HOMEWORK #2

University of Southern California

Lecturer: C.-C.JayKuo | Issued: 1/20/2019 | Due:2/12/2019

Report

1 Problem 1: Edge Detection

1.1 Sobel Edge Detector

1.1.1 Abstract and Motivation

The early stages of vision processing identify features in images that are relevant to estimating the structure and properties of objects in a scene. Edges are one such feature. Edges are significant local changes in the image and are important features for analyzing images. Edges typically occur on the boundary between two different regions in an image. Edge detection is frequently the first step in recovering information from images. Due to its importance, edge detection continues to be an active research area.

An edge in an image is a significant local change in the image intensity, usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity. The gradient is a measure of change in a image. In this Section, I will implement the Sobel edge detector to compute the gradient and do the edge detector.

1.1.2 Approach and Procedures

Sobel's filter detects horizontal and vertical edges separately on a grayscale image. By analogy, significant changes in the gray values in an image can be detected by using discrete approximately to the gradient. Thus, the colour image should be convert to grayscale image. I use the following equation 1:

$$WB(i,j) = 0.2989 \times R(i,j) + 0.5870 \times G(i,j) + 0.1140 \times B(i,j) \tag{1}$$

The gradient is the two-dimensional equivalent of the first derivative and is defined as the vector

And the magnitude of the gradien, given by

$$|G(j,k)| = \sqrt{G_x^2 + G_y^2} \tag{2}$$

A way to avoid having the gradient calculated about an interpolated point between pixels is to use a 3×3 neighborhood for the gradient calculations. Considet the arrangement of pixels about pixel $F(j,k)$ shown in Equation 3, The Sobel operator is the magnitude of the gradient computed by equation 2, where the discrete partial gradient is approximately compute by

$$\begin{pmatrix} A_0 & A_1 & A_2 \\ A_7 & F(j,k) & A_3 \\ A_6 & A_5 & A_4 \end{pmatrix} \tag{3}$$

$$G_x = \frac{1}{k+2}\left[A_2 + kA_3 + A_4 - (A_0 + kA_7 + A_6)\right] \tag{4}$$

$$G_y = \frac{1}{k+2}\left[A_0 + kA_1 + A_2 - (A_6 + kA_5 + A_4)\right] \tag{5}$$

While Sobel operator is $k = 2$. S_x and S_y can be implemented using convolution masks:

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{6}$$

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{7}$$

For the input figure *tiger.raw*, after the image extension, use S_x and S_y to convolute the image and compute the gradient map G_x and G_y, then compute $G(j,k)$ by Equation 2. In order to show the gradient result, the gradient value are normalzed between 0 to 255.

$$\hat{I}(i,j) = \frac{255 \times (I(i,j) - I_{min})}{I_{max} - I_{min}} \tag{8}$$

By accumulating the Cumulative Distribution Function($cdf(x)$) of the gradient image, the threshold can be decided. Figure 1 shows the $cdf(x)$ of gradient image.

