Yusufu Feierdawusi
4929-7851-40
Feierdaw@usc.edu

EE569 Homework #2
February 20, 2022

Problem 1: Edge Detection

a. Sobel Edge Detector

1. Motivation

In an image, there is usually a lot of information, and we need to be selective about what we want to extract from the image based on our goals. This incentivizes us to identify objects/backgrounds and omit the rest. One of the most basic edge detection methods is 1st Order Derivative Edge Detector, this method first finds the gradient of horizontal and vertical directions, then combines them to find the gradient magnitude (the rate of change) and direction of that pixel [1].

2. Approaches and Procedures

In this problem, we are using the Sobel Operator with two 3x3 kernels, one for gradient of vertical direction (y), one for gradient of horizontal direction (x). In the G_x kernel the 3rd column with positive numbers represents the “right” direction, and the negative number in the first column represents the “left” direction. The G_y kernel follows the same logic with “up” and “down” directions [2]. We first convert the RGB images into grayscale using the given conversion formula. Once we have both kernels, we convolve the image with boundary extension with these two filters, thus for every pixel we have two magnitudes with +/- indicating directions. These normalized results are shown in Figure 1 and 2 below. We then take the square root of the sum of these magnitudes squared to find the gradient magnitude of the pixel as a whole. Furthermore, we need a threshold value to determine if a gradient magnitude is an edge or not. We find the CDF of the gradient map and determine the threshold value to cut off edges.

Finally, we compare every pixel in the gradient map with the threshold value and if it is greater, we say it is an edge and set its value to 0, if it is not, we set it to 255 [3].

