HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

EE569-HW2

Problem 1: Edge Detection

- **Abstract and Motivation:**

  Edges can be seen as features which can distinguish one object from another in an image. Edges give rise to contours which are further widely used in various tasks such as object detection, object classification, object tracking etc. Hence, its important to detect edges in an image. For the same we have various methods. We'll see and implement 3 different methods, namely Sobel Edge Detection, Canny edge detection and Structured Edge. Also, to check the performance of these algorithms we'll compute F-score for each method. F-score is an industry standard metric to evaluate the performance of various algorithms (mainly classification related). The inverse of F-score is equal to the sum of inverse of Precision and Recall. It can be seen as a trade-off in getting well balanced Precision and recall. In simple terms, Recall is a measure of how many positives of the total true positives did the algorithm detect and Precision is a measure of how many of the total detected positives were true positives.
  Sobel Edge: This algorithm is based on the fundamentals that there is a measurable difference in the intensity across the edge pixel. This algorithm will identify a pixel as edge if the gradient magnitude of that pixel is above some threshold.
  Canny Edge: This algorithm comprises of 3 steps, smoothing, differentiation and non-maximum suppression. We do double thresholding to ensure only potential edges get detected.
  Structured Edge: This algorithm is a supervised learning technique which is pre-trained on a sample of images and detects/classifies various types of edges.
  We further see the detailing and implementation of algorithms in below sub-sections.

- **Approaches and Procedures:**

  Method 1: Sobel Edge
  - Input: RGB raw image (24bits); Output1: Gradient_x map; Output2: Gradient_y map; Output3: Gradient magnitude map (Probability edge map); Output4: Binary Edge map
  - Threshold values experimented: 90%, 95% (cap on CDF)
  - Algorithm steps:
    - We convert RGB image to grayscale by using the given formula
    - We pad the input image by adding reflected rows and columns as boundary extension.

- We store the input image data as 'double' datatype to deal with signed operations.
- We convolve the image with Sobel_x and Sobel_y filter to generate the 2 componets, Gradient_x and Gradient_y . Sobel_x = $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ Sobel_y = $\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$. These components are signed values.
- We normalize the Gradient_x and Gradient_y values to (0-255) for display purpose.
- Now, we take square root of sum of squares of these signed components which will give us the gradient magnitude. We normalize these values to (0-255) and output the Gradient Magnitude (Probability edge map).
- We compute the CDF of gradient magnitude and plot them. We cap the CDF plot at a chosen threshold % and find the corresponding value. This value is our Threshold.
- Finally, we generate a binary image by using the above found threshold value in the following way $Output(i,j) = 0 \ \ if \ gradient \ magnitude(i,j) > Threshold \ else \ 255$
- Hence, in our output image, all the edges identified the algorithm will be marked in black and the background as white.

Method 2: Canny Edge

- Input: RGB raw image (24bits); Output1: Binary Edge map
- Threshold values experimented: (100,200); (100, 250); (100, 300)
- Blur kernel: 3x3
- Canny filter kernel: 3x3
- Algorithm: (using open source code from OpenCV)
  - We first read the raw RGB image and then convert it to OpenCV matrix object.
  - The image is converted to grayscale image for further operation.
  - The first operation is Gaussian blurring. We use kernel size=3. This is done in order to remove noise and hence apply smoothing.
  - We then use the OpenCV function 'Canny' to implement Canny edge detector. We try different values for Low and High threshold. We set the filter size to 3
  - We then invert the binarized image in order to have black edges and white background.

Method 3: Structured Edge

- Input: Grayscale jpg image; Output1: Probability Edge map(.jpg); Output2: Binary edge map(.jpg)
- Threshold: 0.15
- Algorithm:
  - We use a pre-trained model for detecting edges. The trained model is loaded inside the workspace.
  - We forward pass our input jpg image through the model.
  - The model classifies each pixel with a confidence of being an edge. This is our Probability edge map.
  - We apply a threshold and binarize the probability edge map to generate Binary edge map.

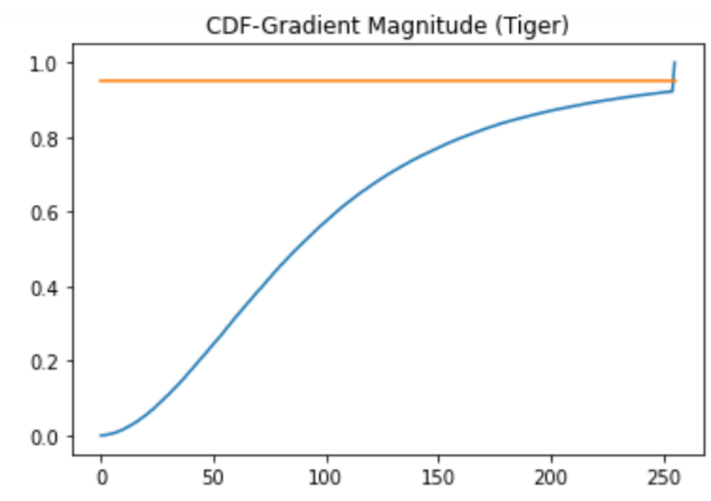
Method 4: Performance Evaluation

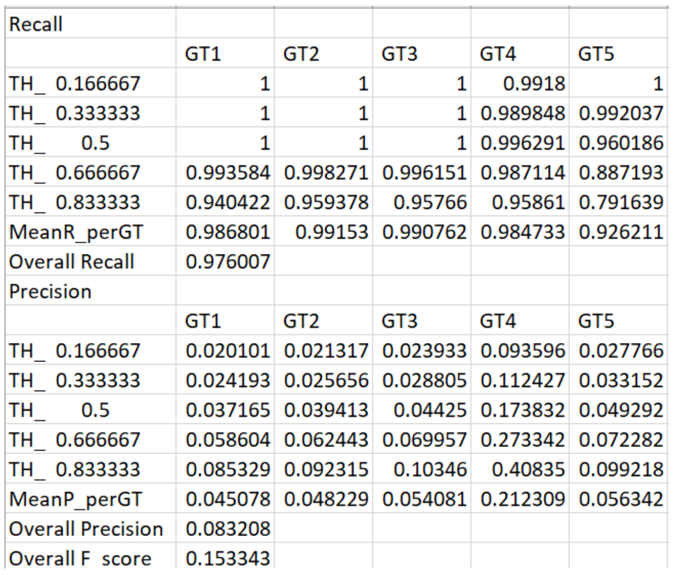
- Input1: Probability Edge map(.jpg); Input2: GT.mat; Output1: mean/precision/recall score (.csv)
- The below 2 approaches are carried out for both Sobel and Structured edge outputs
- For Canny, we directly read binarized map (in E1 variable) and then run the program (both approaches) for 5 different Binarized maps (different threshold pair)
- Approach1:
  - We read Probability edge map and the 5 GT files.
  - We set the additional parameters as follows: {‘Out’, “score.csv”, ‘thrs’, 5, 'maxDist',.0075, 'thin',1}
  - This will lead to experimenting with 5 thresholds: 0.1667, 0.3333, 0.5000, 0.6667, 0.8333
  - We run nested loop with the outside loop running over each threshold value and inside loop running over each GT.
  - For each threshold, we generate Binarized edge map. Then for each GT, we compare our edge map and find the counts of matching pixels.
  - We thus compute the Precision and Recall per threshold per each GT.
  - Outside both the loops, we iterate over each GT idx, to find mean Precision/Recall per GT across all thresholds.
  - We take a mean of average precision/recall per GT to get Overall Precision/Recall.
  - Finally, we compute Overall F_score by using the overall Precision/Recall =(2*Precision*Recall)/(Precision+Recall).
  - These all values are exported to csv file for documenting.

- Approach2:
  - The first 6 steps of Aprroach1 are repeated here.

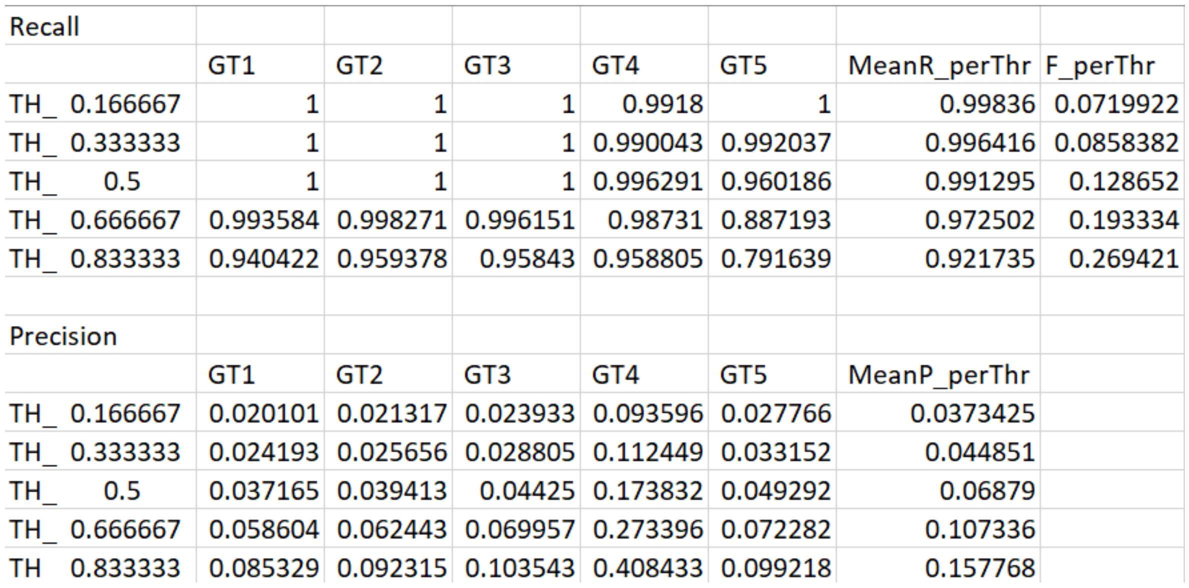
- We then iterate over each threshold idx, to find mean Precision/Recall per Threshold across all GTs.
- We use these 5 Precision/Recall quantities to compute F_score per threshold.
- These values are exported to csv file for documenting.

- **Experimental Results:**
  - **Sobel Edge (Tiger)**



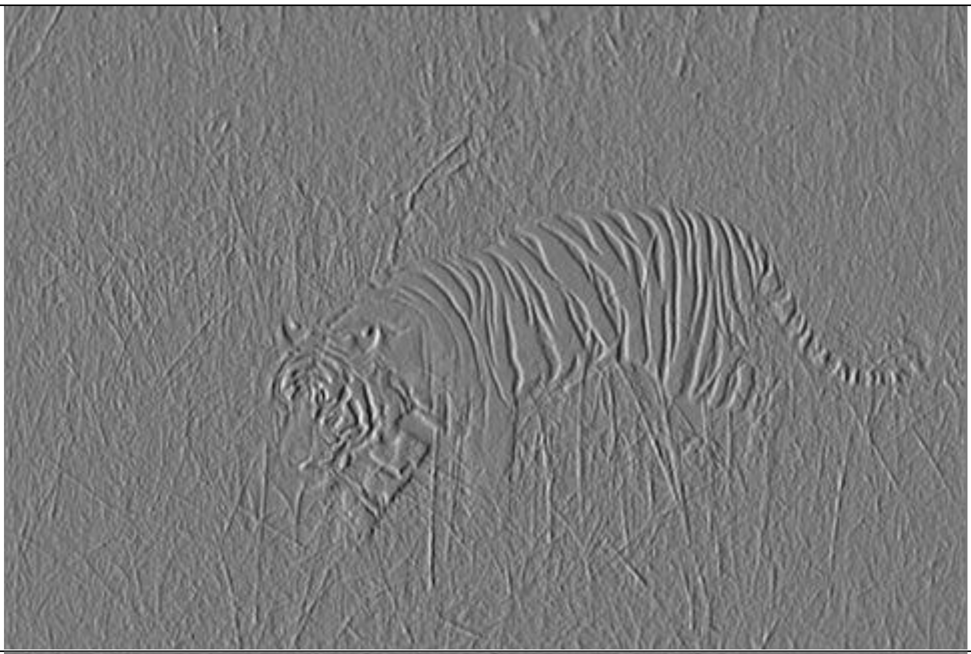
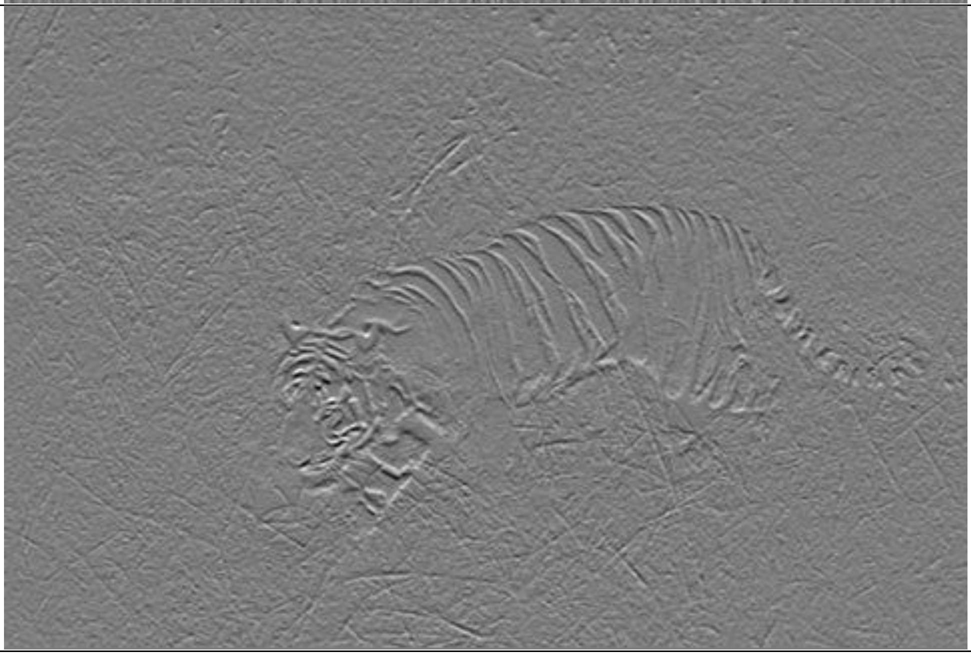


Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	1	1	1	0.9918	1
TH_ 0.333333	1	1	1	0.989848	0.992037
TH_ 0.5	1	1	1	0.996291	0.960186
TH_ 0.666667	0.993584	0.998271	0.996151	0.987114	0.887193
TH_ 0.833333	0.940422	0.959378	0.95766	0.95861	0.791639
MeanR_perGT	0.986801	0.99153	0.990762	0.984733	0.926211
Overall Recall	0.976007				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.020101	0.021317	0.023933	0.093596	0.027766
TH_ 0.333333	0.024193	0.025656	0.028805	0.112427	0.033152
TH_ 0.5	0.037165	0.039413	0.04425	0.173832	0.049292
TH_ 0.666667	0.058604	0.062443	0.069957	0.273342	0.072282
TH_ 0.833333	0.085329	0.092315	0.10346	0.40835	0.099218
MeanP_perGT	0.045078	0.048229	0.054081	0.212309	0.056342
Overall Precision	0.083208				
Overall F_score	0.153343				

