability map by threshold value s = 0.4. For each top left point ptl, we find the closet bottom right point pbr and group them together to obtain the bounding box. This is most common to CornerNet.

- Pie charts:
  To get the location of the pie center and arc points, we use the same key point detection network as decribed in Section 3.1. We replace the corner pooling layer by center pooling layer from [7] to capture the 360-degree neighborhood information. We filter the key points prediction probability map by threshold s = 0.3 to get binarized heat map. For each sector element, the key point detection network extracts the center point pv and arc point parc. When grouping the key points to form the sectors, we consider two cases: (1) Tight pie chart where all the sectors are laying together to form one circle (oval) (2) Exploded pie chart where one or more sectors are separated from each other.

  Previous works [6][23] can process the pie charts in the first case but fail to deal with the charts in the second case. In this work, we design a algorithm SECTOR COMBINING to find the key points in each sectors for both cases. For the first case, we only need to sort the arc points in clock-wise order and calculate the portion of each sector. For the second case, we include the pie radius estimation step where we find the optimal radius that can link all center and arc points. The center and arc points has 1:N mapping, meaningly, one or more sectors can be attached with one center point. We check if the distance between a center point and the candidate arc points is within some threshold. If yes, then this pair belongs to the same sector, otherwise not.

- Line charts:
  The key point detection network predicts the locations of pivot points on the line. In order to group the key points according to the lines that they belong to, we attach a convolutional layer in the key point extraction branch (after conv1 in Figure 3) as the embedding layer.

  We enforce the feature embeddings of points in the same line to be as close as possible, and the embeddings from different lines to be as far as possible. The total loss of key point detection network for line chart is defined as lossPoint = lossPoint +·lossEmbedding, where lossEmbedding = lossPull + lossPush.

To form lines given key points, we adopt the hierarchical clustering strategy to group the embedding of the key points with the classical union-find algorithm. (The details of this algorithm can be found in the supplemental material.) In this way, each cluster contains points that belong to the same line. However, some points may be belong to two (or more) lines and they are usually treated as outliers in the clustering algorithm. We call these points intersection points and propose the QUERY network to predict which lines they should be assigned to.

3 Introducing the code and files related to the project

Before we go over the experiments, it's very important to get an acquaintance about the code and files related to this project.

As mentioned earlier, the ChartOCR repository is a big complex jigsaw puzzle. It's highly undocumented and contains a number of issues that need to be solved in order to run the code. The ChartOCR repository adopts the baseline code from the CornerNet repository. Thus, many of the assumptions made by CornerNet authors are adopted by ChartOCR authors. However, this also creates problems when these assumptions and conventions are extrapolated to concepts that are literally and fundamentally different from the original intent of the assumptions made by CornerNet authors. Hence, there's a lot of mixing of code that needs to be taken care of.

Inside the entry folder, go inside a folder named "Project". We'll cover the Project folder first.

The main folder related to this project is obviously "Deeprule" folder. The main file related to this project is "Chart_OCR.ipynb" notebook. The notebook contains the actual implementations of each phase.

At the same level of ChartOCR notebook, there are 3 folders "bardata", "linedata", and "piedata" that are assumed to contain the appropriate data for the Chart-OCR model. These folders are basically renamed versions of original Excel dataset folders of "bardata(1031)", "linedata(1028)", and "piedata(1008)". Although this can be changed very easily by changing the "–data_dir" input to a respective data folder.

Now I'll explain the directory structure of linedata. linedata folder contains 2 main folders, "nnet" and "line". "line" is the main folder that contains 2 more folders, "images" and "annotations". "images" contains all the image data related to line class charts. The images folder should contain 3 folder,

"train2019", "val2019", and "test2019". All these 3 folders contain the images for training, validating, and testing the line class ChartOCR model. Similarly "annotations" folder contains all the annotations files related to annotations for the 3 folders in the "images" folder. The naming convention of everything's very important. So as a general rule of thumb, try to mimic the same naming pattern as provided in the Excel dataset. If any missing file error occurs, look for the existence and name-pattern accuracy of the mentioned file.