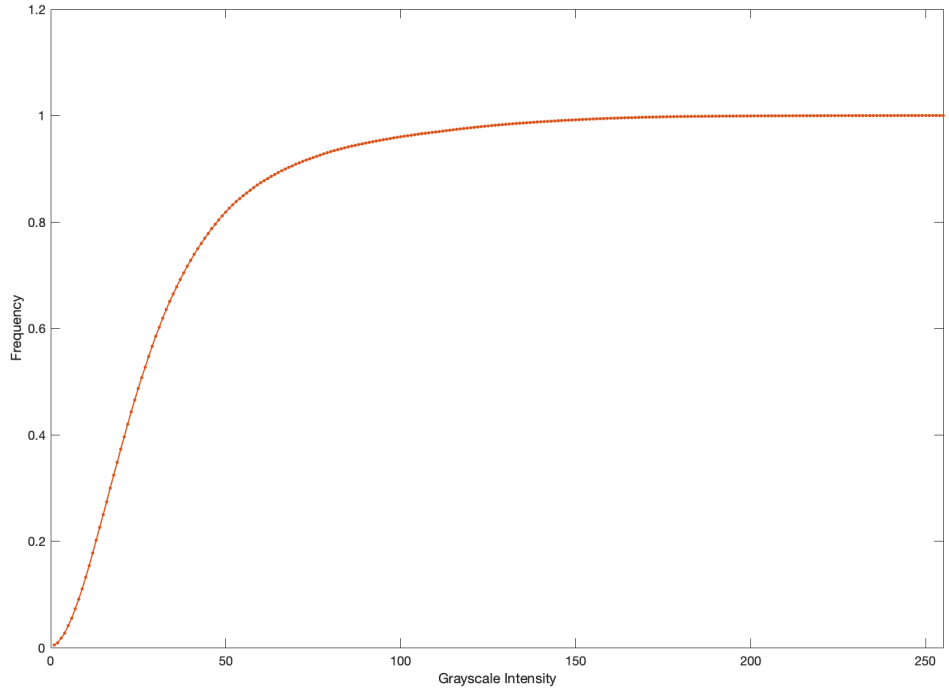

Figure 1: $cdf(x)$ of gradient map

In order to keep the most edges of the object and remove the background noise, it is a good way to choose the "stable point". From the Figure 1, the "stable point" is around 60, thus, the threshold will be turned around 60.

1.1.3 Experimental Result

Figure 2 and 9 the input image. Figure 3 and Figure 4 is the gradient G_x and G_y map. Figure 5 is the origianl edge map. Figure 6 to 15 is the edge image with different threshold.

Figure 2: *tiger.raw*

Figure 3: G_x map

1.1.4 Discussion

Sobel detector is a first order detector. From Figure 7, we can observe a clear outline of the tiger and patterns with threshold = 0.20, although there is still some noises remain in the background. And from Figure 15 we can see the best result of *Pig.raw* with thres hold = 0.20. The Sobel Operator is very quick to execute as well. It takes only $O(mn)$ times to excute all the process. By increase the threshold, some of the noise pattern is filterd and the strong edge remains. However, some weak edge is remain unfound, too. To remove the background noise and keep the weak edge, more restriction should be taken into consideration.

