Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Problem 1: Edge Detection

I. Abstract and Motivation

Edge detection is a fundamental task in image processing area. For more than 50 years, the detect results from different methods are not good enough until the convolution neural network (CNN) method. If we detected edges in an image, we can do further processing such as segmentation, classifying and recognition. While, the definition of edge is not clear even for human, different people give different results. In different scenario, we need different level of edge detail, that means this task is hard for human let along computers. However, people can tell the difference of textures and edges by utilizing their knowledge and only pick out the most important part as edges. Blurred edge is also a challenge, it is hard to tell a clear boundary of edge. After all, edge detection algorithms should detect the edge like human does, because it is used to assist out daily work. Many edge detectors are developed to get more reasonable results. In this article, I will talk about 3 edge detectors, Sobel, Canny, Structured Edge. Implements in this article can be seen as understanding the revolution process of edge detectors before the powerful while mysterious CNN.

II. Approach and Procedures

Sobel Edge Detector

Sobel edge detector uses Sobel filters to detect edges. There are two Sobel filters,

$$F_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \text{ and } F_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad (1\text{-}1)$$

Sobel filters is one kind of edge detection operators. Do the convolution operation on a given image with these filters separately, then we can get the first-order derivative of pixel values. So, the Sobel edge detector is a first-order derivative edge detector. Using filter F_x, we can get the gradient approximation of horizontal direction, while using filter F_y, we can get the gradient approximation of vertical direction.

$$\mathrm{G}_x = F_x * A \text{ and } \mathrm{G}_y = F_y * A \quad (1\text{-}2)$$

Where A is the input image, here we only consider gray scale images, G_x and G_y are horizontal gradient and vertical gradient.

Since the edge in an image is some place with dramatic changes in values, the gradient of edges will also large. That is

why we can detect edges after we get the gradient of an image, we can consider those pixels with high gradient as edge pixels.

We can calculate the direction of the gradient change by utilizing x-gradient and y-gradient

$$\theta = \tan^{-1}\frac{G_y}{G_x} \quad (1\text{-}3)$$

As we do the convolution operation from the top-left corner to the right-bottom corner of the image, F_x detect value increasements in right direction, and F_y detect value increasements in down direction. Since the derivative of an edges pixel will always go up then go down (or opposite), we do not need to flip the F_x and F_y horizontally or vertically and do the convolution operation again.

To get a gradient magnitude without considering x-gradient and y=gradient separately, we can use the following equation

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad (1\text{-}4)$$

After this, we have a gradient magnitude image. We can do a thresholding on this gradient magnitude image, only keep pixels as edges with gradient magnitude higher than the threshold. This threshold can be set by different ways, such as we can first decide the percentage of edge pixels in the image, and then sort gradient magnitudes descendent then get a threshold by picking out the value at certain percentage of all the values.

Sobel edge detector is very simple, comparing with other detectors that I will talk about later, it only has two steps – calculate the gradient & thresholding.

In my implementations of all the edge detectors, the output edge data will use 1 (true) to represent an edge pixel and 0 (false) to represent an non-edge pixel. If we want to display the edge detection result, it should be converted to an 8-bit gray scale image with 0 being the edge and 255 being the background.

Canny Edge Detector

Canny edge detector [1] is a popular edge detector except data-driving detectors. The input image should be a gray scale image. Basically, it has 4 steps. First, applying a Gaussian filter to remove noises. Before Canny edge detector, other edge detectors (such as Laplacian of Gaussian, LoG) also use Gaussian filter to remove noises at first, so this step is a quite common routine. Second, calculate the gradient magnitude using a kind of edge detection operator. The third step is called *Non-maximum suppression*, it keeps edge pixels with strongest gradient magnitude and kill edge pixels around the strongest ones with lower gradient magnitudes, only local maximum is kept. This step thinning the edge, since if we do the thresholding, those pixels with lower gradient magnitudes now have value 0 and will never become an edge pixel at last. Like what we did in Sobel edge detector, the final step in Canny edge detector is thresholding, while here we use

two thresholds. This step is called *double threshold*. There is a higher threshold and a lower threshold. Pixels with gradient magnitudes higher than the higher threshold will definitely become edge points. Similarly, pixels with gradient magnitudes lower than the lower threshold will definitely become non-edge points. The remaining pixels are those have gradient magnitude between two thresholds, further procedure should be done on each pixel to decide whether it can become an edge pixel. If the neighbor with highest gradient magnitude is higher than the higher threshold, it become an edge pixel, otherwise it become a non-edge pixel. Adding this mechanism can deal with those weak parts among strong edges if we choose good thresholds. For the last two steps, see the detailed explanation in the discussion part of this article.

Although Canny edge detector has more steps comparing with Sobel edge detector, procedures are actually quite simple and intuitive. However, we are not asked to implement Canny edge detector. There are many resources that we can use to try Canny edge detector without implementing it, because this detector is really popular. Here, I use the Canny edge detect function in image processing tool box in Matlab.

Structured Edge Detector

Structured edge detector [2] is a data driven edge detector, it should be trained at first. This detector can take color image as input. Give this detector training data and the algorithm can learn what kind of feature can be called an edge. Images are sliced into small pieces. Within each piece, the algorithm needs to find whether there is a feature. This algorithm assumes that edges are related to features, so it can use features to decide which pixels should be edge pixels. Features are pre-defined. In the training procedure, *Random Forest* algorithm is used which is a machine learning algorithm. I will discuss it later.

In this part, I use the paper author's published Matlab tool box to get edge detect results. There is already a pre-trained model using BSDS500 data set, and I will use that model and skip the training step since the training step needs huge amount of computing resources. There are some options that can be changed to affect the edge detect results, I enabled the non-maximum suppression function to get clear edges, and I leave other options unchanged based on the recommended default options. The default output of this detector is the edge probability image, since I only need binary edge image, so enabling non-maximum suppression function will always be a good choice. At last, I add a simple thresholding procedure to obtain a binary edge image from edge probability image.

Performance Evaluation

We want a criterion to evaluate how good the detectors do. Thankfully, the BSDS500 data set contains labeled data that I can use. For each image in that data set, there are corresponding edge data called ground truth which is made by human. Each image has five different ground truths, since the definition of edge is not very clear even for humans, so different person will give different answer about which part will be the edge. Here we just consider the ground truths are standard answers for edge detection.

Zongjian Li

6503-3789-43

zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

If we have an output from one of detectors, we can compare pixels one by one between the output and one ground truth. Since each pixel has two states, true and false, now we have two images, so there are total four kinds of comparing results. They are true positive, true negative, false positive and false negative. If the result named ends in "positive" or "negative", that means the edge detector says that pixel is an edge pixel or not. And the "true" or "false" tells whether the detector said is correct. We can show these results as a color image with four colors assigned to four states.

We can see the performance visually which is intuitionistic but we still need a criterion that can be directly compared by values. So we have the following indices.

Precision: $P = \frac{\#True\ Positive}{\#True\ Positive+\#False\ Positive}$ Recall: $\mathrm{R} = \frac{\#True\ Positive}{\#True\ Positive+\#False\ Negative}$ F measure: $\mathrm{F} = (\frac{1}{2}(P^{-1}+R^{-1}))^{-1} = 2\frac{P \cdot R}{P+R}$	(1-5)

We finally have F measure which combined precision and recall, because using only precision or recall the result will be biased. Each of P and R only take 2 kinds out of 4 results into consideration.

Since for each image there are 5 ground truths, so we can calculate the mean of 3 indices among ground truths. And we can also take different threshold in detectors, so we can also calculate the 3 indices among ground truths. After we calculate all of these, we can draw graphs to visualize the results or do other comparations. And these are the performance evaluation procedures I use in this article.

Values are calculated by following steps, while there are two way to define the threshold in Sobel and Structured Edge detectors, percentage of maxima gradient magnitude after stretching gradient magnitudes to 0 to 1, and percentage of edge pixels. Since I use the Canny Edge detector provided by Matlab, there are only one kind of threshold definition, the first one.

```
for each GT g:
    for each threshold t:
        compute Precision(g,t) and Recall(g,t)

for each GT g:
    compute mean_Precision_over_thresholds(g), mean_Recall_over_thresholds(g) #"mean precision, recall for each ground truth"
compute mean_F = 2 * mean(mean_Precision_over_thresholds) * mean(mean_Recall_over_thresholds) /
                     (mean(mean_Precision_over_thresholds) + mean(mean_Recall_over_thresholds))

for each thredhold t:
    compute mean_Precision_over_GTs(t), mean_Recall_over_GTs(t)
    compute F(t) = 2 * mean_Precision_over_GTs(t) * mean_Recall_over_GTs(t) / (mean_Precision_over_GTs(t) + mean_Recall_over_GTs(t))
plot F #x-axis would be threshold values
find the best F-score max(F)
```

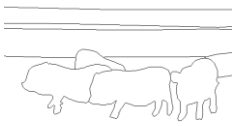
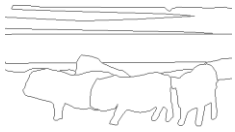
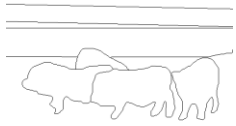
Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

III. Experimental Results

Input Images

	
Fig 1-1. Pig image *pig.raw* (481*321)	Fig 1-2. Tiger image *tiger.raw* (481*321)

Ground Truth

				
Fig 1-3. Pig ground truth #1	Fig 1-4. Pig ground truth #2	Fig 1-5. Pig ground truth #3	Fig 1-6. Pig ground truth #4	Fig 1-7. Pig ground truth #5
				
Fig 1-8. Tiger ground truth #1	Fig 1-9. Tiger ground truth #2	Fig 1-10. Tiger ground truth #3	Fig 1-11. Tiger ground truth #4	Fig 1-12. Tiger ground truth #5

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Sobel Edge Detector

Gradients

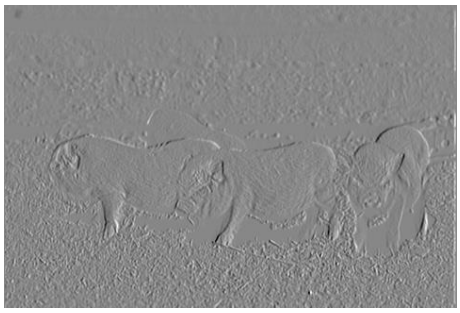
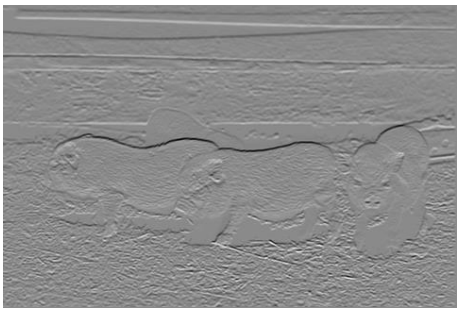
	

Fig 1-13. Pig image x-gradient (normalized to 0~255)	Fig 1-14. Pig image y-gradient (normalized to 0~255)	Fig 1-15. Pig image gradient magnitude (normalized to 0~255)

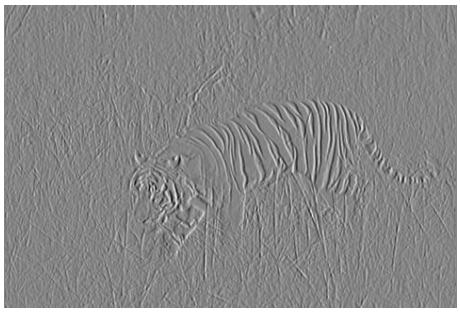
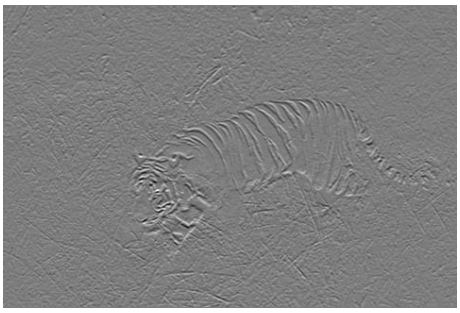

	

Fig 1-16. Tiger image x-gradient (normalized to 0~255)	Fig 1-17. Tiger image y-gradient (normalized to 0~255)	Fig 1-18. Tiger image gradient magnitude (normalized to 0~255)

Edges

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

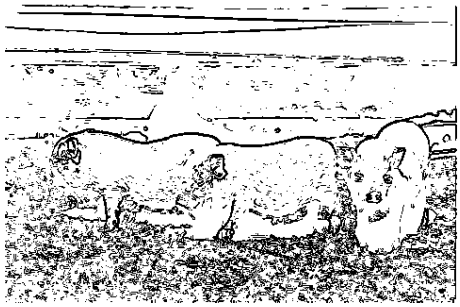
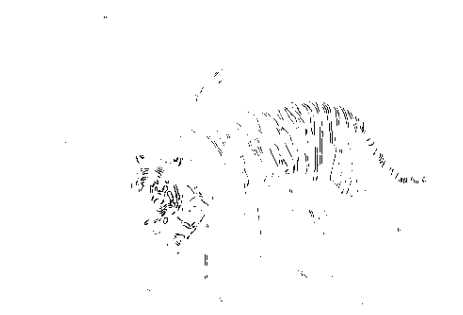


Fig 1-19. Pig image edges (T=0.16, %=0.1843, P=0.1474, R=0.8172, F=0.2498)	Fig 1-20. Pig image edges (T=0.29, %=0.0432, P=0.2551, R=0.4144, F=0.3158) **BEST**	Fig 1-21. Pig image edges (T=0.43, %=0.0099, P=0.4511, R=0.1742, F=0.2513)
		