The "nnet" folder contains the pre-trained model ".pkl" file. This model file, like the rest of the files, follows a specific naming pattern. So make sure, whichever model you train is renamed to a version expected by Deeprule code.

During runtime, some of the cache files get created in the "linedata" folder. Make sure to delete these files, in case you want to train or test on a different dataset. Else, there'll be a mismatch between the actual data and the data read during runtime.

The rest of the classes follow a similar pattern as that of the line class and "linedata" folder. Make sure to follow these assumptions clearly else things will break down easily. However, "clsdata" folder needs to be placed inside the "data" folder inside "DeepRule" folder.

The rest of folder (i.e. "excel", "pmc", "pmc_ensemble") were used by me to store the actual data. I used to copy the required folders and files into a specific folder as required by the tasks. The excel folder contained the ExcelChart400K dataset. The pmc folder contained the "bar" and "line" specific datasets. These datasets have been created using one of the utility functions provided in the "utility" folder in the entry directory. The details of "utility" folder will be provided later on. Similary, pmc_ensemble contained the 5 UB-PMC dataset splits for the "bar" and "line" classes. However, depending upon the size, the excel folder might be emptied in order to ease the submission of the project.

Now, I'll explain some of the important files in "DeepRule" folder. So there are basically 3 main files to train a new model. They are "train_chart.py", "train_pmc.py", and "train_ensemble.py". "train_pmc.py", and "train_ensemble.py" have been created by me for Phase-2 and Phase-3 specifically.

There are 2 main files to test a model. They are "test_pipe_type_cloud.py" and "test_ensemble.py". "test_ensemble.py" has been created by me. "test_pipe_type_cloud.py" deals with phase-1 and phase-2. "test_ensemble.py" deals with phase-3.

I won't be explaining the files related to internal working of DeepRule repository. But I'll be commenting on some of the files as and when required. I would also be providing the link of data files and models to scorers via an alternative medium in order to maintain the privacy of that link.

4 Experiments

As mentioned above, there 5 phases for the project. Out of these 5, 4 were experiments related. Now I'll go over each phase sequentially. This part would contain the instructions on how to replicate each conducted experiment.

4.1 Phase-0

Not directly related to experiments

4.2 Phase-1

Phase-1 was related to the feasibility study of the ChartOCR paper. The idea was to get the same results on the Excel dataset, as per the claims made by the author. The next task of the phase was to get results by applying the ChartOCR model on the UB-PMC dataset and benchmark those results using the evaluation script from ChartInfo-2020. We used pre-trained models for all the tasks during phase-1. I used "json_[bar/line]_[train].py" scripts provided in the "utility" folder, to filter and convert UB-PMC dataset into an Excel format.

To carry out Phase-1, follow the specific instructions:

- Make sure the appropriate "test2019" folder and annotation file is appropriately placed in the referenced folders.
- Follow dependencies installation steps given in Chart_OCR notebook. Make sure to change the paths/values of Constants "DEEPRULE", "ROOT_DIR", and "IMG_DIR" according to your settings.
- During testing, the code pre-loads all models, considering the fact that image can be of any class. Thus, make sure the appropriate files and modesl of all classes are placed accordingly.
- Run the Phase-1 commands accordingly and sequentially.
- ChartOCR Metrics can be evaluated using the functions provided in the notebook.
- Many times the iteration suffix in the name of the saved model will be different from "50000". Make sure to change that to "50000", so that model names are accurate.

NOTE: To get the pmc metrics from the predicted json output, we copy the file in the "metric" folder inside "utility" folder and rename it as "pred.json". Make sure the "pred" and "gt" folders inside metrric folders are empty. Than run the "metric_[bar/line].py" file. This should populate the "pred" and "gt" folders. Now runs the "metric6a.py" file appropriatly, where "pred" folder contains the prediction files and "gt" folder contains the ground truth files.

4.3 Phase-2

In Phase-2, we had to train a new ChartOCR model using UB-PMC dataset. I used "json_[bar/line]_[train].py" scripts provided in the "utility" folder, to filter and convert UB-PMC dataset into an Excel format. You can train the a new model by running the command given in the notebook.