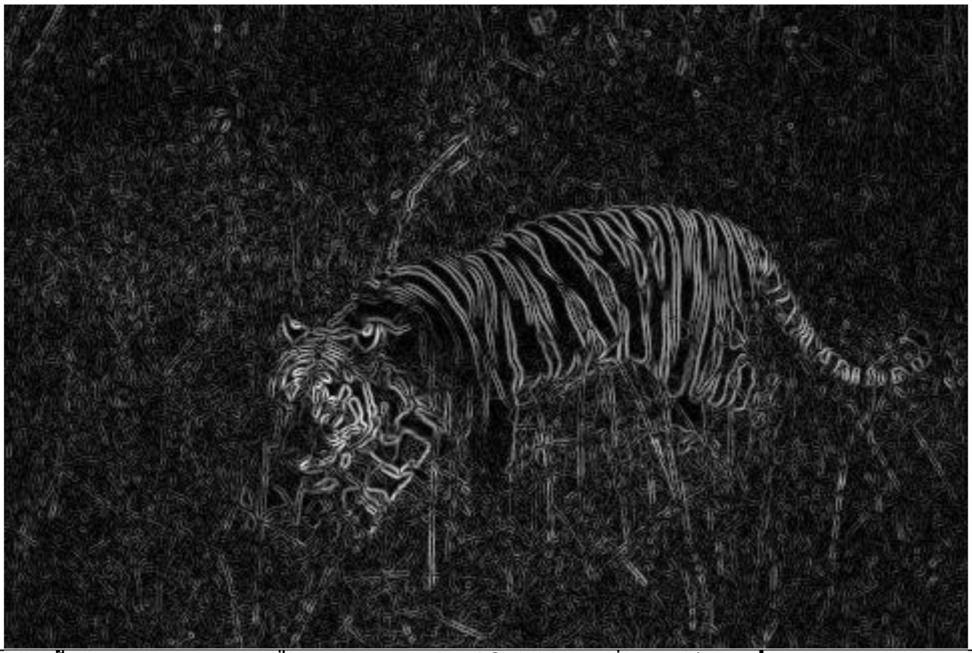
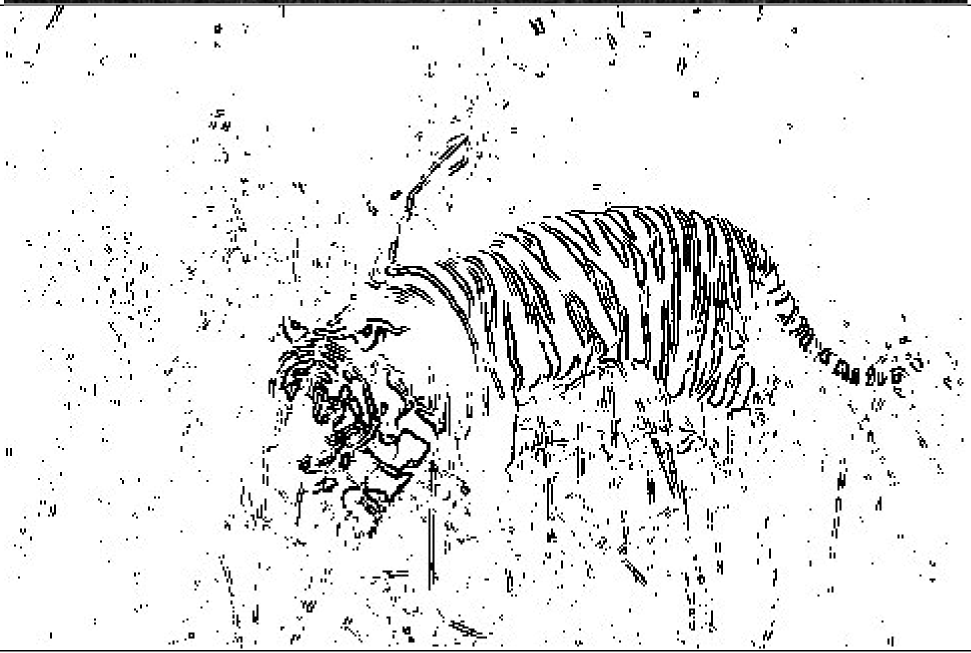


Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ 0.166667	1	1	1	0.9918	1	0.99836	0.0719922
TH_ 0.333333	1	1	1	0.990043	0.992037	0.996416	0.0858382
TH_ 0.5	1	1	1	0.996291	0.960186	0.991295	0.128652
TH_ 0.666667	0.993584	0.998271	0.996151	0.98731	0.887193	0.972502	0.193334
TH_ 0.833333	0.940422	0.959378	0.95843	0.958805	0.791639	0.921735	0.269421
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ 0.166667	0.020101	0.021317	0.023933	0.093596	0.027766	0.0373425	
TH_ 0.333333	0.024193	0.025656	0.028805	0.112449	0.033152	0.044851	
TH_ 0.5	0.037165	0.039413	0.04425	0.173832	0.049292	0.06879	
TH_ 0.666667	0.058604	0.062443	0.069957	0.273396	0.072282	0.107336	
TH 0.833333	0.085329	0.092315	0.103543	0.408433	0.099218	0.157768	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Sobel Edge Output	**Tiger**
Gradient_x	
Gradient_y	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Gradient Magnitude	
Binary edge map (%thr =0.95) (thr=254)	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

- **Sobel Edge (Pig)**

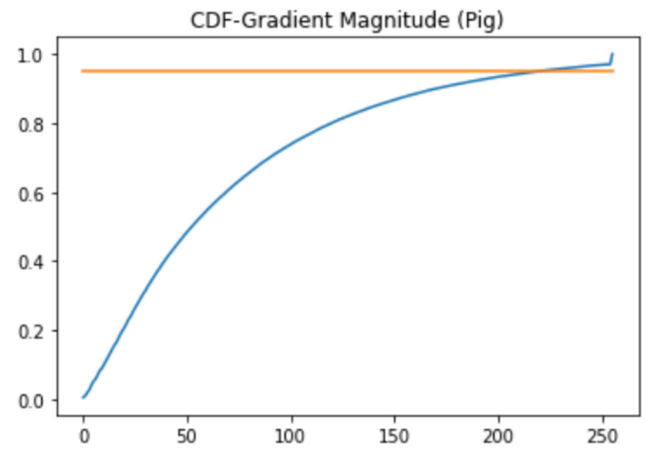
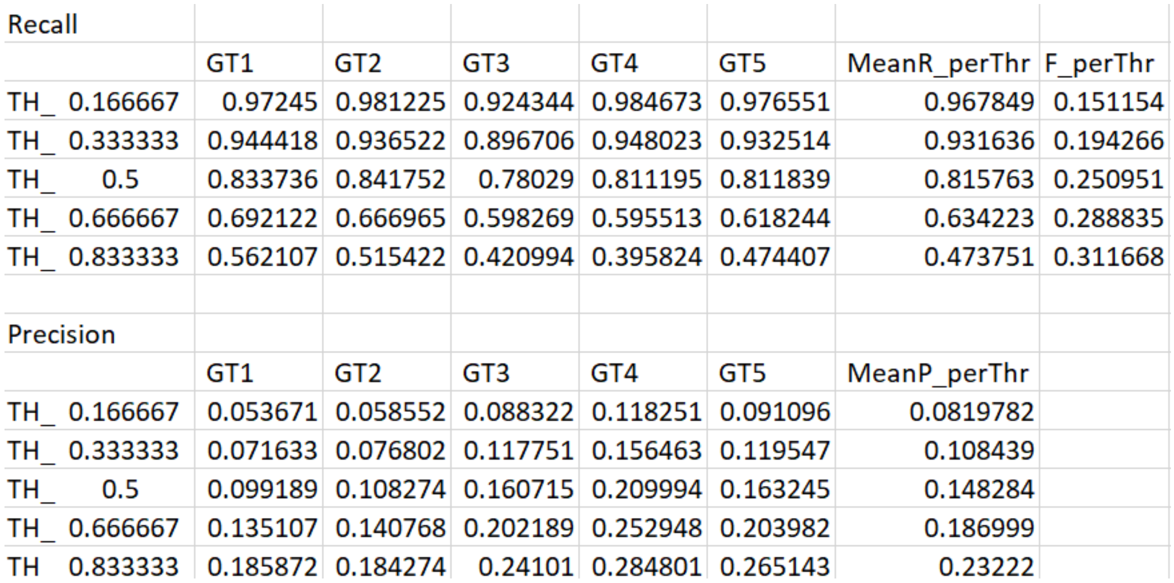


Table 1 Evaluation_1 (Sobel)

Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.97245	0.981225	0.924344	0.984896	0.976265
TH_ 0.333333	0.944418	0.936522	0.896706	0.948245	0.9328
TH_ 0.5	0.834219	0.842199	0.78057	0.810973	0.811839
TH_ 0.666667	0.692122	0.666965	0.598269	0.595735	0.618244
TH_ 0.833333	0.563074	0.515422	0.420994	0.395824	0.474407
MeanR_perGT	0.801257	0.788467	0.724176	0.747135	0.762711
Overall Recall	0.764749				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.053671	0.058552	0.088322	0.118278	0.091069
TH_ 0.333333	0.071633	0.076802	0.117751	0.1565	0.119584
TH_ 0.5	0.099247	0.108332	0.160773	0.209936	0.163245
TH_ 0.666667	0.135107	0.140768	0.202189	0.253043	0.203982
TH_ 0.833333	0.186191	0.184274	0.24101	0.284801	0.265143
MeanP_perGT	0.10917	0.113745	0.162009	0.204512	0.168605
Overall Precision	0.151608				
Overall F_score	0.25305				

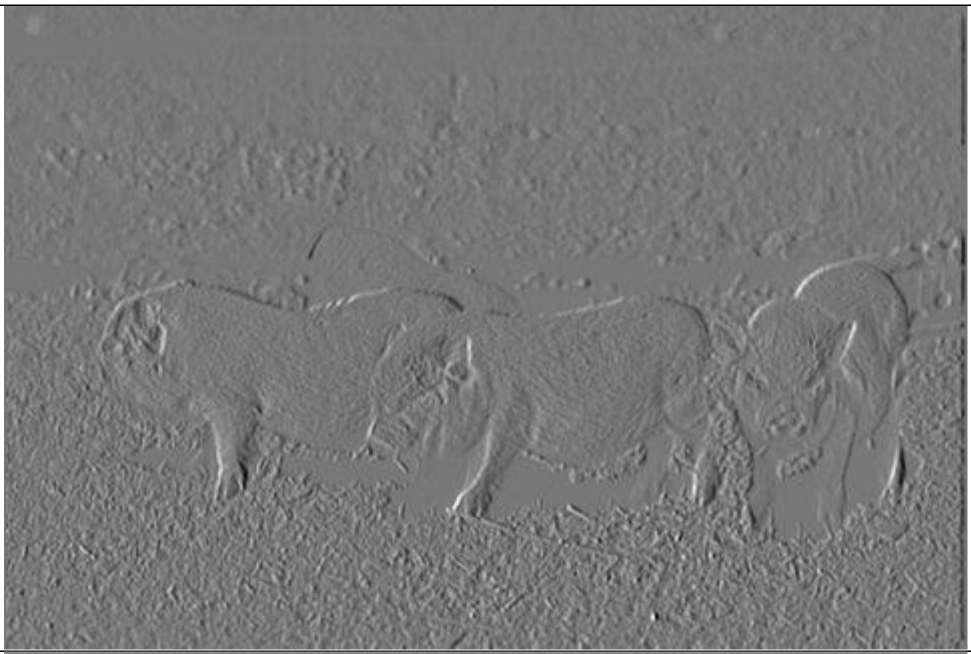
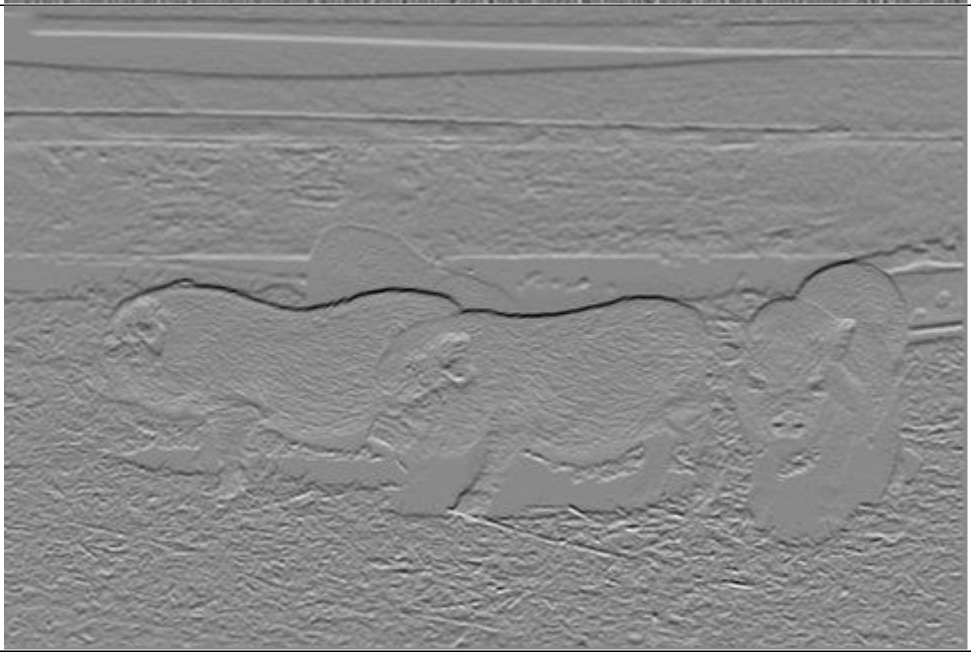
HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Table 2 Evaluation_2(Sobel)

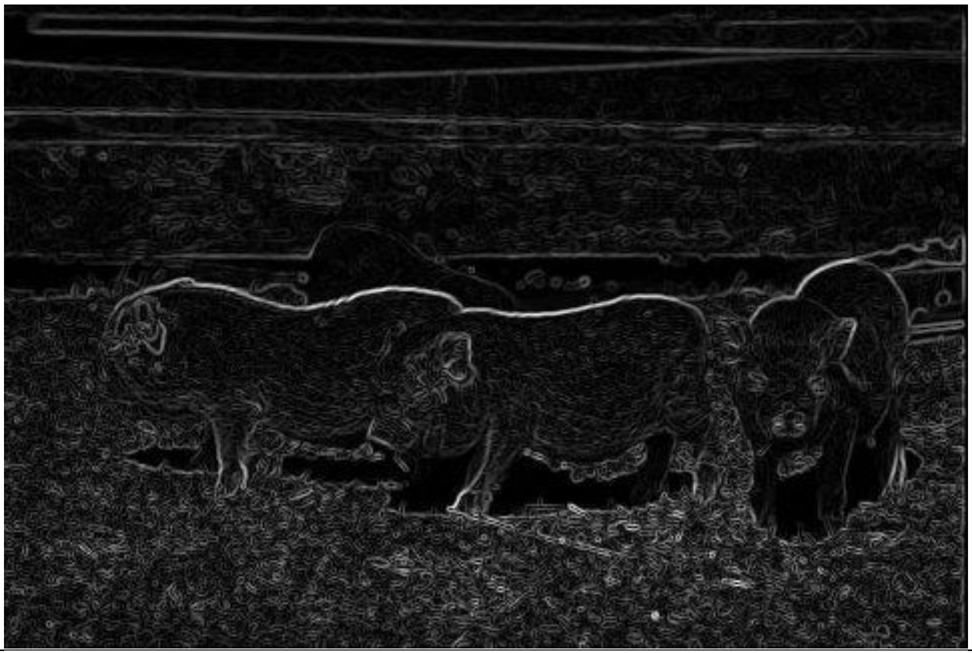
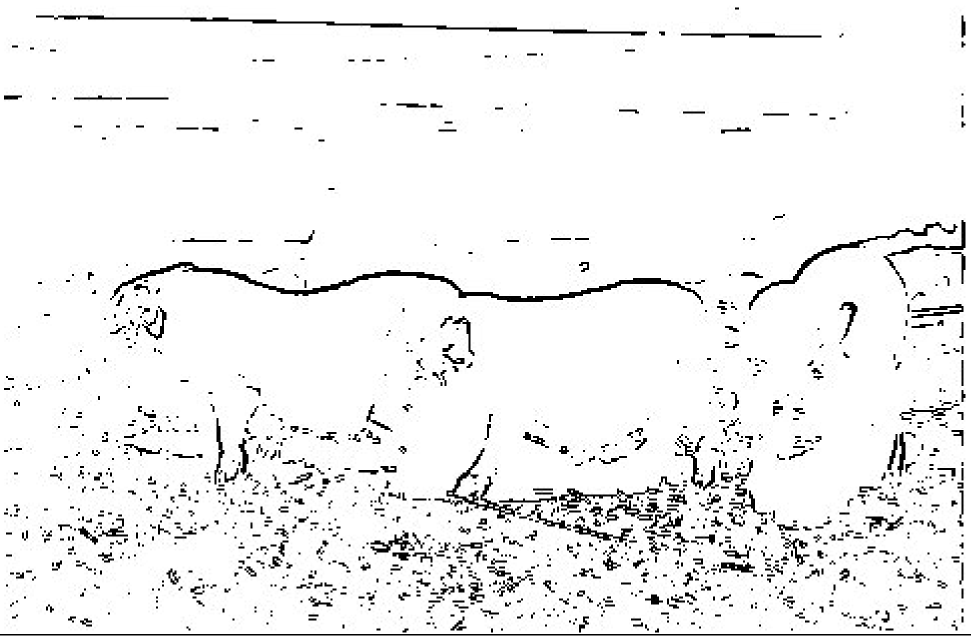


Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ 0.166667	0.97245	0.981225	0.924344	0.984673	0.976551	0.967849	0.151154
TH_ 0.333333	0.944418	0.936522	0.896706	0.948023	0.932514	0.931636	0.194266
TH_ 0.5	0.833736	0.841752	0.78029	0.811195	0.811839	0.815763	0.250951
TH_ 0.666667	0.692122	0.666965	0.598269	0.595513	0.618244	0.634223	0.288835
TH_ 0.833333	0.562107	0.515422	0.420994	0.395824	0.474407	0.473751	0.311668
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ 0.166667	0.053671	0.058552	0.088322	0.118251	0.091096	0.0819782	
TH_ 0.333333	0.071633	0.076802	0.117751	0.156463	0.119547	0.108439	
TH_ 0.5	0.099189	0.108274	0.160715	0.209994	0.163245	0.148284	
TH_ 0.666667	0.135107	0.140768	0.202189	0.252948	0.203982	0.186999	
TH 0.833333	0.185872	0.184274	0.24101	0.284801	0.265143	0.23222	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