3. Experimental Results

i) Tiger.raw

Figure 1: Normalized X-Gradient Result

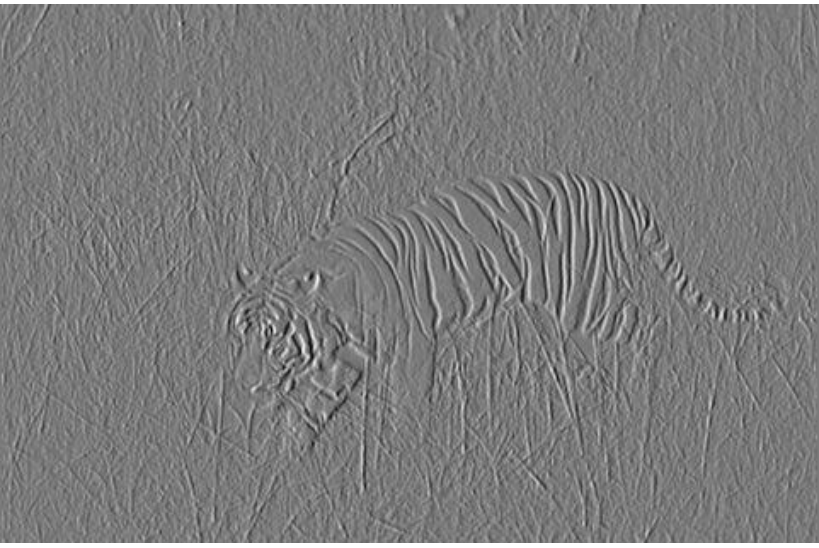


Figure 2: Normalized Y-Gradient Result

Figure 3: Gradient Magnitude Map

Figure 4: CDF of Gradient Magnitude

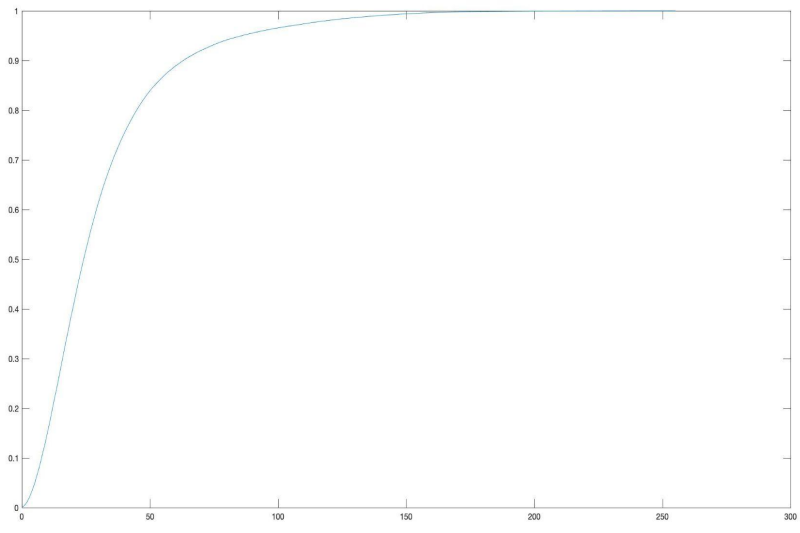


Figure 5: Edge Map with 92% Threshold

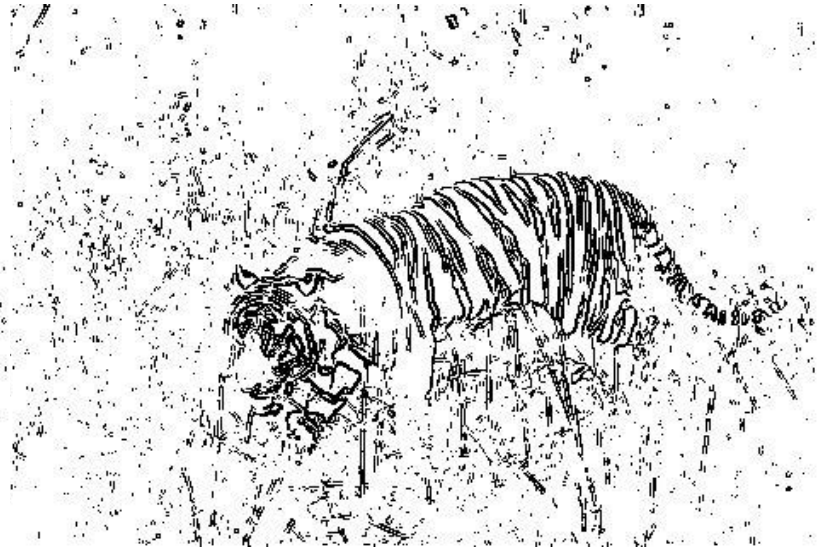


Figure 6: Edge Map with 88% Threshold

i) Pig.raw

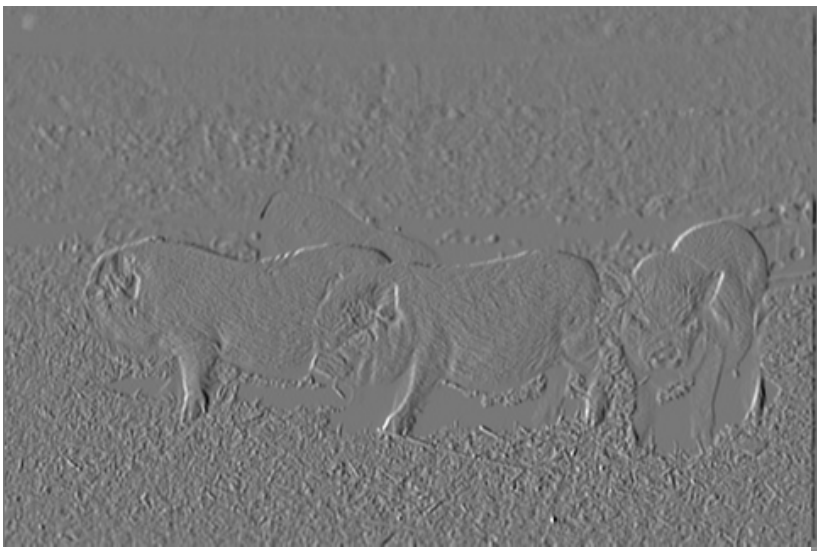


Figure 7: Normalized X-Gradient Result

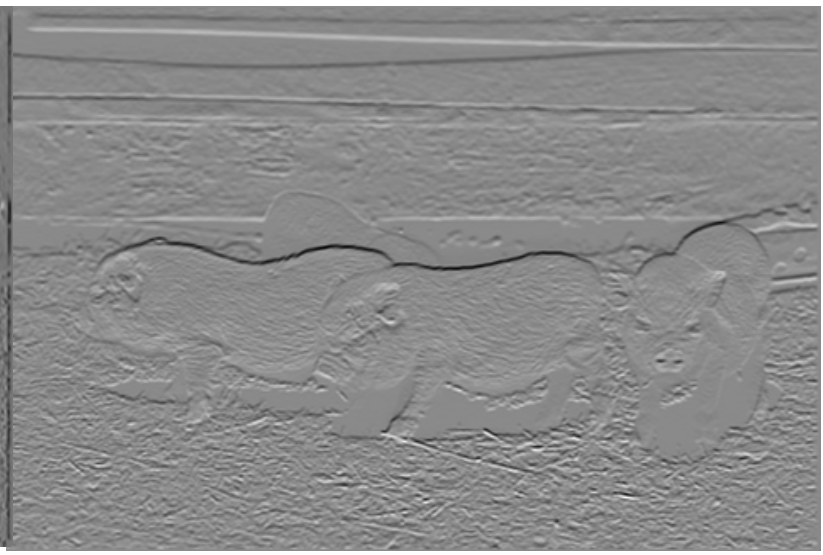


Figure 8: Normalized Y-Gradient Result

Figure 9: Gradient Magnitude Map

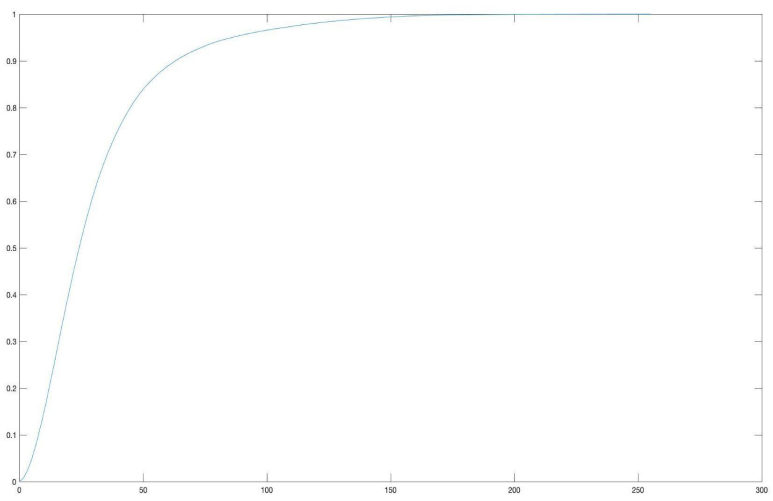


Figure 10: CDF of Gradient Magnitude

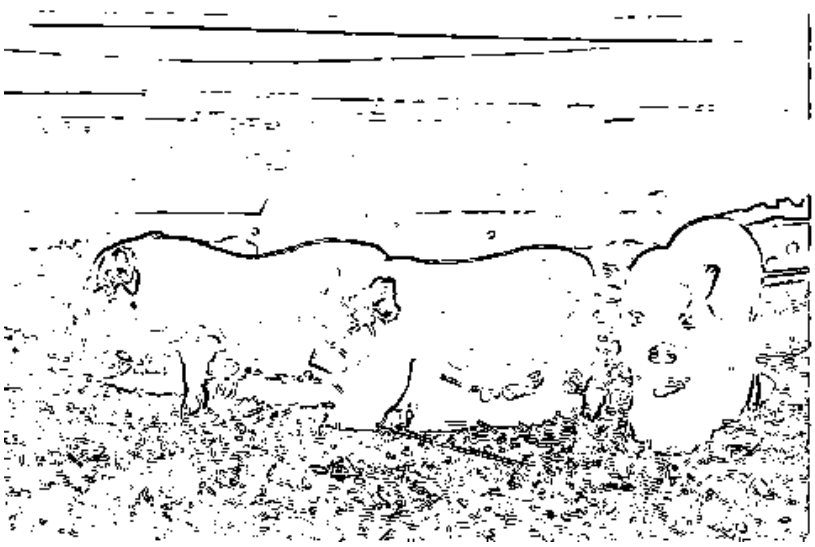


Figure 11: Edge Map with 92% Threshold

Figure 12: Edge Map with 88% Threshold

4. Discussion

1). As we can observe from figures 1 and 7, the normalized x gradient mapping highlights the vertical edges (meaning the lines in the vertical direction are more visible), or the rate of change in the vertical direction. And from figures 2 and 8, we can see that the normalized y gradient mapping highlights the horizontal edges, or the rate of change in the horizontal direction.

2). The gradient map in figure 3 and 9 combines the x and y gradients and highlights edges in both directions. However, as we can see in these figures, there are lots of noises generated by background lines and ill-defined edges.

3). The X-axis on both CDF graphs are pixel values and the Y-axis are the thresholds. If we observe the cumulative probability distribution of both images (figure 4 and 10), we can see the cumulative probability slowly converging to 1 around the pixel value 60. Thus, we can use the corresponding 88% as a threshold, meaning that any pixel gradient magnitude above the grayscale value 60 is considered an edge and assigned a value of 0 and any value below 60 is not an edge and assigned 255. The comparison between different threshold values are also shown in figures 5,6 and figures 11, 12. We can see in both cases that if we increase the threshold we see much less noise because those wrongly defined edges fall below the threshold. However, we also see a decrease of actual edges of objects because although they are an essential edge, their

gradient value is still lower than the threshold. We can conclude that the Sobel Operator is an efficient method to find well-defined edges, but lacks when it comes to poorly defined ones.

b. Canny Edge Detector

1. Motivation

As we can tell from part a, the edge detection performed by the sobel operator cuts off some edges that we want to identify. To solve this problem, we can use the Canny method edge detection. The difference between this method and sobel is that it uses a gaussian filter to remove the noise first, then it uses Non-maximum Suppression to create thinner edges and it also has two cut off thresholds for edges (detailed explanation in section 4).

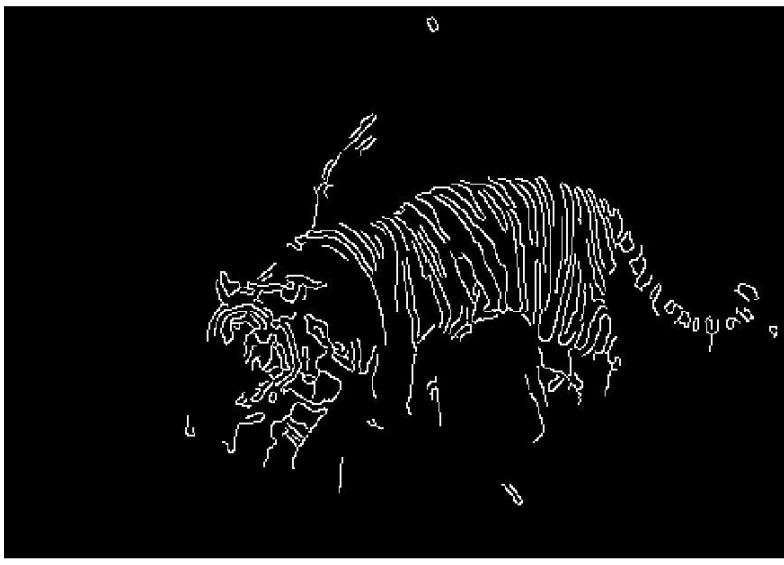
2. Approaches and Procedures

We first convert the RGB image into a grayscale image using the rgb2gray() function in Matlab. Then we use the edge detection function edge() from Matlab Image Processing ToolBox to perform Canny edge detection. This function takes in a grayscale image and performs edge detection using the method indicated by the user (i.e. Sobel, Prewitt, Canny and other methods). It also has another parameter for users to specify threshold values. For this problem, I have tested two different thresholds values for low and high and the results are shown in the next section below [4].

3. Experimental Results

Figure 1: Canny Edge Detection (low=0.2, high=0.6)

Figure 2: Canny Edge Detection (low=0.2, high=0.4)



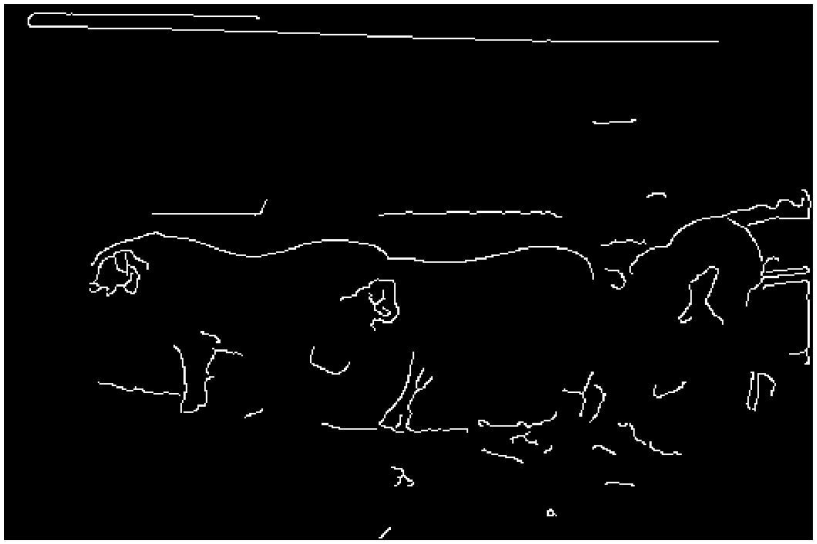


Figure 3: Canny Edge Detection (low=0.2, high=0.4)

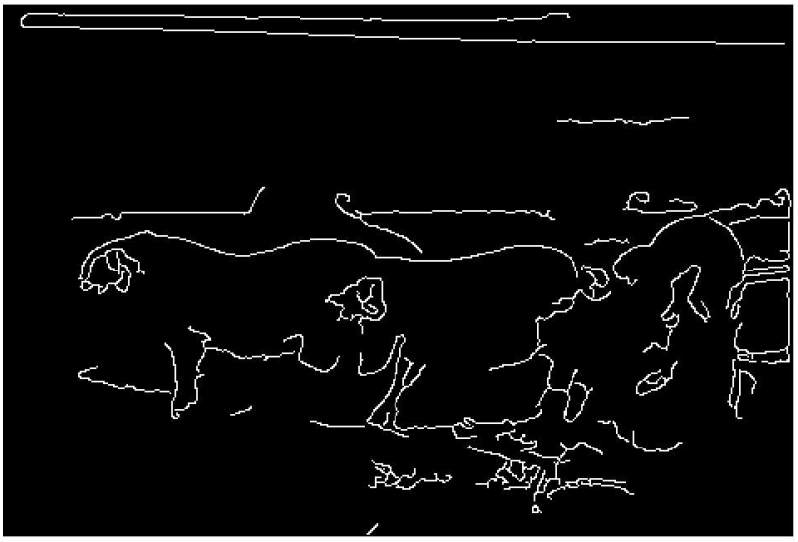


Figure 4: Canny Edge Detection (low=0.1, high=0.4)

4. Discussions

1). The main purpose of Non-maximum suppression is to find a line of pixels that we can confidently say is an edge and omits other less confident pixels surrounding those local maximum pixels, sequentially making the edges thinner and less blurry compared to the Sobel edge detector. On the technical side, after smoothing the image using a gaussian filter, we find an edge, then we compare the gradient magnitude of every pixel following that gradient direction. During the comparison, we suppress the value of a pixel to 0 if there is another pixel with a higher magnitude [5].

2). The high threshold works the same way as the Sobel method where any gradient magnitude higher than the high threshold value is confidently considered an edge. Similarly, any gradient magnitude that is lower than the low threshold is omitted and not considered an edge. For the gradient magnitudes between the low and high thresholds, they are considered an edge if and only if they are connected to an edge above the high threshold [5].

3). When we compare both figure 1 and 2 with the result we got from the Sobel detector, we can see that it almost reduces all noise (the grass in the background). And when we compare them to each other, we can see that a bigger high threshold removes the background noise completely while preserving most of the essential edges on the tiger. Not only does the smaller high threshold preserve almost all edges on the tiger including its tail, it also includes what seems like a broken tree branch in the original image. For figure 3 and 4, the difference is in the low threshold value. As we can observe, the smaller the low threshold is, the more edges and

noise it contains. However, if we focus on 2 of the edges at the top of the image, we can see that the edge in the 10% threshold image is longer, which implies that the magnitude is between 10% and 40%, and connected to a confident edge. Furthermore, this magnitude cannot possibly be above 40% because it does not show up in the first figure, which also has a 40% high threshold.

c. Structured Edge

5. Motivation

When we perform classical edge detection with kernels like the Sobel filter, we measure the gradient magnitude on one single point. Thus, this would sometimes lead to edges being omitted, especially when the color is the same inside and outside that edge. For example, if the background is red and there is also a red object on that background, classic detection would not be able to recognize that edge. However, we can use contour detection methods to overcome this problem [6].