- Clean all cache and run-time files created during previous runs. For example, a dataset cache ".pkl" file might be created in the data folder during the run.
- During testing, the code pre-loads all models, considering the fact that image can be of any class. Thus, make sure the appropriate files and modesl of all classes are placed accordingly.
- You can change the number iterations and frequency of model saving by changing the parameters in "train_[chart/ensemble].py" file.
- Many times the iteration suffix in the name of the saved model will be different from "50000". Make sure to change that to "50000", so that model names are accurate.

NOTE: To get the pmc metrics from the predicted json output, we copy the file in the "metric" folder inside "utility" folder and rename it as "pred.json". Make sure the "pred" and "gt" folders inside metrric folders are empty. Than run the "metric_[bar/line].py" file. This should populate the "pred" and "gt" folders. Now runs the "metric6a.py" file appropriatly, where "pred" folder contains the prediction files and "gt" folder contains the ground truth files.

4.4 Phase-3

Phase-3 expected us to train an ensemble of 5 models for each class on the Excel dataset. However, I made a very silly and embarrassing mistake. I ended up training on splits of the UB-PMC dataset instead of the Excel dataset. I used "fed_[bar/line].py" scripts provided in the "utility" folder, to filter and convert the UB-PMC dataset into an Excel format. You can train a new model by running the command given in the notebook. Training on Excel would've been

much easier as the data was already in the correct format. However, life is full of errors and this one was pretty embarrassing.

As mentioned earlier, many files inside the DeepRule directory were changed to make "train_ensemble" work. Most importantly "Testfiles" for the respective configuration files were changed to make things work.

ChartOCR paper has mentioned that their model uses a combination of the deep learning methods and the rule-based methods to get the final predictions. Thus to get predictions for the ensemble of models, the detected key points from different models were combined using the common rules-based methods used in DeepRule code. So basically, we take the detected key points and combine them as an input for the rest of the process. The rest of the rule-based process is not directly dependent upon the learning of the model.

- Make sure the precautions mentioned in the earlier parts are correctly followed. You might need to change the names of splits data folders and files.
- You have to use "train_ensemble.py" file to train a new split.
- Make sure you follow the split naming pattern mentioned in the notebook.
- Make sure to rename the trained models according to the pattern mentioned in the notebook. For example, split-0 will be renamed to "CornerNetXXX_50000.pkl", split-1 will be renamed to "CornerNetXXX_150000.pkl" , split-2 will be renamed to "CornerNetXXX_250000.pkl" and split-4 will be renamed to "CornerNetXXX_450000.pkl".
- The testing part is similar to Phase-2.

4.5 Phase-4

Phase-4 wasn't as such a stand-alone experiment. Phase-4 could be considered as a side task to Phase-1, 2, and 3. Phase-4 tasked to search for the appropriate "Hyper-parameters" and "Modularize" the code so that it can be used in a plug-and-play setting for different datasets. There were many changes that were made during the course of the projects. One of the interesting Hyper-parameter changes was related to changing the "ae_threshold" value for Bar class during training a new model on the UB-PMC dataset. The changed value was surprising and radical, however, it provided some results and predictions. However, that effect might have been due to the fact that I didn't fully train the models due to time limitations and training time.

But the code is highly modular, as very minimum new code was added on my part to the original repository. Thus, it allows the plug-and-play scenario for different datasets, given the fact that one needs to convert the new dataset into an

appropriate excel format expect by ChartOCR code. So the level of modularity has been maintained given to the constraints of the original codebase.

5 Evaluation

Original ChartOCR paper presented their metrics using a custom formula for each class. The details of this formula are provided in the paper along with their claimed metric numbers for each class. The implementation of these custom formulas is available inside the Chart_OCR notebook.

Likewise, ChartInfo-2020 used its own evaluation script to evaluate the predictions on the UB-PMC dataset. The details of the evaluation scripts can be found in the references. I'll be using "metric6a.py" to calculate the UB-PMC metric. The script is available inside the "metric" folder. I won't be attempting task-6b of ChartInfo-2020.