Fig 1-22. Tiger image edges (Sobel, T=0.22, %= 0.1297, P= 0.1446, R= 0.9477, F= 0.2510)	Fig 1-23. Tiger image edges (Sobel, T=0.35, %= 0.0462, P= 0.2614, R= 0.6362, F= 0.3706) **BEST**	Fig 1-24. Tiger image edges (Sobel, T=0.53, %= 0.0115, P= 0.3018, R= 0.2054, F= 0.2444)

Performance Evaluation (here the threshold is the percentage of maxima gradient magnitude)

	Pig	Tiger

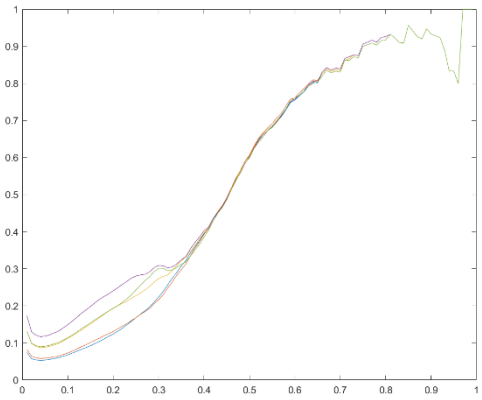
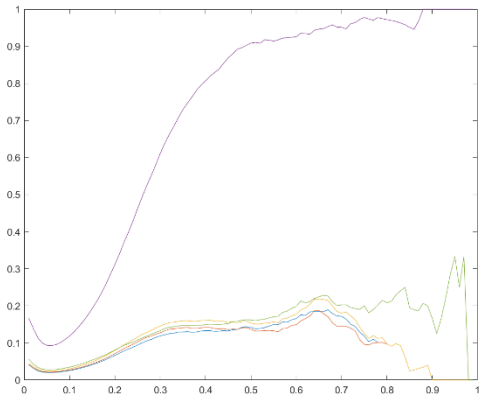
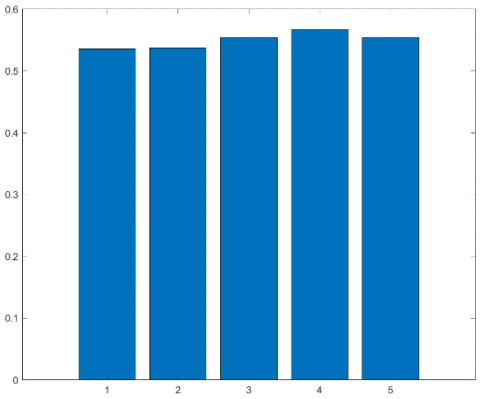
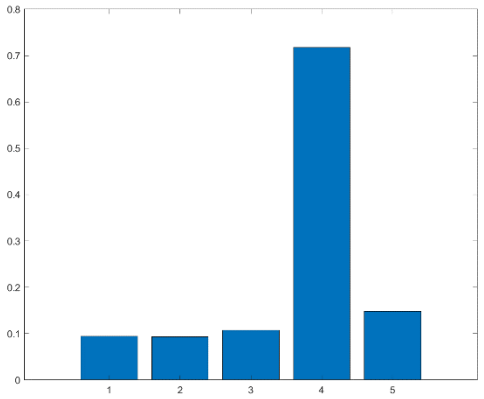
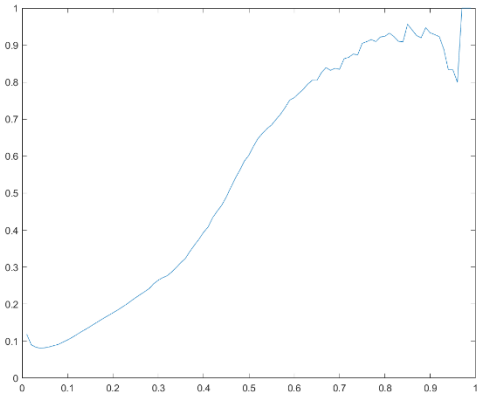
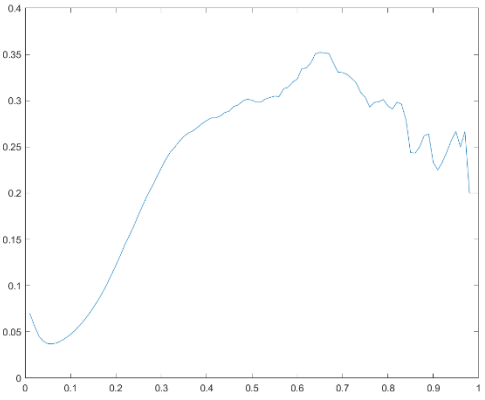
Zongjian Li

6503-3789-43

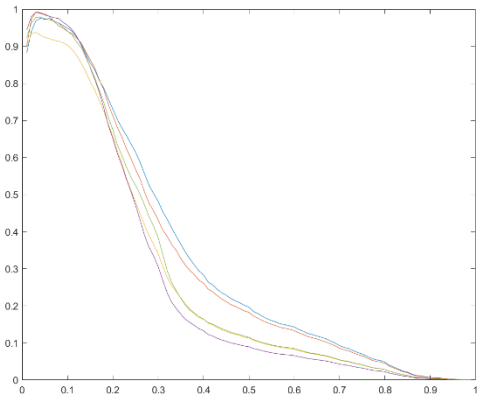
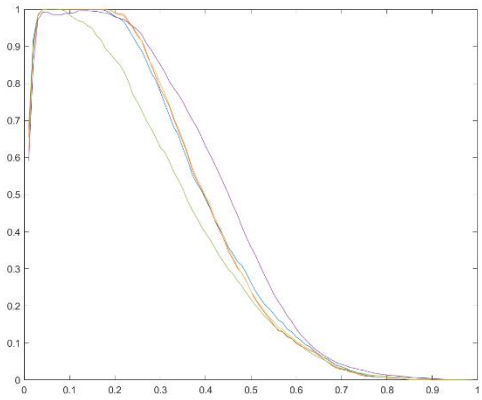
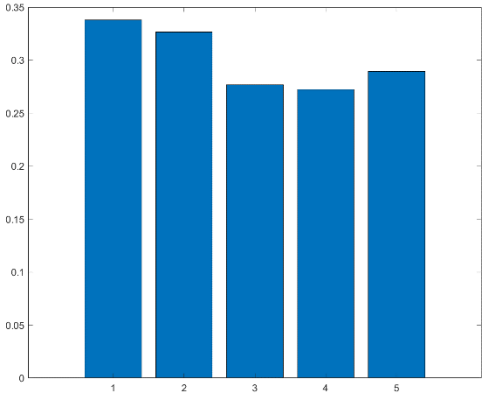
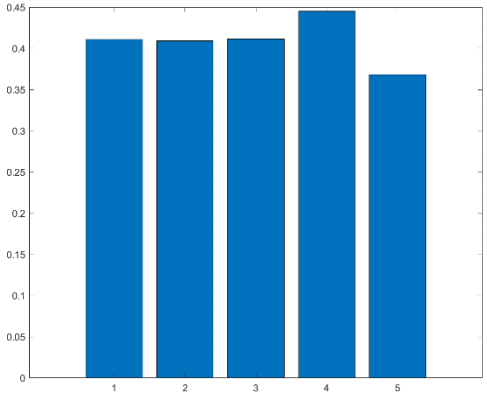
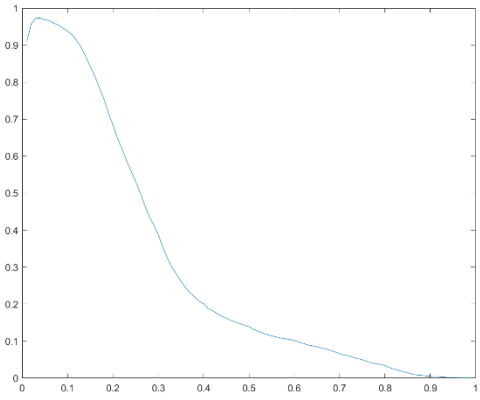
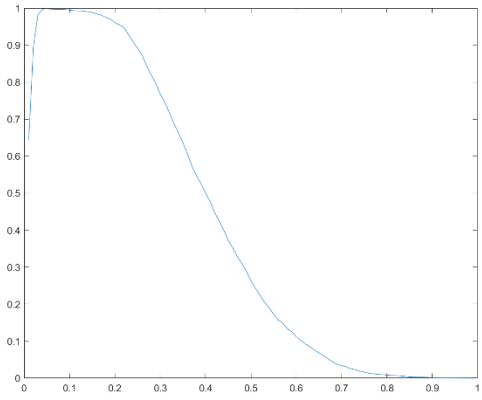
zongjian@usc.edu

EE 569 Homework #2

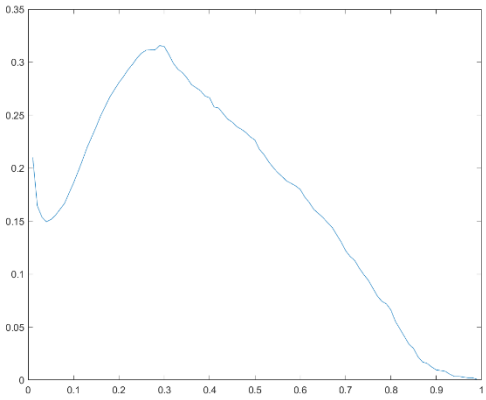
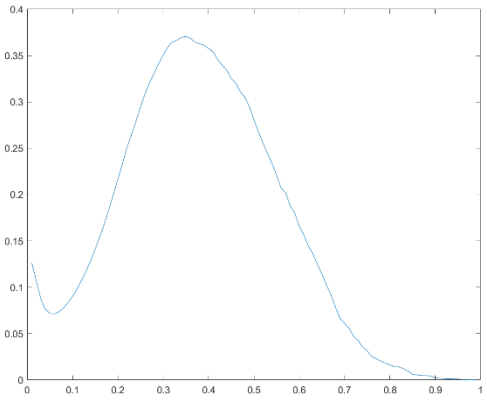
February 12th, 2019

Precision (g, t)		
Mean Precision over thresholds (g)		
Mean Precision over ground truths (t)		

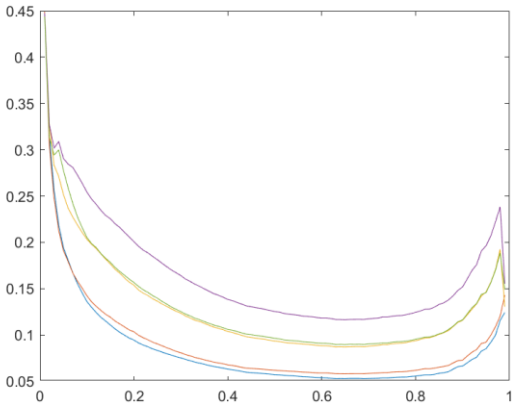
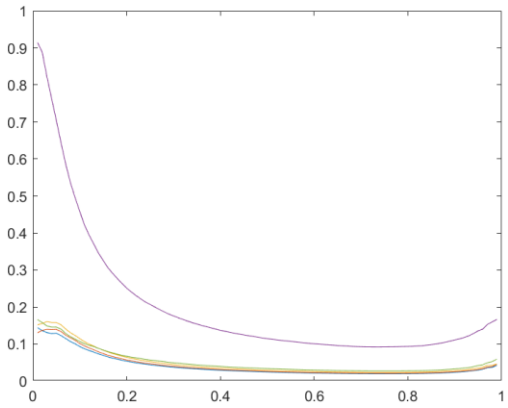
Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Recall (g, t)		
Mean Recall over thresholds (g)		
Mean Recall over ground truths (t)		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

F (t)		
Mean F	0.3728	0.2963
Best F	0.4765	0.3705

Performance Evaluation (here the threshold is the percentage pixels become edge points, not a good idea but requested)

	Pig	Tiger
Precision (g, t)		

