

Department of Electrical Engineering, IIT Jodhpur
BTP Report
August 2020

Title: **Scientific Chart Reader**

Names & Roll Numbers: 1. Arpit Gupta (B18EE006)
2. Raghav Ranjan (B18EE037)

Name of Supervisor: **Dr. Anand Mishra**

Abstract

Most scientific documents contain charts such as bar charts, pie charts, and other plots to visualize and present the results. These charts are information-rich. However, most state-of-the-art OCR engines label them as graphics and do not read and interpret them. In this work, we take a baby step towards developing methods for reading one of the prominent scientific charts, namely bar charts. Bar Charts are a great way to visualize and understand data and have vast applications in this data-driven world. In this project, we propose an effective, fully automated system that returns the data extracted from a bar graph in the form of tables and latex script when given an input image of a bar graph. In our solution, we identify the type of bar graph, extract textual data followed by determining the height of the bars and displaying the results on our web app. Our solution achieved a classification accuracy of 91.2%, and the accuracy of 95.25% in extracting data tested over 100 unknown test images, in extracting the data from the bar chart. Our project repo is available [here](#).

Keywords: Computer Vision, object detection, Information Retrieval, Chart data extraction.

1. Motivation

Bar Charts are useful tools for helping people visualize and comprehend large amounts of data. Bar graphs are used extensively to visualize data in presentations, data analysis, reports, and other fields where data visualization is required. Currently, there isn't any available method designed specifically for reading and extracting data from an image of a bar graph. We wanted to take up a challenging project with vast applications and found this a perfect choice. Being able to accomplish this, we would be able to extract data from large datasets consisting of bar charts and obtain them in tabular form, which is ideally suited for data analysis.

2. Methodology

We have divided our project into three main sections.

1. Classification of Bar Graphs (Deep Learning)
2. Extraction of Data (Image processing)
3. A web application to host this project (Web Development)

We divided the bar graphs into four subtypes namely,

1. Simple Vertical Bar graph
2. Simple Horizontal Graph
3. Stacked Vertical Bar Graph
4. Stacked Horizontal Bar Graph

We pass the input bar graph image to our API hosted on the server. When the API receives the input image, it is passed through a deep learning classifier, which identifies the bar graph's subtype. Then the respective algorithm is applied accordingly, which gives us the readings of the graph. These readings are then passed to the front end and displayed on our website and the latex script of the table.

2.1 Tool Used

- Python: Python is a high-level programming language that is platform-independent, user friendly, and open-sourced.
- Tesseract OCR: Tesseract is an Optical Character Recognition (OCR) engine, we used Pytesseract, a wrapper of Tesseract OCR engine for python. It is the best available open-source OCR engine.
- OpenCV: Open Source Computer Vision Library provides a common infrastructure for computer vision applications and comes packed with various image processing tools.

- Tensorflow: Tensorflow is a python friendly open-sourced library developed by google used for numerical computations that are used for machine learning applications such as CNN.
- Django: Django is an open-source framework for backend web applications based on Python. Its main advantages are simplicity, flexibility, reliability, and scalability.