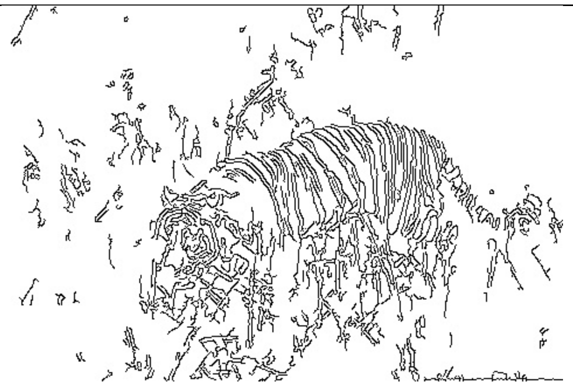
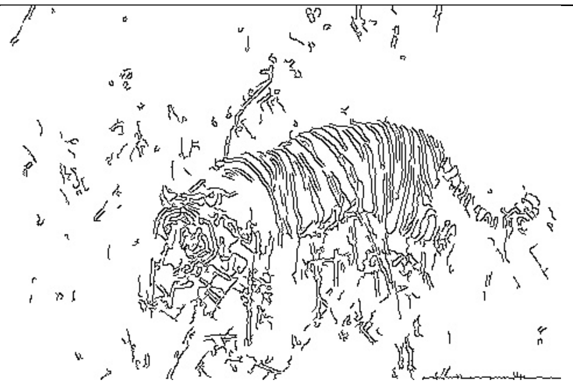
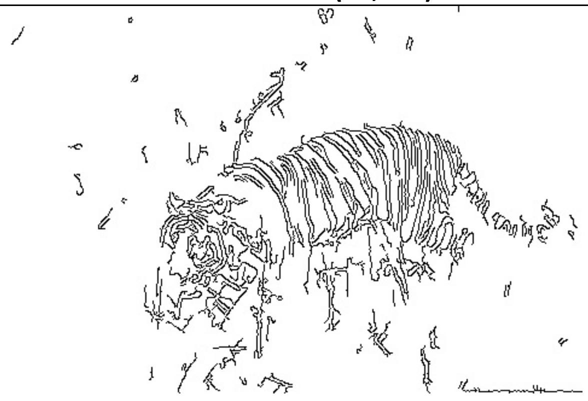
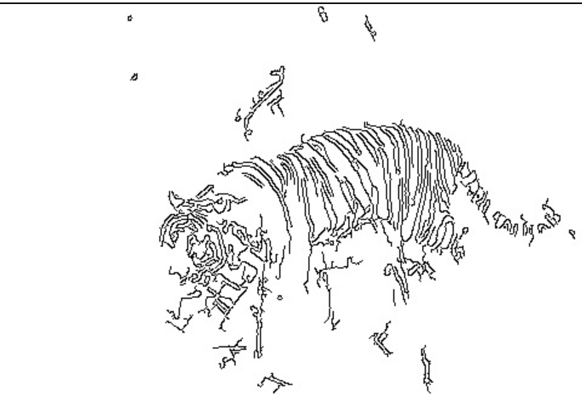
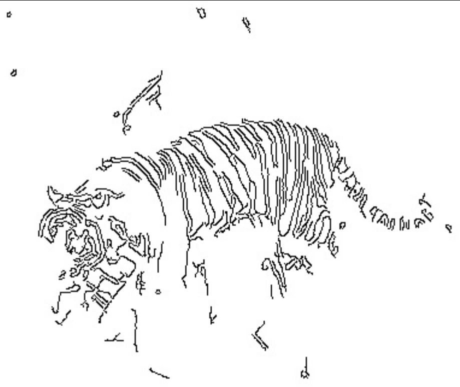
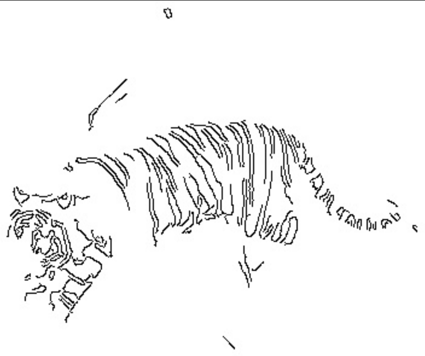
Sobel Edge Output	**Pig**
Gradient_x	
Gradient_y	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Gradient Magnitude	
Binary edge map (%thr =0.95) (thr=218)	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

- Canny Edge (Tiger)

 (80,200)	 (100,200)
 (100,250)	 (100,300)
 (130,300)	 (180,360)

EE569-HW2

Table 3 Tiger_Evaluation_1(Canny)

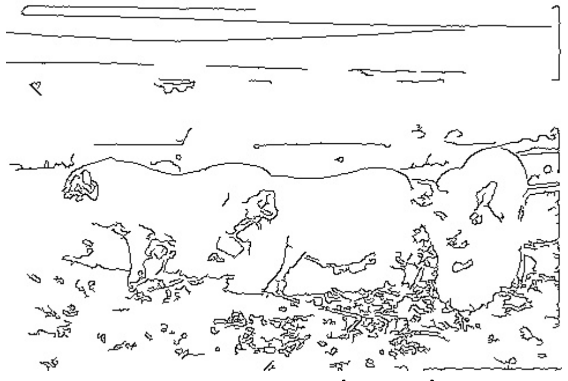
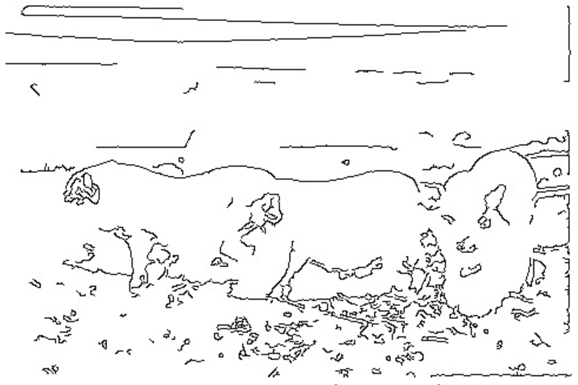
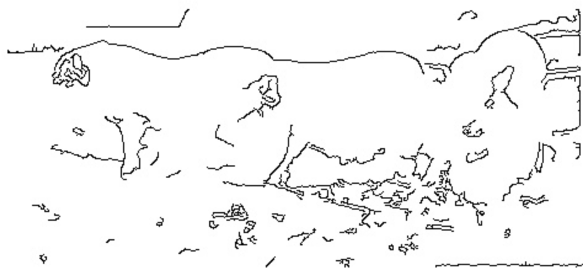
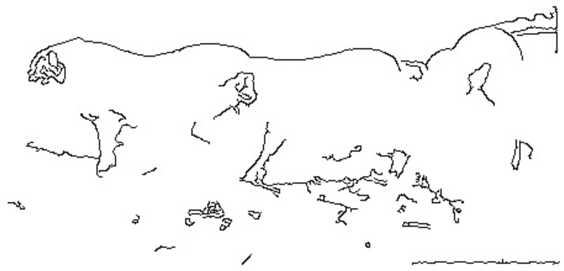
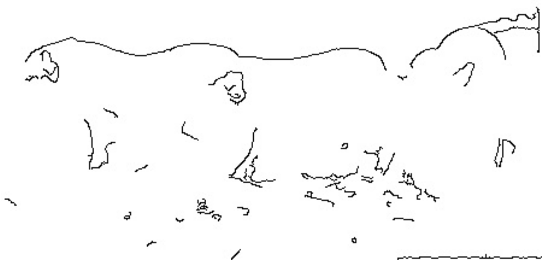
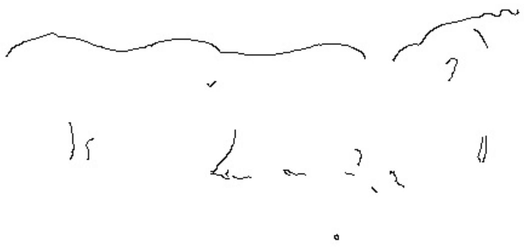
Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ (80,200)	0.958753	0.969749	0.961509	0.959196	0.806238
TH_ (100,200)	0.91659	0.936906	0.931486	0.94221	0.77505
TH_ (100,250)	0.826764	0.858254	0.856813	0.90492	0.690113
TH_ (100,300)	0.776352	0.81936	0.790608	0.865092	0.631055
TH_ (130,300)	0.713107	0.770959	0.747498	0.801835	0.584605
MeanR_perGT	0.838313	0.871046	0.857583	0.894651	0.697412
Overall Recall	0.831801				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ (80,200)	0.0901802	0.096733	0.107682	0.423571	0.10475
TH_ (100,200)	0.105541	0.114406	0.127704	0.50934	0.123272
TH_ (100,250)	0.122289	0.134626	0.150895	0.628389	0.140998
TH_ (100,300)	0.139792	0.156461	0.1695	0.731309	0.156957
TH_ (130,300)	0.15318	0.175625	0.191179	0.808624	0.173459
MeanP_perGT	0.122196	0.13557	0.149392	0.620247	0.139887
Overall Precision	0.233458				
Overall F_score	0.364589				

Table 4 Tiger_Evaluation_2(Canny)

Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ (80,200)	0.958753	0.969749	0.961509	0.959391	0.806238	0.931128	0.279748
TH_ (100,200)	0.91659	0.93777	0.931486	0.94182	0.77505	0.900543	0.321975
TH_ (100,250)	0.826764	0.858254	0.856813	0.90492	0.690113	0.827373	0.366567
TH_ (100,300)	0.776352	0.81936	0.789838	0.865092	0.631055	0.77634	0.401505
TH_ (130,300)	0.714024	0.770959	0.747498	0.80164	0.583942	0.723613	0.424526
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ (80,200)	0.0901802	0.096733	0.107682	0.423657	0.10475	0.1646	
TH_ (100,200)	0.105541	0.114512	0.127704	0.509129	0.123272	0.196032	
TH_ (100,250)	0.122289	0.134626	0.150895	0.628389	0.140998	0.235439	
TH_ (100,300)	0.139792	0.156461	0.169335	0.731309	0.156957	0.270771	
TH_ (130,300)	0.153377	0.175625	0.191179	0.808427	0.173262	0.300374	

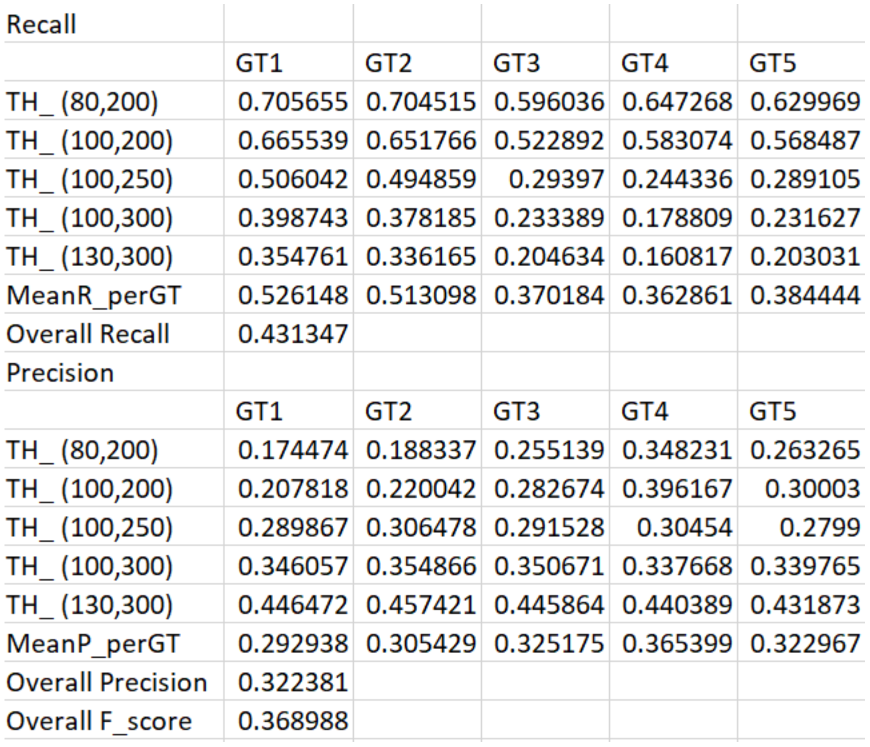
HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

- **Canny Edge (Pig)**

 **(80,200)**	 **(100,200)**
 **(100,250)**	 **(100,300)**
 **(130,300)**	 **(180,360)**

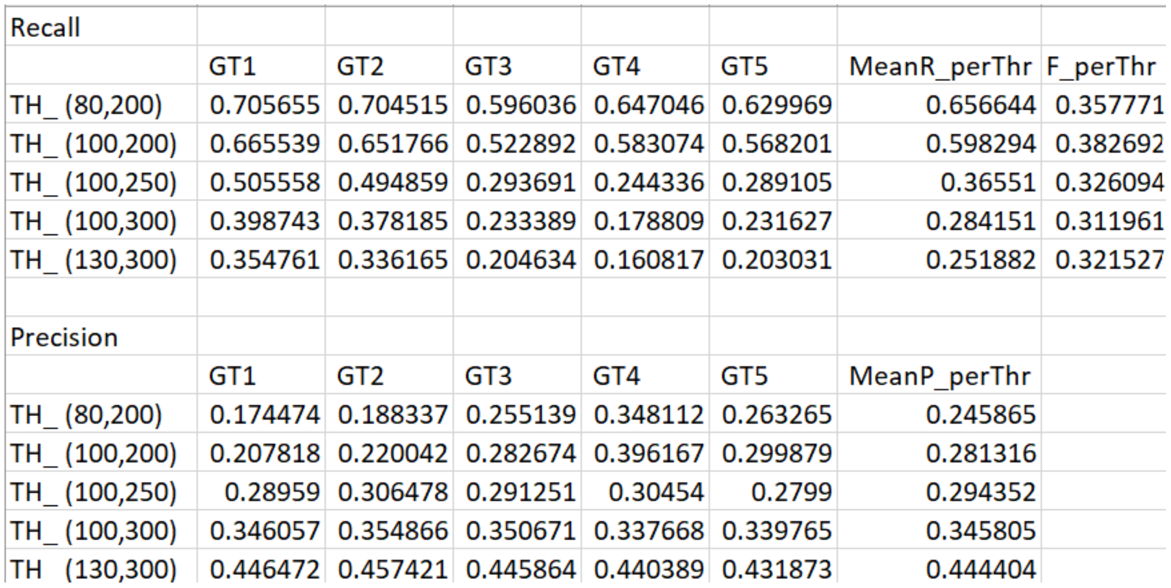
HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Table 5 Pig_Evaluation_1(Canny)