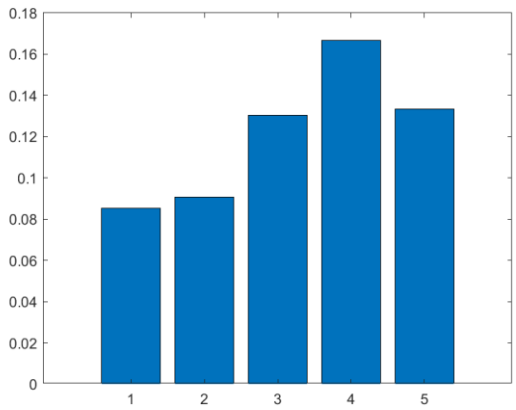
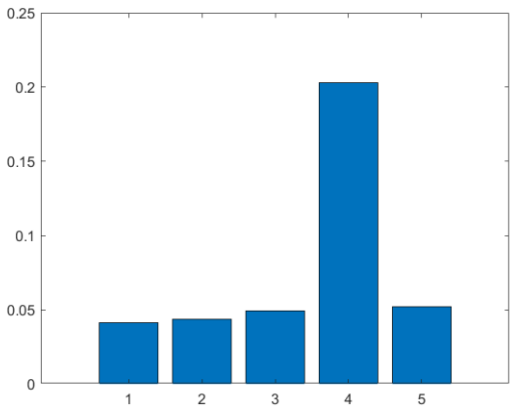
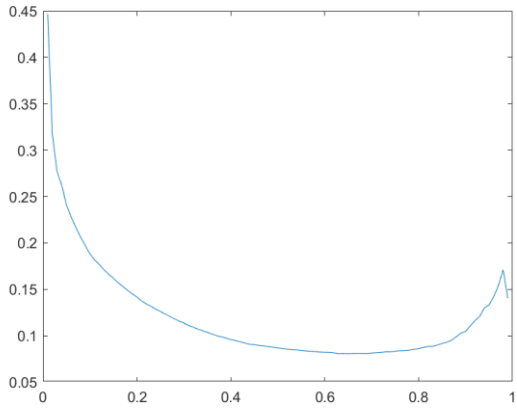
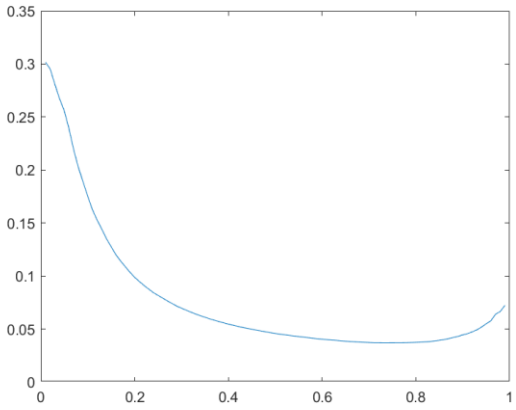
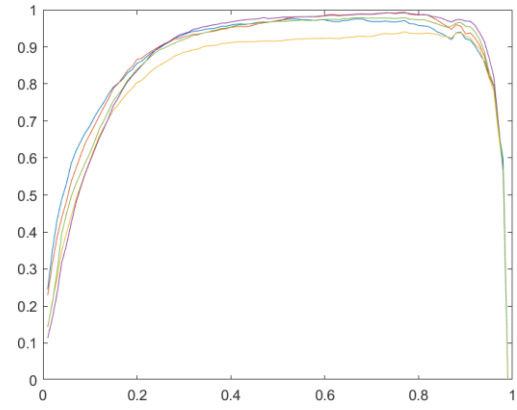
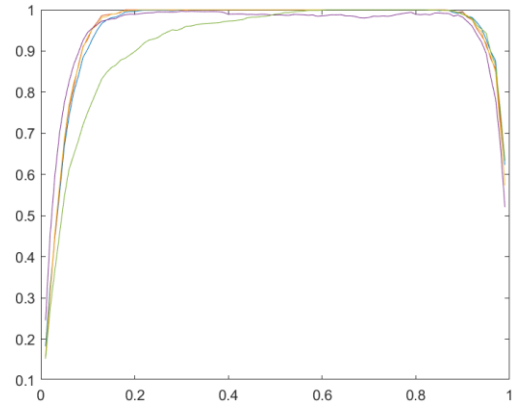
Zongjian Li

6503-3789-43

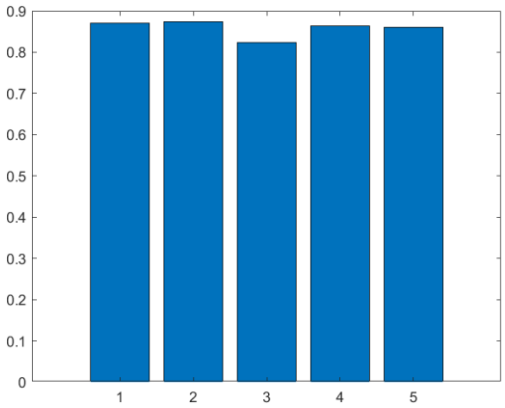
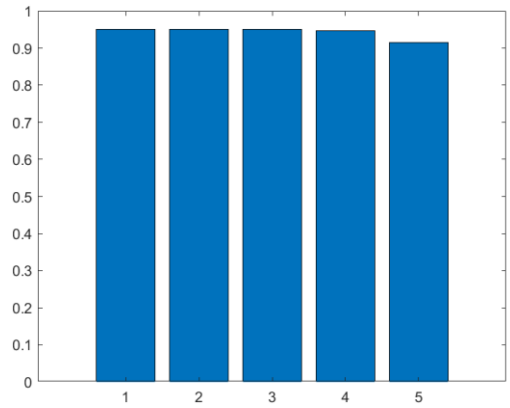
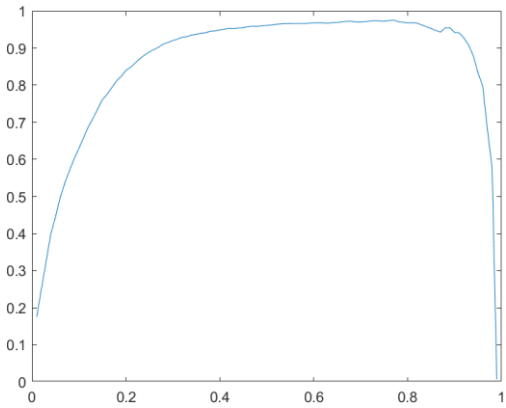
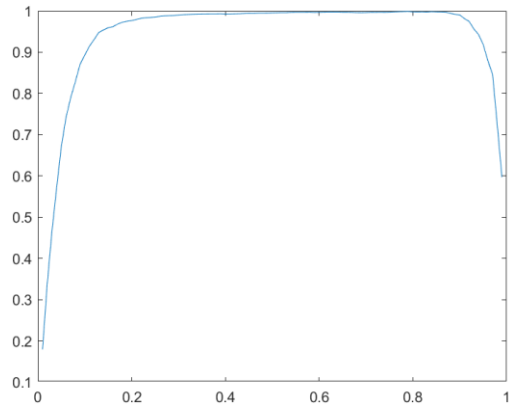
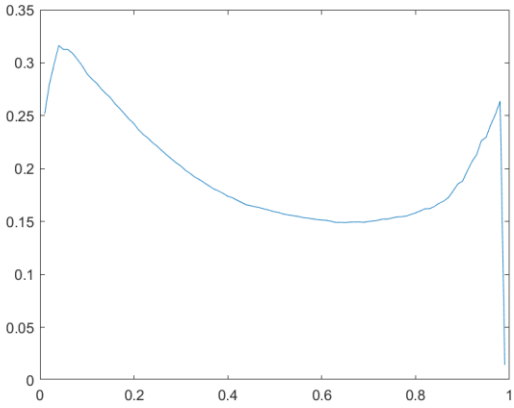
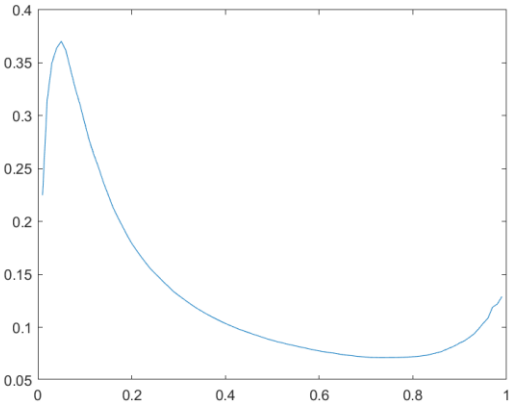
zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

Mean Precision over thresholds (g)		
Mean Precision over ground truths (t)		
Recall (g, t)		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Mean Recall over thresholds (g)		
Mean Recall over ground truths (t)		
F (t)		
Mean F	0.2126	0.1438
Best F	0.3163	0.3703

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Canny Edge Detector

Edges

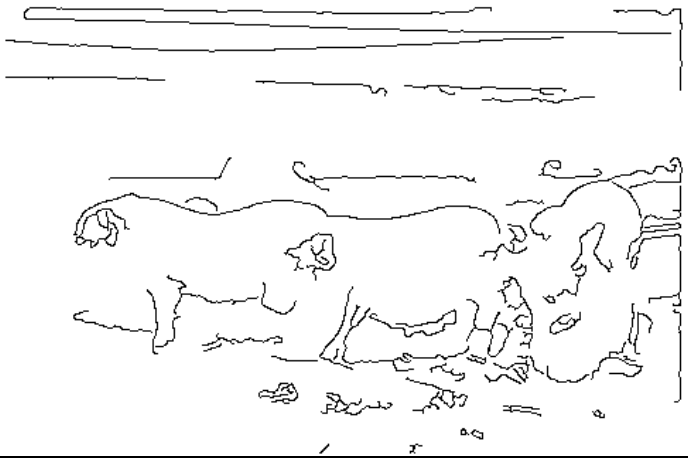
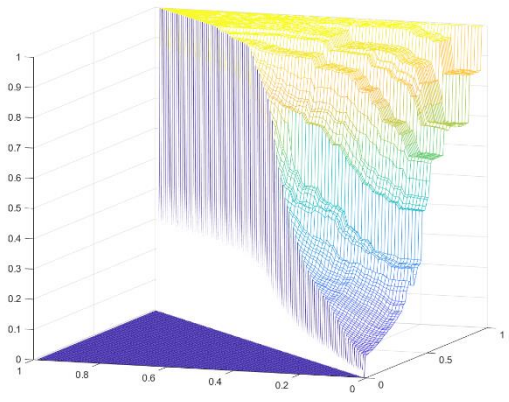
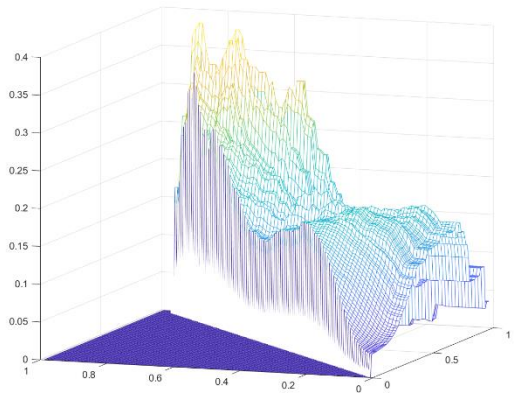
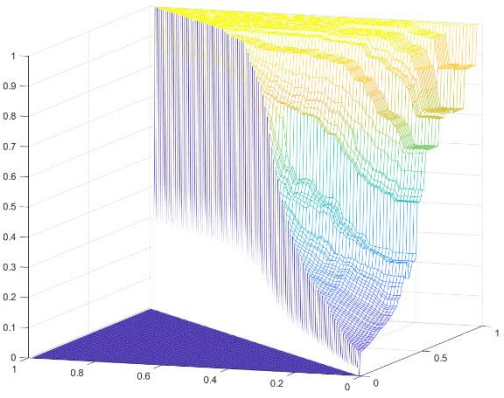
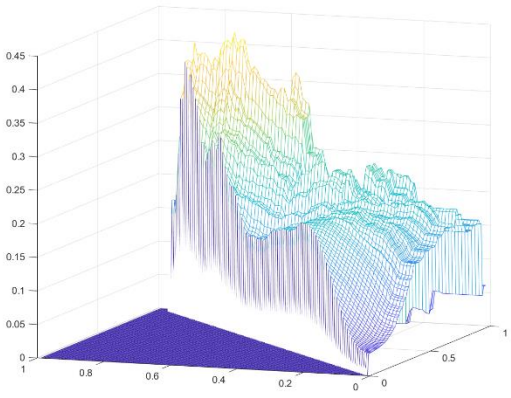
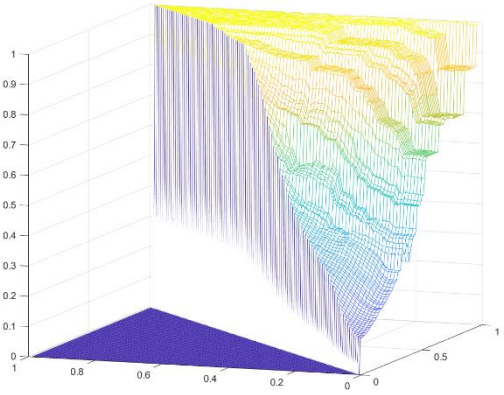
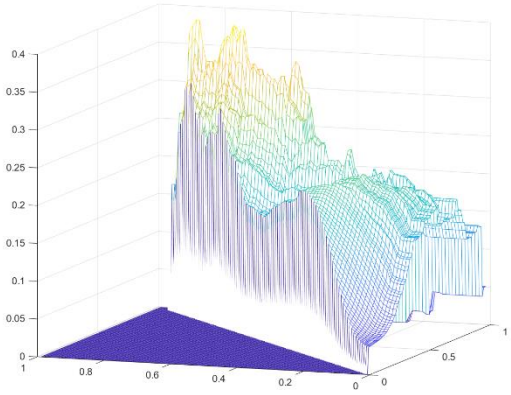
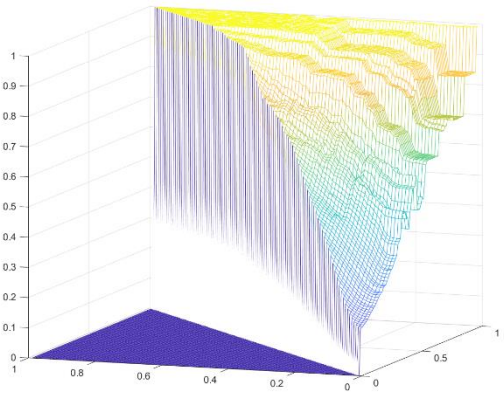
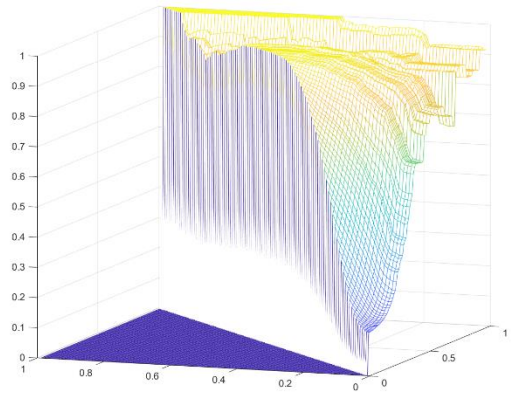