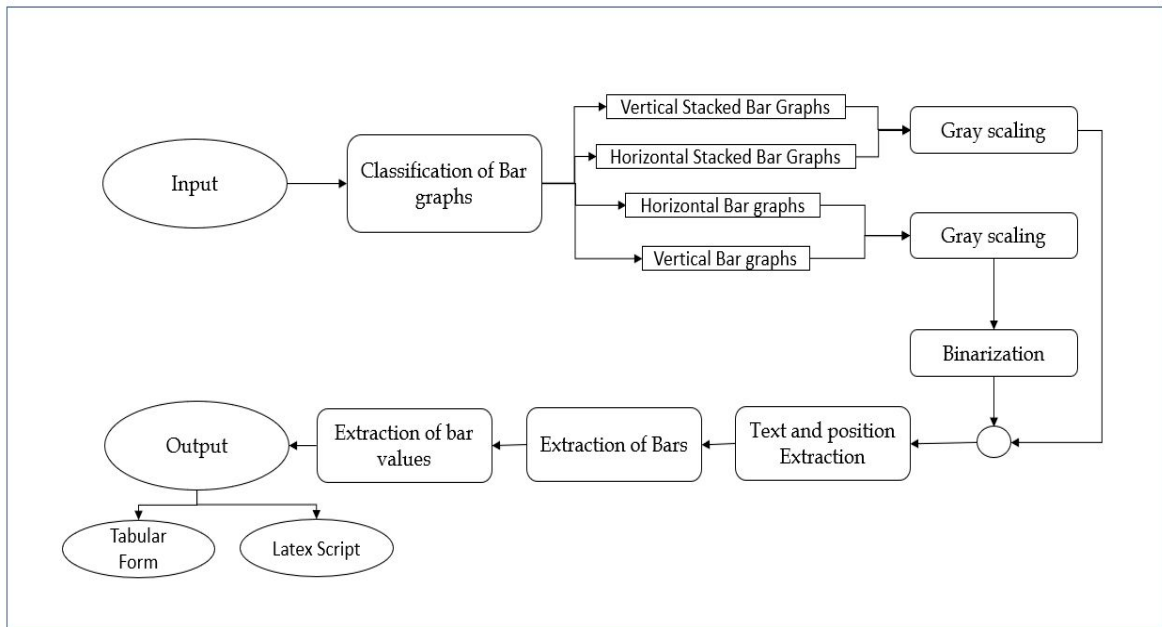


Fig 1: Flow Chart- Scientific Chart Reader

A. Classification of Bar Graph

I. Data Collection:

We couldn't find a publicly available dataset that could cater to our needs, so we had to create it ourselves. We used Matplotlib, a graph plotting library for python, and wrote a python script that randomizes different variables of a graph such as Size, fonts, number of bars, colors of the bars and legends. We used 4000 images for the four types of bar charts to train our deep learning classifier and further generated 1000 images for testing our model.

II. Deep Learning Model:

We used a sequential model. We used different pre-trained models and didn't include its top layer and added some layers like dense layers and flatten layers to detect the type of bar graph and classify it according to the class given to it.

For pre-trained models, we used the Keras application. Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. We trained our classifier model using different pre-trained models like VGG16, VGG19, InceptionResNetV2 as a base model, and got better results with VGG16 with an accuracy of 91.2% for 1000 unknown test data.

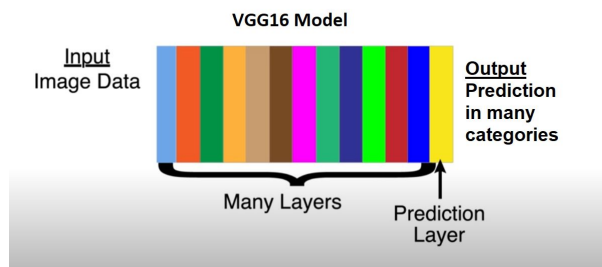


Fig 2.1: VGG16 model

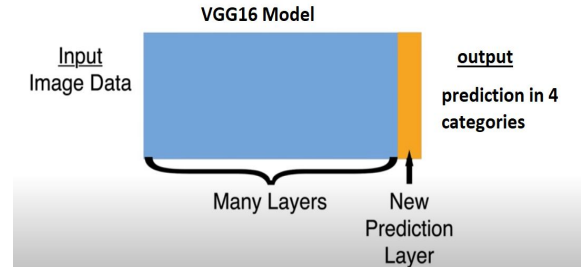


Fig 2.2: Model after modification

So basically the architecture of our model is:

The base model is VGG16. We removed its top layer and then added a Flatten layer to convert the output of the previous layer to 1D representation. Then we added two Dense layers with 256 and 4 filters (no. of Convolution), respectively. We used the SGD optimizer.

We had a total of 16000 images, 4000, for each class.

We choose batch sizes of 25 and 50 epochs to train and validate our model.