6. Approaches and Procedures

We use the open source Structured Edge Detection Toolbox by Piotr Dollar for this question. We first reference the pre-trained model with model = load(ground truth file), we then configure the given parameters such as number of forest tree evaluation, Non-maximum suppression and maximum number of threads for evaluation. Next we use edgeEval(), which uses a validation set of images to evaluate the SE detector. Lastly, we read the image with imread and use edgeDetect(model) to obtain the final results.

7. Experimental Results

Figure 1: Probability Edge Map for pig.png

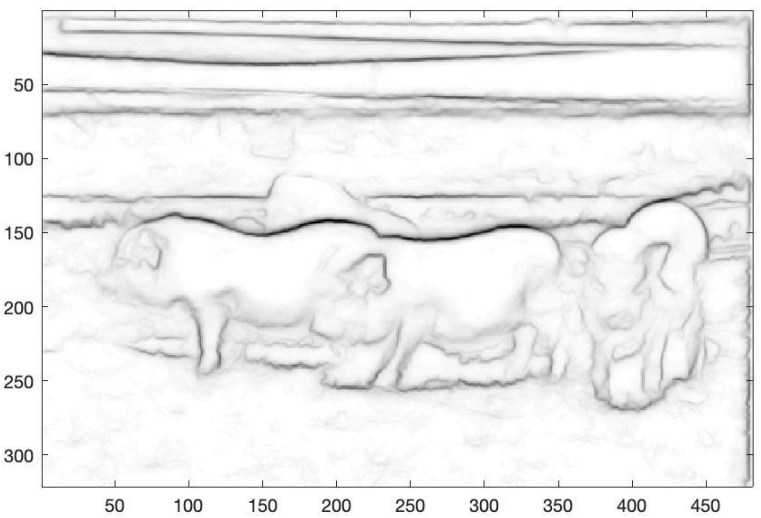
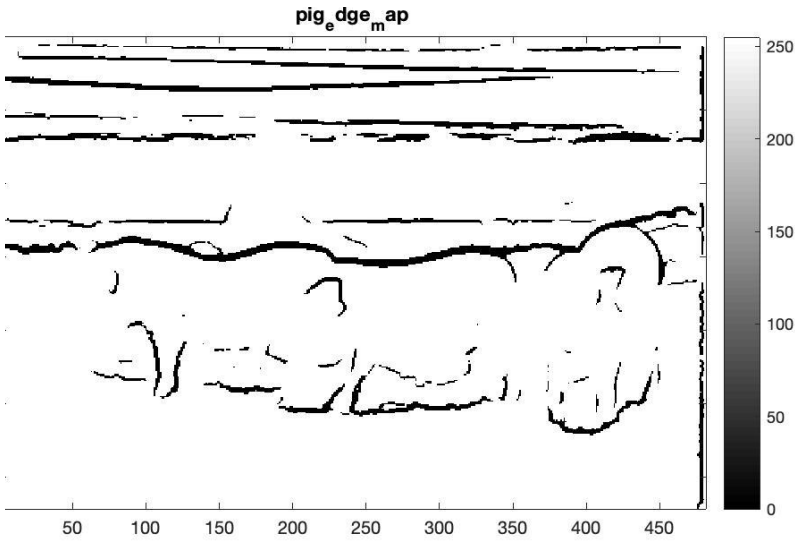


Figure 2: Binary Edge map for pig.png with p > 0.2



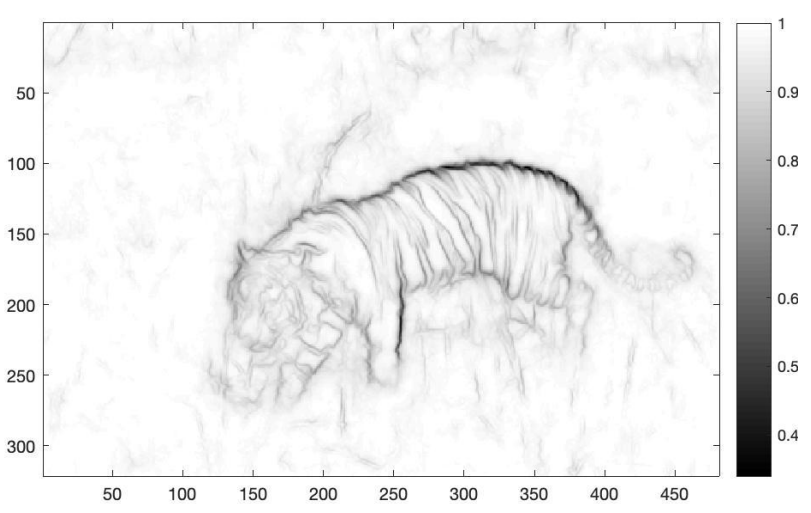


Figure 3: Probability Edge Map for tiger.png

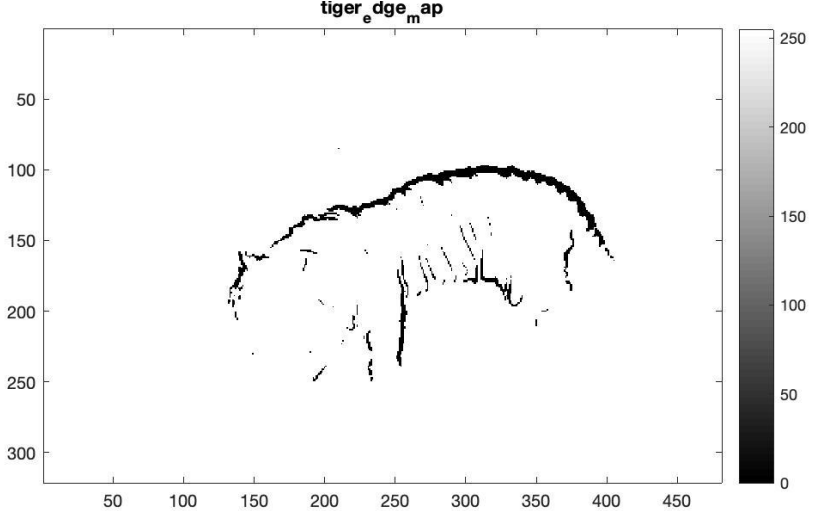
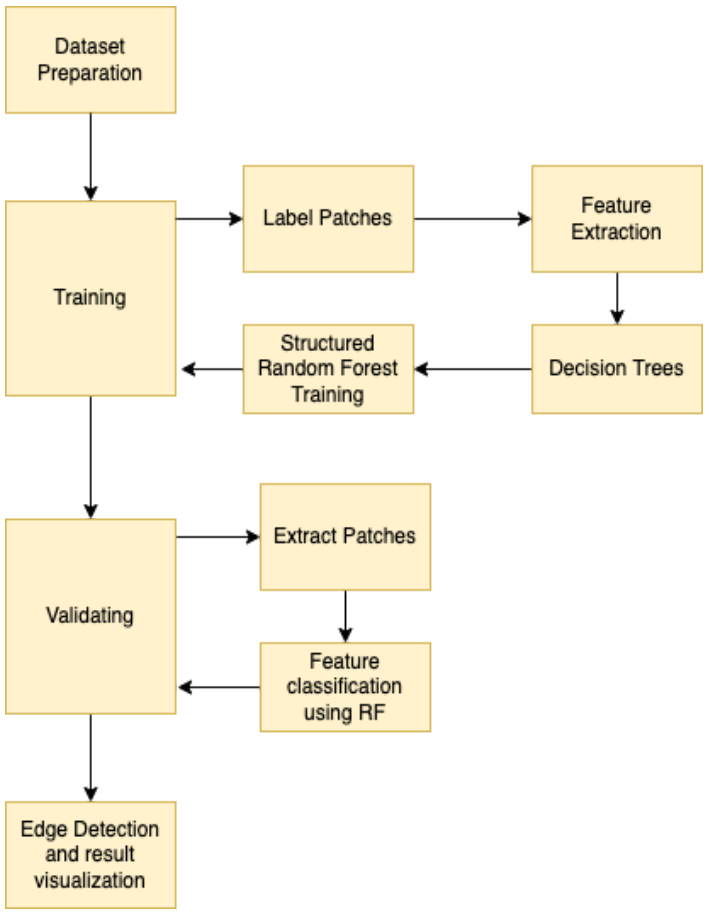


Figure 4: Binary Edge map for tiger.png with p > 0.2

8. Discussions

1). Following the flowchart of the Structured Edge algorithm, the first step is dataset preparation, meaning separating the data into a training set and testing set, and obtaining the ground truth annotations from multiple sources and averaging them into one file. We then can start the training process, we label patches that are 35 pixels by 35 pixels while comparing these patches with the corresponding ground truth patch, if it is an edge in the ground truth file, we label it as such. Next, we can extract many different features like gradient or color channels for the edge patches and use them to perform the structured random forest training (See #2 for a detailed explanation of RF). After the training, we have an edge detection model we need to test to verify its correctness. To validate, we take our testing dataset and extract all the 35px by 35px patches in the image, then use the trained Structured Random Forest classifier to extract the features at each patch. Finally, we can use our validated Random Forest classifier to perform the edge detection on the target image [7].



2). The goal of a decision tree is to provide us with a prediction based on the features provided. We iterate through the given dataset and group nodes that give us the same outcome as a tree branch. We continue to apply this concept until we reach leaf nodes that cannot obtain a different outcome using any features. Thus, any new observations that include all the features used until these leaf nodes will give us the same prediction. The Random Forest Classification contains multiple decision trees like these to produce a prediction. Each decision tree within the "forest" of trees is specifically selected to be uncorrelated or lightly correlated with any other decision tree in the forest. This means that these trees will use different **features extracted from the patches** in the SE algorithm to predict if it has an edge or not. Thus, this will allow different trees to produce different predictions. From these different predictions, the RF classifier produces a probability function that favors the outcome that was predicted the most. And this is the main principle of the RF classifier where if one decision tree produces a wrong prediction, there are other uncorrelated decision trees with correct results to lower the probability of that individual prediction [8].

