Eric Gomez, Peyton Slater
Computer Vision
December 16th, 2023

Implementation of Edge Detection and Object Detection Method

Introduction:

The main goal of this project is to create a method for edge detection, then to identify the five largest objects of the image and display them to the user. Secondary objectives include the implementation of other edge detection methods and comparing their results with our design.

Implementation:

For our edge detection method, we aimed to improve upon the traditional sobel edge detection by enhancing the visibility of prominent edges while minimizing noise and unwanted details. First, we decided that by smoothing the image with a 3x3 gaussian kernel would be good because it would remove a lot of the noisy edges from the final result. We then applied a simple dark-channel-prior to get rid of hazy and foggy elements that would appear in the background of the image. Then, we convert the image to grayscale to remove most of the color that would interfere with the contour detection that we used. To find the edges, we used a simple sobel operator in the x and y gradient. After finding the gradient magnitude of the sobel operators, we further filtered the gradient with a threshold of 75 , which removed pixels with gradient magnitude values above 75. This removed a lot of the less prominent edges at the cost of losing some detail in the final image. Now that we had the edges, we used cv2's findContours operation to find the contours of the edges array (The boundaries of the objects). We then filtered out the contours with a threshold of '3', which gets rid of contours that are not thicker than 3 pixels. This removed less prominent objects. We then displayed the edge detected image to the user and sent the contour image to our object detection algorithm.

For the object detection method, we mainly utilized built-in cv2 methods to detect the objects. We first start by converting the image to a binary image. We then use cv2 to find the contours again, as well as sort those contours by area. We then just simply draw the 5 largest contours with cv2 drawContours and show them to the user.

Code explanation:

Our code has 2 runnable files, main.py and evaluation.py. Main.py opens a UI where you can input any image and it will display the 5 largest objects of that image as well as the edge detected image. It will also open a separate window with the edge detection results of all the other edge detection methods we implemented. Evaluation.py will show the runtimes of the edge detection methods, as well as save the results of our edge detection method on BDSD500 to a directory titled 'output'

Evaluations:

To properly evaluate our implementation, we decided it would be best to test our results on the Berkeley Segmentation Dataset 500 (BDSD500). This dataset contains 200 test images that we can test our algorithm with. We also implemented the following edge detection methods: Canny, Sobel, Kirsch, Lapplachan of Gaussian, PreWitt, and Holistic Edge Detection(HED). Note that HED is a deep-learning based method, and we decided to use a pre-trained model to save time. We used multiple algorithms to gather more qualitative results to compare our results with. Here is the result for some sample images from the dataset. The first image is shown in Figure 1, with a side-by-side comparison with a human drawn representation of the expected edges. In Figure 2, we see the results of the various edge detection methods in comparison with our own algorithm.

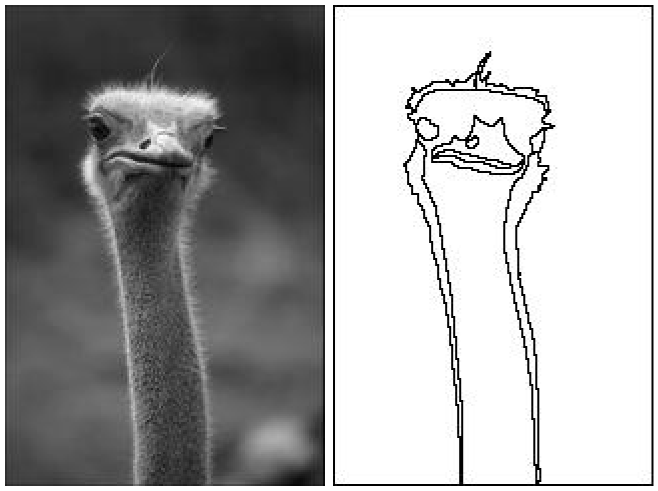

Figure 1. On the left, the original image from BDSD500. On the right, a drawing from a human subject from UC Berkeley for the expected edges of the image.