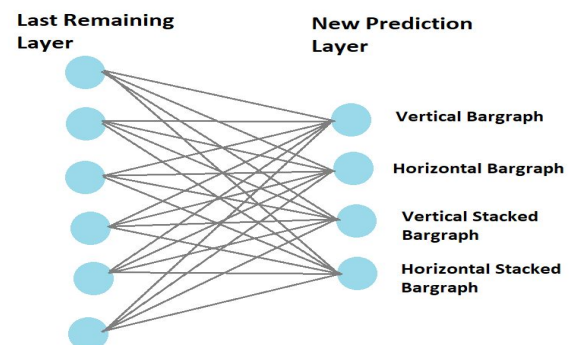


Fig 3: Basic Node representation

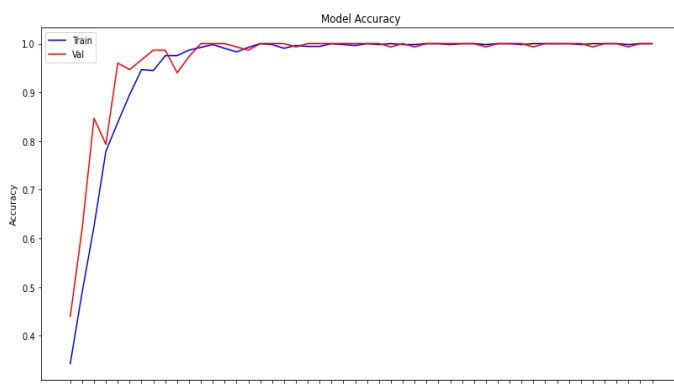


Fig 4.1: Model Accuracy v/s epoch

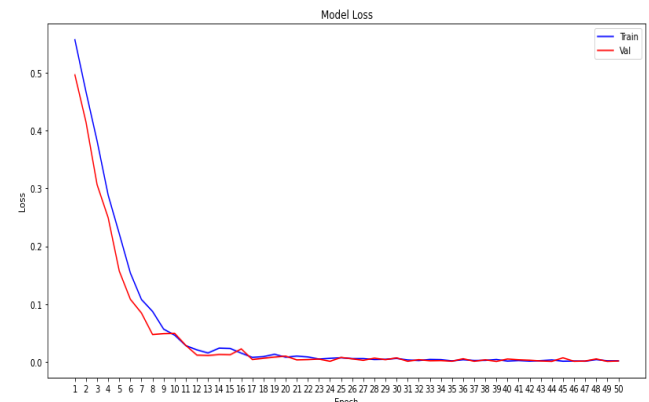


Fig 4.1: Model Loss v/s epoch

A. Extraction of Data

I. Detecting text and Position

i) Preprocessing:

We apply gray scaling and thresholding on the input image to improve the accuracy of Pytesseract, a wrapper for Google's Tesseract OCR engine.

- **Gray Scaling:** In RGB images, there are three color channels and have three dimensions, while grayscale images are single-dimensional. It is easier to work with single-dimensional data. We used the OpenCV library to grayscale the Input image.
- **Thresholding:** In the grayscale image representation, pixel values can be any value between 0 to 255. For simpler graphs (other than stacked), we don't need to consider a color variation to calculate the heights of bars. So we just binarized the grayscale image, i.e., assign a pixel either a value of 0 (Black) or 255 (white). To do this, we can perform thresholding. In this, we set a threshold Let us say 200 and then all pixels with values higher than 200 would be assigned a value of 255, and all pixels with values lesser than or equal to that would be assigned a value of 0.

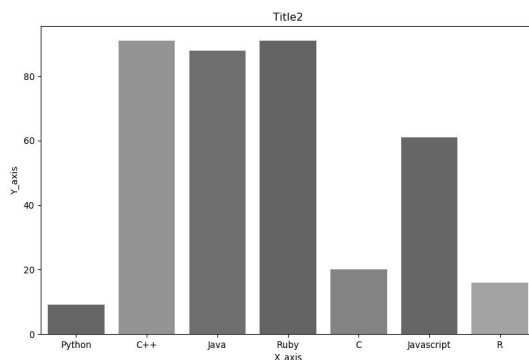


Fig 7: Grayscale Image

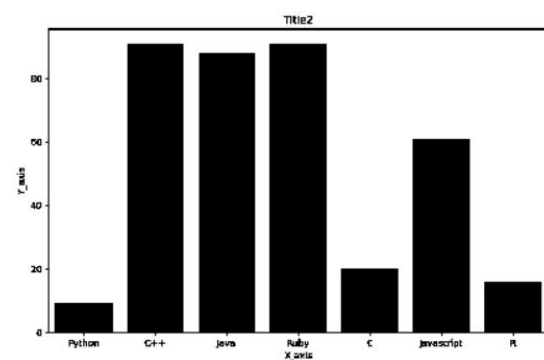


Fig 8: Binarized Image

ii) Text Detection:

After the preprocessed image is passed through by tesseract, it returns a list of each character that it could detect and the coordinates of a box around it.