Results for Phase-1 (Testing pre-trained models on ExcelChart400K dataset):

Description	Bar	Line	Pie
Results provided in the paper	0.919	0.962	0.918
Results evaluated by me	0.9089	0.901	0.88

Results for Phase-1 (Testing Bar class pre-trained models on UB-PMC dataset):

Description	Net Avg. Bar	Vertical Bar	Horizontal Bar
Results evaluated by me	0.65	0.71	0.34

Results for Phase-1 (Testing Line class pre-trained models on UB-PMC dataset):

Description	Net Avg. Line	Scatter	Line
Results evaluated by me	0.74	0.43	0.83

Results for Phase-2 (Testing Bar class trained models on UB-PMC dataset):

Description	Net Avg. Bar	Vertical Bar	Horizontal Bar
Results evaluated by me	0.801	0.826	0.669

Results for Phase-1 (Testing Line class trained models on UB-PMC dataset):

Description	Net Avg. Line	Scatter	Line
Results evaluated by me	0.716	0.59	0.76

Results for Phase-2 (Testing Ensembled Bar class trained models on UB-PMC dataset):

Description	Net Avg. Bar	Vertical Bar	Horizontal Bar
Results evaluated by me	0.58	0.60	0.55

Results for Phase-1 (Testing Ensembled Line class trained models on UB-PMC dataset):

Description	Net Avg. Line	Scatter	Line
Results evaluated by me	0.61	0.53	0.63

6 Conclusion

The whole project was challenging from a practical point of view. Although it wasn't a huge coding task, understanding the nitty-gritty details related to the DeepRule repository and CornerNet repository was quite challenging. There were issues that were related to the redundancy of the code. There were issues that were related to deprecated versions of code. There were issues related to the misnomer of certain functions due to the legacy code of CornerNet being forced onto the concepts of DeepRule code. As such the strategy was to advance incrementally by solving each error sequentially. This debugging proved to be a boon in disguise as it made my understanding of the code more clear. Hence, to sum up it was a big and complex jigsaw puzzle.

Another challenge from a practical perspective was that I was doing it alone instead of a team, so the workload was more for me as most other teams had 3-4 students each.

Another challenge was that I was doing everything on Google Colab and Drive as my laptop(i.e. Mac M1 pro) didn't support many libraries during the time of project execution. So editing the code on google drive wasn't a pleasant experience.

However, I was still able to get decent results along with not decent results. The final metrics for Phase-1(Excel Data) were almost the same. I had no benchmark for the rest of the metrics. But taking a cue from the results of the participants of the ChartInfo-2020, I can say that there's a huge scope for improvement and better solutions might be available in the methods adopted by winners of the ChartInfo-2020 competition. However, This paper provided an alternative method to tackle the problem. It's also to be noted that I couldn't fully train my models due to practical limitations of training on Google Colab.

Otherwise, the results might have been better. However, the experience from this project was extremely valuable.

7 References

References

[1] Luo, Junyu and Li, Zekun and Wang, Jinpeng and Lin, Chin-Yew (2021): *ChartOCR: Data Extraction from Charts Images via a Deep Hybrid Framework*, The Computer Vision Foundation, URL: https://www.microsoft.com/en-us/research/publication/chartocr-data-extraction-from-charts-images-via-a-deep-hybrid-framework/

[2] Law, Hei and Deng, Jia (2018): *CornerNet: Detecting Objects as Paired Keypoints*, Proceedings of the European Conference on Computer Vision (ECCV)

[3] Newell, Alejandro and Yang, Kaiyu and Deng, Jia", editor="Leibe, Bastian and Matas, Jiri and Sebe, Nicu and Welling, Max (2016): *Stacked Hourglass Networks for Human Pose Estimation*, Springer International Publishing

[4] Kenny Davila, Chris Tensmeyer, Hrituraj Singh, Sumit Shekhar, Srirangaraj Setlur, Venu Govindraju (2020): *CHART-Infographics 2020*, URL: https://chartinfo.github.io/