Image Processing and Object Detection in Plotly-Dash

Moe Hein Aung and Regina Sahani

Abstract

With the rise of big data and the advancements in computer hardware, the need for the visualization of data has become extremely as it not only helps with finding new trends and gaining new insights from data but also to communicate these results to an audience. In an effort to understand some of various new technologies that have also emerged as a consequence, we have built a dashboard application using Plotly-Dash framework to perform object detection and classification in images as well as apply image segmentation and post-processing such as gamma correction.

1 Introduction

The work aimed to build simple but meaningful dashboard user interface (UI) that can load in an image through a URL and then immediately allow a user to adjust the gamma on the image. A random button is provided that will allow a user to get a randomized image from a set of images hosted on a public GitHub repository . This was mainly achieved through Plotly-Dash which is a Python open-source framework that allows users to build UI that can collect data from external sources and show visualizations in the form of plots and graphs with their built-in tools.

For image segmentation, two implementations were provided: thresholding using binary segmentation based on a user-selected value and thresholding with black and white dithering. Refer to section 5 for more details regarding the algorithms. For object detection and classification, we utilized Facebook's pre-trained DETR model as described in section 6 of this report. As our testing dataset, 30 images that include a variety of objects were selected and converted to JPEG format where 11 of the images are personal and 19 are open-source images from Flickr. Some of the selected objects for detection include motorcycle, airplane, train, snowboard, pizza and clock among many others.

2 Gamma Correction

After the display of the selected original image, a user can adjust gamma on the image using a slider. The default value is set to be 2.2 as it is common in many modern displays. Figure 1 shows the before and after gamma correction of an image.

(a) Before (b) After

Figure 1: Before and After Gamma Correction

3 Image Segmentation

One provided implementation for image segmentation is based on the binary black and white thresholding operation. The user can interactively select a threshold value using a slider and if the intensity of a pixel is greater than the selected threshold, it is rendered white and vice versa. Figure 2 shows an example of a segmented image using binary thresholding.

Figure 2: Thresholding based on Binary Segmentation

To better assist the user to select threshold value, a histogram of RGB channels of the original image is plotted immediately below. Based on the Gaussian distributions, the user can selected a good threshold value at the demarcation of the Gaussian curves. Figure 3 shows the RGB histogram of the original image.

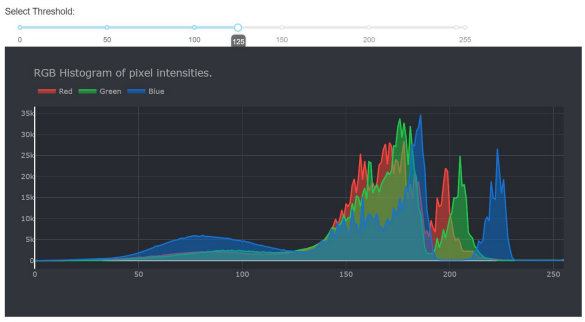

Figure 3: RGB Histogram

The second method for image segmentation is based on black and white dithering and is implemented using the Pillow image library. This thresholding method uses Floyd-Steinberg dither method which approximates the image's luminosity levels. All values larger than 127 are set to be white and values less than 127 are set to be black. Figure 4 below demonstrates image segmentation based on black and white dithering.

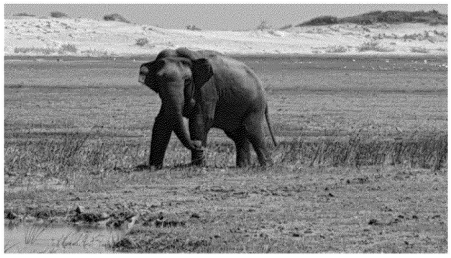

Figure 4: Black and White Dithering

4 Object Detection

For object detection and classification of images, the Detection Transformer Algorithm (DETR) by Facebook was utilized. DETR was released in 2020 and uses a combination of Convolutional Neural Network (CNN) architecture and Transformers. The details of the algorithm are out of the scope of this paper but it is worth noting that the performance of this model is extraordinary. In Figure 5 below, it can be seen that the algorithm can detect the truck in front of a car as well as the potted plants and vase as well as the human all in one tight image.