Figure 4: G_y map

Figure 5: Normalized Gradient Map

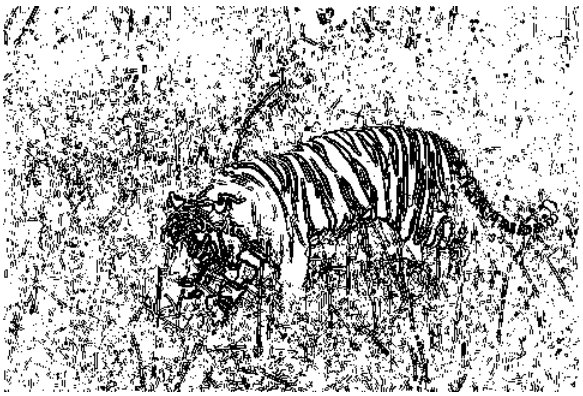

Figure 6: *Tiger_sobel.raw*, threshold = 0.15

Figure 7: *Tiger_sobel.raw*, threshold = 0.20

Figure 8: *Tiger_sobel.raw*, threshold = 0.30

Figure 9: *pig.raw*

Figure 10: G_x map

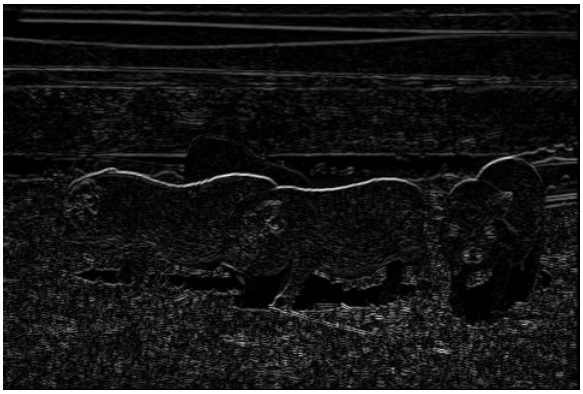

Figure 11: G_y map

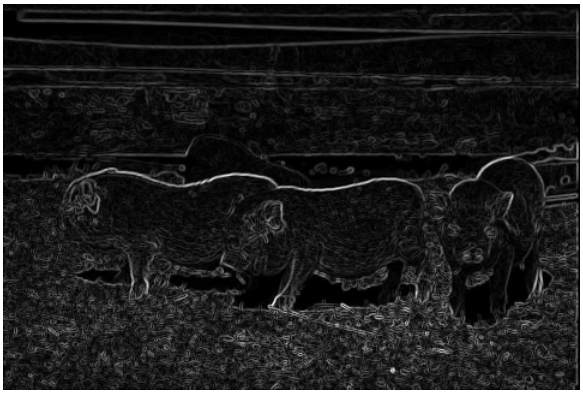

Figure 12: Normalized Gradient Map

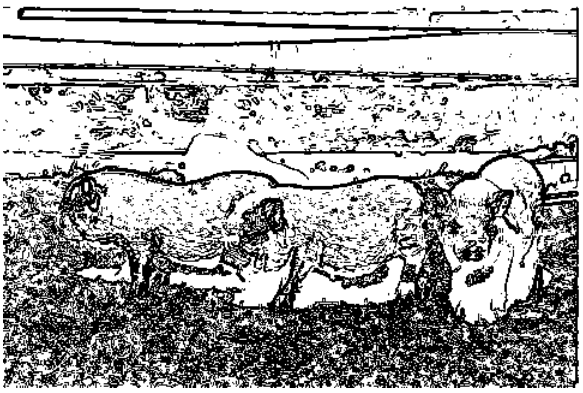

Figure 13: *Pig_sobel.raw*, threshold = 0.1

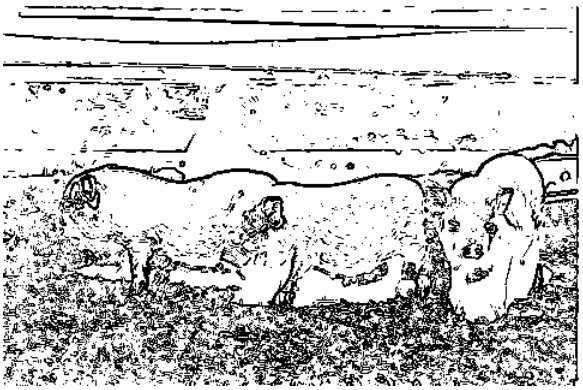

Figure 14: *Pig_sobel.raw*, threshold = 0.15

Figure 15: *Pig_sobel.raw*, threshold = 0.20

1.2 Canny Edge Detector

1.2.1 Abstract and Motivation

In order to remove the background noise and get a clear edge map, The Canny edge detector was invented by J. Canny. The Canny edge detector is the first derivative of a Gaussian and closely approximates the operator that optimizes the product of signal-to-noise ratio and localization. The Canny edge detection algorithm is summarized by the following notation. Let $I(i,j)$ denote the image. The result from convolving the image with a Gaussian smoothing filter using separable filtering is an array of smoothed data,

$$Y(i,j) = I(i,j) * Gaussian(i,j) \tag{9}$$

Where $G(i,j)$ denotes the Gaussian filter.

Then find magnitude and orientation of gradient by convolution.

$$|G(j,k)| = \sqrt{G_x^2 + G_y^2} \tag{10}$$

$$\theta(j,k) = \arctan \frac{G_y}{G_x} \tag{11}$$

To identify edges, the broad ridges in the magnitude array must be thinned so that only the magnitudes at the points of largest local change remain. This is called Non-maximum Suppression (NMS).

Non − maximum suppression thins the edges of gradient magnitude in $G(i,j)$ by suppressing all values along the line of the gradient that are not peak values of a ridge. This step is considet to remove the pixel that not the part of the edge. What it do is to supress all the gradient value to be zeors except the local optimal. The algorithm begins by reducing the angle of the gradient $\theta(i,j)$ to one. It passes a 3×3 neighborhood across the magnitude matrix $G(i,j)$. If the magnitude matrix value $G(i,j)$ at the center is not greater than both of the neighbor magnitudes along the gradient line, then $G(i,j)$ is set to zero. This process thins the broad edges of gradient magnitude in $G(i,j)$ into edges that are only one pixel wide.

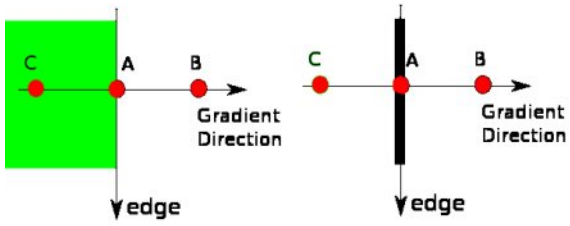

Figure 16: Non-maximum supression

Algorithm :

for each pixel $I(A)$
  if $I(A) \leq I(B)$ or $I(A) \leq I(C)$
    $I(A) = 0$
  end if
end for

1.2.2 Approach and Procedures

Download the MATLAB toolbox "Canny Edge Detection", and input the image *Tiger.jpg* and *Pig.jpg*. Tune the higher threshold and lower threshold.