Fig 1-25. Pig image edges (Canny, T1=0.12, T2=0.37, %= 4.00, P=0.3784, R=0.3784, F=0.4765)	Fig 1-26. Tiger image edges (Canny, T1=0.19, T2=0.47, %= 3.27, P=0.3488, R=0.6871, F=0.4627)

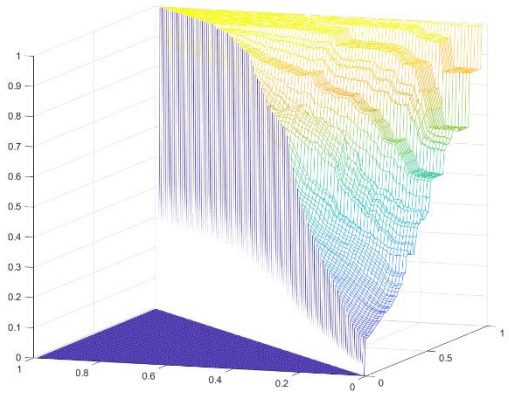
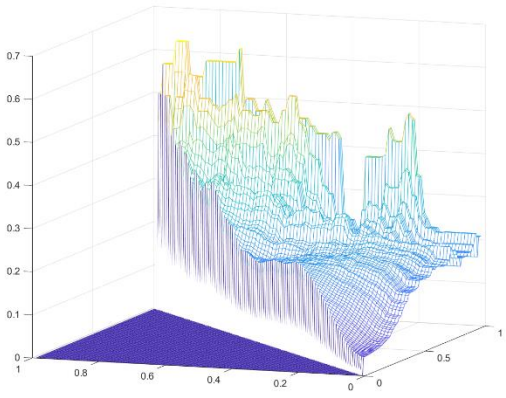
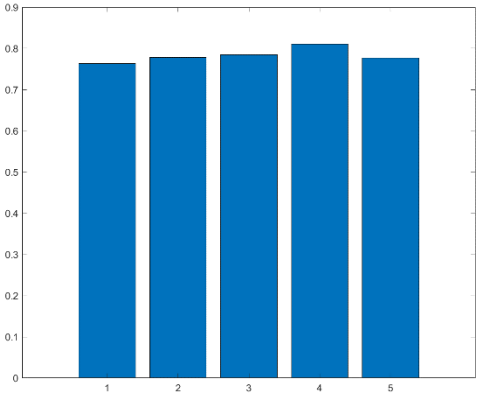
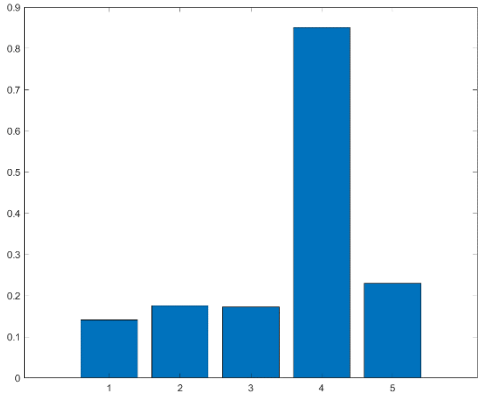
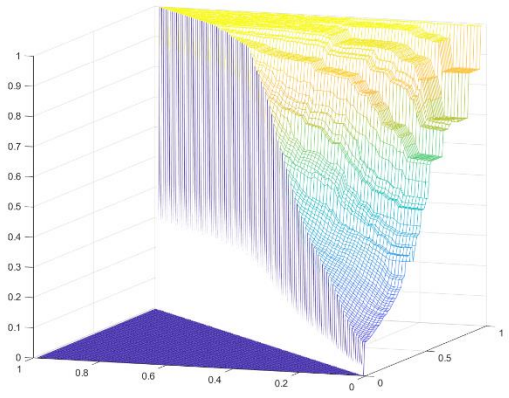
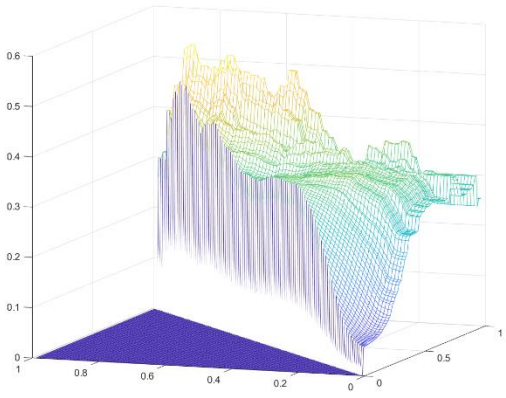
Performance Evaluation (here the thresholds are the percentage of maxima gradient magnitude, only this kind of threshold is available)

	Pig	Tiger
Precision (g, t1, t2)		

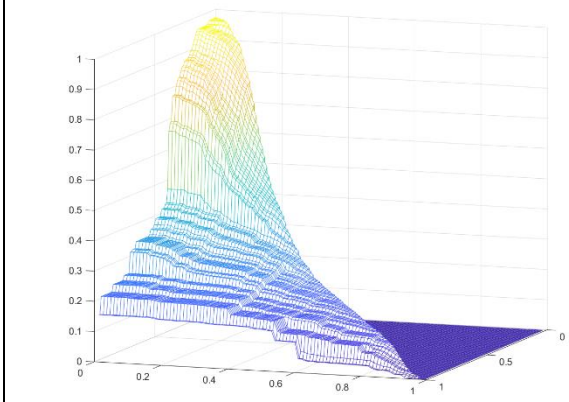
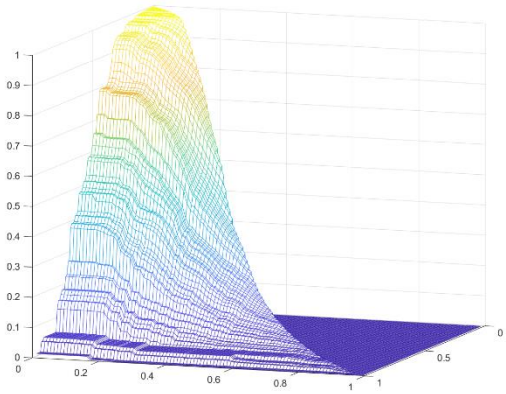
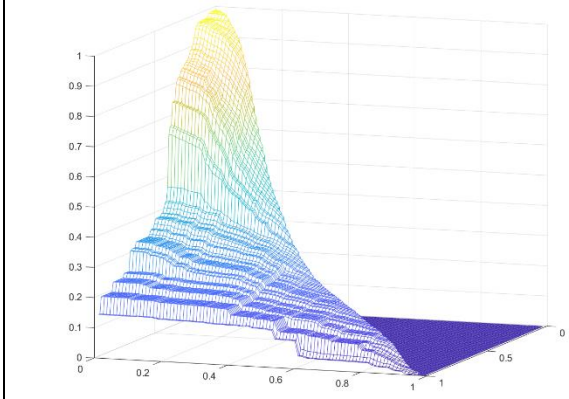
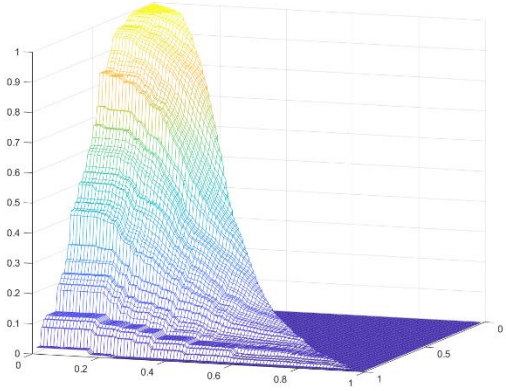
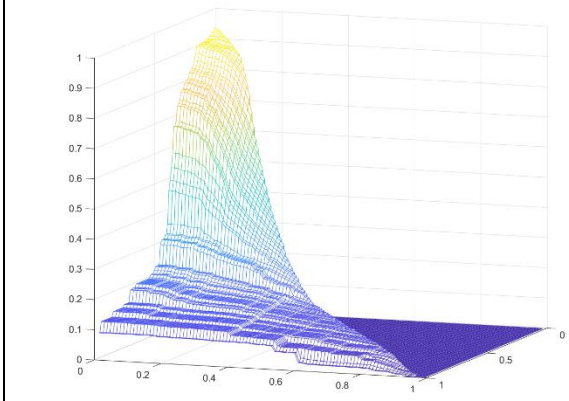
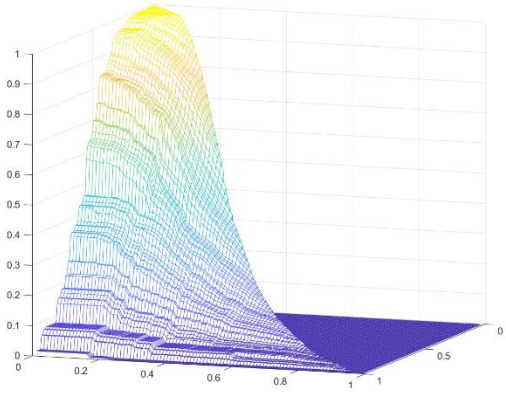
Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

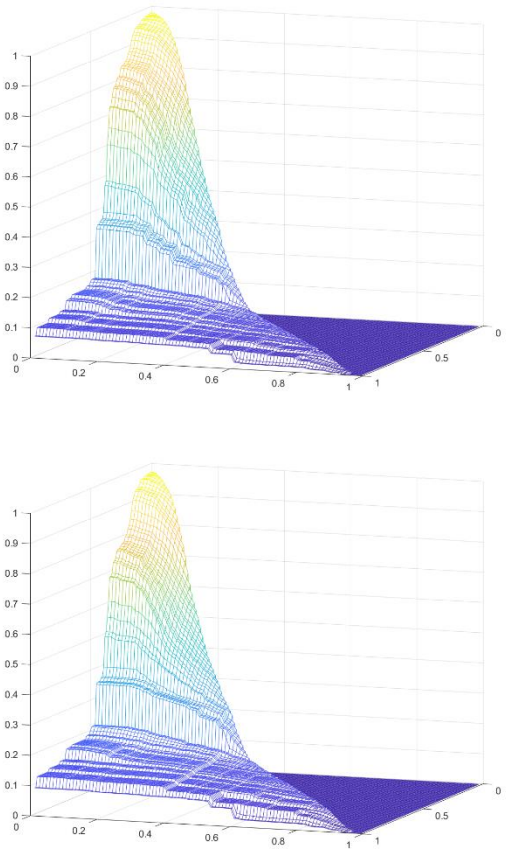
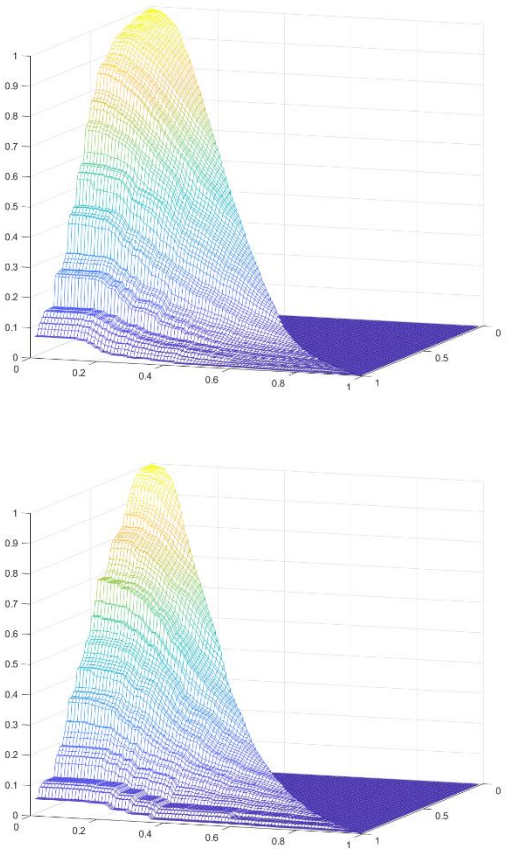
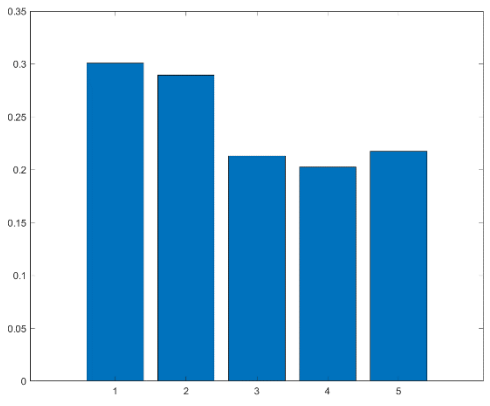
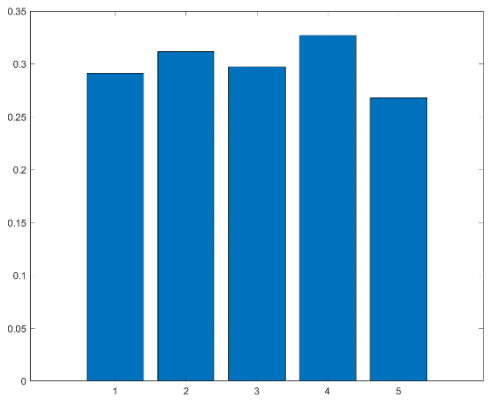
Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

		
Mean Precision over thresholds (g)		
Mean Precision over ground truths (t1, t2)		