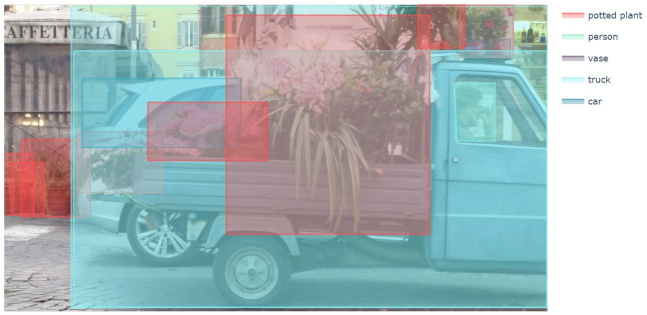

Figure 5: Object Detection with DETR

5 Application Performance

Our application generally runs well on a desktop equipped with i7 CPU (6 cores) and a Nvidia RTX 2070 Super GPU. We also tested with just the CPU on a laptop. Both devices had 16GB of RAM. Table 1 shows that there is a huge decrease in performance without the use of a GPU. It is interesting to note however that when the image size is very large, about 5MB, the hardware with GPU resulted in even slower performance than the laptop. This could be an interesting area of investigation for the future.

Table 1: Performance metrics of dataset

Image	CPU time (s)	GPU time (s)	Total no. of objects detected	No. of object types	Average objects per classification	CPU time per object	GPU time per object	Largest class size	Image Size (kB)
im1	12.04	2.33	28	6	4.67	0.43	0.08	20	70
im2	11.07	2.27	3	2	2.8	3.69	0.76	2	82
im3	8.11	2.23	4	2	2	2.03	0.56	3	73
im4	8.09	2.36	13	5	2.6	0.62	0.18	9	90
im5	10.65	2.38	8	4	2	1.33	0.3	4	125
im6	8.61	2.09	21	7	3	0.41	0.1	6	87
im7	9.44	2.27	3	2	1.5	3.15	0.76	2	84
im8	7.99	2.54	3	3	1	2.66	0.85	3	69
im9	9.5	2.29	2	2	1	4.75	1.15	2	70
im10	7.45	2.07	2	2	1	3.73	1.04	2	48
im11	10.39	2.17	26	6	4.33	0.4	0.08	13	194
im12	7.58	2.09	2	2	1	3.79	1.05	2	56
im13	9.04	2.51	3	3	1	3.01	0.84	3	84
im14	9.37	2.1	33	3	11	0.28	0.06	16	92
im15	11.35	2.16	2	2	1	5.68	1.08	2	83
im16	16.42	2.41	6	4	1.5	2.74	0.4	2	83
im17	7.5	2.45	28	3	9.33	0.27	0.09	14	141
im18	9.65	2.31	16	7	2.29	0.6	0.14	4	89
im19	25.12	0.25	14	5	2.8	1.79	0.02	8	1175
im20	19.29	8.18	5	1	5	3.86	1.64	5	267
im21	15.32	5.77	1	1	1	15.32	5.77	1	116
im22	18.18	7.69	14	1	14	1.3	0.55	14	109
im23	22.37	7.93	3	2	1.5	7.46	2.64	2	142
im24	17.49	0.45	1	1	1	17.49	0.45	1	5216
im25	24.47	25.7	1	1	1	24.47	25.7	1	5253
im26	24.79	57.17	3	3	1	8.26	19.06	1	5416
im27	31.77	0.43	1	1	1	31.77	0.43	1	7765
im28	25.92	38.82	25	7	3.57	1.04	1.55	18	970
im29	29.5	14.13	6	4	1.5	4.92	2.36	3	674
im30	48.11	0.49	6	2	3	8.02	0.08	4	3530
sum	**1545**	**9**	**632**	**310**	**50**	**123**	**25**	**2694**	**2546**

6 Conclusion

Visualization dashboards and visualization algorithms will continue to become more relevant and applicable to a wide range of industries ranging from artificial intelligence, healthcare, finance and many more. There is ongoing research to continually come up with the next best visualization technique that is optimized better than its predecessor and leverages the latest hardware.

Through this project, we have learned to create a visualization dashboard using one of the most popular frameworks and learned to host image processing functions such as gamma correction. We demonstrated image segmentation through the use of two separate algorithms as well as perform object detection and classification on images using one of the latest computer vision algorithms.

ACKNOWLEDGMENTS

The authors wish to thank A, B, and C. This work was supported in part by a grant from XYZ.

REFERENCES