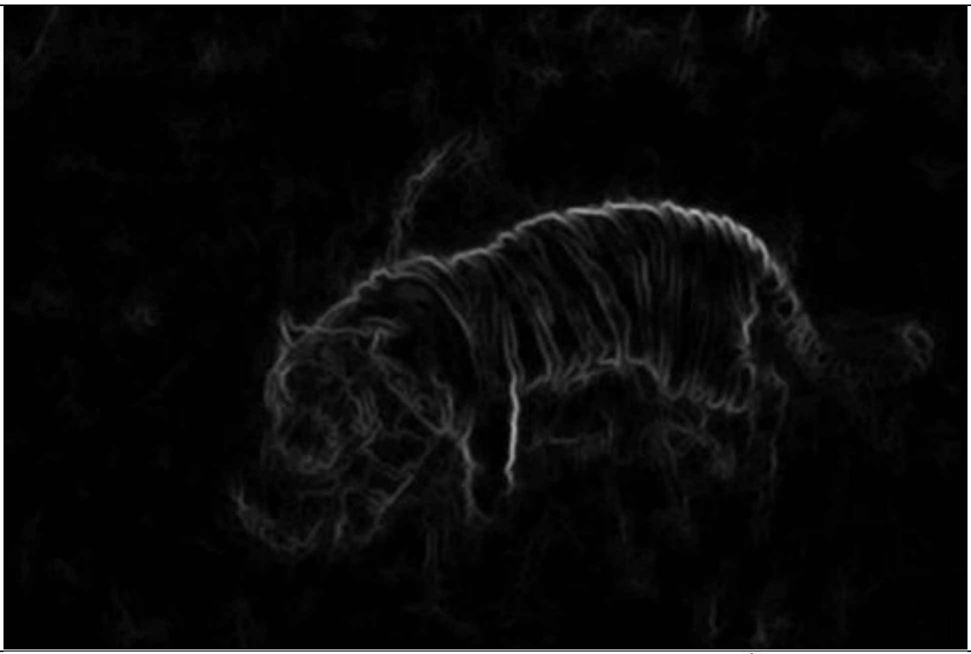
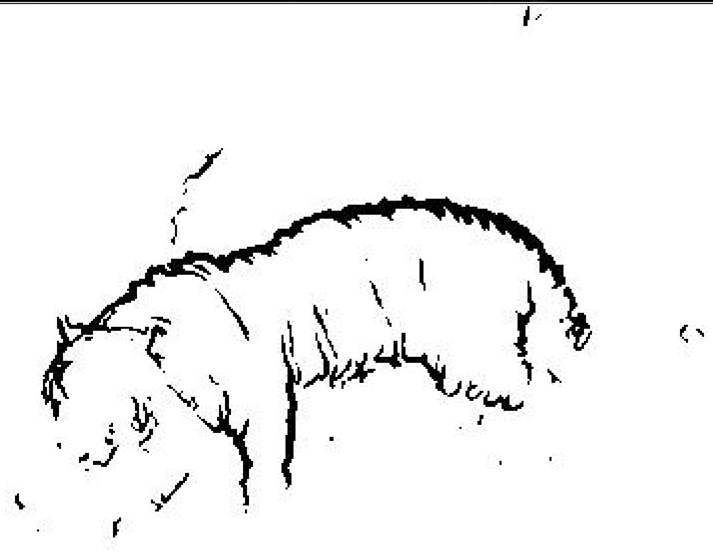
Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ (80,200)	0.705655	0.704515	0.596036	0.647268	0.629969
TH_ (100,200)	0.665539	0.651766	0.522892	0.583074	0.568487
TH_ (100,250)	0.506042	0.494859	0.29397	0.244336	0.289105
TH_ (100,300)	0.398743	0.378185	0.233389	0.178809	0.231627
TH_ (130,300)	0.354761	0.336165	0.204634	0.160817	0.203031
MeanR_perGT	0.526148	0.513098	0.370184	0.362861	0.384444
Overall Recall	0.431347				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ (80,200)	0.174474	0.188337	0.255139	0.348231	0.263265
TH_ (100,200)	0.207818	0.220042	0.282674	0.396167	0.30003
TH_ (100,250)	0.289867	0.306478	0.291528	0.30454	0.2799
TH_ (100,300)	0.346057	0.354866	0.350671	0.337668	0.339765
TH_ (130,300)	0.446472	0.457421	0.445864	0.440389	0.431873
MeanP_perGT	0.292938	0.305429	0.325175	0.365399	0.322967
Overall Precision	0.322381				
Overall F_score	0.368988				

Table 6 Pig_Evaluation_2(Canny)



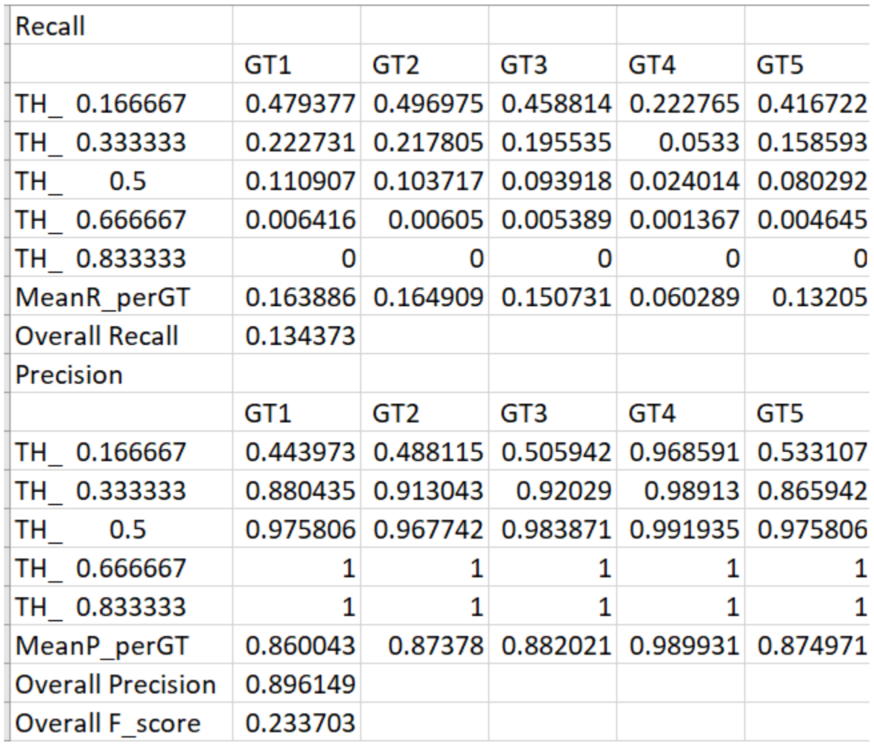
Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ (80,200)	0.705655	0.704515	0.596036	0.647046	0.629969	0.656644	0.357771
TH_ (100,200)	0.665539	0.651766	0.522892	0.583074	0.568201	0.598294	0.382692
TH_ (100,250)	0.505558	0.494859	0.293691	0.244336	0.289105	0.36551	0.326094
TH_ (100,300)	0.398743	0.378185	0.233389	0.178809	0.231627	0.284151	0.311961
TH_ (130,300)	0.354761	0.336165	0.204634	0.160817	0.203031	0.251882	0.321527
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ (80,200)	0.174474	0.188337	0.255139	0.348112	0.263265	0.245865	
TH_ (100,200)	0.207818	0.220042	0.282674	0.396167	0.299879	0.281316	
TH_ (100,250)	0.28959	0.306478	0.291251	0.30454	0.2799	0.294352	
TH_ (100,300)	0.346057	0.354866	0.350671	0.337668	0.339765	0.345805	
TH_ (130,300)	0.446472	0.457421	0.445864	0.440389	0.431873	0.444404	

- **Structured Edge (Tiger)**

Probability Edge Map	
Binary Edge Map	

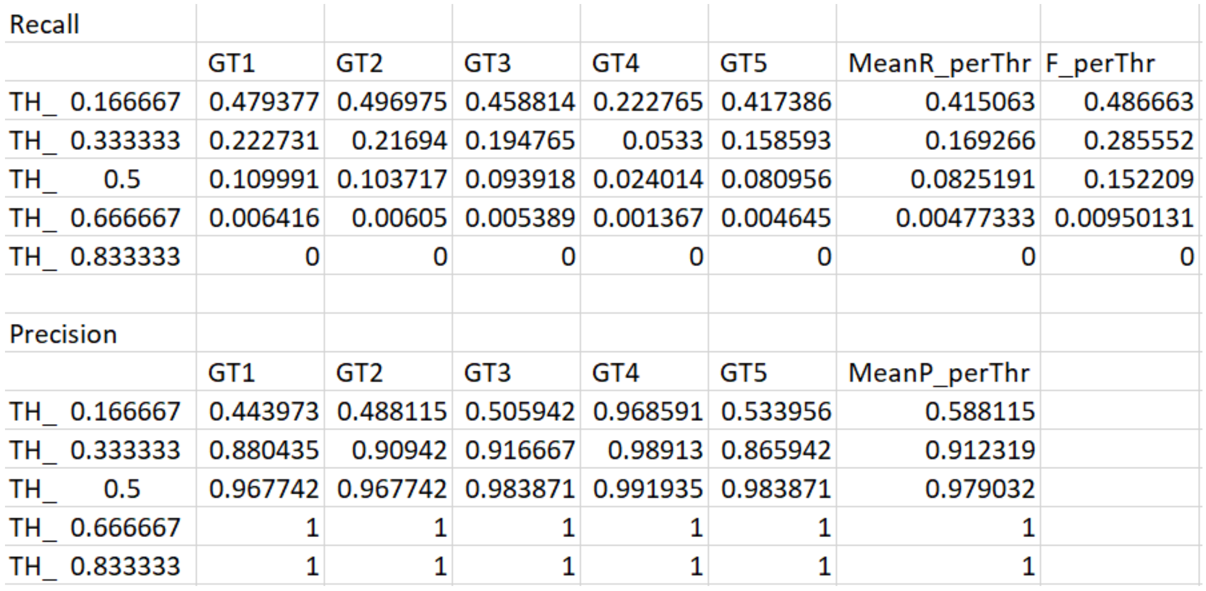
HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

Table 7 Tiger_Evaluation_1(SE)



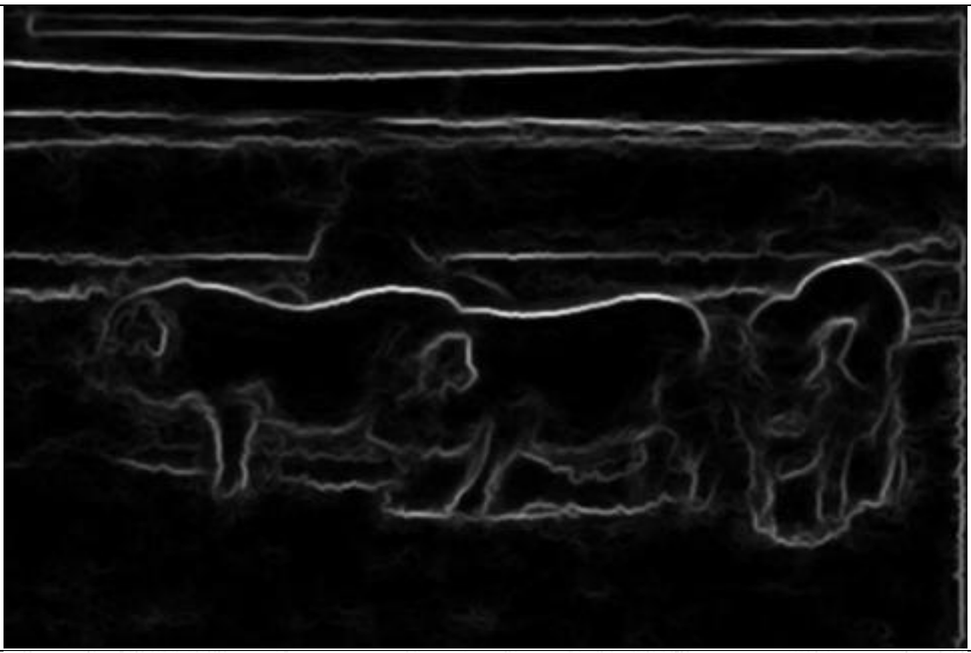
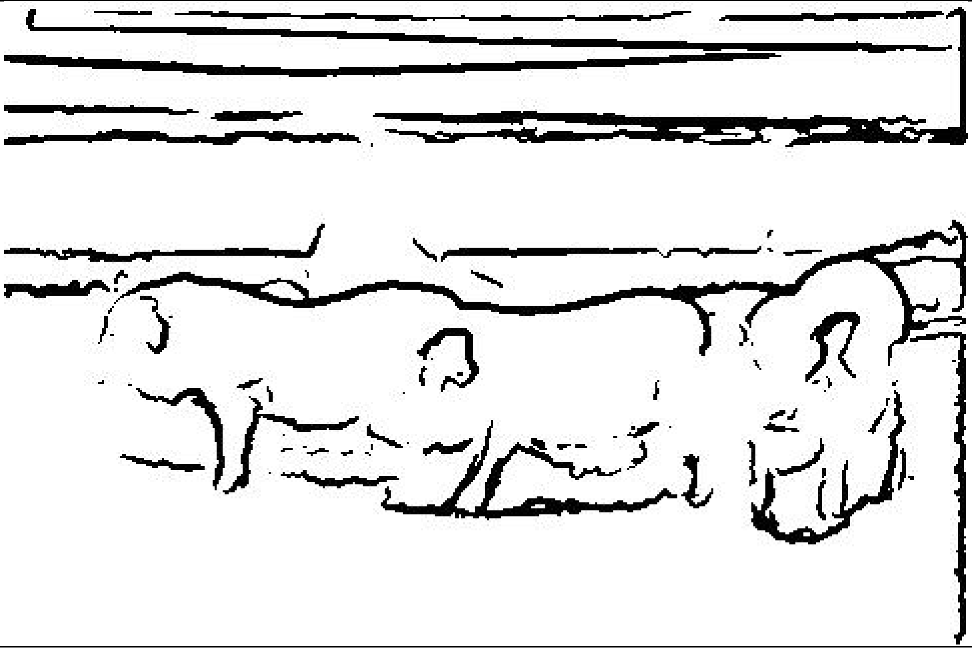
Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.479377	0.496975	0.458814	0.222765	0.416722
TH_ 0.333333	0.222731	0.217805	0.195535	0.0533	0.158593
TH_ 0.5	0.110907	0.103717	0.093918	0.024014	0.080292
TH_ 0.666667	0.006416	0.00605	0.005389	0.001367	0.004645
TH_ 0.833333	0	0	0	0	0
MeanR_perGT	0.163886	0.164909	0.150731	0.060289	0.13205
Overall Recall	0.134373				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.443973	0.488115	0.505942	0.968591	0.533107
TH_ 0.333333	0.880435	0.913043	0.92029	0.98913	0.865942
TH_ 0.5	0.975806	0.967742	0.983871	0.991935	0.975806
TH_ 0.666667	1	1	1	1	1
TH_ 0.833333	1	1	1	1	1
MeanP_perGT	0.860043	0.87378	0.882021	0.989931	0.874971
Overall Precision	0.896149				
Overall F_score	0.233703				

Table 8 Tiger_Evaluation_2(SE)



Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ 0.166667	0.479377	0.496975	0.458814	0.222765	0.417386	0.415063	0.486663
TH_ 0.333333	0.222731	0.21694	0.194765	0.0533	0.158593	0.169266	0.285552
TH_ 0.5	0.109991	0.103717	0.093918	0.024014	0.080956	0.0825191	0.152209
TH_ 0.666667	0.006416	0.00605	0.005389	0.001367	0.004645	0.00477333	0.00950131
TH_ 0.833333	0	0	0	0	0	0	0
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ 0.166667	0.443973	0.488115	0.505942	0.968591	0.533956	0.588115	
TH_ 0.333333	0.880435	0.90942	0.916667	0.98913	0.865942	0.912319	
TH_ 0.5	0.967742	0.967742	0.983871	0.991935	0.983871	0.979032	
TH_ 0.666667	1	1	1	1	1	1	
TH_ 0.833333	1	1	1	1	1	1	

- **Structured Edge (Pig)**

Probability Edge Map	
Binary Edge Map	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

EE569-HW2

Table 9 Pig_Evaluation_1(SE)

Recall					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.660706	0.663835	0.703797	0.742559	0.680011
TH_ 0.333333	0.309328	0.318283	0.3512	0.392492	0.369745
TH_ 0.5	0.166747	0.155565	0.152708	0.193025	0.150415
TH_ 0.666667	0.096182	0.088958	0.060581	0.074634	0.0620532
TH_ 0.833333	0.0232	0.021457	0.0134	0.010662	0.0137261
MeanR_perGT	0.251232	0.24962	0.256337	0.282674	0.25519
Overall Recall	0.259011				
Precision					
	GT1	GT2	GT3	GT4	GT5
TH_ 0.166667	0.285744	0.31041	0.526965	0.698788	0.497074
TH_ 0.333333	0.281442	0.313105	0.55321	0.777045	0.568602
TH_ 0.5	0.379538	0.382838	0.60176	0.955996	0.578658
TH_ 0.666667	0.590504	0.590504	0.643917	0.997033	0.643917
TH_ 0.833333	1	1	1	1	1
MeanP_perGT	0.507446	0.519371	0.66517	0.885772	0.65765
Overall Precision	0.647082				
Overall F_score	0.369943				

Table 10 Pig_Evaluation_2(SE)

Recall							
	GT1	GT2	GT3	GT4	GT5	MeanR_perThr	F_perThr
TH_ 0.166667	0.660222	0.663388	0.703518	0.742115	0.680583	0.689965	0.554623
TH_ 0.333333	0.307878	0.317836	0.350642	0.392492	0.370031	0.347776	0.409628
TH_ 0.5	0.166747	0.155565	0.152708	0.193025	0.150701	0.163749	0.255392
TH_ 0.666667	0.096182	0.088958	0.060581	0.074856	0.062053	0.0765259	0.137847
TH_ 0.833333	0.0232	0.021457	0.0134	0.010662	0.013726	0.016489	0.0324431
Precision							
	GT1	GT2	GT3	GT4	GT5	MeanP_perThr	
TH_ 0.166667	0.285535	0.310201	0.526756	0.69837	0.497492	0.463671	
TH_ 0.333333	0.280123	0.312665	0.552331	0.777045	0.569041	0.498241	
TH_ 0.5	0.379538	0.382838	0.60176	0.955996	0.579758	0.579978	
TH_ 0.666667	0.590504	0.590504	0.643917	1	0.643917	0.693769	
TH 0.833333	1	1	1	1	1	1	