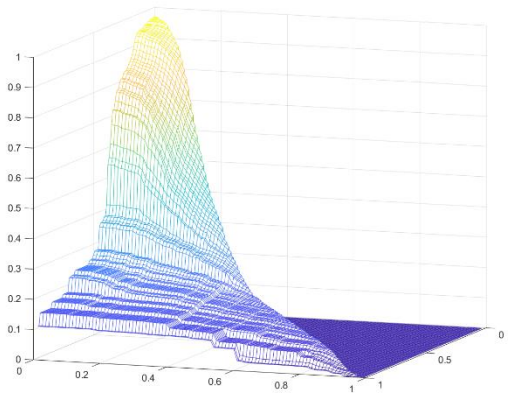
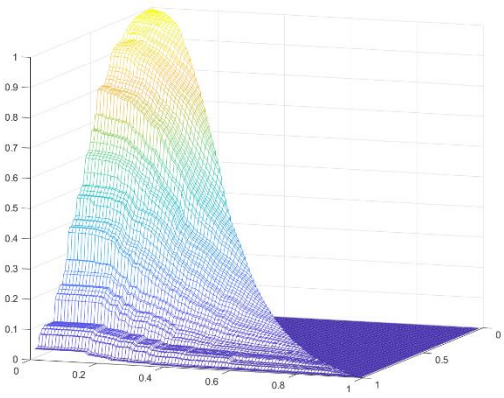
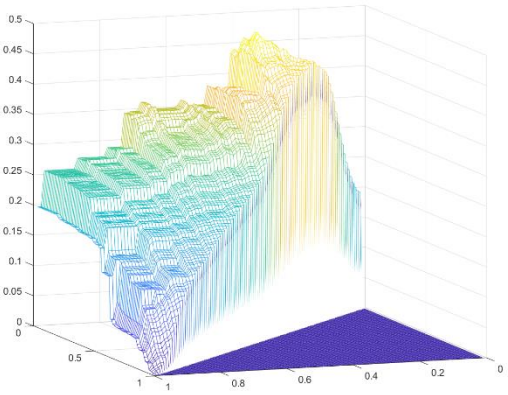
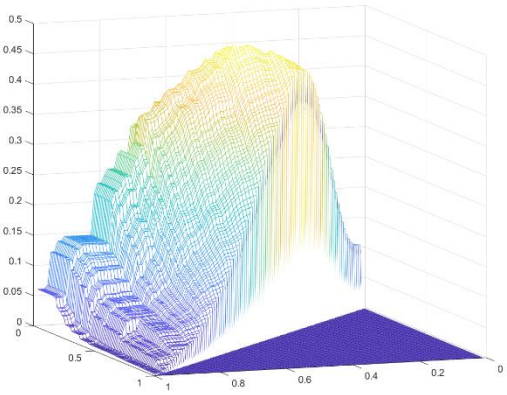
Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Recall (g, t1, t2)		
		
		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

		
Mean Recall over thresholds (g)		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Mean Recall over ground truths (t1, t2)		
F (t1, t2)		
Mean F	0.3728	0.3064
Best F	0.4765	0.4627

Structured Edge Detector

Edges

Zongjian Li

6503-3789-43

zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

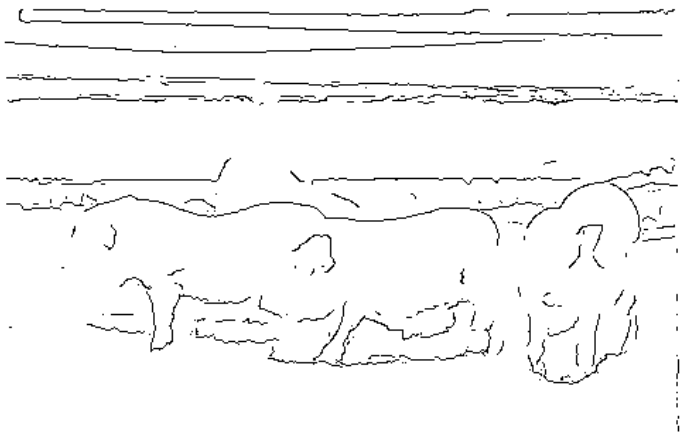
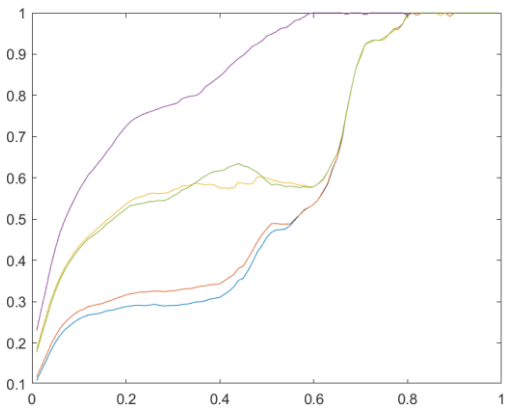
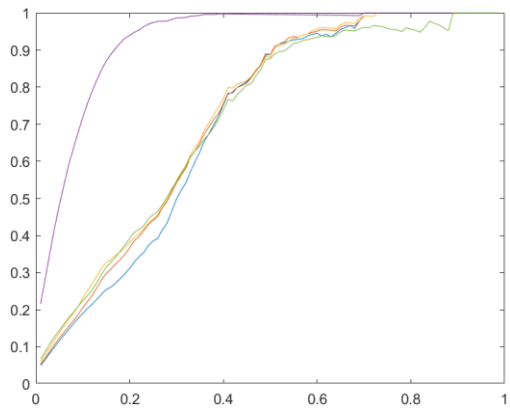


Fig 1-27. Pig image edges (Structured, non-maximum suppression & multiscale enabled, T=0.19, %=3.45, P=0.4700, R=0.7240, F=0.5703)	Fig 1-28. Tiger image edges (Structured, non-maximum suppression & multiscale enabled, T=0.15, %=2.34, P= 0.4130, R= 0.7251, F=0.5263)

Performance Evaluation (here the threshold is the percentage of maxima gradient magnitude)

	Pig	Tiger
Precision (g, t)		

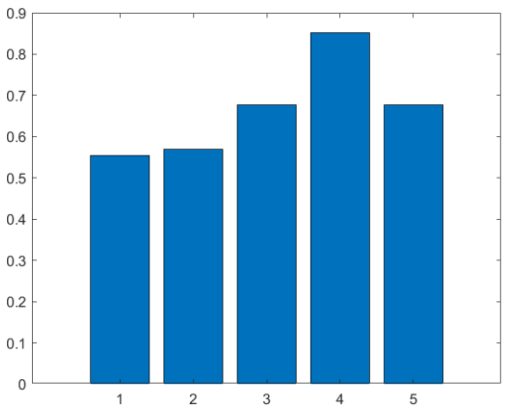
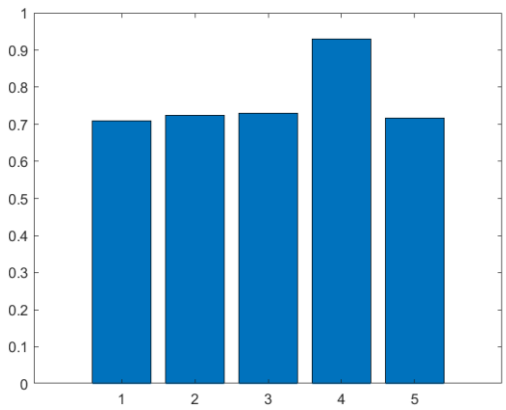
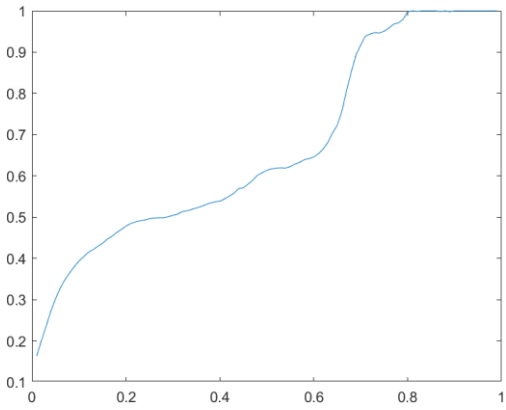
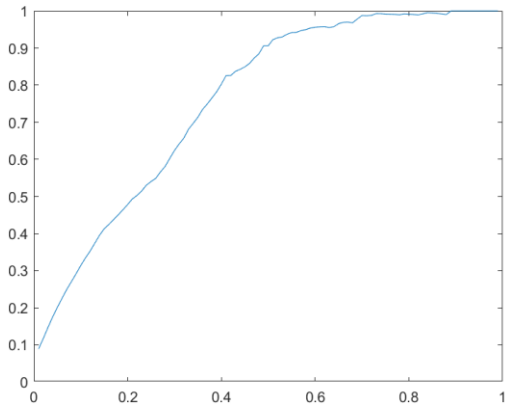
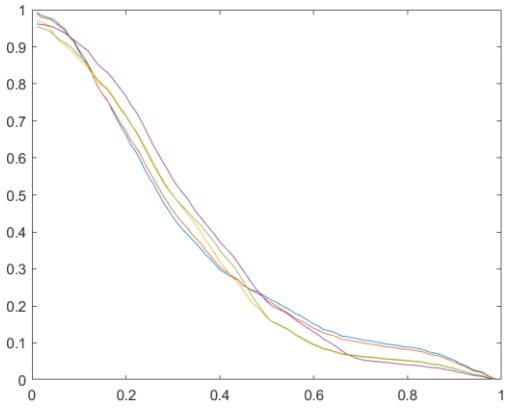
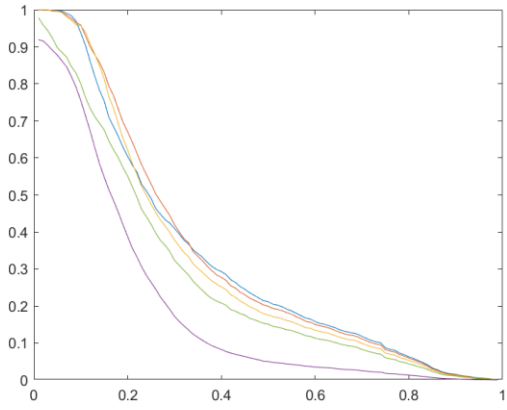
Zongjian Li

6503-3789-43

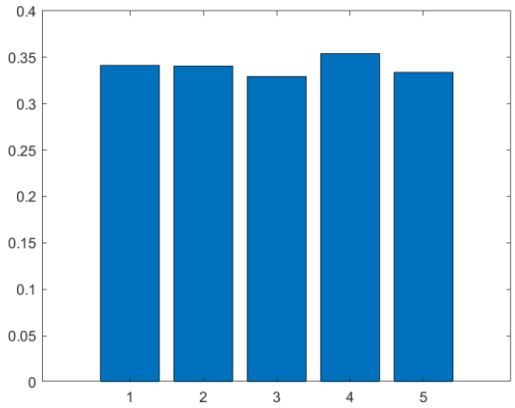
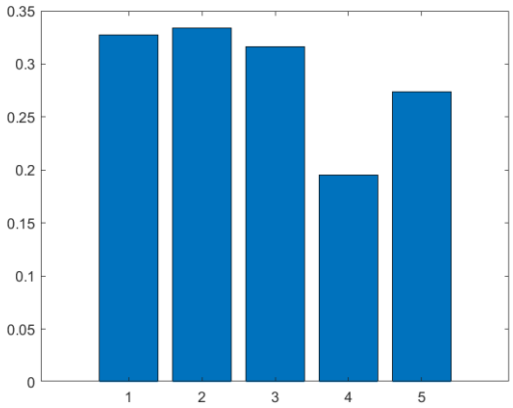
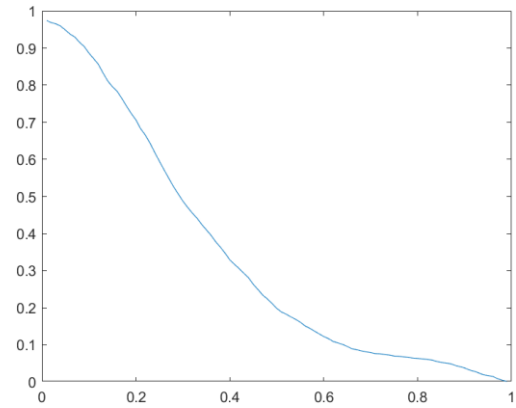
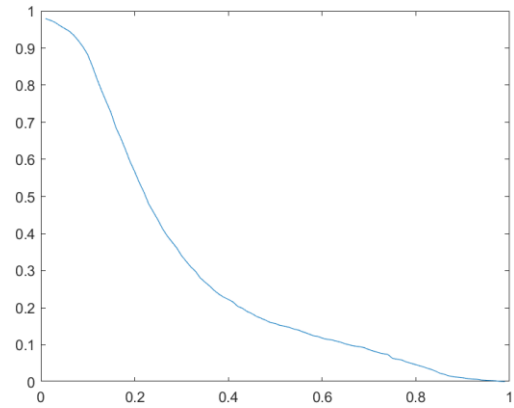
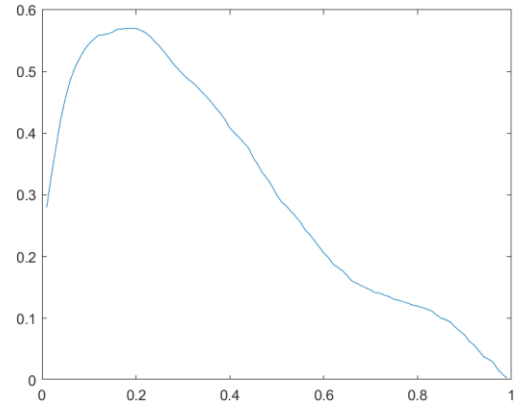
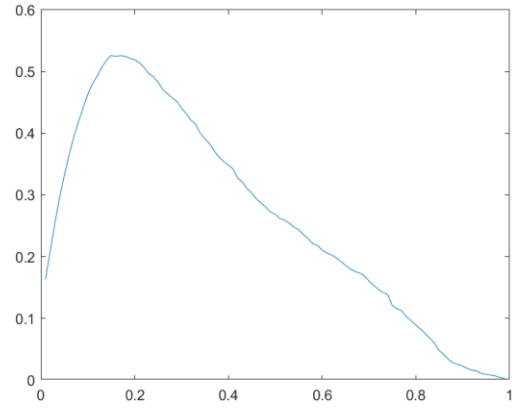
zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