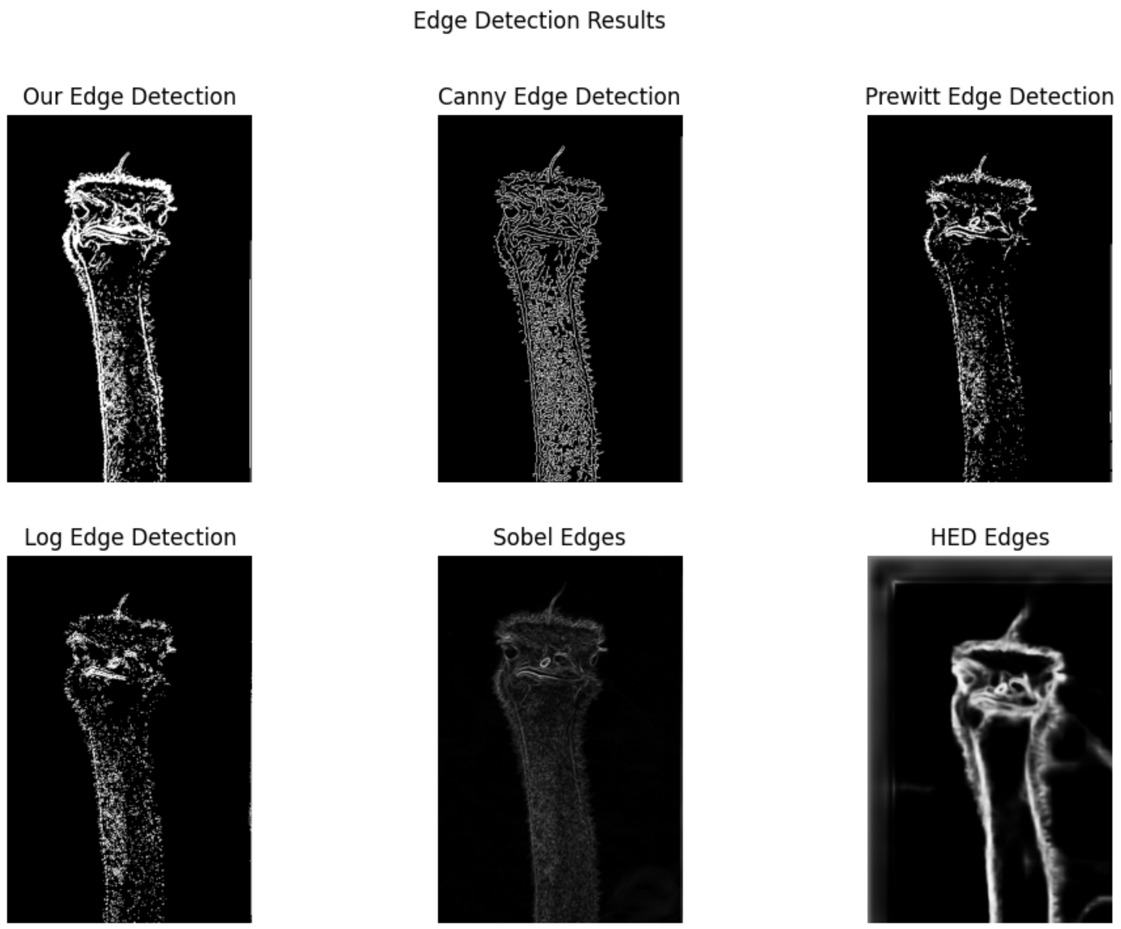

Figure 2. Expected output from our program for the original image in Figure 1.
As we can see, our edge detection does a good job at capturing edges from the image and improving clarity in comparison with other edge detection methods. Now, as for the object detection portion of our code we will examine the next set of results. In Figure 3, the original image with a side by side comparison of the expected objects that we expect to capture in our program. The actual results from our program are shown in Figure 4.

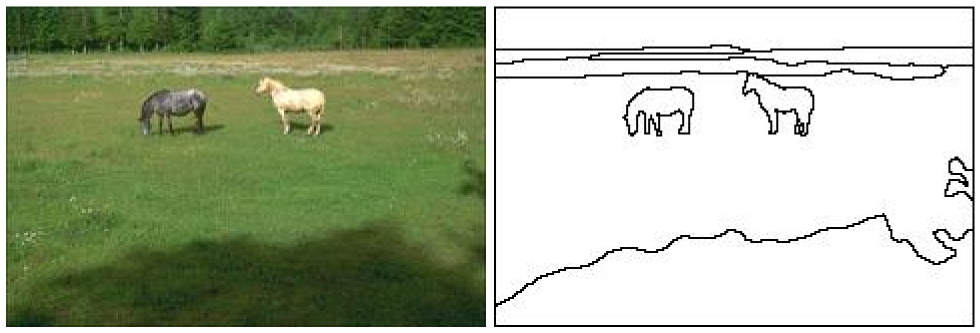

Figure 3. On the left, the original image from BDSD500. On the right, a drawing from a human subject from UC Berkeley for the expected edges of the image.