3). The parameters that I think work the best are sharpen=2, nTreesEval=5 and nms=0. I set the sharpening of the image to the max value, 2, because sharpening would make the edges in an image more clearly defined. Since time is not a big concern for this problem, I have maximized the number of trees to evaluate at each patch so that I can get the most accurate results. If we compare figure 4 from part b to figure 2 from this part, we can see that although the Canny detector shows more edges than the SE algorithm, some of these edges are false positive edges that should not be considered. On the other hand, the SE algorithm signifies most confident edges by stronger/thicker lines.

d. Performance Evaluation

1.

Figure 1: Evaluation Table for Sobel detector with Pig.png

	1 Thresholds	2 GT_1_Precision	3 GT_2_Precision	4 GT_3_Precision	5 GT_4_Precision	6 GT_5_Precision	7 GT_1_Recall	8 GT_2_Recall	9 GT_3_Recall	10 GT_4_Recall	11 GT_5_Recall	12 GT_P_Means	13 GT_R_Means	14 F_Score	15
1	0.0323	0.0542	0.0600	0.0911	0.1210	0.0925	0.9633	0.9875	0.9358	0.9889	0.9734	0.4537	0.3932	0.4012	
2	0.0645	0.0543	0.0595	0.0894	0.1194	0.0918	0.9720	0.9839	0.9235	0.9820	0.9720	0.4563	0.3805	0	
3	0.0968	0.0605	0.0650	0.0995	0.1331	0.1019	0.9633	0.9580	0.9151	0.9738	0.9605	0.4746	0.3217	0	
4	0.1290	0.0715	0.0765	0.1172	0.1564	0.1195	0.9430	0.9334	0.8931	0.9485	0.9325	0.4910	0.3177	0	
5	0.1613	0.0861	0.0932	0.1403	0.1869	0.1439	0.8922	0.8936	0.8403	0.8903	0.8825	0.4747	0.3367	0	
6	0.1935	0.1007	0.1098	0.1628	0.2130	0.1656	0.8265	0.8333	0.7719	0.8032	0.8038	0	0	0	
7	0.2258	0.1179	0.1255	0.1853	0.2330	0.1857	0.7598	0.7479	0.6896	0.6901	0.7080	0	0	0	
8	0.2581	0.1370	0.1440	0.2043	0.2558	0.2061	0.6897	0.6705	0.5941	0.5920	0.6140	0	0	0	
9	0.2903	0.1646	0.1669	0.2250	0.2805	0.2378	0.6259	0.5869	0.4941	0.4902	0.5350	0	0	0	
10	0.3226	0.1910	0.1893	0.2469	0.2896	0.2730	0.5515	0.5056	0.4118	0.3843	0.4664	0	0	0	
11	0.3548	0.2246	0.2188	0.2738	0.3111	0.3020	0.4896	0.4412	0.3448	0.3116	0.3895	0	0	0	
12	0.3871	0.2695	0.2616	0.2925	0.3079	0.2978	0.4248	0.3813	0.2663	0.2230	0.2777	0	0	0	
13	0.4194	0.3134	0.3055	0.3292	0.3296	0.3155	0.3644	0.3286	0.2211	0.1761	0.2170	0	0	0	
14	0.4516	0.3687	0.3642	0.3727	0.3790	0.3590	0.3122	0.2852	0.1823	0.1475	0.1799	0	0	0	
15	0.4839	0.4096	0.4110	0.4118	0.4147	0.4031	0.2726	0.2530	0.1583	0.1268	0.1587	0	0	0	
16	0.5161	0.4588	0.4606	0.4588	0.4588	0.4532	0.2392	0.2222	0.1382	0.1100	0.1398	0	0	0	
17	0.5484	0.5220	0.5255	0.5197	0.5185	0.5150	0.2180	0.2030	0.1253	0.0995	0.1273	0	0	0	
18	0.5806	0.5819	0.5833	0.5775	0.5775	0.5789	0.1924	0.1784	0.1103	0.0877	0.1132	0	0	0	
19	0.6129	0.6277	0.6313	0.6295	0.6259	0.6241	0.1687	0.1569	0.0977	0.0773	0.0992	0	0	0	
20	0.6452	0.6660	0.6745	0.6702	0.6660	0.6660	0.1503	0.1408	0.0874	0.0691	0.0889	0	0	0	
21	0.6774	0.6938	0.7037	0.6963	0.6963	0.6988	0.1358	0.1274	0.0787	0.0626	0.0809	0	0	0	
22	0.7097	0.7348	0.7431	0.7376	0.7376	0.7403	0.1286	0.1203	0.0745	0.0593	0.0766	0	0	0	
23	0.7419	0.7682	0.7781	0.7715	0.7715	0.7715	0.1121	0.1051	0.0650	0.0518	0.0666	0	0	0	
24	0.7742	0.7879	0.7917	0.7879	0.7917	0.7879	0.1005	0.0934	0.0581	0.0464	0.0595	0	0	0	
25	0.8065	0.8117	0.8117	0.8117	0.8161	0.8072	0.0875	0.0809	0.0505	0.0404	0.0515	0	0	0	
26	0.8387	0.8218	0.8218	0.8218	0.8276	0.8218	0.0691	0.0639	0.0399	0.0320	0.0409	0	0	0	
27	0.8710	0.8433	0.8433	0.8433	0.8433	0.8358	0.0546	0.0505	0.0315	0.0251	0.0320	0	0	0	
28	0.9032	0.8900	0.8900	0.8900	0.8900	0.8800	0.0430	0.0398	0.0248	0.0198	0.0252	0	0	0	
29	0.9355	0.8904	0.8904	0.8904	0.8904	0.8767	0.0314	0.0291	0.0181	0.0144	0.0183	0	0	0	
30	0.9677	0.8889	0.8889	0.8889	0.8889	0.8889	0.0155	0.0143	0.0089	0.0071	0.0092	0	0	0	
31															
32															

Figure 2: Evaluation Table for SE detector with Pig.png

	1 Thresholds	2 GT_1_Precision	3 GT_2_Precision	4 GT_3_Precision	5 GT_4_Precision	6 GT_5_Precision	7 GT_1_Recall	8 GT_2_Recall	9 GT_3_Recall	10 GT_4_Recall	11 GT_5_Recall	12 GT_P_Means	13 GT_R_Means	14 F_Score	15
1	0.0323	0.1738	0.1907	0.3029	0.3837	0.2749	0.7289	0.7398	0.7339	0.7397	0.6823	0.6355	0.2625	0.3838	
2	0.0645	0.2721	0.2883	0.4402	0.5617	0.4111	0.8951	0.8771	0.8364	0.8492	0.8001	0.6468	0.2613	0	
3	0.0968	0.2854	0.3041	0.4881	0.6279	0.4569	0.8405	0.8283	0.8303	0.8498	0.7961	0.7380	0.2550	0	
4	0.1290	0.2859	0.3066	0.5127	0.6663	0.4858	0.7540	0.7479	0.7811	0.8076	0.7581	0.8780	0.2722	0	
5	0.1613	0.2820	0.3108	0.5309	0.7042	0.5037	0.6626	0.6755	0.7205	0.7606	0.7003	0.7349	0.2525	0	
6	0.1935	0.2806	0.3126	0.5356	0.7247	0.5166	0.5853	0.6030	0.6452	0.6946	0.6374	0	0	0	
7	0.2258	0.2774	0.3086	0.5316	0.7301	0.5149	0.4978	0.5123	0.5511	0.6022	0.5468	0	0	0	
8	0.2581	0.2756	0.3089	0.5415	0.7401	0.5283	0.4239	0.4394	0.4810	0.5231	0.4807	0	0	0	
9	0.2903	0.2828	0.3164	0.5588	0.7606	0.5626	0.3625	0.3751	0.4137	0.4480	0.4267	0	0	0	
10	0.3226	0.2929	0.3227	0.5591	0.7920	0.5898	0.3137	0.3196	0.3459	0.3898	0.3737	0	0	0	
11	0.3548	0.3191	0.3464	0.5610	0.8359	0.5997	0.2716	0.2727	0.2758	0.3270	0.3020	0	0	0	
12	0.3871	0.3904	0.4160	0.5886	0.9129	0.6213	0.2426	0.2392	0.2113	0.2608	0.2285	0	0	0	
13	0.4194	0.4533	0.4706	0.5872	0.9391	0.5842	0.2160	0.2074	0.1616	0.2057	0.1647	0	0	0	
14	0.4516	0.4821	0.4898	0.5921	0.9770	0.5882	0.1822	0.1712	0.1293	0.1697	0.1315	0	0	0	
15	0.4839	0.5298	0.5298	0.5877	0.9851	0.5877	0.1547	0.1430	0.0991	0.1322	0.1015	0	0	0	
16	0.5161	0.6031	0.6031	0.6368	1	0.6368	0.1300	0.1203	0.0793	0.0991	0.0812	0	0	0	
17	0.5484	0.7508	0.7508	0.7571	1	0.7571	0.1150	0.1064	0.0670	0.0704	0.0686	0	0	0	
18	0.5806	0.9149	0.9149	0.9149	1	0.9149	0.1039	0.0961	0.0600	0.0522	0.0615	0	0	0	
19	0.6129	0.9340	0.9340	0.9340	1	0.9340	0.0957	0.0885	0.0553	0.0471	0.0566	0	0	0	
20	0.6452	0.9791	0.9791	0.9791	1	0.9791	0.0904	0.0836	0.0522	0.0424	0.0535	0	0	0	
21	0.6774	1	1	1	1	1	0.0788	0.0729	0.0455	0.0362	0.0466	0	0	0	
22	0.7097	1	1	1	1	1	0.0638	0.0590	0.0369	0.0293	0.0377	0	0	0	
23	0.7419	1	1	1	1	1	0.0391	0.0362	0.0226	0.0180	0.0232	0	0	0	
24	0.7742	1	1	1	1	1	0.0242	0.0224	0.0140	0.0111	0.0143	0	0	0	
25	0.8065	1	1	1	1	1	0.0039	0.0036	0.0022	0.0018	0.0023	0	0	0	
26	0.8387	1	1	1	1	1	0	0	0	0	0	0	0	0	
27	0.8710	1	1	1	1	1	0	0	0	0	0	0	0	0	
28	0.9032	1	1	1	1	1	0	0	0	0	0	0	0	0	
29	0.9355	1	1	1	1	1	0	0	0	0	0	0	0	0	
30	0.9677	1	1	1	1	1	0	0	0	0	0	0	0	0	
31															