Mean Precision over thresholds (g)		
Mean Precision over ground truths (t)		
Recall (g, t)		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Mean Recall over thresholds (g)		
Mean Recall over ground truths (t)		
F (t)		
Mean F	0.4498	0.4194
Best F	0.5703	0.5263

Zongjian Li

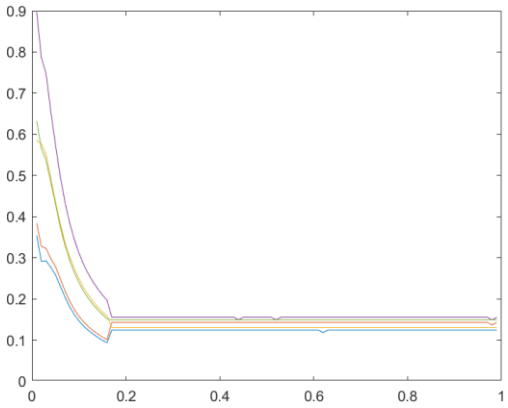
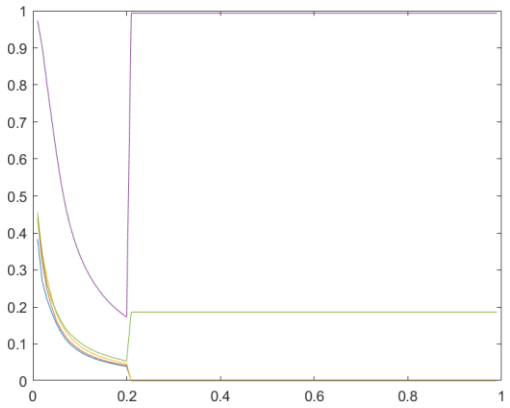
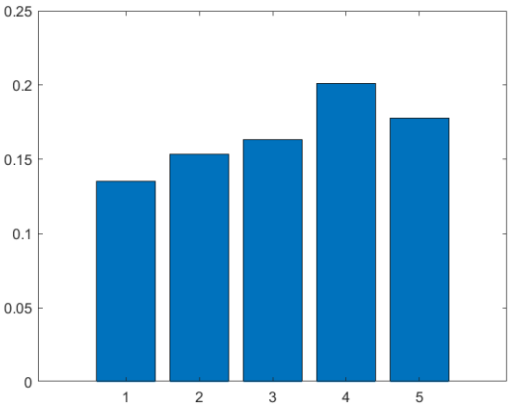
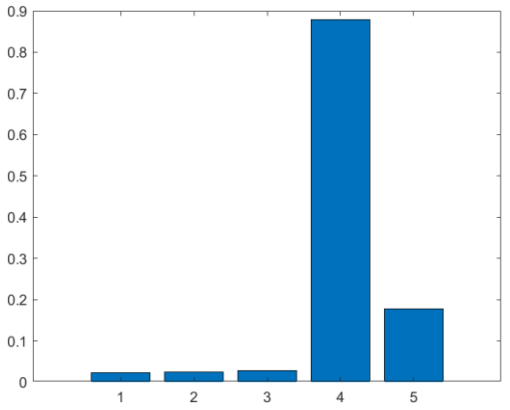
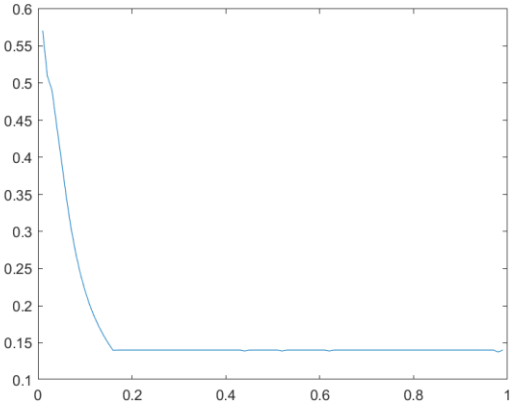
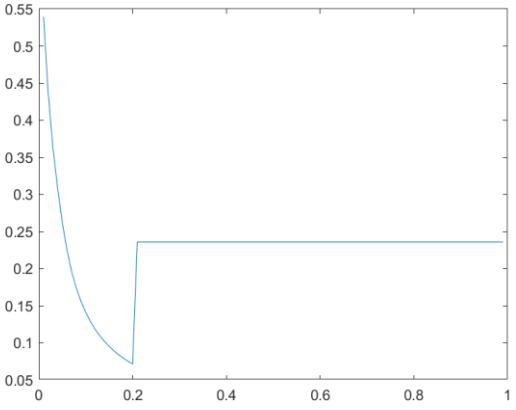
6503-3789-43

zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

Performance Evaluation (here the threshold is the percentage pixels become edge points, not a good idea but requested)

	Pig	Tiger
Precision (g, t)		
Mean Precision over thresholds (g)		
Mean Precision over ground truths (t)		

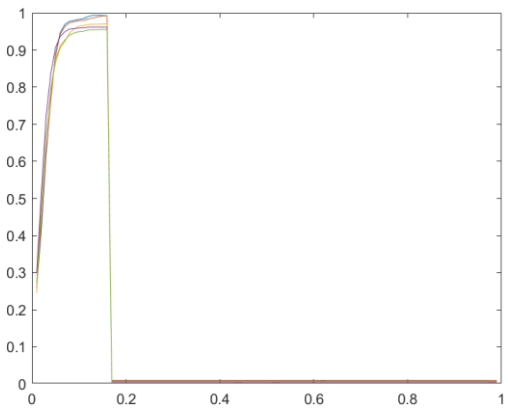
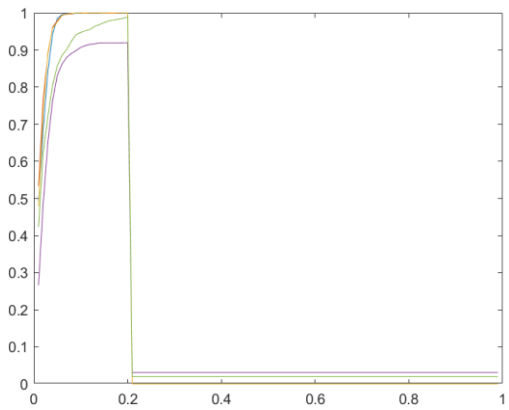
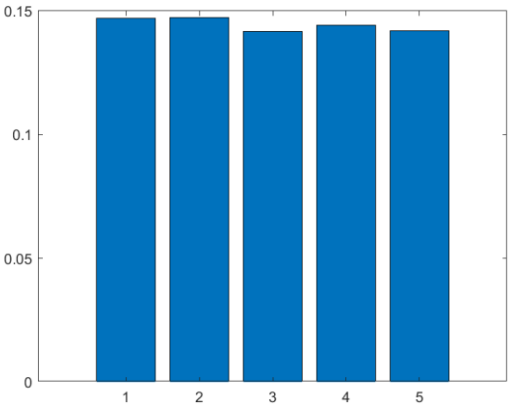
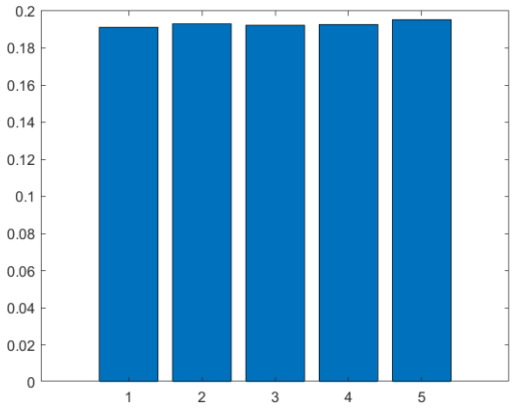
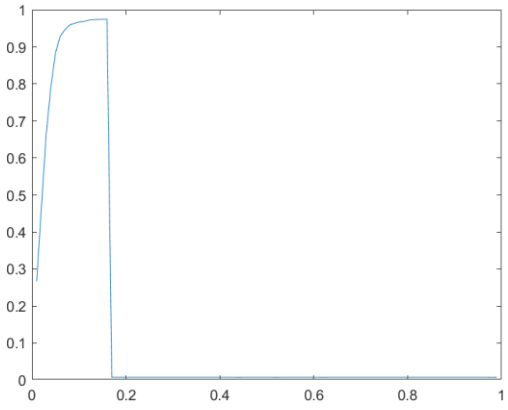
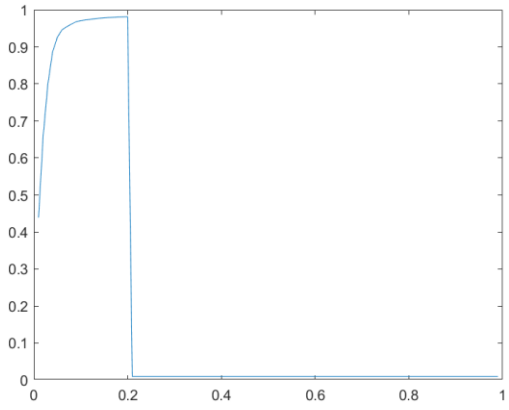
Zongjian Li

6503-3789-43

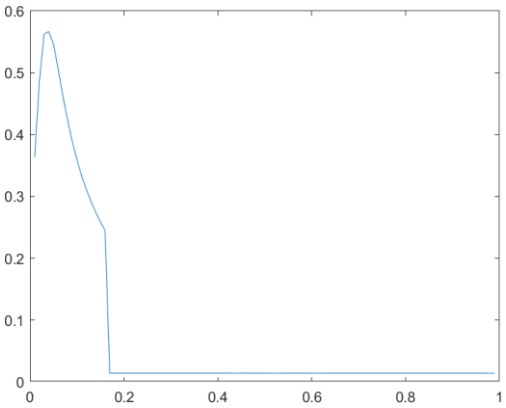
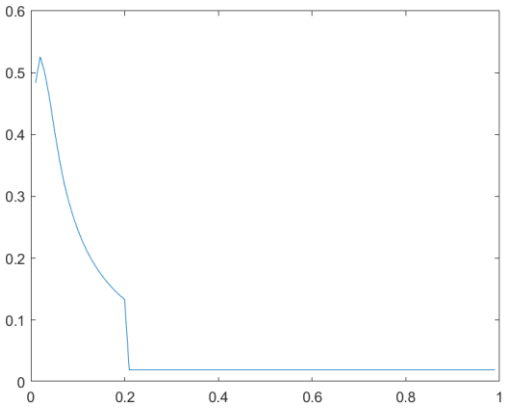
zongjian@usc.edu

EE 569 Homework #2

February 12th, 2019

Recall (g, t)		
Mean Recall over thresholds (g)		
Mean Recall over ground truths (t)		

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

F (t)		
Mean F	0.1546	0.2083
Best F	0.5669	0.5260

IV. Discussion

[Question a-1] Normalize the x-gradient and the y-gradient values to 0-255 and show the results.

Results are shown above. Sobel operators are used.

[Question a-2] Thune the thresholds (in terms of percentage) to obtain your best edge map.

Results are shown above. My best threshold for pig image is 29% of the maximum gradient value and 4.32% pixels become edge pixels. My best threshold for tiger image is 35% of the maximum gradient value and 4.62% pixels become edge pixels.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

[Question b-1] Explain Non-maximum suppression in Canny edge detector in your own words.

Only Sobel edge detector in my implementation does not have non-maximum suppression procedure. And we can see the edges are thicker than edges in other methods' results. Non-maximum suppression thins the edge by deleting edges points (set the gradient value to 0) with local non-maxima gradient value, only local maximum edges pixels are remained. When the program traversing pixels in gradient magnitude map, it checks the current pixel's neighbors. If one of its neighbors has larger gradient magnitude, that means current pixel is not the local maximum one, and this edge pixel should be suppressed by setting its value to 0. The neighbors here are pixels in opposite sides among the gradient direction of current pixel.

[Question b-2] How are high and low threshold values used in Canny edge detector?