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

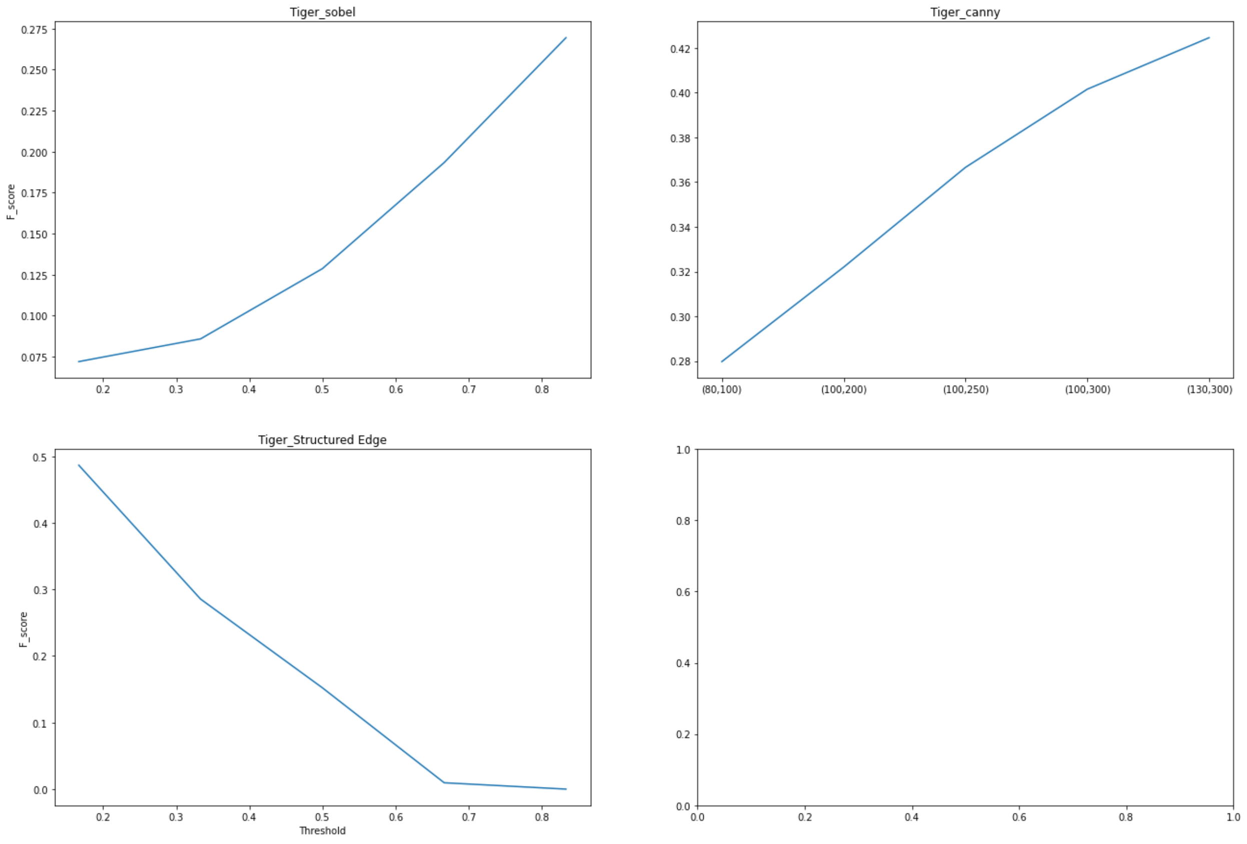


Figure 1 F_score vs Threshold (Tiger)

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

EE569-HW2

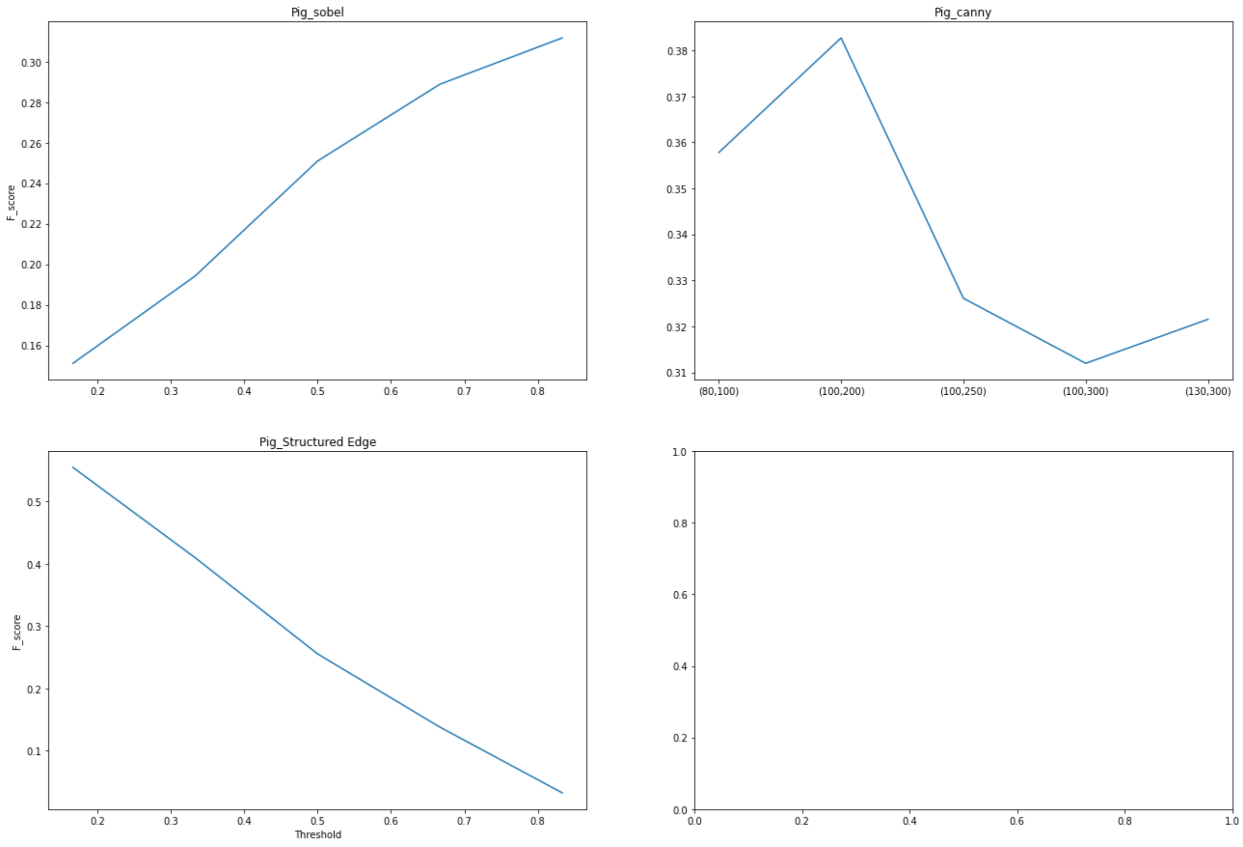


Figure 2 F_score vs Threshold (Pig)

- **Discussion:**
  - Sobel Edge Detector:
    - We need to work in grayscale domain, especially with Luminance channel.
    - It's based on the first derivative of intensity. The gradient magnitude will have a spike if a pixel is an edge point.
    - It identifies an edge based on local neighborhood window.
    - We finds the X & Y gradient components and the resultant of it gives the Gradient magnitude.
    - Sobel filters calculate slopes in only 2 directions. Would be interesting to convolve with filters in 45° direction.
    - The single thresholding is not very definitive and sensitive to background noise. Like in Tiger image, the big grass is very denser and that is detected as edges which is unwanted.
  - Canny Edge Detector:

    (1) Non-Maximum suppression:
      a. This is the 4th step of 5 steps process after finding gradient magnitude followed by double thresholding.
      b. This is done to fine tune the edges, in laymen language to thin the edge.
      c. This will ensure that all edges are only 1 pixel wide.
      d. NMS compares neighborhood pixel gradient magnitude of identified edge pixels and fires the current pixel if its value is greater than the rest. It's important to understand that the neighborhood of current pixel is not the regular 3x3 window, but it checks in the direction of gradient vectors.

    (2) High/Low Threshold:
      a. This is the final step of Canny Edge detector. This overcomes the shortcomings of sobel edge detector to output only strong candidates of being identified as edge.
      b. Those pixels having gradient magnitude higher than higher threshold are confirmed as edge pixels.
      c. Those pixels having gradient magnitude less than lower threshold are not identified as edge.
      d. Those pixels having gradient magnitude in the range of (low_thr,High_thr) are considered to be edges only if they have any connecting pixels having gradient magnitude higher than High threshold, rest are dropped to be considered as edge.

    (3) Result discussion:
      a. We have shown 6 results each for pig and tiger in the above section.
      b. The thresholds chosen for experimentation were (80,200); (100,200); (100,250); (100,300); (130,300); (180,360).

c. Tiger:

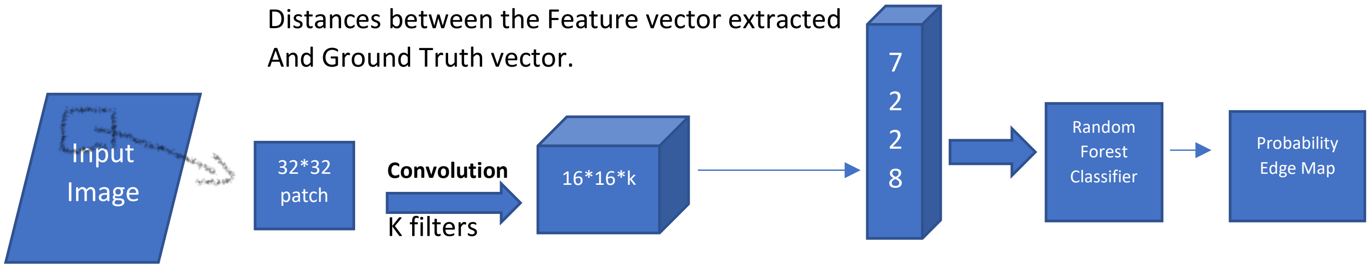
1. Human Perception: Visually, I find the best result as the one with threshold pair (100,250). The structure of tiger's face stays intact and some part of legs are also visible with minimum background.
2. F_score metric: The F-score table in above section suggests that the best result is the one with threshold pair (130,300) as it has highest F-score of 0.42

d. Pig:

1. Human Perception: Visually, I find the best result as the one with threshold pair (100,200). The structure of (right most) pig's face is quite clearly visible. Although lot of unwanted background is classified as edge.
2. F_score metric: The F-score table in above section also suggests that the best result is the one with threshold pair (100,200) as it has highest F-score of 0.38

- Structured Edge Detector:

1. SE detection algorithm:
   a. This is a supervised learning algorithm. It's a data driven approach.
   b. We slice an image into 32*32 pixel sized patches.
   c. Each of these patches are convolved with various size and type of filters. This will result into a vector/tensor. We call this tensor as our feature.
   d. This feature vector is then fed to Random Forest Classifier which classifies each pixel as an edge or not with a probability.
   e. According to the paper [5] we have 7228 total candidate features.
   f. The detection algorithm uses Criterion/Error function as sum of squared Distances between the Feature vector extracted And Ground Truth vector.

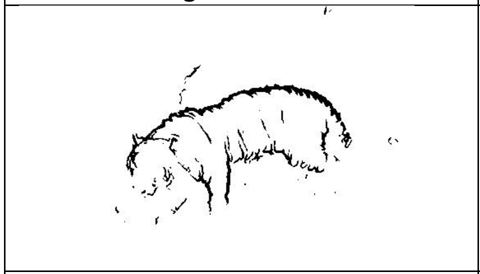
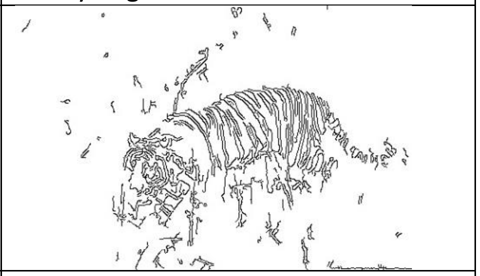


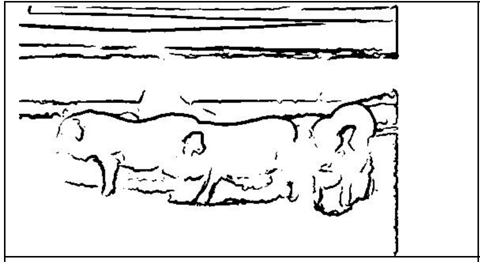
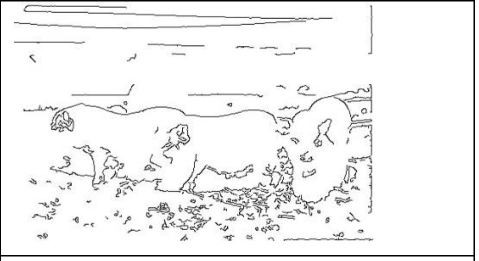
   g. The input data consists of Input image, with annotations for each 16*16 patch.
   h. This way the model is trained on BSDS 500 dataset.
2. Random Forest Classifier:
   a. Random Forest classifier in an ensemble type of Learning algorithm. It combines votes from various decision trees.
   b. A decision tree is a supervised learning algorithm with selects a feature from feature space and divides the region into 2 spaces. This is carried out at every node and the tree branches build up.
   c. The criterion function of decision tree is dependent on 4 parameters:

i) Region selection for further splitting
ii) Feature_j selection
iii) Threshold value of Feature_j to split the region
iv) Classifying the split region as positive or negative class.

d. The tree is built until some stopping criterion function is matched. For example,
   i) Total number of nodes
   ii) 0 classification error
   iii) No change in classification error between previous and current split
   iv) Total number of iterations.
e. Decision Tree is prone to overfitting if the number of nodes are large. Hence it's experimentally proved to keep max_nodes =3
f. Random Forest is built upon many such Decision Trees.
   i) We randomly select many subsets of Dataset with replacement.
   ii) Each of these subsets give rise to one decision tree and one classifier/regressor function.
   iii) We then take a vote/average of these decisions from each tree and give our final decision.

3. Parameters and comparison with Canny
   a. Parameters: (I have chosen the default parameters)
      i) Multiscale : The paper states that it works at 3 different resolution(original,half and double). This is done to include fine features as well as multi-scale objects/patterns.
      ii) Sharpen: This is used to align the predicted segmentation mask along with the underlying image data in order to get sharp, strong edges. This helps to reduce the effect of diffusion.
      iii) nTreeEval: [1,4] The depth of each tree.
      iv) Nms: Non-Maximal suppression is used to thin the edges. Sets the edge width to 1px.
   b. Structured Edge Vs. Canny

Structured Edge	Canny Edge
	
Less Background noise	Relatively more background noise
Outlines the structure of tiger which is easy human interpretable	Outlines the structure of tiger with too many minute features which is difficult for human interpretation

	
Thick edges as nms was set to False	Fine thin edges due to nms operation
Almost no background noise	Lot of background noise
Edges are well shaped/structured and identifiable	Edges are smooth but get mixed up with other edges

- Performance Evaluation
  1. Comparison between Different Edge Detectors

	Sobel	Canny	Structured Edge
Pros	Easy implementation Easy to interpret Works on any image and any type of edge	Only outputs Strong candidates of edges Fine thin edges due to nms Filters out a lot of noise at the start	Very fast, works in real-time Fine crisp edges Negligible background noise
Cons	Bakground Noise is too large. Thick edges which merge in each other	Need to fine tune 2 threshold values Noise still persists Possibility of missing out on fine features due to double thresholding	Data-driven approach. Classifies only those edges on which the model has been trained. Need to train model.

  2. F_score comparison:

	Sobel		**Canny**		**Structured Edge**	
Best F-score (tiger)	0.27	The plot increases monotonically. With increase in threshold there is considerable increase in precision as total positives detected decreases due to decrease in false positive. The curve has an upward opening.	0.42	The plot also increases monotonically. Although, it starts from best Sobel F-score and goes as high as 0.42. The curve has a downward opening and it seems that further increase in lower threshold would achieve a saturation.	0.48	The F-score plot decreases monotonically with increase in threshold. It has a steep descent.
Best F-score (pig)	0.31	The plot increases monotonically. The curve has a downward	0.38	It's difficult to examine the nature of plot with both changing high and low	0.55	The plot decreases monotonically with increase in threshold. It has a steep descent.

		opening which suggests the curve might saturate with higher threshold value.		threshold. The curve achieves the max score for threshold pair (100,200). This ratio justifies the suggestions given in the paper. For the same low threshold=100, curve has a downward fall for higher values of High Threshold.		