Figure 3: Evaluation Table for Sobel detector with Tiger.png

	1 Thresholds	2 GT_1_Precision	3 GT_2_Precision	4 GT_3_Precision	5 GT_4_Precision	6 GT_5_Precision	7 GT_1_Recall	8 GT_2_Recall	9 GT_3_Recall	10 GT_4_Recall	11 GT_5_Recall	12 GT_P_Means	13 GT_R_Means	14 F_Score
1	0.0323	0.0234	0.0248	0.0278	0.1085	0.0323	0.9973	0.9991	0.9969	0.9854	0.9960	0.0951	0.4130	0.2985
2	0.0645	0.0202	0.0214	0.0240	0.0930	0.0279	1	1	1	0.9818	1	0.0938	0.4118	0
3	0.0968	0.0248	0.0263	0.0295	0.1150	0.0338	1	1	1	0.9887	0.9887	0.1086	0.4143	0
4	0.1290	0.0329	0.0349	0.0392	0.1539	0.0440	1	1	1	0.9951	0.9662	0.7203	0.4494	0
5	0.1613	0.0466	0.0495	0.0556	0.2177	0.0598	0.9991	1	1	0.9932	0.9277	0.1532	0.3695	0
6	0.1935	0.0632	0.0675	0.0757	0.2955	0.0774	0.9863	0.9931	0.9931	0.9826	0.8746	0	0	0
7	0.2258	0.0813	0.0881	0.0984	0.3857	0.0955	0.9560	0.9767	0.9715	0.9660	0.8129	0	0	0
8	0.2581	0.0987	0.1073	0.1211	0.4857	0.1117	0.8937	0.9162	0.9207	0.9367	0.7319	0	0	0
9	0.2903	0.1151	0.1239	0.1409	0.5855	0.1287	0.8011	0.8133	0.8237	0.8678	0.6483	0	0	0
10	0.3226	0.1259	0.1376	0.1556	0.6696	0.1435	0.7039	0.7252	0.7306	0.7973	0.5806	0	0	0
11	0.3548	0.1288	0.1401	0.1580	0.7348	0.1465	0.6068	0.6223	0.6251	0.7372	0.4997	0	0	0
12	0.3871	0.1319	0.1410	0.1606	0.7919	0.1477	0.5225	0.5264	0.5343	0.6679	0.4234	0	0	0
13	0.4194	0.1302	0.1387	0.1587	0.8270	0.1491	0.4345	0.4365	0.4450	0.5879	0.3603	0	0	0
14	0.4516	0.1329	0.1353	0.1553	0.8729	0.1532	0.3538	0.3397	0.3472	0.4949	0.2953	0	0	0
15	0.4839	0.1419	0.1402	0.1572	0.9001	0.1616	0.2970	0.2766	0.2764	0.4012	0.2449	0	0	0
16	0.5161	0.1378	0.1312	0.1488	0.9085	0.1604	0.2291	0.2057	0.2079	0.3217	0.1931	0	0	0
17	0.5484	0.1491	0.1327	0.1559	0.9142	0.1685	0.1833	0.1538	0.1609	0.2394	0.1500	0	0	0
18	0.5806	0.1571	0.1421	0.1688	0.9220	0.1870	0.1347	0.1150	0.1216	0.1685	0.1161	0	0	0
19	0.6129	0.1753	0.1612	0.1909	0.9343	0.2097	0.1027	0.0890	0.0939	0.1166	0.0889	0	0	0
20	0.6452	0.1797	0.1843	0.2143	0.9424	0.2189	0.0715	0.0691	0.0716	0.0799	0.0630	0	0	0
21	0.6774	0.1871	0.1599	0.2075	0.9558	0.2143	0.0504	0.0406	0.0470	0.0549	0.0418	0	0	0
22	0.7097	0.1589	0.1402	0.1776	0.9486	0.1963	0.0312	0.0259	0.0293	0.0396	0.0279	0	0	0
23	0.7419	0.1241	0.1103	0.1379	0.9793	0.1931	0.0165	0.0138	0.0154	0.0277	0.0186	0	0	0
24	0.7742	0.1053	0.0947	0.1158	0.9684	0.1895	0.0092	0.0078	0.0085	0.0180	0.0119	0	0	0
25	0.8065	0.0882	0.0882	0.0882	0.9706	0.2059	0.0055	0.0052	0.0046	0.0129	0.0093	0	0	0
26	0.8387	0.0612	0.0612	0.0612	0.9592	0.2449	0.0027	0.0026	0.0023	0.0092	0.0080	0	0	0
27	0.8710	0.0323	0.0323	0.0323	0.9677	0.1935	9.1659e-04	8.6430e-04	7.6982e-04	0.0059	0.0040	0	0	0
28	0.9032	0	0	0	1	0.1176	0	0	0	0.0033	0.0013	0	0	0
29	0.9355	0	0	0	1	0.2500	0	0	0	0.0016	0.0013	0	0	0
30	0.9677	0	0	0	1	0.3333	0	0	0	5.8571e-04	6.6357e-04	0	0	0
31														

Figure 4: Evaluation Table for SE detector with Tiger.png

	1 Thresholds	2 GT_1_Precision	3 GT_2_Precision	4 GT_3_Precision	5 GT_4_Precision	6 GT_5_Precision	7 GT_1_Recall	8 GT_2_Recall	9 GT_3_Recall	10 GT_4_Recall	11 GT_5_Recall	12 GT_P_Means	13 GT_R_Means	14 F_Score
1	0.0323	0.1328	0.1369	0.1542	0.4590	0.1682	0.8662	0.8418	0.8445	0.6376	0.7943	0.8213	0.2004	0.2882
2	0.0645	0.1929	0.2094	0.2327	0.7012	0.2192	0.8799	0.9006	0.8915	0.6814	0.7240	0.8316	0.2046	0
3	0.0968	0.2576	0.3038	0.3368	0.8527	0.3047	0.7516	0.8358	0.8253	0.5299	0.6437	0.8366	0.1929	0
4	0.1290	0.3152	0.3720	0.3934	0.9393	0.3844	0.6095	0.6785	0.6390	0.3870	0.5382	0.9622	0.1066	0
5	0.1613	0.3886	0.4464	0.4507	0.9721	0.4486	0.4986	0.5402	0.4858	0.2657	0.4167	0.8202	0.1620	0
6	0.1935	0.4900	0.5375	0.5470	0.9863	0.5375	0.4253	0.4399	0.3988	0.1824	0.3378	0	0	0
7	0.2258	0.6089	0.6471	0.6502	0.9936	0.6200	0.3511	0.3518	0.3149	0.1220	0.2588	0	0	0
8	0.2581	0.7339	0.7583	0.7627	0.9956	0.7118	0.3034	0.2956	0.2648	0.0877	0.2130	0	0	0
9	0.2903	0.8309	0.8338	0.8457	0.9941	0.8071	0.2566	0.2429	0.2194	0.0654	0.1805	0	0	0
10	0.3226	0.8962	0.9000	0.9000	0.9962	0.8654	0.2136	0.2022	0.1801	0.0506	0.1493	0	0	0
11	0.3548	0.9330	0.9330	0.9286	0.9955	0.9018	0.1916	0.1806	0.1601	0.0435	0.1340	0	0	0
12	0.3871	0.9430	0.9482	0.9534	0.9948	0.9275	0.1668	0.1582	0.1416	0.0375	0.1188	0	0	0
13	0.4194	0.9455	0.9515	0.9576	0.9939	0.9394	0.1430	0.1357	0.1216	0.0320	0.1029	0	0	0
14	0.4516	0.9710	0.9710	0.9855	0.9928	0.9565	0.1228	0.1158	0.1047	0.0267	0.0876	0	0	0
15	0.4839	1	1	1	1	0.9608	0.0935	0.0882	0.0785	0.0199	0.0650	0	0	0
16	0.5161	1	1	1	1	0.9474	0.0697	0.0657	0.0585	0.0148	0.0478	0	0	0
17	0.5484	1	1	1	1	0.9583	0.0440	0.0415	0.0370	0.0094	0.0305	0	0	0
18	0.5806	1	1	1	1	0.9474	0.0174	0.0164	0.0146	0.0037	0.0119	0	0	0
19	0.6129	1	1	1	1	1	0.0073	0.0069	0.0062	0.0016	0.0053	0	0	0
20	0.6452	1	1	1	1	1	9.1659e-04	8.6430e-04	7.6982e-04	1.9524e-04	6.6357e-04	0	0	0
21	0.6774	1	1	1	1	1	0	0	0	0	0	0	0	0
22	0.7097	1	1	1	1	1	0	0	0	0	0	0	0	0
23	0.7419	1	1	1	1	1	0	0	0	0	0	0	0	0
24	0.7742	1	1	1	1	1	0	0	0	0	0	0	0	0
25	0.8065	1	1	1	1	1	0	0	0	0	0	0	0	0
26	0.8387	1	1	1	1	1	0	0	0	0	0	0	0	0
27	0.8710	1	1	1	1	1	0	0	0	0	0	0	0	0
28	0.9032	1	1	1	1	1	0	0	0	0	0	0	0	0
29	0.9355	1	1	1	1	1	0	0	0	0	0	0	0	0
30	0.9677	1	1	1	1	1	0	0	0	0	0	0	0	0
31														

The first column of these tables represents the threshold values, columns 2-6 are precision scores for each threshold, columns 7-11 are recall scores for each threshold, column 12-13 are mean precision and recall scores for each ground truth and column 14 is the final F measure for overall precision and recall. The clear comparison between these two edge detectors is that Sobel has a higher **overall average** F-score than SE. However, as we can see in the next section that this may not be true for the **max** F-scores. The cons of the Sobel detector is that it has significantly less precision scores across all ground truths compared to the SE detector. Vice versa, the cons of the SE detectors is that it has a bit less recall scores across all ground truths compared to the Sobel detectors. This means that the SE method is better at identifying edges but also contains more false negative edges. Where the Sobel method is better at avoiding false edges but fails to include all the true edges.

2.