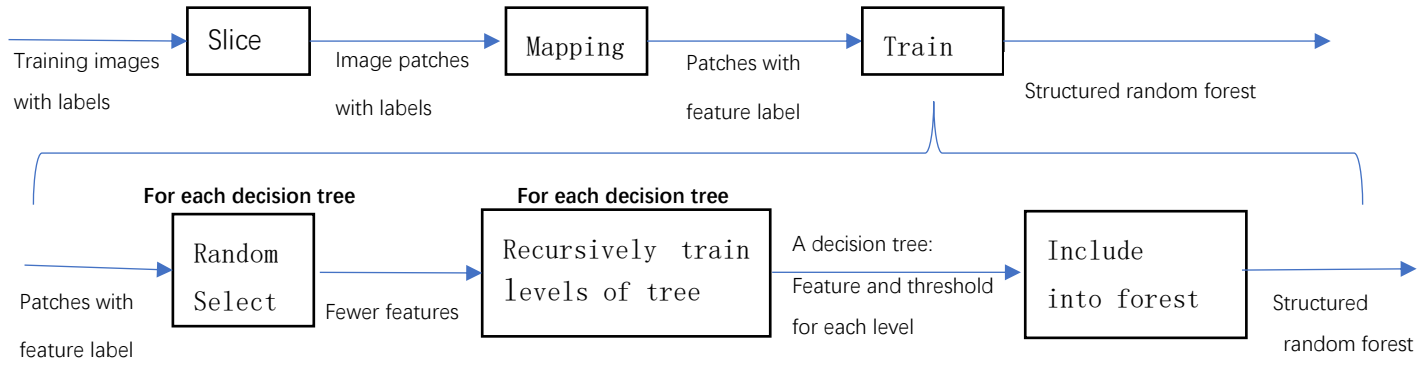
Double threshold uses two thresholds rather than only one threshold, there are one higher threshold and a lower threshold. Although, non-maximum suppression removed some non-edge pixels, there are still some unwanted pixels with not too low and not too high gradient values caused by noise or color changing, and it is hard to distinguish between weak edge pixels. To tell whether a pixel with relatively high gradient value is a weak edge pixel that is needed to keep or not, we need to use the information from its neighbor pixels. Only if one of its neighbors is a strong edge pixel, the current pixel will become an edge pixel in the output. In the implementation, when the program traversing the edge gradient map, if the current pixel's value higher than the higher threshold, it becomes an edge point. And if the value is lower than the lower threshold, it will not become an edge point. If it is lower than the higher threshold and higher than the lower threshold, further examinations are required. Current pixel becomes an edge point in the output only if at least one of its neighbors has gradient magnitude higher than the higher threshold.

[Question c-1] Summarize Structured Edge detection algorithm with a flow chart and explain it in your own words.

There are two steps in Structured Edge detector.

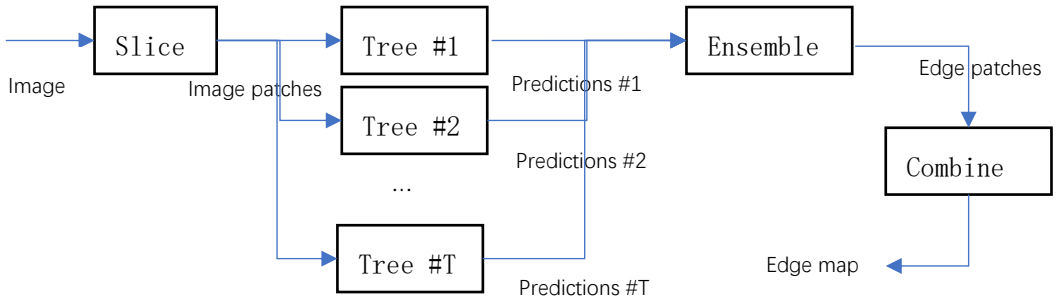
First, we need to train a structured random forest.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019



At first, we have training data with labels, we need to slice images into small 32*32 image patches with 16*16 segmentation mask information. Here we have all the features in feature space Y. The Mapping step has two sub-steps. Using a mapping function can reshape the features into vectors belonging result space Z. We need to calculate the distance between features, so after vectorizing we can calculate the distance (e.g. Euclidean distance). Distances are used for Training, if the distance is large that means two features has very large differences. An approximate distance is enough. While there are too many features that we cannot handle them, so we need to reduce the number of features. We can sample the features by certain stingray and get fewer features that we can handle. Now each patch has the labeled data of selected features. At last, using patches to train a structured random forest. There are many details in this step. We need to train trees by feeding all the training data while contains different labels of random selected features. After the trees are trained, It becomes a structured random forest. (Training details are discussed in the answer of next question)

Then we can use the trained structured random forest to detect edges.



When we get an image, we need to slice it into pieces with size same as that in training step. Then we can feed sliced images to the structured random forest. Each of the decision tree in the forest will give a prediction for each sliced image. Then an ensemble model is used to combine predictions from all decision trees for each sliced image. One possible ensemble model is to sum all the predictions respect to patterns and get the pattern with highest total prediction value. So far, we have slices of edge map. Then we combine slices of edge map to the edge detect result.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

[Question c-2] Explain the process of decision tree construction and the principle of the Random Forest classifier.

Decision tree is a kind of machine learning algorithm, which can be used as a classifier. In structured edge detector, we need to classify the pattern of given image slices.

We can start to construct a decision tree after the training data is preprocessed. The training data here is sliced images with selected patterns' vector as their labels, which is discussed above. Then we need to decide the number of trees T in the forest. To get better results, it is better to train $2\mathrm{T}$ trees and select T trees that can produce best results at last. Since we need a structured random forest, we cannot feed all features into every decision tree, since that will cause overfitting. So, for each tree, we randomly select a sub set of selected features, which has fewer features. There are some techniques to select patterns to make trees have enough diversity to become de-correlated. For each tree, we train branches recursively. For each level, we need to find a pattern and a threshold. The pattern and the threshold are for splitting training data into two parts with largest difference between two parts. And these two parts will be the left and right branch. We can calculate differences by calculating distance between vectors (mentioned above). There are many algorithms to tell how to select a pattern for each level. Training stops if the depth reaches the maximum value or all the leaf nodes are not separable. So far, a decision tree is constructed.

The random forest classifier contains multiple decision trees. As mentioned above, these trees should be de-correlated. Trees can be binary classifiers or multiclass classifiers. If we give same input for each tree, each tree will produce a predication (binary-class or multiclass). While we need only one combined prediction. To combine the predictions, an ensemble model should be defined, which is a task dependent hard work. Here we can simply pick the pattern with highest average precision. The structured random forest "use the default ensemble model during training but utilize a custom approach for merging outputs over multiple overlapping image patches" [2].

[Question c-3] Apply … State the chosen parameters clearly and justify your selection. Compare and comment on the visual results of the Canny detector and the Structured Edge detector.

The model is pre-trained, and I cannot modify parameters for training. I can only change parameters for edge detection. My chosen parameters are

Zongjian Li
6503-3789-43
zongjian@usc.edu


```
model = edgesTrain(opts); % load existing model, don't train it!
model.opts.multiscale = true; % Changed! The default option is false
model.opts.sharpen = 2;
model.opts.nTreesEval = 4;
model.opts.nThreads = 4;
model.opts.nms = true; % Changed! The default option is false
```

One change from the default parameter is that I enabled non-maximum suppression. The default output of this detector is the edge probability map. Enabling non-maximum suppression does not make sense because this procedure will destroy some potential edges' information by setting values to 0. However, I only need a binary edge map, so enabling non-maximum suppression can always give me clear edges.

Another parameter I changed is the multiscale switch, I enabled it. Since it increases the accuracy by down scale and up scale the input images, which is good for a machine learning classifier.

Number of trees "should be $1 \leq T \leq 4$ in practice" [2], so 4 is the best to avoid overfitting. Other parameters are just some kinds of optimizations for speed. The default values are recommended and give good results, changing them does not cause fundamental changings in results.

After utilizing non-maximum suppression, results from both Structured Edge and Canny have thin edges. Structured Edge detector gives clearer body boundaries of pigs or the tiger than Canny. However, they both cannot find all four pigs very well, especially the black pig which is similar with the background color. Best F measures for Structured Edge is also higher. Due to double thresholding, Canny's edge are connected rather than broken to small parts. Canny is more sensitive to stripes on tiger's body. While, 4 of 5 ground truth do not consider stripes as edges. Structured Edge also detected stripes, but less. And, Canny's edge contains more small parts that is not edges in any of ground truths.

[Question d-1] Calculate … Please use a table to show the precision and recall for each ground truth, their means and the final F measure. Comment on the performance of different edge detectors (i.e. their pros and cons.)

The results are shown above, since there are too many different measurements to be shown, one table is not enough.

	Sobel	Canny	Structured Edge
Pros	• Easy to implement • Fast	• Thin edges • Connected edges	• Best results (clean, complete, etc) • Thin edges

cons	• Worst results. • Thick edges • Broken edges • Difficult to select a threshold	• A little bit difficult to implement • Two thresholds should be tuned	• Need training • Rely on the training data set • Difficult to implement • Many parameters to select

Three detectors all can detect edges. While Sobel edge detector cannot give good body boundaries of pigs or the tiger in the input image. All detectors are failed to detect the black pig and one pig's head. Stripes on the tiger's body are detected by all the detectors, while whether this part is edge is confusing.

[Question d-2] The F measure is image dependent. Which image is easier to get a high F measure – Tiger or Pig? Please provide an intuitive explanation to your answer.

Pig image has higher F measures than tiger image.

I think there are two reasons to make the tiger image get low F measures. The first reason is that there are large stripes on tiger's body and will always caught by detectors, while 4 of 5 ground truths do not seem those as edges, that will make the F measures lower. However, there is no stripes on pig's bodies. The second reason is the background, there are many long grasses in the whole background of tiger image which will cause edges of grasses be detected, and the ground truths do not do like that.

[Question d-3] Discuss the rationale behind the F measure definition. Is it possible to get a high F measure if precision is significantly higher than recall, or vice versa? If the sum of precision and recall is a constant, show that the F measure reaches the maximum when precision is equal to recall.

F measure is a combination of precision and recall $\frac{1}{F}=\frac{1}{2}(\frac{1}{P}+\frac{1}{R})$. Precision and recall are both smaller or equal to 1, so F is smaller or equal to 1. Here $\frac{1}{2}$ means we give equal weights to precision and recall. We do can give different weights for precision and recall, but this article will not cover that. There are four kinds of results, true/false positive/negative. Precision and recall both take only two of them into consideration, which gives biased results. So, we need F measure to

give us overall consideration.

It is impossible to get a high F measure if precision is significantly higher than recall, or vice versa. This is confirmed by my results above. By increasing the threshold, the precision becomes higher and higher while the recall becomes lower and lower, and the F measure goes up then goes down. The highest F measure is obtained by only precision and recall with intermediate values. And we can analysis it by using the equation. If one of precision and recall is the highest value 1, the F measure will be a function w.r.t the other measurement (P or R) with monotone decreasing values by increasing the other measurement's value. So, it is impossible to get high F measure in the question's scenarios, only both high precision and recall will lead to a high F measure.

If the sum of precision and recall is a constant, we can replace one measurement by using the constant value and the other measurement by $\mathrm{c}-\mathrm{m}$. where c is the constant and m is one of the precision and recall. Then the equation for F measurement will become $\mathrm{F}=2\frac{m(c-m)}{m+(c-m)}=\frac{2}{c}m(c-m)$. It reaches the highest value at $\mathrm{m}=\frac{c}{2}$, I get this by utilizing equation's derivative $\mathrm{F}'=2-\frac{4}{c}m=0$. That means in question's scenario, only when precision equals to recall, F measure reaches the maximum.

Computational Complexity Analysis

These algorithms have basically the same time complexity $\mathrm{o}(mn)$ if the input image size m and n is large enough (no optimization). While, the Structured Edge Detector need to be trained at first, that is a time-consuming procedure.

Two Kind of Thresholds

As mentioned above, 2 kinds of percentage thresholds are used in this article. The first one is the percentage of the maxima gradient magnitude, and the second one is the percentage of the pixels being edge pixels. Since they can be converted to each other easily, either of them can be used.

However, I prefer the first definition. As we can see in my tables, if I separate the threshold value range by using equidistance thresholds, 2nd definition gives large portion of non-sense result, since after a certain threshold all the pixels with 0 gradient magnitude are selected as edge points, that means all the pixels are edge pixels after a certain threshold. That is why there are horizontal lines at the right part of each figure, this part is useless although I still draw them on the figures. This will compress the space for meaningful thresholds. The granularity for meaningful thresholds becomes larger and cannot get good results. In this article's discussion, only results from first kind of definition are used.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Problem 2: Digital Half-toning

I. Abstract and Motivation

Half-toning is a technique for printing images using only few kinds of inks. For a black-white printer, each pixel is whether black or white, no other gray scale inks. For a color printer, the number of kinds of inks are only four, cyan, magenta, yellow and black, each pixel color can be one of the mix of these ink colors, basically 8 colors. To show a gray or color image with these limited colors, half-toning is used. In a small region, it will use more color dots to make that region seems darker and fewer color dots to make that region seems brighter. Since the density of these solid color is high due to the development of print technologies, people can feel like the image is actually printed in multiple colors and good quality. While, the results from these methods are different. With some methods, we can see some defects in the results such as introduce unwanted repeated pattern into the image. To get good half-toned images, we need good methods. In this article, I will discuss several half-toning methods, including methods for gray scale images then color images.

II. Approach and Procedures

Depth of pixel value

In this article, each channel takes 8-bit space, using unsigned char type, that means the possible values are integers from 0 to 255. However, during the processing, some detailed values should be kept to get precision results. That means the intermediate images will be stored using float type. The minimum and maximum is still 0 and 255, but values can be real numbers now. I will not convert them to 0 to 1 like some people usually do, because it is an inessential procedure.

After the image processing procedures, to get a standard output image, those real numbers should be converted to integers. There are 3 basic methods to do this, round, ceiling and floor. Here I use round, since it has a maximum error less than 0.5. After that, I also check whether the integer values are out of range. A clip function needs to be used at here.

Fixed Thresholding

Fixed thresholding is the simplest method, and its results can not be used in any reality scenario. Here I just use it to as a sign post in comparison part in this article.

This method can be described as following equation

Zongjian Li
6503-3789-43
zongjian@usc.edu

$$G(i,j)=\begin{cases}0, & 0\le F(i,j)\le T\\ 255, & T<F(i,j)\le 255\end{cases} \quad (2\text{-}1)$$

Where F is the input gray scale image and $F(i,j)$ is one of the pixels in F, G is the output image, T is a threshold. Normally T should be set to 127, which is the nearest integer of half of 255 that can separate the gray scale values half to half.