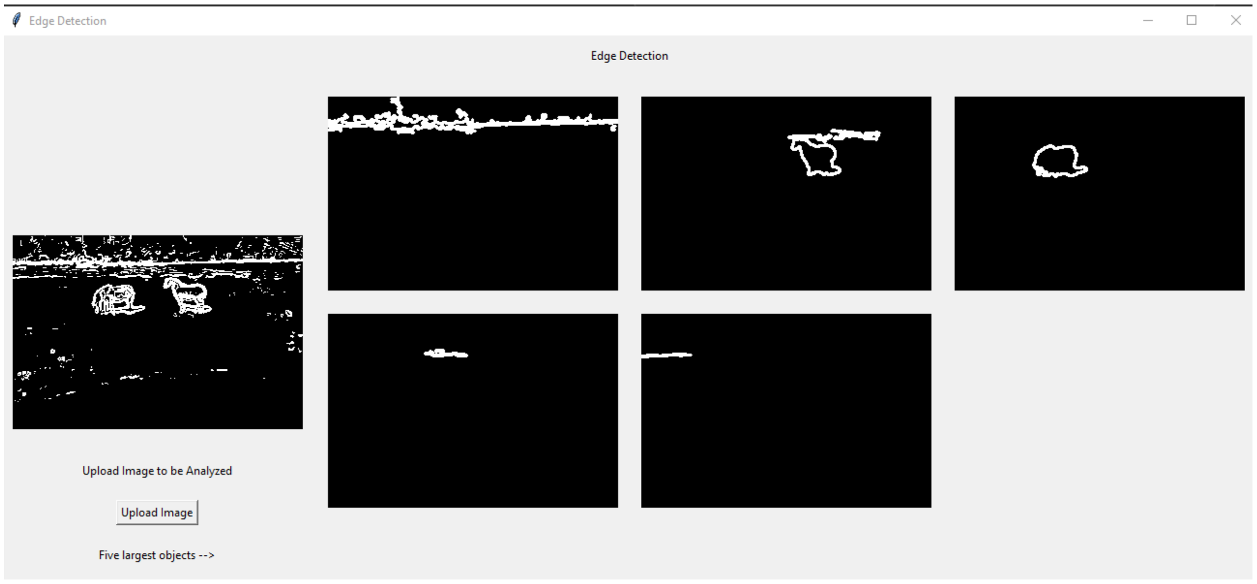

Figure 4. Expected output from our program for the original image in Figure 3.

As we can see, our program did a good job at capturing the two horses from the original picture as well as parts of the tree line in the background. However, when it came to separating the shadow on the grass from the sunny parts, the program did not do as well. We also see that for the 4th and 5th largest objects the result is just a line. This is because we detect objects on the area inside contours, and most of the objects actually are not ‘closed’ entirely despite appearing so. Further improvements should prioritize the closure of detected objects to ensure more precise boundary detection. Also more focus on refining shadow and lighting detection would be beneficial for creating more objects.

Now for some qualitative results, in our evaluation.py program we captured the runtime of all of the edge detection programs we implemented and compared them to the runtime of our own program. This is shown in Figure 5.