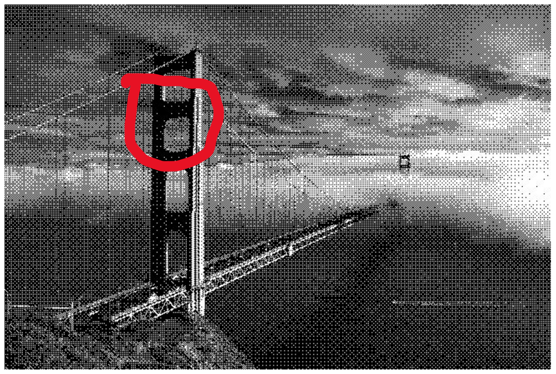
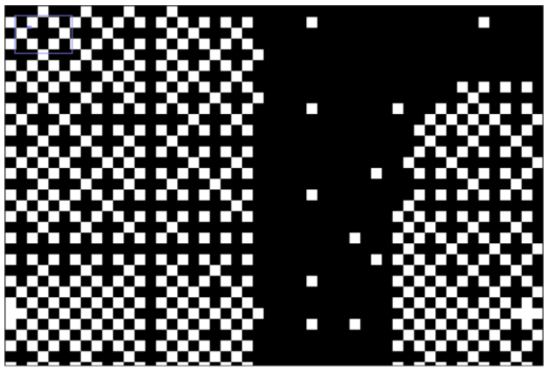
3. Tiger Or Pig F-score comparisons:
   a. From the experimental results, it seems that its easier to receive high F-scores for Pig image.
   b. This can be reasoned as follows: The Pig image covers larger portion of image as fore-ground objects. There are 3 pigs in the image compared to one tiger in another image.
   c. Edge Detectors are usually prone to unwanted background edges. Hence, if the image is covered more by the object (recognized by Ground Truth) would ensure less occupancy of background and hence less chances of detecting unwanted background edges.
   d. This would mean decrease in false positive and hence increase in Precision.
   e. Another explanation that I can think of is, tiger is occluded by a lot of tall grass whereas pigs are standing on ground which has no tall grass on it.
4. F-Measure and relationship between Precision and recall:
   a. F-score ensures balance between a good Precision and a good Recall. It's a tradeoff between precision and recall.
   b. If one is only dependent on either precision or recall, then the model is not the desirable. For example. in above edge detection algorithms, one can set all pixels in output image as 1 (edges). This would mean that we have Recall=1 as all edges have been eventually detected. But we know that it is not desirable or what our end goal is.
   c. NO, it would not be possible to get a high F-score if either one of Precision and Recall is significantly higher than the other.
      Let's say Precision >> Recall then,
      $F = 2 * \frac{P*R}{P+R} ==> F \approx 2 * \frac{P*R}{P} ==> F \approx 2 * R$ But we know than Recall takes smaller value. Hence F would be smaller.
   d. If (Precision + Recall) = k (some constant) then,
      $F = 2 * \frac{P*R}{P+R} ==> F = 2 * \frac{(k-R)*(k-P)}{k}$
      Now, taking partial derivative of F wrt P and R,
      $$\frac{df}{dP} = \frac{2}{k} * (k-R) * (-1) \quad \& \quad \frac{df}{dR} = \frac{2}{k} * (k-P) * (-1)$$
      Now, equating the above derivatives to 0 we get,
      $\frac{df}{dP} = 0 ==> \frac{2}{k} * (k-R) * (-1) = 0 ==> k = R$ --- (i)

$$\frac{df}{dR} = 0 ==> \frac{2}{k} * (k - P) * (-1) = 0 ==> k = P \text{ --- (ii)}$$

From (i) & (ii) we get,
P = R
Hence proved ■
F measure reaches maximum when Precision = Recall if their sum is kept constant.

Problem 2: Digital Halftoning

- **Abstract and Motivation:**
  Printing a digital image gives rise to Digital Halftoning. The image printed consists of only black dots on white paper, but our eyes visualize it as a grayscale image. A grayscale image has 256 tones which is not possible to install inside a printer. So, in order to do that, we break down image into cells. Now a cell with many black dots would look blacker than the cell with a smaller number of black dots. This is heavily used in printing newspapers. There are various methods of implementing the same and we implement 2 such methods: Dithering and Error diffusion. For example, the below picture does not look binary but rather some tones of grayscale. But if we zoom in a particular region (right image), we see that its either a black dot or nothing. (This is result of I32 threshold matrix based Dithering)

- **Approaches and Procedures:**
  Method 1: Dithering
  - **Fixed Thresholding:**
    - Input1: Grayscale image (raw); Output1: Half-toned image (raw)
    - We compare each pixel intensity of input image with a fixed threshold and map it as 255 if it has higher intensity then threshold else 0.
    - This divides the 256 tones of grayscale into 2 ranges.
    - Threshold values experimented: 100, 128, 200
  - **Random Thresholding:**

- Input1: Grayscale Image (raw); Output1: Half toned image (raw)
- This method is similar to Fixed thresholding except for the part of Threshold value.
- Here, we generate random number in range of (0-255) as threshold for each pixel location.
- And then map a pixel to 255 if its value is greater than the randomly generated threshold value.
- Computation increases compared to fixed thresholding.
- I use srand(time(0)) to initialize the random generator and then generate new threshold for each pixel by using rand()%255.

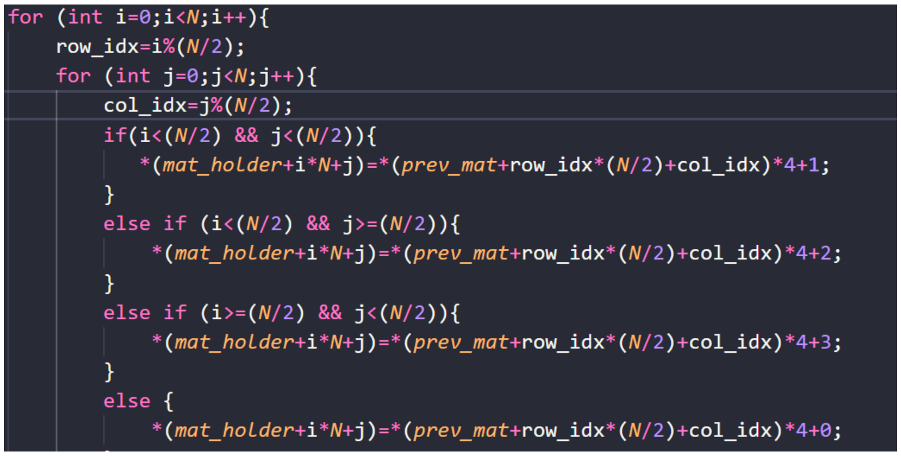
o **Dithering Matrix:**

- Input1: Grayscale Image (raw); Output1: Half toned image_I2; Output2: Half toned image_I8; Output3: Half toned image_I32
- We first iteratively build index matrix I4, I8, I16, I32 using the initial given I2 matrix. This is done using the given formula

  $I2(i,j)= \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix}$

$$I_{2\mathrm{n}}(i,j) = \begin{bmatrix} 4 \times I_{\mathrm{n}}(i,j) + 1 & 4 \times I_{\mathrm{n}}(i,j) + 2 \\ 4 \times I_{\mathrm{n}}(i,j) + 3 & 4 \times I_{\mathrm{n}}(i,j) \end{bmatrix}$$

- This is implemented as followed



```
for (int i=0;i<N;i++){
    row_idx=i%(N/2);
    for (int j=0;j<N;j++){
        col_idx=j%(N/2);
        if(i<(N/2) && j<(N/2)){
           *(mat_holder+i*N+j)=*(prev_mat+row_idx*(N/2)+col_idx)*4+1;
        }
        else if (i<(N/2) && j>=(N/2)){
            *(mat_holder+i*N+j)=*(prev_mat+row_idx*(N/2)+col_idx)*4+2;
        }
        else if (i>=(N/2) && j<(N/2)){
            *(mat_holder+i*N+j)=*(prev_mat+row_idx*(N/2)+col_idx)*4+3;
        }
        else {
            *(mat_holder+i*N+j)=*(prev_mat+row_idx*(N/2)+col_idx)*4+0;
```

To build the next index matrix, I first check whether the position of the current pixel I_2n(i,j) is present in the top left/top right/bottom left/bottom right quadrants of the I_2n matrix. And then the current pixel value is computed from previous iteration index matrix I_n. For example
I_4(i,j) = 4*I_2(i%(4/2), j%(4/2)) + quadrant   where,
Quadrant = 1, if top left quadrant
              2, if top right quadrant

3, if bottom left quadrant
0, if bottom right quadrant

- Next, we build the Threshold matrix, which is given by

$$T(x,y) = \frac{I_N(x,y) + 0.5}{N^2} \times 255$$

- Now, once we have this we iterate over each pixel of our input image and compare with the corresponding threshold value from the respective Threshold matrix. The threshold value for particular pixel (i,j) is located by Threshold_matrix_N (i%N, j%N).
- Finally, we map the pixel to 0 if its value is lower than the corresponding threshold value else 255.
- We repeat this for I2, I8, I32. Note in order to build I8, I4 is also constructed and similarly for I32, I16 is additionally constructed.

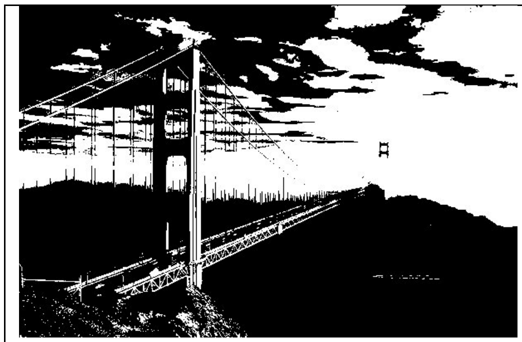
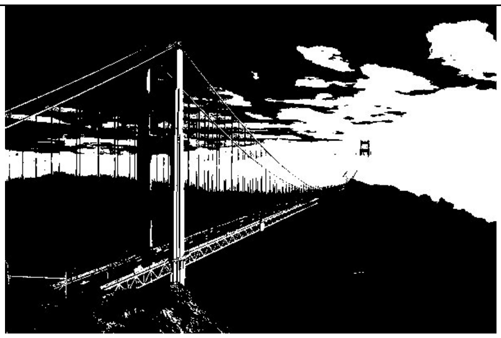
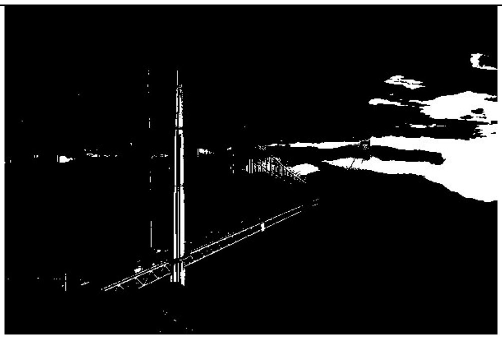
Method 2: Error Diffusion

- Input1: Grayscale image (raw); Output1: Half-toned Floyd (raw); Output2: Half-toned JNN (raw); Output3: Half-toned Stucki (raw)
- We first initialize f_hat matrix as input image matrix. The datatype of f_hat matrix is double in order to deal with negative and float values.
- Then, we iterate over each pixel in f_hat (baring the boundary pixels). For each pixel, we binarize using a fixed Threshold (=128).
- Then, we compute signed error given by difference of f_hat(i,j) – Binarized(i,j).
- This error is diffused to the future pixels. For this we experiment with 3 techniques: Floyd-Steinberg, JNN, Stucki.
- To do the above step we multiply the desired filter with signed error and add it to f_hat around the neighborhood of the current pixel. Hence our f_hat is updated.
- We then move forward.
- The scanning (iterating) over each pixel is done in serpentine manner. To do this we check the row index. On basis whether its even/odd and Size or filter, we scan from left to right or right to left.
- Also, when going from right to left, we mirror the filter in order to diffuse error to only the future pixels.
- Once the iteration is completed, I binarize the entire updated f_hat matrix with the same threshold and the resultant of this is my output image.

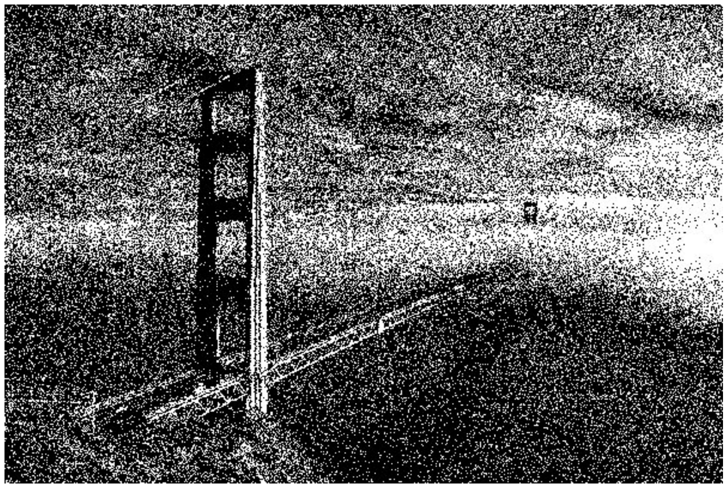
HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