Figure 1: F-measure Graph for Sobel detector with Pig.png

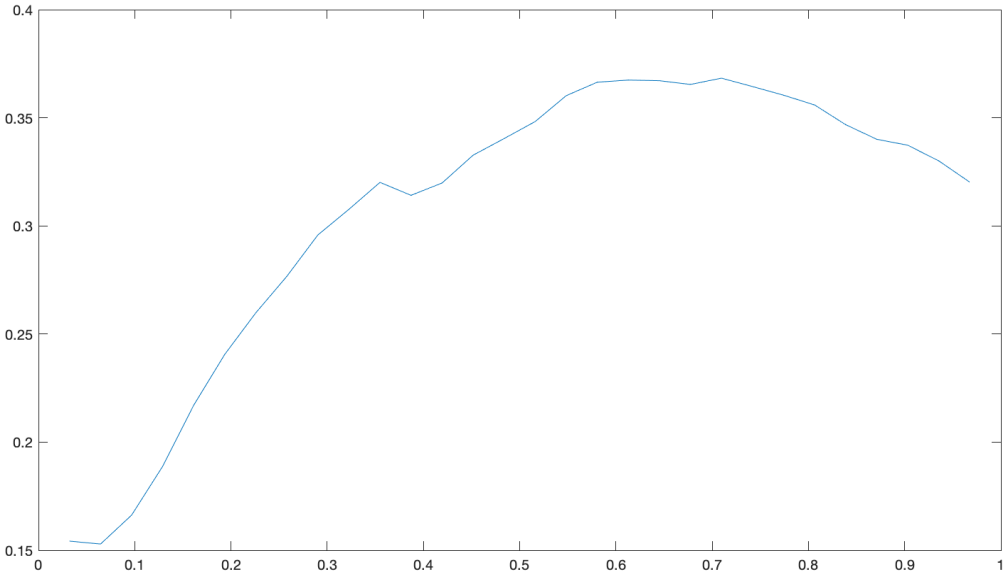


Figure 2: F-measure Graph for SE detector with Pig.png

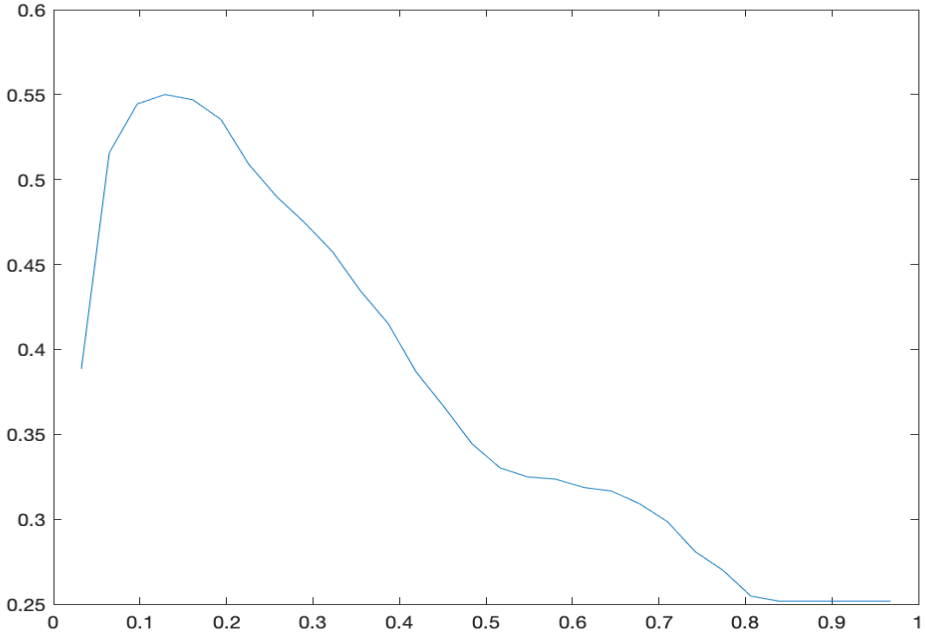


Figure 3: F-measure Graph for Sobel detector with Tiger.png

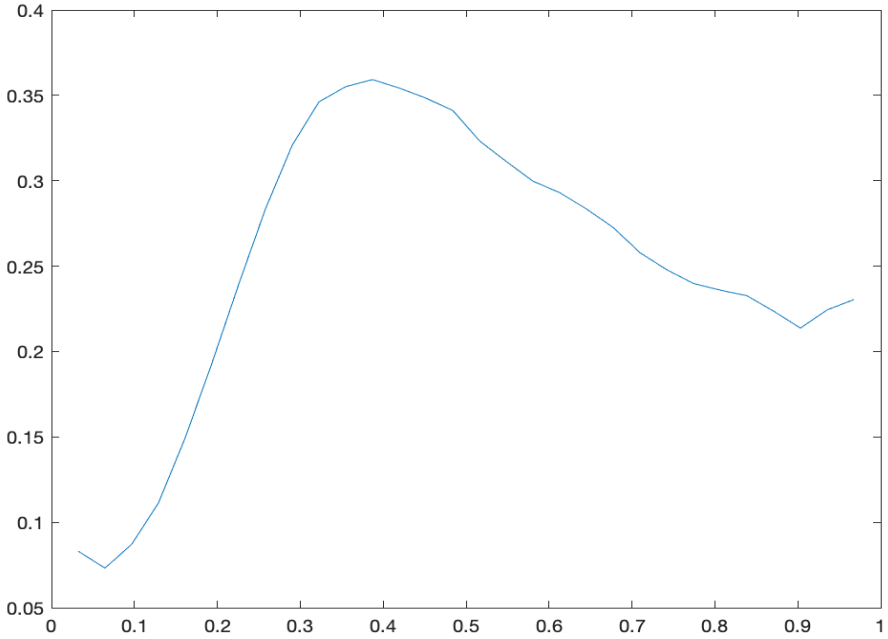
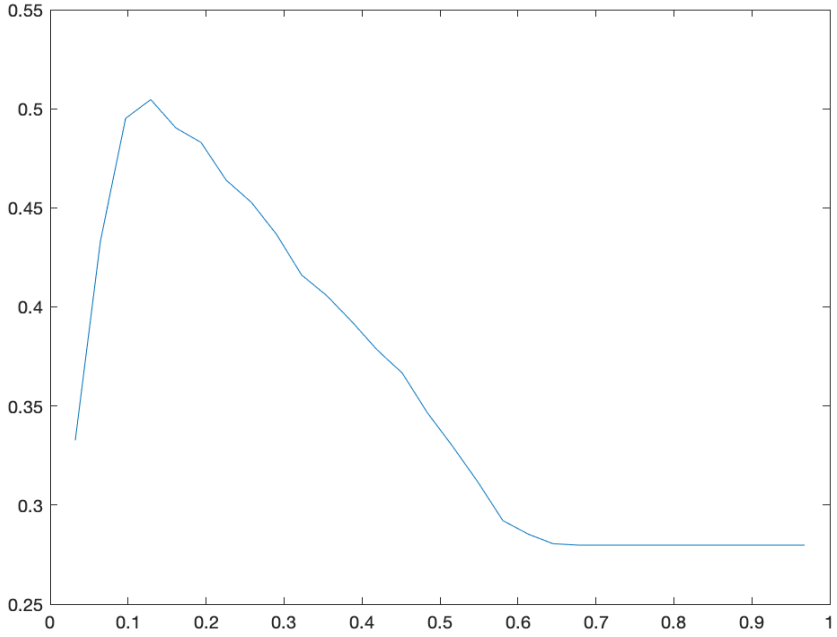


Figure 4: F-measure Graph for SE detector with Tiger.png

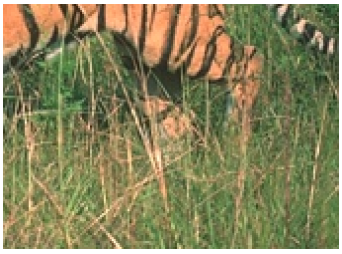


The X-axis in these graphs represent the threshold levels, and the y-axis represents the F measures. The approximate best F measures in figure 1-4 in order is 37%, 55%, 36%, 51%. When we compare the corresponding thresholds (0.73, 0.15, 0.41, 0.15), we can see that the Sobel detector has better F measures (meaning bigger AUC) only with higher threshold levels compared to the SE detector. From figure 2 and 4, we can also conclude that the F measures of the SE detector becomes the constant as it reaches a certain threshold, meaning that after that threshold, the SE detector is unable to detect any more edges.

Connecting back to section 1, we can see that the SE detector has a higher **max** F measure compared to the Sobel detector.

3.

Based on the F measures we observed from the Sobel detector graphs, the tiger image reaches the max F measure with a threshold of 41% instead of 73% with the pig image. Thus, it is easier to get a higher F measure for the tiger image. I think that the reason for this is that there are more true edges drawn in the ground truth of the pig image while in the tiger image there are lots of grass (as shown below) and we tend to omit insignifiant edges like these when constructing the ground truth files (lecture slides, the slippers example). Consequently, when we increase the threshold just low enough to omit most of the grass edges and retain the tiger, we will have a high matching percentage with the ground truth files (which mostly only contain the tiger). In contrast, the pig image has a lot of edges in the background, thus requiring a higher threshold level to identify. It is more difficult to see this in the SE detector because they both reach the max F measure at around 15%.



4.

The simplified definition of the F score is the measurement of how accurate a model is at testing a dataset. In mathematical terms, we are trying to find the harmonic mean between these variables. We double the product of precision and recall then divide by the sum of precision and recall, the numerator represents the combined performance of both precision and recall, the denominator is used to average this performance [9]. We cannot get a higher F measure if precision is significantly higher than recall, as we can see from the evaluation table for SE detector with Pig.png (figure 2 from section 1). The ground truth precision mean is a lot higher than the ground truth recall mean, yet the F measure is lower than figure 1 from the same section, which has a much similar average between precision and recall. However, having a much higher recall may improve F measure (1-figure 3 and 4).

Problem 2: Digital Half-toning

1. Motivation

In an image, we can represent different levels of gray with the range from 0 to 255, meaning that we would need one of the 255 different colors to display/print that pixel. Although our electronic devices are more than capable of representing those pixel values on our RGB screens, many printers lack this power. This is mainly because instead of using RGB displays, printers are "displaying" to white (empty) paper with black ink [10]. Thus, we need a way to represent the grayscale image with only black dots (0) and no dots (255). The first way to perform half-toning is dithering. Dithering is performed with an index matrix where the values in each index represents how likely the pixel at the corresponding index is activated. We then apply a threshold to quantize the image [11]. However, when we binarize a grayscale image with a threshold, we tend to lose a lot of information. Fortunately, we have another method called error diffusion to counteract this effect. The main idea of error diffusion is that after quantizing a pixel, the algorithm calculates the error between this new value to the original pixel, then spreads this error to neighboring pixels using different diffusion matrices such as Floyd-Steinberg, Jarvis and Stucki [12].