Tesseract also has a configuration to return blobs of texts forming words directly, but we found out that it was more accurately able to detect characters than words; hence we followed this approach. From the list of characters obtained, we formed them into

words by combining the characters whose coordinates are adjacent. We then obtain the midpoint of each label of a bar to mark its position.

We obtained the value of the graph's pixel ratio by extracting the numerical values and finding the ratio of the difference between the actual values between two adjacent value labels to the number of pixels separating them. We multiply this ratio by the obtained heights of the bar to get the bar's final value.

$$\text{Height of Bar} = \frac{\text{Numbers of pixels in height range of a bar}}{\text{Number of pixels in given length of unit on y-axis}} \times (\text{Length of that particular unit on y-axis})$$

Pytesseract can correct 5 degrees skew error by itself so the labels can diverge from the horizontal axis by about 5 degrees and still be read correctly by the OCR engine.

II. Extracting Data from Graphs:

i) Preprocessing:

For our algorithm to run accurately, we first need to get rid of all the unwanted regions from the image such as texts, random lines, legends, noise leaving only the bars for the algorithm to work with. Hence this step is very important.

In the case of simple graphs, we applied thresholding while for stacked graphs, we grayscale the image because, in stacked graphs, the color of different stacks over a bar is of importance to us, but in case of simple graphs, we don't need their color hence we binarize them.

We assumed that a bar is a rectangle with its width being smaller than its height. To detect these bars, we perform Morphological Operations. We chose the kernel length as 1/100th of the width of the image and defined the rectangular kernel of

- 1) Kernel length \times 10 in case of vertical bars to detect vertical bars
- 2) kernel of 10 \times Kernel length in the case of horizontal bars to detect horizontal bars.

We first inverted the binary image and then performed Opening ($(A \circ B) = (A \ominus B) \oplus B$: First Erosion and then Dilation) over the image. It is done to obtain an image that only contains bars, and all noise has been eliminated. Then we again inverted the image to get black bars.

Erosion (\ominus): It is used for probing and reducing the shapes contained in the input image. Basically, it acts as a local minimum filter. Using erosion, we can eliminate the holes, and gaps between different regions become larger and small details. The value of the output pixel is the minimum value of all pixels in the neighborhood. In a binary image,

a pixel is set to 0 if any of the neighboring pixels have the value 0. It removes islands and small objects so that only substantive objects remain.

Dilation (\oplus): It is used for probing and expanding the shapes contained in the input image. Basically, it acts as a local maximum filter. It makes objects more visible and fills in small holes in objects.

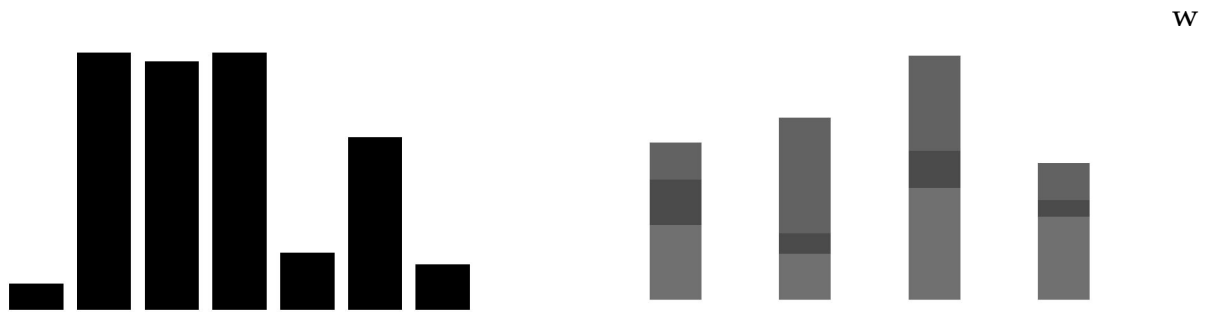


Fig 9.1

Fig 9.2

Fig 9: Extracted Bars after Morphological a).on Binarized Image b). Grayscale image

We followed the same process for extracting the text positions and preprocessing for the four types of bar graphs, but the algorithms we used on them are different.

The algorithm for simple horizontal and vertical bar graphs is similar, and also the algorithm for stacked vertical and horizontal bar charts is closely related to each other.

ii) Obtaining Values:

• Vertical and Horizontal bar Charts:

After preprocessing, we obtain a binarized image containing only bars. The bars are black colored (pixel value=0), and the background is white (pixel value=255).