EE569-HW2

- **Experimental results:**
- **Dithering:**
    - **Fixed Threshold:**

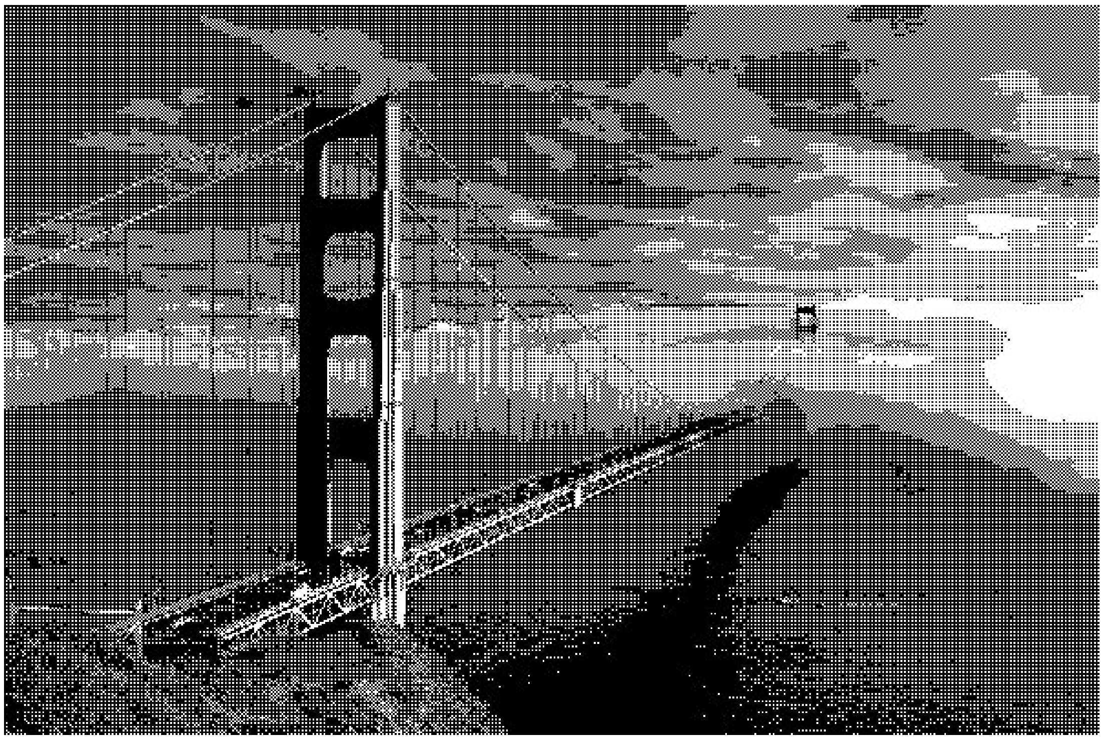
		
Threshold = 100	**Threshold = 128**	**Threshold = 200**

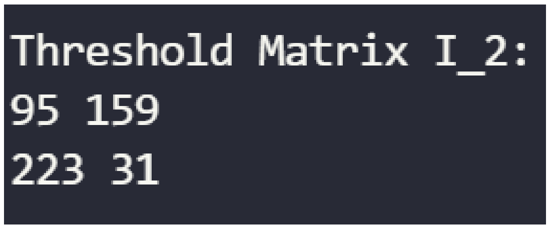
- **Random Threshold:**



HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

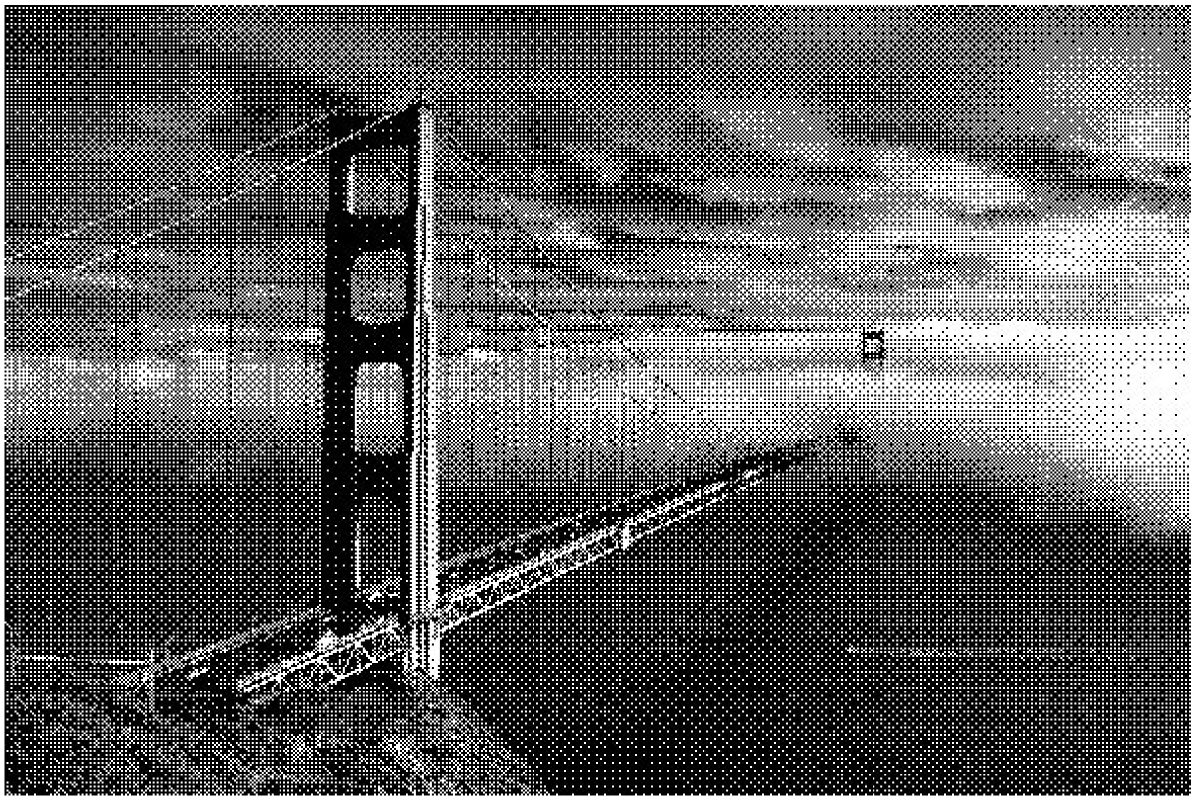
- **Dithering Matrix:**
    - **I2 dithering Matrix:**

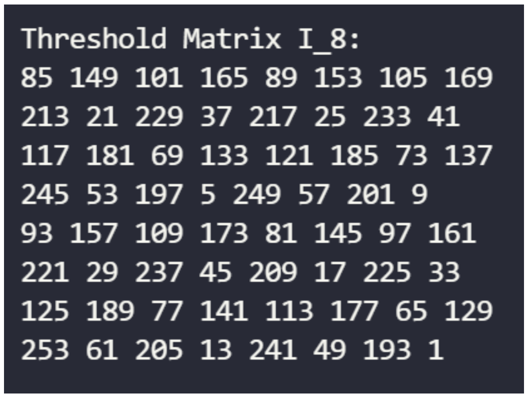




```
Threshold Matrix I_2:
95 159
223 31
```

- **I8 dithering Matrix:**

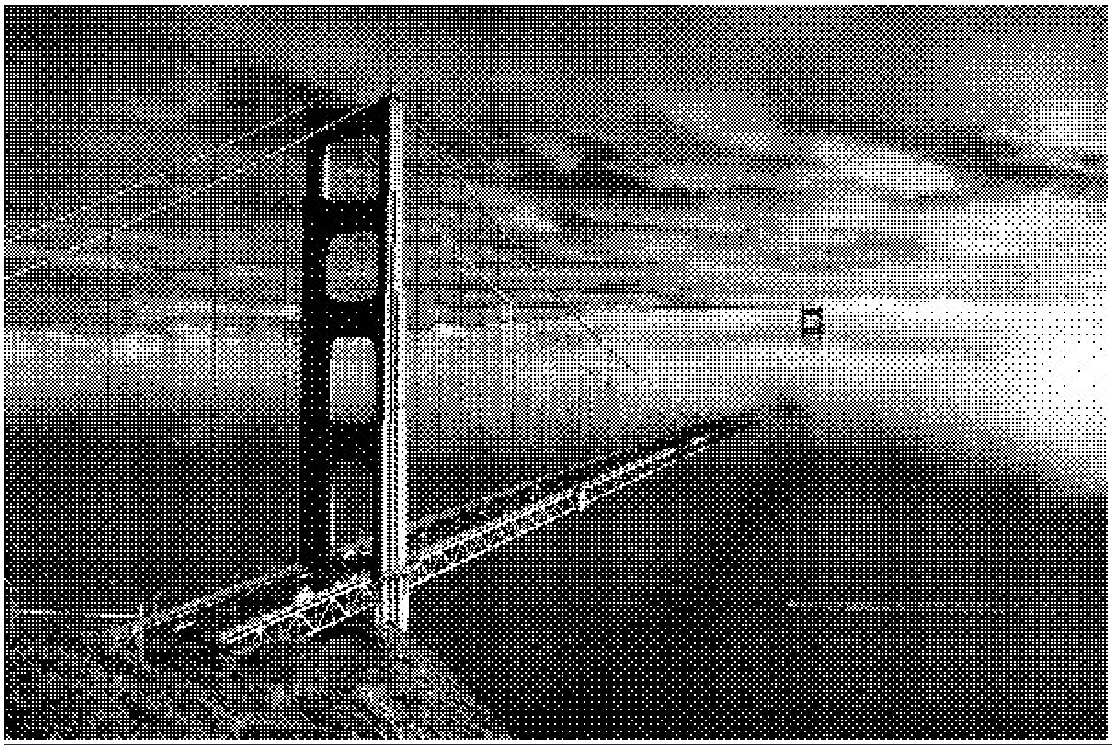


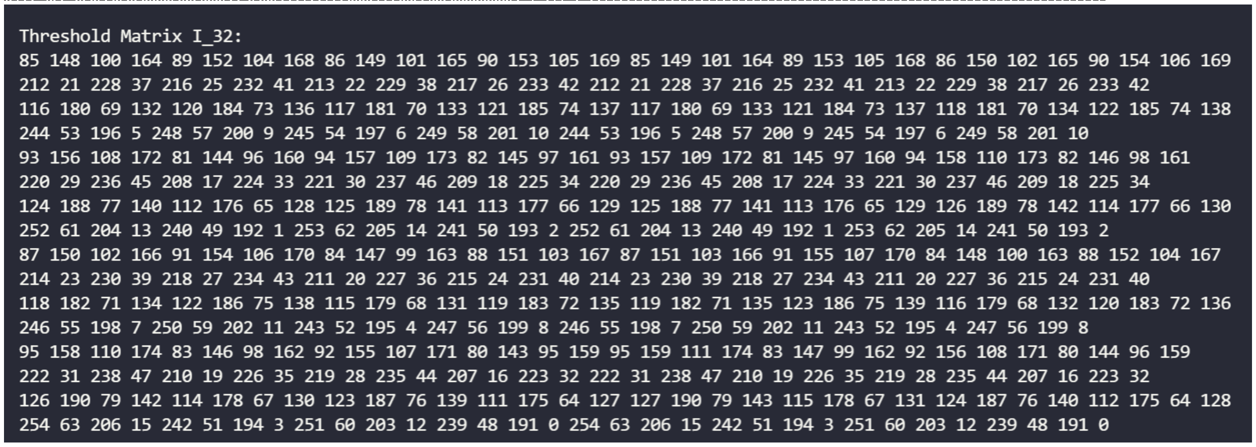


```
Threshold Matrix I_8:
85 149 101 165 89 153 105 169
213 21 229 37 217 25 233 41
117 181 69 133 121 185 73 137
245 53 197 5 249 57 201 9
93 157 109 173 81 145 97 161
221 29 237 45 209 17 225 33
125 189 77 141 113 177 65 129
253 61 205 13 241 49 193 1
```

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

- **I32 dithering Matrix:**

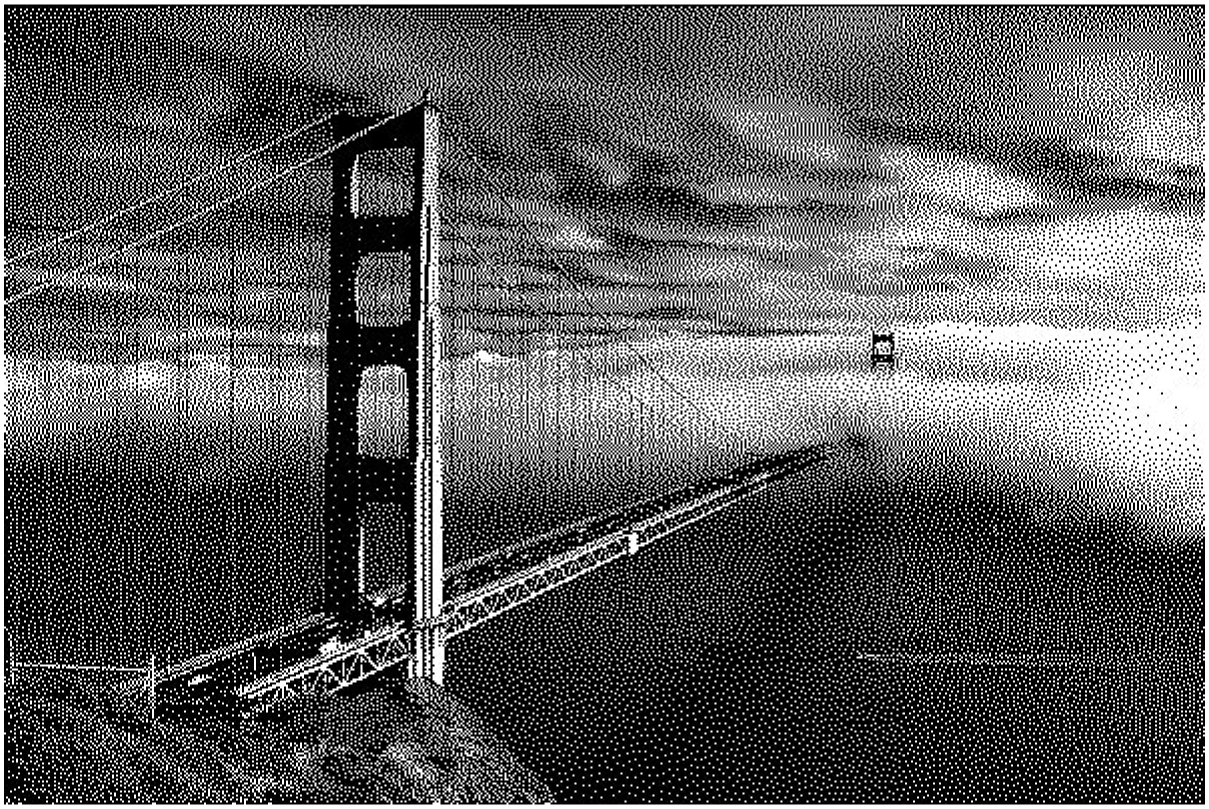




```
Threshold Matrix I_32:
85 148 100 164 89 152 104 168 86 149 101 165 90 153 105 169 85 149 101 164 89 153 105 168 86 150 102 165 90 154 106 169
212 21 228 37 216 25 232 41 213 22 229 38 217 26 233 42 212 21 228 37 216 25 232 41 213 22 229 38 217 26 233 42
116 180 69 132 120 184 73 136 117 181 70 133 121 185 74 137 117 180 69 133 121 184 73 137 118 181 70 134 122 185 74 138
244 53 196 5 248 57 200 9 245 54 197 6 249 58 201 10 244 53 196 5 248 57 200 9 245 54 197 6 249 58 201 10
93 156 108 172 81 144 96 160 94 157 109 173 82 145 97 161 93 157 109 172 81 145 97 160 94 158 110 173 82 146 98 161
220 29 236 45 208 17 224 33 221 30 237 46 209 18 225 34 220 29 236 45 208 17 224 33 221 30 237 46 209 18 225 34
124 188 77 140 112 176 65 128 125 189 78 141 113 177 66 129 125 188 77 141 113 176 65 129 126 189 78 142 114 177 66 130
252 61 204 13 240 49 192 1 253 62 205 14 241 50 193 2 252 61 204 13 240 49 192 1 253 62 205 14 241 50 193 2
87 150 102 166 91 154 106 170 84 147 99 163 88 151 103 167 87 151 103 166 91 155 107 170 84 148 100 163 88 152 104 167
214 23 230 39 218 27 234 43 211 20 227 36 215 24 231 40 214 23 230 39 218 27 234 43 211 20 227 36 215 24 231 40
118 182 71 134 122 186 75 138 115 179 68 131 119 183 72 135 119 182 71 135 123 186 75 139 116 179 68 132 120 183 72 136
246 55 198 7 250 59 202 11 243 52 195 4 247 56 199 8 246 55 198 7 250 59 202 11 243 52 195 4 247 56 199 8
95 158 110 174 83 146 98 162 92 155 107 171 80 143 95 159 95 159 111 174 83 147 99 162 92 156 108 171 80 144 96 159
222 31 238 47 210 19 226 35 219 28 235 44 207 16 223 32 222 31 238 47 210 19 226 35 219 28 235 44 207 16 223 32
126 190 79 142 114 178 67 130 123 187 76 139 111 175 64 127 127 190 79 143 115 178 67 131 124 187 76 140 112 175 64 128
254 63 206 15 242 51 194 3 251 60 203 12 239 48 191 0 254 63 206 15 242 51 194 3 251 60 203 12 239 48 191 0
```

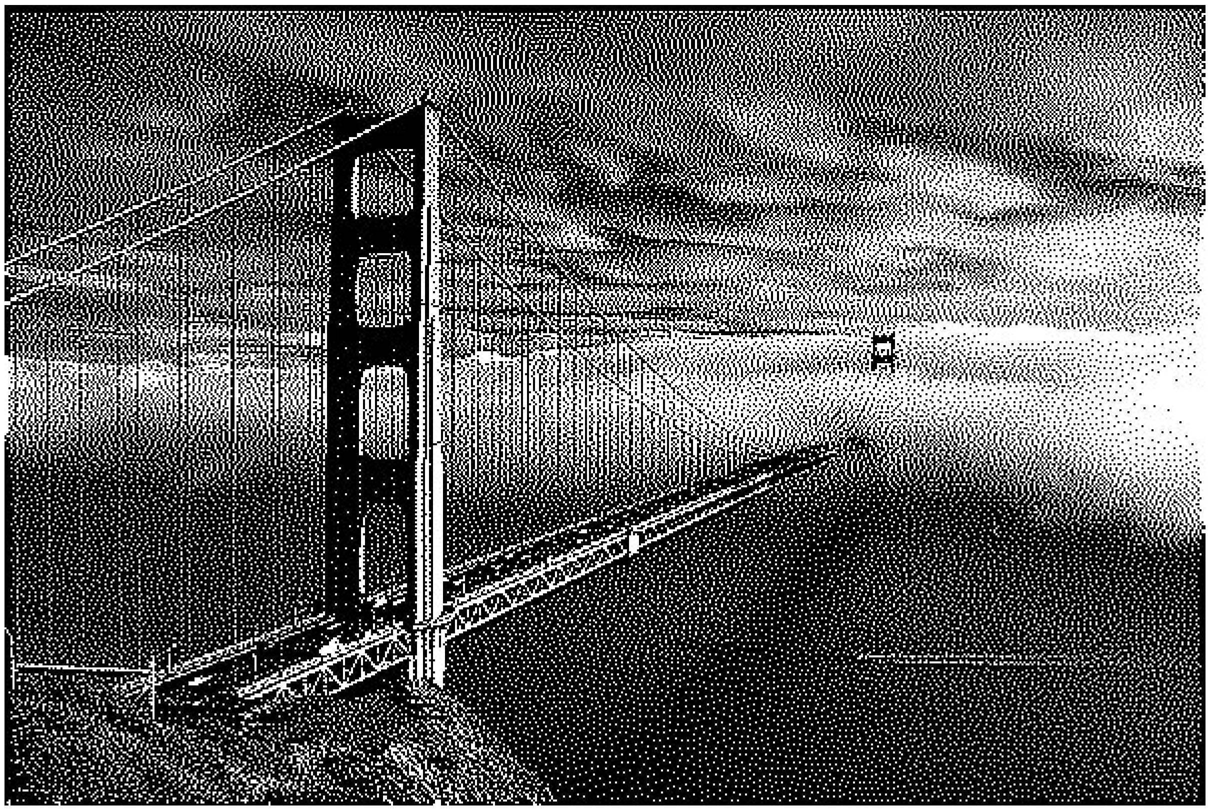
Note: This is just a snippet of the I32 threshold matrix. Not all rows are present in this image.