2. Approaches and Procedures

a. Dithering

We first read the image, then for fixed thresholding we pick a threshold (128) and loop through the entire image pixel by pixel. For every pixel, if the pixel value is greater than or equal to the threshold, the pixel becomes 255 (white), otherwise it becomes 0 (black). For random thresholding we perform the same steps except the threshold is chosen at random. For dithering, we first create a 2x2 Bayer index matrix. Then based on this matrix we can create the 4x4, 8x8, 16x16 and 32x32 index matrix (using the formula provided in the lecture). We can normalize each of these matrices and transform them into threshold matrices. Finally, we can obtain the output image by applying these kernels on the entire image.

b. Error Diffusion

After we read the image, we initialize three different error diffusion matrices (Floyd-Steinberg, Jarvis and Stucki). While we are looping through the entire image matrix using Serpentine scanning, we also perform fixed thresholding and find the error of that pixel by subtracting the original pixel value by the binarized value. Then we diffuse this error to neighboring pixels based on the chosen diffusion matrix. My logic behind serpentine scanning is as follows: if it is an even row, we traverse through the matrix left to right while applying the diffusion matrix, otherwise, we traverse through the matrix right to left while applying the inverse diffusion matrix.

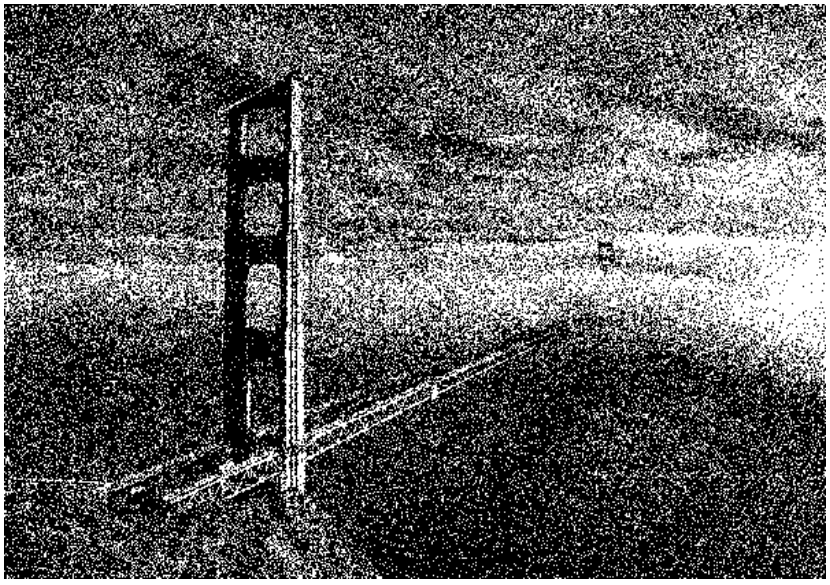
3. Experimental Results

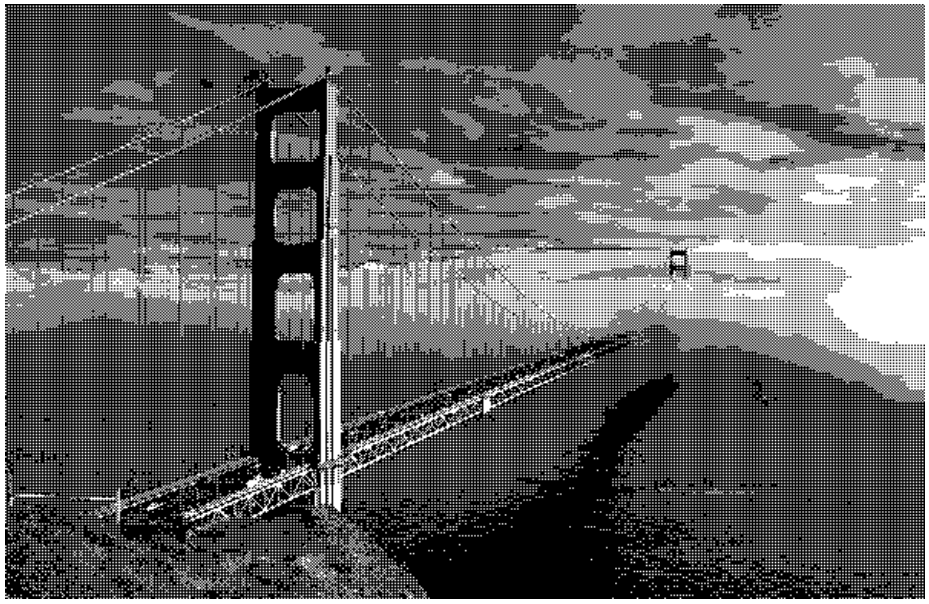
a. Dithering

Figure 1: Original Bridge Image

Figure 2: Fixed Threshold Bridge

Figure 3: Random Threshold Bridge





```
95,223,
159,31,
```

Figure 5: Dithering 2x2 Threshold Matrix

Figure 4: Dithering 2x2 Bridge

```
125,253,93,221,125,253,93,221,
189,61,157,29,189,61,157,29,
77,205,109,237,77,205,109,237,
141,13,173,45,141,13,173,45,
125,253,93,221,125,253,93,221,
189,61,157,29,189,61,157,29,
77,205,109,237,77,205,109,237,
141,13,173,45,141,13,173,45,
```

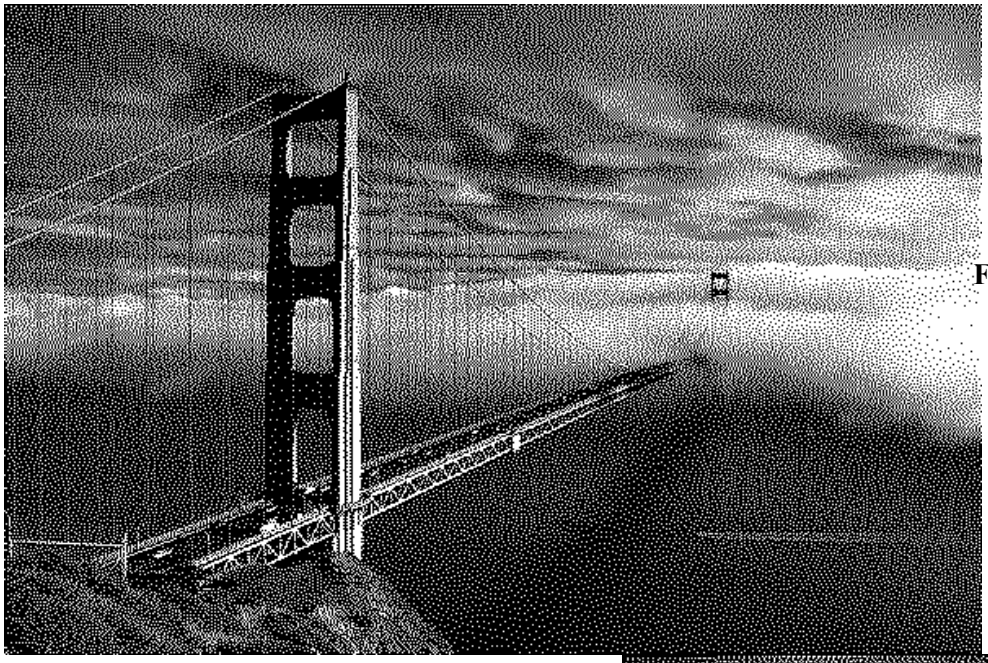
Figure 7: Dithering 8x8 Threshold Matrix

Figure 6: Dithering 8x8 Bridge

Figure 8: Dithering 32x32 Bridge

Figure 9: Dithering 32x32 Threshold Matrix
(Only shown 16x16 for clarity)

```
127,251,95,220,127,251,95,220,127,251,95,220,127,251,95,220
191,60,159,28,191,60,159,28,191,60,159,28,191,60,159,28,191
79,204,111,235,79,204,111,235,79,204,111,235,79,204,111,235
143,12,175,44,143,12,175,44,143,12,175,44,143,12,175,44,143
115,239,83,208,115,239,83,208,115,239,83,208,115,239,83,208
179,48,147,16,179,48,147,16,179,48,147,16,179,48,147,16,179
67,192,99,223,67,192,99,223,67,192,99,223,67,192,99,223,67,
131,0,163,32,131,0,163,32,131,0,163,32,131,0,163,32,131,0,1
124,254,92,223,124,254,92,223,124,254,92,223,124,254,92,223
188,63,156,31,188,63,156,31,188,63,156,31,188,63,156,31,188
76,207,108,238,76,207,108,238,76,207,108,238,76,207,108,238
140,15,172,47,140,15,172,47,140,15,172,47,140,15,172,47,140
112,242,80,211,112,242,80,211,112,242,80,211,112,242,80,211
176,51,144,19,176,51,144,19,176,51,144,19,176,51,144,19,176
64,195,96,226,64,195,96,226,64,195,96,226,64,195,96,226,64,
128,3,160,35,128,3,160,35,128,3,160,35,128,3,160,35,128,3,1
```