We start traversing the image linearly from the bottom till we come across the beginning of a bar, a black pixel. We call this point say $(x1, y1)$, then we traverse along the length of the bar till we reach its maximum height, i.e., the background. The difference of this point from the $y1$ gives us the height of that bar. We continue traversing the bar's width while also checking that the region just above the bar is white-colored and below it is black. If we encounter a black pixel above the point of a white pixel just below the point, then we mark this point $x2$, and the midpoint $(x1+x2)/2$ gives us the location of the bar. We continue traversing in the direction of the change until we reach a white region or a black region, respectively. The difference between this point and $y1$ gives us the height of the bar, and similarly, we continue traversing the width until we reach a white region.

If we reach y_1 , then it means that we have reached the foot of the bar, and then we start traversing in the direction of the width of the bar until we encounter black pixels. We mark this point (x_1, y_1) , and the process goes on until we reach the end of the image.

In the case of Horizontal bar graphs, we start traversing from the top downwards, and everything else remains the same.

After this algorithm has finished running, we get the heights of the bars in pixels and the coordinates of the bars' midpoints.



Fig 10.1



Fig 10.2

Fig 10 (a). Ratio, X-titles, Bars positions (b). Final values (heights) of bars

- **Stacked Vertical and horizontal Bar Charts:**

After preprocessing, we obtain a grayscale image consisting of only bars with white-colored (255) background and the bars less than 240 pixels.

We start traversing from the bottom until we encounter a bar. We mark this point (x_1, y_1) . We note the pixel value of the encountered coordinate and start traversing along the length of the bar. If there is a change in more than 10 color pixels, it means we have reached a different stack of the graph. We calculate the difference in this point and y_1 to get the height of this stack. We then start traversing along the length till either a different color or the background is encountered. We calculate the height of the current stack, which is the difference between the y value of the previous stack. This way, we obtain the height of each stack over a bar. Once we reach the background pixel, we traverse along the length until we reach the background again.

We mark this point x_2 and then obtain the midpoint of the bar as $(x_1 + x_2) / 2$. Then we travel downwards along the length till we reach y_1 . We then continue traversing along the width of the bar until we reach another bar. We again mark this point (x_1, y_1) . We keep repeating this process until we reach the end of the image.

For horizontal stacked bar graphs, we start traversing from the top downwards, and everything else remains the same.

After this algorithm has finished running, we get the heights of each stack overall bars in pixels and the coordinates of the midpoints of the bars.

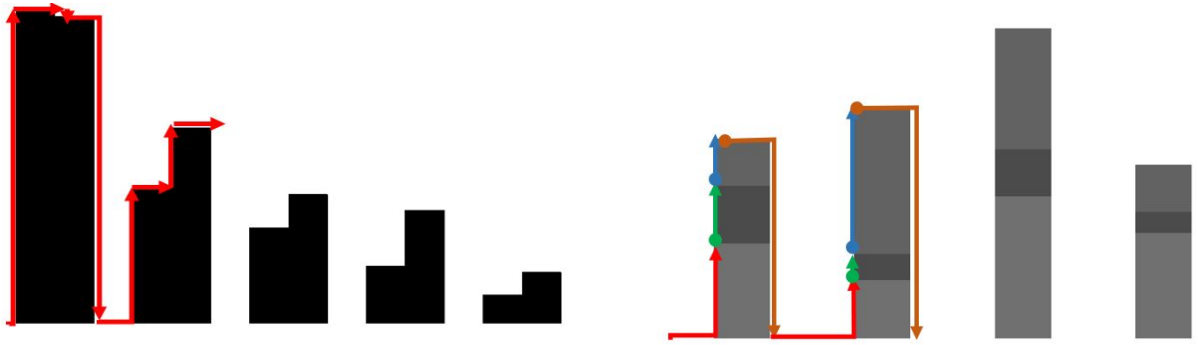


Fig 11: Depicting the algorithm to obtain values of bars

After we have obtained all the positions and heights of the bars, we combine the positions of each label obtained and the locations of the bars obtained from the above algorithms. We assumed that the bars' locations should be very close to their labels, and over this assumption, we applied the following logic: if the difference between a label and bar is large, then it implies that the bar at that label has value 0.