Random thresholding

In this method, the only difference between Fixed Thresholding is the threshold value. For each pixel, a random threshold will be generated and used as the threshold.

$$G(i,j)=\begin{cases}0, & 0\le F(i,j)\le \mathrm{rand}(i,j)\\ 255, & \mathrm{rand}(i,j)<F(i,j)\le 255\end{cases} \quad (2\text{-}1)$$

The random thresholds range from 0 to 255. The distribution of random value can be uniform distributed. I use C standard random function, so in my implementation, the thresholds are uniformly distributed. For the comparing purpose, a random seed can be set in my implementation and the default random seed is 0. Same random seeds always give us the same output.

Dithering Matrix

A Dithering Matrix can be seen as multiple thresholds. Do the dithering through an image without overlaps, then the number of 1 in corresponding dithering matrix area in the output can be seen as the darkness level of that area. As a result, a dithering matrix should be designed to show the difference of areas with different gray scale values. The Bayer index matrices are good ones, they are defined by

$$I_2(i,j)=\begin{bmatrix}1 & 2\\ 3 & 0\end{bmatrix}$$

$$I_{2n}(i,j)=\begin{bmatrix}4I_n(i,j)+1 & 4I_n(i,j)+2\\ 4I_n(i,j)+3 & 4I_n(i,j)\end{bmatrix} \quad (2\text{-}2)$$

They are matrices with different sizes. Using dithering matrices with different sizes can result to different outputs at last. Pick one the Is as I and compute the dithering matrix
T with thresholds in it using following equation

$$T(i,j)=\frac{I(i,j)+0.5}{N^2}\times 255 \quad (2\text{-}3)$$

Where N is the width or height of I, and 255 is the maximum quantized value.

Then apply the T to the input image F to get output image G with equation.

$$G(i,j) = \begin{cases} 0, & 0 \le F(i,j) \le T(i \bmod N, j \bmod N) \\ 255, & T(i \bmod N, j \bmod N) < F(i,j) \le 255 \end{cases} \quad (2\text{-}4)$$

In my implementation, all the intermedia values are stored in float point data types.

Error Diffusion

Error diffusion is another half-toning method. It passes the error of half-toned pixel to future pixels to make up the error. For choosing which future pixels and how much should these pixels should make up, a error diffusion matrix is used. There are three kinds of popular error diffusion matrices

$$\text{Floyd-Steinberg's: } \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

$$\text{Jarvis, Judice and Ninke's: } \frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

$$\text{Stucki's: } \frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix} \quad (2\text{-}5)$$

Using larger error diffusion matrices can pass the error to further pixels. These given matrices are used in left-to-right up-to-down scanning order. Other scanning order can be used, while these matrices should be flip horizontally or vertically. Raster and serpentine scanning order are two easy scanning orders that I implemented.

For the errors at the boundary, here I just discard them so that operations will not exceed the image area.

Separable Error Diffusion

Methods above are basic ones for gray scale image. We can extend the usage of them to color images by doing half-toning in different color channels separately. Separable Error Diffusion is one choice.

In my implementation, I do the separable error diffusion in CMY color space, I convert the input from RGB color space to

Zongjian Li
6503-3789-43
zongjian@usc.edu

CMY color space before applying error diffusion and convert back at last. Converting are based on following equations

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{2-6}$$

MBVQ-based Error Diffusion

MBVQ-based Error Diffusion [3] is an advanced color image half-toning method, it uses 6 color sets containing 4 colors out of 8 with smallest brightness differences to minimalize the local brightness fluctuation. Since human eyes are sensitive to brightness changes, this method can give more comfortable results.

Also, this method is based on Error Diffusion method, other parts except output color selecting are same as those in Error Diffusion method.

III. Experimental Results

Gray Scale Input Image

Fig 2-1. Input gray scale image *bridge.raw* (600 * 400)

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Fixed and Random Thresholding

Fig 2-2. Fixed thresholding (T = 127.5)

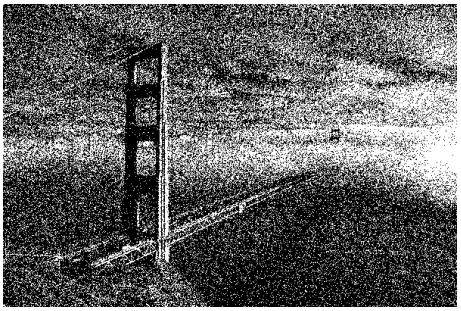


Fig 2-3. Random thresholding (uniform distribution)

Dithering Matrix

Fig 2-4. Bayer dithering matrix (N=2)

Fig 2-5. Bayer dithering matrix (N=8)

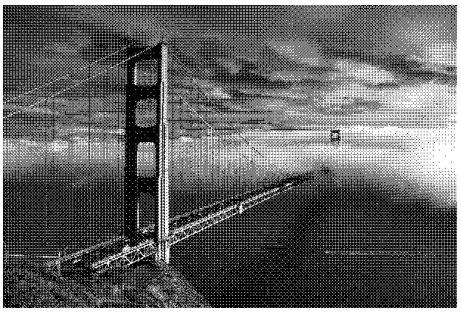


Fig 2-6. Bayer dithering matrix (N=32)

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Error Diffusion

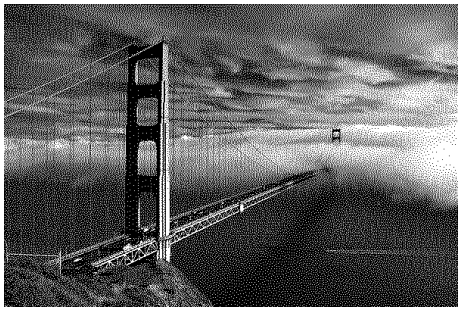


Fig 2-7. Error diffusion (Floyd-Steinberg, Serpentine)	Fig 2-8. Error diffusion (Jarvis Judice Ninke, Serpentine)	Fig 2-9. Error diffusion (Stucki, Serpentine)
		
Fig 2-10. Error diffusion (Floyd-Steinberg, Raster)	Fig 2-11. Error diffusion (Jarvis Judice Ninke, Raster)	Fig 2-12. Error diffusion (Stucki, Raster)

Color Input Image

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

Fig 2-13. Input color image *bird.raw* (500*375)

Separable Error Diffusion

Fig 2-14. Separable error diffusion (Floyd-Steinberg, Serpentine)

Fig 2-15. Separable error diffusion (Jarvis Judice Ninke, Serpentine)

Fig 2-16. Separable error diffusion (Stucki, Serpentine)

Fig 2-17. Separable error diffusion (Floyd-Steinberg, Raster)

Fig 2-18. Separable error diffusion (Jarvis Judice Ninke, Raster)

Fig 2-19. Separable error diffusion (Stucki, Raster)

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

MBVQ-based Error Diffusion

		
Fig 2-20. MBVQ-based error diffusion (Floyd-Steinberg, Serpentine)	Fig 2-21. MBVQ-based error diffusion (Jarvis Judice Ninke, Serpentine)	Fig 2-22. MBVQ-based error diffusion (Stucki, Serpentine)
		
Fig 2-23. MBVQ-based error diffusion (Floyd-Steinberg, Raster)	Fig 2-24. MBVQ-based error diffusion (Jarvis Judice Ninke, Raster)	Fig 2-25. MBVQ-based error diffusion (Stucki, Raster)

IV. Discussion

[Question a-2] Please create I_2, I_8, I_{32} thresholding matrices and apply them to halftone Fig. 4. Compare your results.

Results are shown above.

All the results have some kinds of patterns, we can see horizontal and vertical lines in the outputs. They all have cross-

hatch patterns, regions with different gray values have difference patterns. The clouds in the input image have smooth gray value change, while we can only see color blocks with no smooth variation. Thresholding with matrix size 2 has the worst result since gray values in its result changes abruptly and details are lost.

Results from size 8 and 32 seems similar. However, larger size of error diffusion matrix may distinguish more different gray values since it has more different thresholds, that will help keeping more details. While the larger matrix size decreases its ability to do that, so the results in size 8 and 32 are similar for the bridge image.

[Question b-1] Show the outputs of the following three variations and discuss these obtained results. Compare these results with dithering matrix. Which method do you prefer? Why?

Results from error diffusion methods looks better than those from dithering, since the gray values changes smoothly. Although granular sensation in error diffusion results are unnatural, they do not introduce any pattern. Granular sensation is caused by white (black) dots in larger areas of black (white) background. Computational complexity of error diffusion methods are higher than dithering since to process each pixel 2 more loops for the error diffusion matrix are needed.

Granular sensation in two error diffusion methods with error diffusion matrix size equals to 5 (JJN, Stucki) are stronger, that means the density of white (black) dots are lower. And Floyd-Steinberg's method has high density of white (black) dots, which make the results seems has more salt and pepper noise pixels. That is because, for larger error diffusion matrices, the error values are passed to further pixels, so near pixels make up lower weight of error and the lower make up error value are small that will not flip the white or black output pixel results. Also, a larger error diffusion matrix will cause the compute time grows up.

Results between JJN and Stucki are quite similar. Stucki gives higher weight of error for near pixels than farther ones than JJN does. The effect is similar to the difference between Uniform filter and Gaussian filter. That make results from Stucki's keep more details than JJN.

Two scanning orders, serpentine and raster, are performed, and the results seems have no difference. However, raster method will only pass errors to right or bottom pixels, which make the result unbalanced. Serpentine method is better since it passes errors to left, right or bottom pixels, and will not increase the execution time.

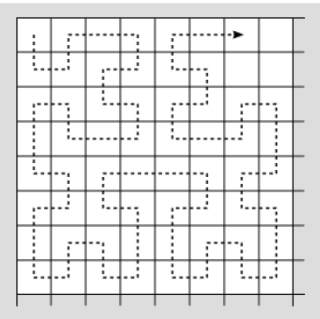
I prefer error diffusion methods with Stucki error diffusion matrix and serpentine scanning order, because it is the most advanced method based on my analysis above in my implementation.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

[Question b-1] Describe your own idea to get better results. There is no need to implement it if you do not have time. However, please explain why your proposed method will lead to better results.

Based on the discussion above, I can change the scanning order, which passes errors to all four directions. Serpentine and raster do not pass error to top pixels since they scan from top to bottom and all top pixels are already processed that make it impossible to pass the error. That makes the error make up values' distribution unbalanced. One possible better scanning order are shown below

- **Hilbert curve**



[Question a-1] Compare your results obtained by these algorithms in your report.

Here I will compare the results from method for gray scale images.

Fixed thresholding is the easiest one to implement, while we can only see a bridge in the result and cannot tell what the different parts is in the background are. Random thresholding does not introduce any pattern, but the noise level is very high that we can hardly see the details. Dithering has fair enough visual quality and fast, while it introduces patterns and details are lost. Images with only large areas with small color change is suitable for this method. Error diffusion methods have good visual quality, while they need more time to process. Detailed differences among error diffusion implementations are discussed in another question.

Zongjian Li
6503-3789-43
zongjian@usc.edu
EE 569 Homework #2
February 12th, 2019

[Question c-1] Please show and discuss the result of the half-toned color bird image. What is the main shortcoming of this approach?

Results are shown above. It ignored relationships between channels. It does not consider the brightness variation between nearby pixels. Channels are processed completely separately. Brightness in small areas may change rapidly which cause a worse visual quality, since human are sensitive to brightness changes.

[Question c-2] Describe the key ideas on which the MBVQ-based Error diffusion method is established and give reasons why this method can overcome the shortcoming of the separable error diffusion method.

MBVQ-based Error Diffusion takes all channels of pixels into consideration, which make it overcome the shortcomings in Separable Error Diffusion. Again, since human eyes are sensitive to brightness changes, MBVQ-based method designed 6 sets of colors which have 4 solid colors (from total 8 colors that a printer can produce) with minimum brightness change, and by using these color sets, brightness variation in small regions are reduced. Visual quality is increased. We can imagine that in a small region in the input image, the colors are similar, so they will use a same color set to produce output, as a result, in this region, the brightness variations are minimized. The computing complexity of MBVQ-based method is the same as Separable method. In total, MBVQ-based method is better.

[Question c-3] Implement … Compare the output with that obtained by the separable error diffusion method.

The results are shown above. A theoretical analysis is given in the answer of last question, now let me show the actual results. Compare the region of bird's head. After we zoom in, we can find lots of white pixels adjoin with black pixels in Separable method results. While MBVQ-based method has less this kind of effect. White and black is the pair of color has largest brightness change. Fewer of white-black adjoin regions are better. In other regions in bird image, we can also find this kind of difference. Thus, the theoretical analysis is confirmed.

Computing Complexity Analysis

Detailed analysis is given in each algorithm's discussion. In conclusion, all algorithms have a time complexity of $o(mn)$,

where the m and n denotes the input image's height and width (gray scale image). But the error diffusion algorithms have larger constant term which make computing consuming more time. And larger error diffusion matrix (JJN, Stucki) introduce larger constant term than Floyd-Steinberg's.

References

[1] Canny, John. "A computational approach to edge detection." IEEE Transactions on pattern analysis and machine intelligence 6 (1986): 679-698.

[2] Dollár, Piotr, and C. Lawrence Zitnick. "Structured forests for fast edge detection." Proceedings of the IEEE International Conference on Computer Vision. 2013.

[3] Shaked, Doron, et al. "Color diffusion: error diffusion for color halftones." Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts IV. Vol. 3648. International Society for Optics and Photonics, 1998.