b. Error Diffusion

Figure 10: Floyd-Steinberg's Diffusion

Figure 11: JJN Diffusion

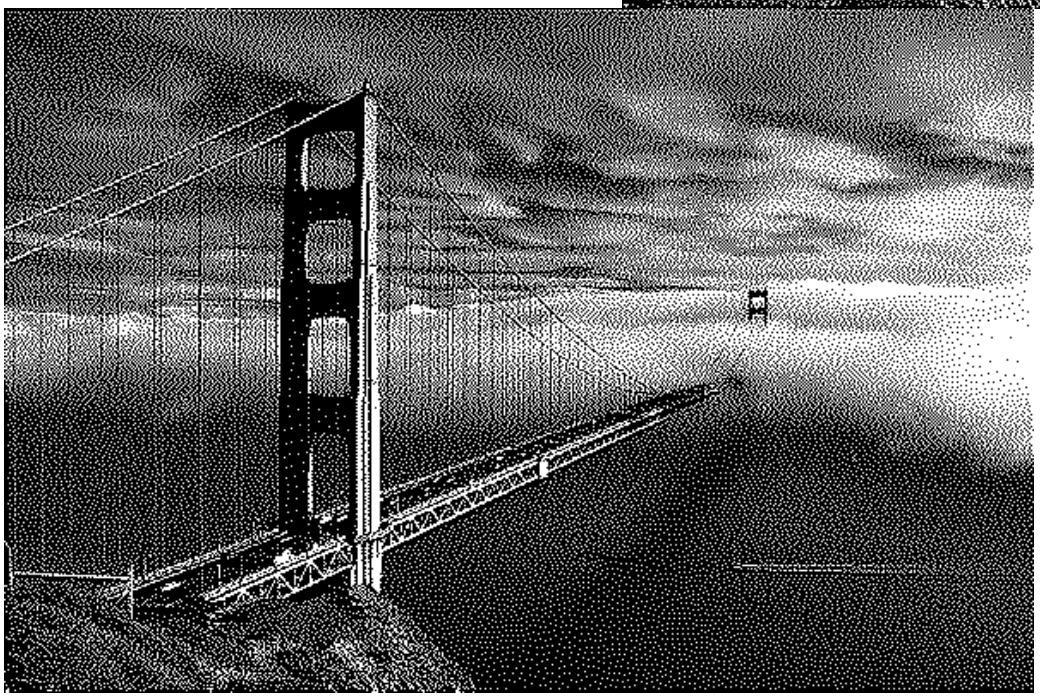


Figure 12: Stucki Diffusion

4. Discussion

a. Dithering

From the results of section a (figures 1-9), I think the best visual result is figure 6 with 8x8 dithering matrix. Although fixed thresholding is the least visible image, it is quite amazing that we are still able to recognize the bridge with just changing the same pixel value to black or white. On the other hand, Random thresholding produces a very heavy amount of unpleasant noise. When we compare all three dithering results, we can see that the image produced by the 2x2 matrix is less detailed than the other 2 matrices. For example, the clouds in the sky are less realistic when compared to the other two images and the sea is very monotone. Another artifact that I have noticed in all dithering images (especially in figure 8) is that there are horizontal and vertical lines appearing in square patterns. I think this is mainly due to the same square matrix being applied throughout the image using Raster Parsing.

b. Error Diffusion

I think the more effective error diffusion matrix is either JJN or Stucki. The result of Floyd-Steinberg's diffusion is less sharp than the other two results. This can be seen when comparing the suspension cables on the bridge. The cables in figure 10 starts from the top and slowly mixes with the background (the sea), however we can see very defined lines in figure 11 and 12.

When comparing the overall method of error diffusion and dithering, I prefer error diffusion a lot more than dithering. This is because error diffusion gives a feel of depth, strengths the shadows and edges on the bridge and clouds are a lot more sharper than dithering results. The thick fog covering the bridge is also more visible than dithering. If I were to design a better algorithm, I would create a 7x7 matrix with similar values in JJN. I think this would perform better because the algorithm is diffusing in a bigger area.

Problem 3: Color Half-toning with Error Diffusion

1. Motivation

The motivation for this section is similar to the last problem because it is also error diffusion, except it is for RGB colored images. In fact, there is even more reason to perform half-toning on colored images. There are 3 color channels for a RGB image, thus naturally we could have 256^3 different types of color tones in that image. Since it is already difficult for a printer device to output grayscale images which only have 256 color tones, it would be impossible to produce 16.7 millions of colors without half-toning [13].

2. Approaches and Procedures

a. Separable Error Diffusion

We first read the image with R,G,B channels and normalize them between 0 - 1, then we convert them to C,M,Y channels by subtracting the RGB channels separately from 1. Then we apply the Floyd-Steinberg's error diffusion algorithm described in problem 2 approaches and procedure section.

b. MBVQ-based Error diffusion

The diffusion process is almost identical to Floyd-Steinberg's error diffusion algorithm mentioned before. The main difference here is that instead of converting the RGB to CMY by simply subtracting them from 1, we first find the Minimum Brightness Variation Quadrant that the RGB pixel belongs to. We do this by assigning them to a quadrant like CMYW based on their sum of different channels like R + B or B + G. Then we take this quadrant and determine the nearest vertex in the MBVQ tetrahedron using a quantization process. These vertices represent 6 different colors, RGB and CMY. After we find which vertex (color) this pixel belongs to, we then find the new error and diffuse this error with the FS algorithm [14].

3. Experimental Results

Figure 1: Original Bird Image

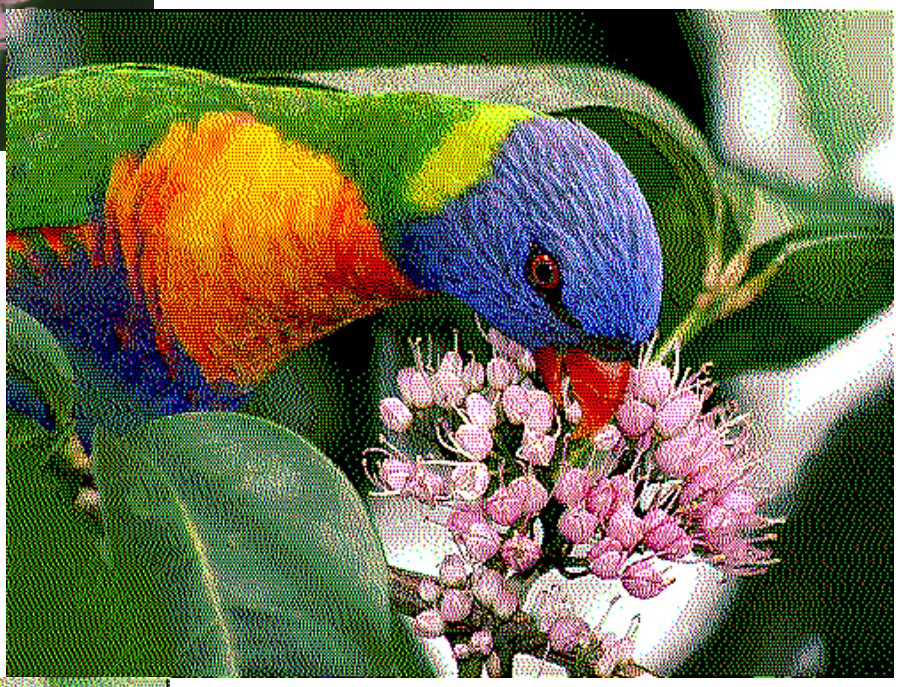


Figure 2: Separable Error Diffusion

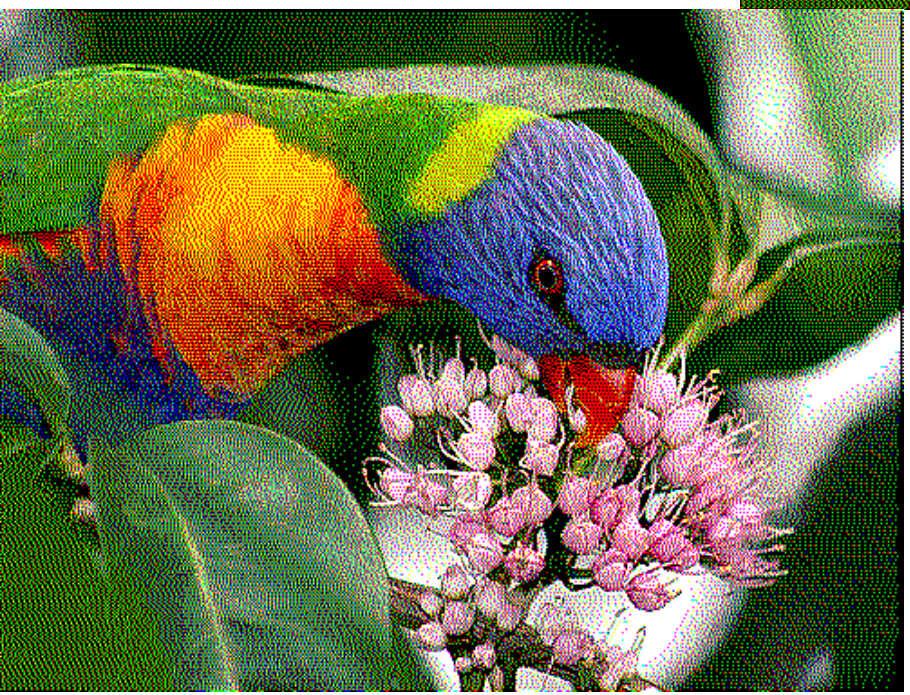


Figure 3: MBVQ-based Error diffusion

4. Discussion

a). Based on my understanding of the separable error diffusion algorithm, the main shortcoming of this approach is that the colors represented are not accurately displayed, meaning the lack of details in the image. Furthermore, we can see this clearly because when comparing figure 2 with figure 1, the former is much brighter than the latter image. I think this is because CMY color channels have higher luminance than RGB channels.

b).

1). The two key ideas that the MBVQ-based Error diffusion algorithm is based on are that which color quadrant (meaning four combined colors like RGMY) does this pixel with RGB values belong to, and based on this quadrant which specific color is predominant. This method is able to counteract the brightness downsides of Separable Error Diffusion because in the second step where we find the closest vertex in the quadrant, we tend to pick a desired color who has a minimal brightness variation [15].

2). Personally, there is not much of a visual difference when comparing this result (figure 3) with the result of the separable error diffusion algorithm (figure 2). However, when we zoom into the pixel level, we can see that the pixels in figure 3 are more spread out than figure 2. For example, when we focus on the little white patch under the flower, we can see that the magenta pixels are more compact and closer to the sides in figure 2.

Image References

All images are either directly given in the homework description or are the results of my programs.

References

[1]: Edge Detection, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Edge_detection

[2]: Sobel Operator, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Sobel_operator

[3]: EE569 Discussion Week 3 Edge Detection, Slide 8.

[4]: Canny Edge Detector, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Canny_edge_detector

[5]: EE569 Discussion Week 3 Edge Detection, Slide 17 - 19.

[6]: EE569 Lecture 4 Learning-based Edge and Contour Detection, Slide 3 - 5.

[7]: EE569 Discussion Week 3 Edge Detection, Slide 21 - 29.

[8]: Tony Yiu, "Understanding Random Forest", *Towards Data Science*, Jun 12, 2019, https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[9]: F-Score, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/F-score

[10]: EE569 Lecture 5 Digital Halftoning, Slide 5.

[11]: Dithering, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Dither

[12]: Error Diffusion, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Error_diffusion

[13]: EE569 Discussion Week 4 Digital Halftoning, Slide 7.

[14]: EE569 Discussion Week 4 Digital Halftoning, Slide 21 - 25.

[15]: EE569 Lecture 5 Digital Halftoning, Slide 36.