- Error Diffusion
  - **Floyd-Steinberg**

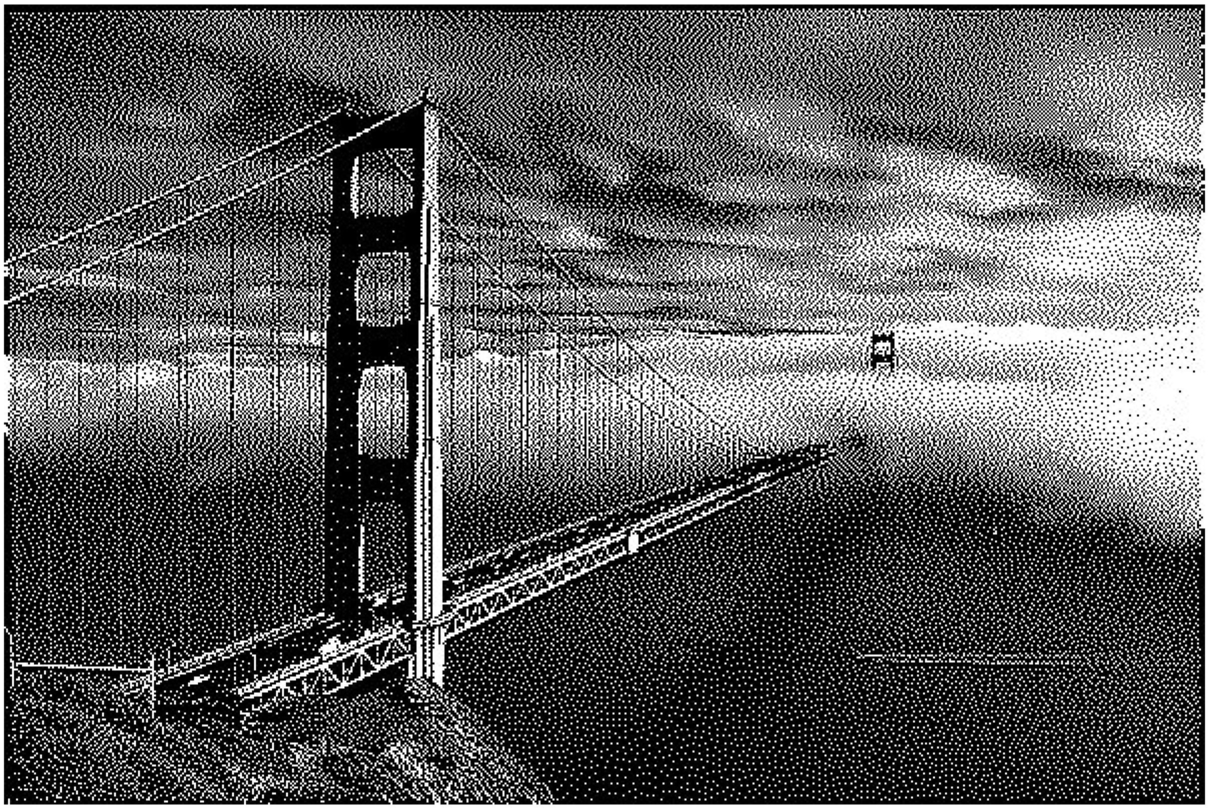


HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

- **Jarvis, Judice and Ninke (JJN) :**



- **Stucki :**



- ## Discussion:
- **Fixed Thresholding:**
  - Very straightforward technique to divide 256 tones of grayscale into 2 regions.
  - Experimented with 3 thresholds: 100, 128, 200. As threshold value increases, the image has more dark areas
  - One can easily make up that the image is binary. Hence it fails the main goal of Half-toning to create a visual of grayscale image when printed.
- **Random Thresholding:**
  - Similar to Fixed thresholding except that threshold is not fixed. Each pixel has a random threshold.
  - Does create a slight visual of half-toning.
  - Adds a lot of Random noise.
- **Dithering Matrix:**
  - We generate 3 threshold matrix of size 2x2, 8x8 and 32x32 and corresponding half-toned output images.
  - They create a good visual of grayscale.

- Lot of checkered patterns.
- As size of Threshold matrix increases, more fine details are visible. Although I feel, I8 and I32 results are very similar. Can be said that the quality of output converges with very large sized Threshold matrix.
- Much better results compared to Fixed and Random threshold techniques.

- **Error Diffusion:**
    - The results are much more detailed than the ones with Dithering matrix.
    - The ropes of bridge, the ship on right are more visible in resultant images due to Error diffusion than the ones due to Dithering matrix
    - Out of the 3 techniques of Error Diffusion, it's tough to make out any notable difference. Although according to me Floyd results have slightly less black dots (or more white dots) compared to JNN and Stucki. This can be reasoned due to size of filter, 3x3 for Floyd and 5x5 for the rest two.
    - All three has some maze kind of pattern. If watched for more time, kind of strain to the eyes.
    - I would prefer Error Diffusion technique with Floyd-Steinberg filter. My reasoning would be that, for Dithering in order to work with I16/I32, I first need to create I2,I4,I8 matrixes which is unnecessary computation. Also, Error diffusion gives much finer details.
    - My idea:
        - I would like to club the Dithering matrix and error diffusion matrix together.
        - In error diffusion, instead of having fixed threshold for binarizing the updated (initial +error) pixel value, I would rather like to use the idea of Dithering Matrix.
        - For binarizing the pixel (initial + error), the threshold value would be found from I16/I32 Threshold matrix.
        - My intuition behind this idea is, it will incorporate the 2 main fundamental core of both the techniques. i.e
            - Error Diffusion to future pixels
            - Thresholding to N^2 tones.
        - So the steps would be:
            - Build the I16 threshold matrix
            - Iterate over each pixel:
                - Binarize: Map to 0 if F_hat(i,j) <= T_16(i mod 16, j mod 16) else 255
                - Calculate error
                - Using Floyd Steinberg filter diffuse error in serpentine scanning

HARDIK PRAJAPATI
HPRAJAPA@USC.EDU
2678294168
2/20/22

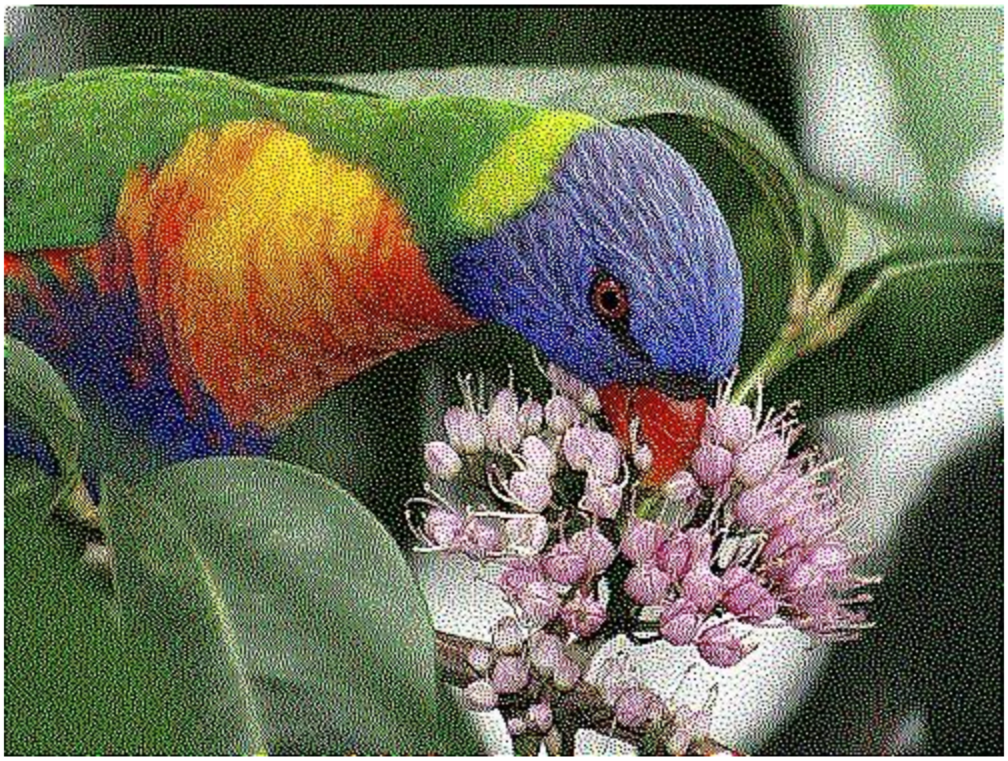
Problem 3: Color Half-toning with Error Diffusion

- **Abstract and Motivation:**
  We have color printers which consists of 3 dyes in them, RGB. If we consider all the shades of RGB, then that would sum up to 256*256*256 tones of colors. This is a huge number and is not practical to have these many color dyes. Hence, we quantize these color tones into 8 bins. They are Red, Green, Blue, Cyan, Yellow, Magenta, Black and white. We see 2 methods for achieving color Half-toning: Separable Error Diffusion and MVBQ based error diffusion.

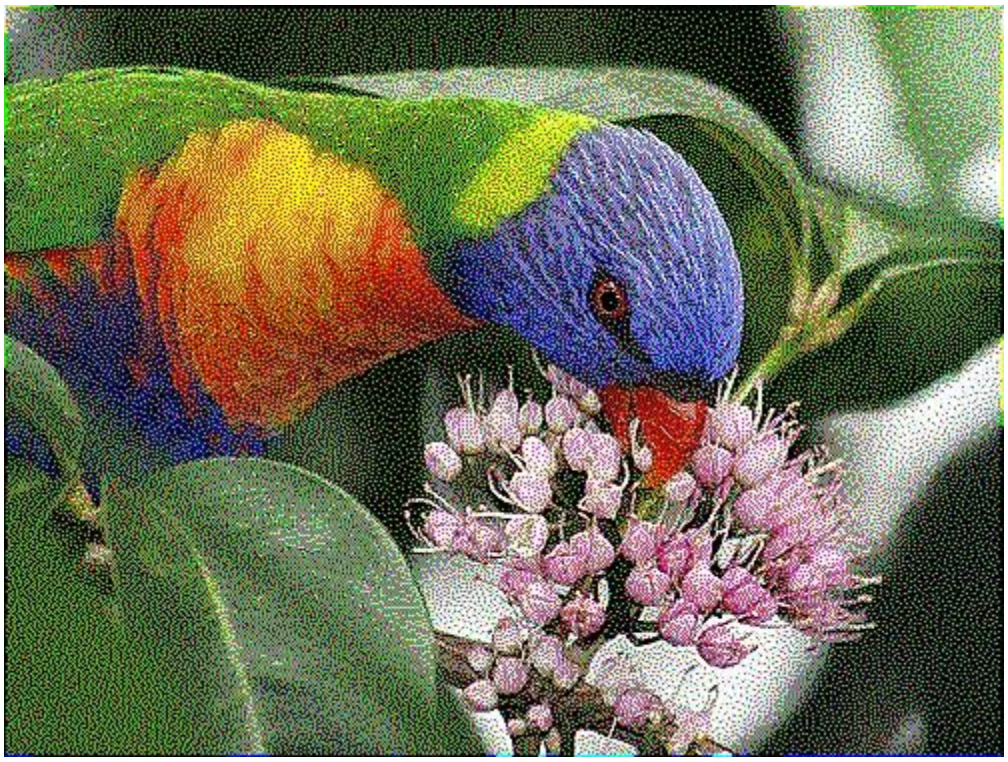
- **Approaches and Procedures:**
  - **Separable Error Diffusion:**
    - Input: RGB Color image (raw); Output1: RGB half-toned image (raw)
    - We first convert RGB image into CMY color space. C = 255 – R ; M = 255-G Y = 255 - B
    - We then separately apply the Error-diffusion technique (Floyd-Steinberg) discussed in Problem 2 above on each channel of CMY.
    - We club all the three channels back together
    - Convert the result back to RGB color space.
    - Hence, the color shades of RGB space have been quantized to 8 colors.
  - **Minimum Brightness Variant Quadrant based Error Diffusion:**
    - Input: RGB color image (raw); Output1: RGB half-toned image (raw)
    - We first separate the 3 channels and copy data to 3 2-d arrays with datatype DOUBLE.
    - Now, I first quantize the boundary pixels. For doing this, I find the quadruple in which the pixel lies and then find the nearest vertex of this quadruple with my boundary pixel.
    - Now, for the core of image, I iterate over each pixel and do the following:
      - Find the quadruple of the pixel based on the original input pixel value – (i)
      - Find the nearest vertex of this quadruple based on the updated RGB(i,j) + error value of the current pixel – (ii)
      - This quantized pixel is also saved in the output image.
      - Find the error channel wise
        err_red(i,j) = Red(i,j) – nearest_vertex(Red channel)
      - Now we diffuse this error to future pixels (channel wise) based on Floyd-Steinberg Error diffusion technique with serpentine scanning.

      - Hence, RGB values of future pixel get updated and error gets accumulated.
    - The tree structures and pseudo codes for (i) and (ii) were provided.

- **Experimental results:**
  - **Separable Error Diffusion:**



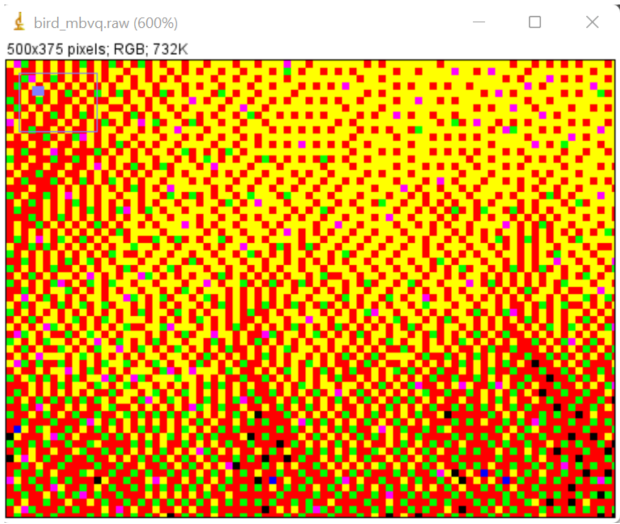
- **MVBQ based Error Diffusion:**

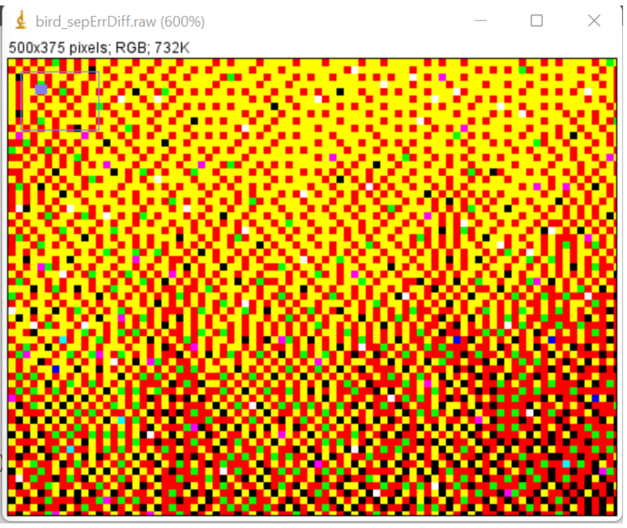


Discussion:

- **Separable Error Diffusion:**
  - This is simple yet effective idea of color half-toning.
  - The main idea to work in CMY color space rather than RGB color space directly can be reasoned as CMY has wider bandwidth as well as better light absorbing property.
  - The results have maze pattern due to FS error diffusion technique as discussed earlier
  - The main shortcoming of this technique is, it does not preserve brightness correctly.

- **MVBQ based error Diffusion:**
  - This technique divides the RGB color space into 6 dis-joint quadruples each of different brightness level.
  - The algorithm ensures that a pixel with accumulated error gets quantized to the nearest vertex of that quadruple where it originally belonged.
  - Hence, MVBQ method preserves brightness better than separable error diffusion and overcomes the shortcoming.
  - As the error diffusion is carried out by FS technique, we have maze patterns in the output of MVBQ too.
  - MVBQ and Separable Error diffusion results look quite similar to naked eye. But on careful observation, one can notice that certain regions have different brightness in both images.
  - Eg1. (Bird's body) Below is the zoomed snippet of same patch of MVBQ (left) and Separable_Error(right). We can observe presence of many white pixels in the separable Error result.





  - Similar observations can be found around leaves in the picture.