After we have obtained the heights of all the bars or stacks, we multiply them with the value to pixel ratio to get the actual height of the bar, which essentially is its reading. Hence we successfully obtain the readings of a bar graph. We then create a table in the form of a latex script.

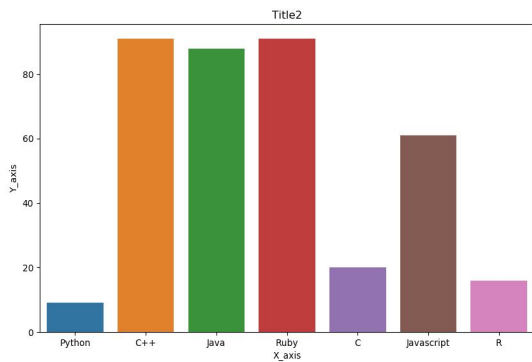


Fig 12.1: Input image

X-axis titles	Values
Python	9.0
C++	91.0
Java	88.0
Ruby	91.0
c	20.0
Javascript	61.0
R	16.0

Fig 12.2: Output Table

C. Web App:

We created an API (Application Programming Interface) using Django, a python based open-source web development framework. The API receives an input image from the frontend and passes the image to our algorithm.

The API then sends the bar chart's output readings and its corresponding latex script back to the user.

3. Results and Discussions:

The accuracy of our system depends on the accuracy of the classifier and those of the readings. We successfully obtained a classification accuracy of 91% over 1000 unknown test data using the AUC metric. We calculate the accuracy of the readings using

$$\text{Accuracy} = (| \text{calculated} - \text{True} | / \text{calculated}) \times 100$$

The calculated accuracy for different types of bar graphs comes out to be different,

Simple Vertical Bar Graph= 99% Simple Horizontal Bar Graph=97%

Stacked Vertical = 94% Stacked Horizontal=91%

An average of 95.25% accuracy tested over 100 test images.

We find that the accuracy of vertical graphs is more than their horizontal counterparts, the reason for this is that the Tesseract engine performs much better when the texts are horizontal. Also, the simple bar graphs have higher accuracy than the stacked ones. This happens because the grayscale of different colors tends to be equal, and hence no change is detected and hence the lower accuracy.

The minor errors in readings are due to the height of the bars being so small that it is considered noise and removed during the preprocessing step.

Our algorithm works best when some of our assumptions are satisfied,

- 1) There are no writings on the labels, and the width of the bars is not very thin.
- 2) The background is light-colored.
- 3) The graph doesn't have too many graphical effects.
- 4) The axes are located near the bottom-left corner of the images.
- 5) The text doesn't have an angle greater than 5 degrees with the horizontal axis.

4. Conclusion:

Our solution performs very well when the graphs are auto-generated. Still, when we tested our system on graphs from the web, the accuracy fell considerably, this happens due to the graphs having various graphical effects to make them look more attractive. Our accuracy can be improved drastically if we could improve the accuracy of the Tesseract engine, which largely controls the efficiency of our model. We can improve this by using RNN to extract regions of texts and then applying OCR on them to be more accurate. Our project's final goal is to be able to read and extract data from any bar graph.

5. References

- [1] Abhijit Balaji1, Thuvaarakkesh Ramanathan, Venkateshwarlu Sonathi “*Chart-Text: A Fully Automated Chart Image Descriptor*”
- [2] Object detection and classification public notebook on kaggle.com. 2019.
<https://www.kaggle.com/navidrashik/object-detection-and-classification>
- [3] Kanan Vyas, “*A Box detection algorithm for any image containing boxes.*” An article on medium.com. July 2018.
<https://medium.com/coinmonks/>
- [4] Nickson Joram, “*Morphological Operations in Image Processing.*” An article on medium.com. Jan 2020.
<https://medium.com/@himnickson/>

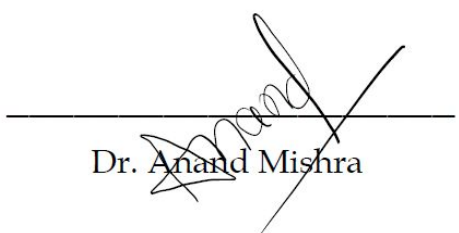
Signatures



Arpit Gupta



Raghav Ranjan



Dr. Anand Mishra