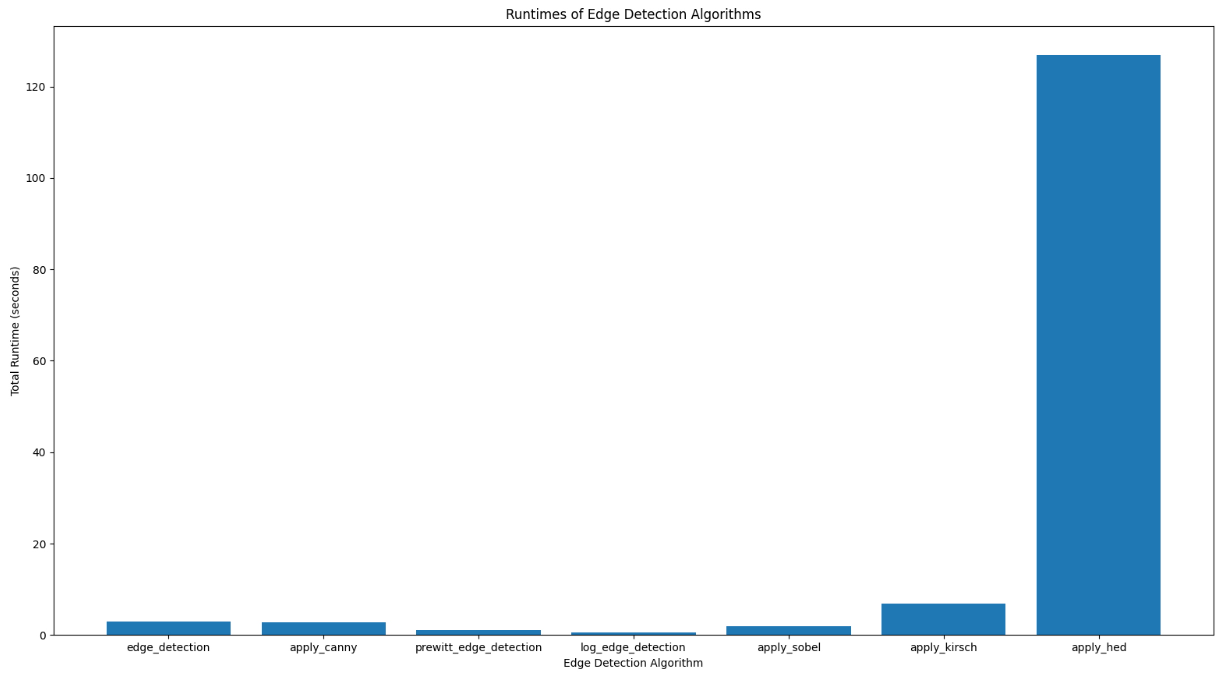

Figure 5. Runtime results for edge detection algorithms

edge_detection - Total Runtime: 2.9661 seconds
apply_canny - Total Runtime: 2.8668 seconds
prewitt_edge_detection - Total Runtime: 1.0700 seconds
log_edge_detection - Total Runtime: 0.6446 seconds
apply_sobel - Total Runtime: 2.0095 seconds
apply_kirsch - Total Runtime: 6.9097 seconds
apply_hed - Total Runtime: 126.9205 seconds

This shows that all of the programs have overall a similar runtime with the exception of apply_hed, which is expected due to it being the only deep-learning based algorithm.

Conclusion:

Overall, the results for our method shows success in certain areas and shortcomings in other areas. We saw that the edge detection struggled in images with a lot of background detail. This is because our edge detection algorithm attempts to filter out less prominent edges, but when the image is almost entirely composed of those small edges as a result most of the image does not get detected by the algorithm. Another shortcoming we noticed was images with backgrounds that blend well with object boundaries. This is more a problem with the contour detection piece, because once again we attempt to get rid of less prominent object boundaries so that we could clearly discern what is or isn't an object. The issue is that when the boundary edge is similar in color to the actual background, the thickness of the edge can be too small for our algorithm to pick up.

As for successes, the first is that our algorithm is fairly efficient. This is because the algorithm has a very similar runtime compared to other traditional edge detection methods as shown in Figure 5. The other success is that the algorithm shows a high accuracy in detecting edges compared to other algorithms. This is shown in Figure 2 when we compare the sharpness of edges and detail retained from the original image. Overall the edge detection method was a success because it is efficient and does pick up edges. As for object detection, we would definitely recommend that more work be done to refine that method because it is very situational to when it performs well or not.

References:

(BSDS500 dataset)
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

(HED documentation and pre-trained model)
https://github.com/s9xie/hed?tab=readme-ov-file

(Literature Review Articles)
https://www-sciencedirect-com.aurarialibrary.idm.oclc.org/science/article/pii/S092523122200248X?via%3Dihub#s0010

A New Method for Edge Detection Under Hazy Environment in Computer Vision | SpringerLink (oclc.org)

https://www.sciencedirect.com/science/article/pii/S221201731200312X?via%3Dihub

https://www-sciencedirect-com.aurarialibrary.idm.oclc.org/science/article/pii/S0925231222008141?via%3Dihub

(Documentation for tkinter interface)
https://docs.python.org/3/library/tk.html

(Documentation for openCV)
https://docs.opencv.org/4.x/

(Documentation for Pillow)
https://pillow.readthedocs.io/en/stable/

(Human drawn edges as shown in the figures)
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/html/dataset/humans/1124-0121-0130.html

https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/html/dataset/humans/1124-0211-0220.html