1. Apply a 5×5 Gaussian Filter to smooth the image. Because noise is high frequency. Thus, it is easy to disturb the edge detection.
2. Apply Sobel detector to compute the magenitude and oritation of gradient.
3. Thin the edge with NMS method.
4. Double threshold to obtain binary edge map.

1.2.3 Experimental Result

Figure 17 is the original image *pig.raw* and the original tiger image is already shown in Figure 2. Figure 19 and 23 shows the gradient probability of *Tiger.raw* and *pig.raw* caculated by convolution. Figure 18 and 22 is the edge map. Figure 21 to 25 is the edge image with different threshold.

Figure 17: *Pig.raw*

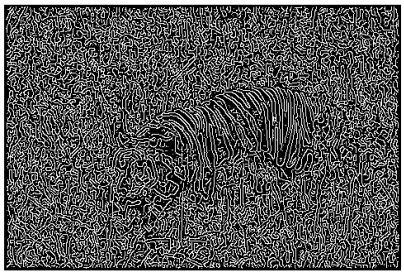

Figure 18: Total edge map of Tiger

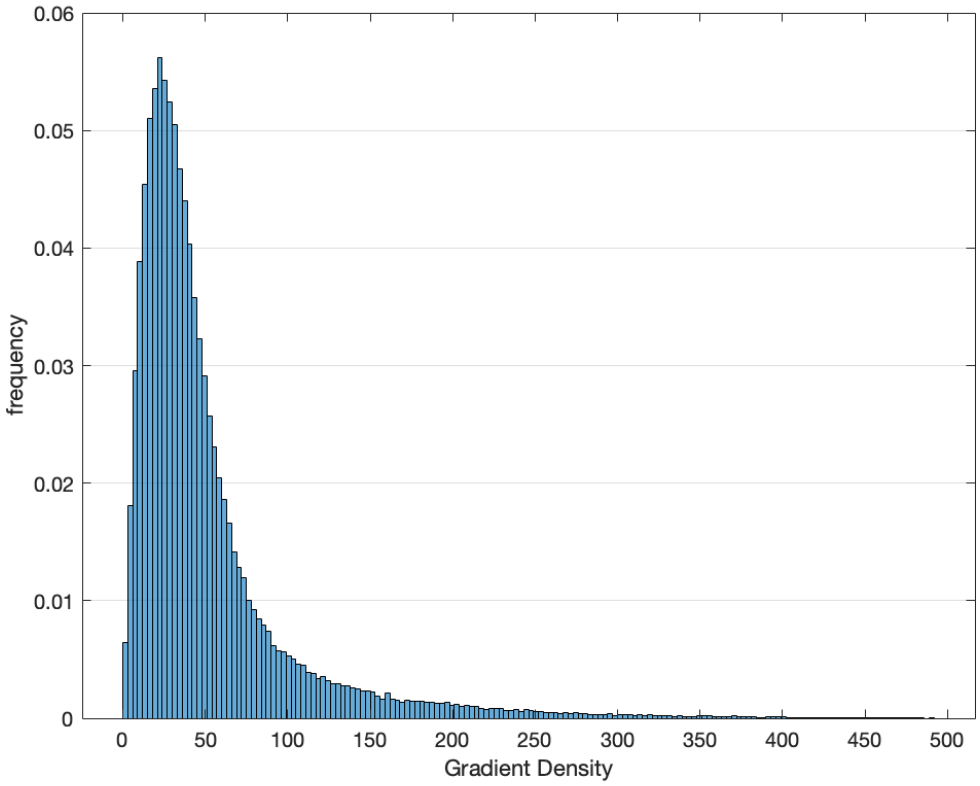

Figure 19: Gradient Probabilty of *Tiger.raw*

Figure 20: *Tiger_canny.raw*, low = 0.25, high = 0.5

Figure 21: *Tiger_canny.raw*, low = 0.245, high = 0.49

Figure 22: Total edge map of Pig

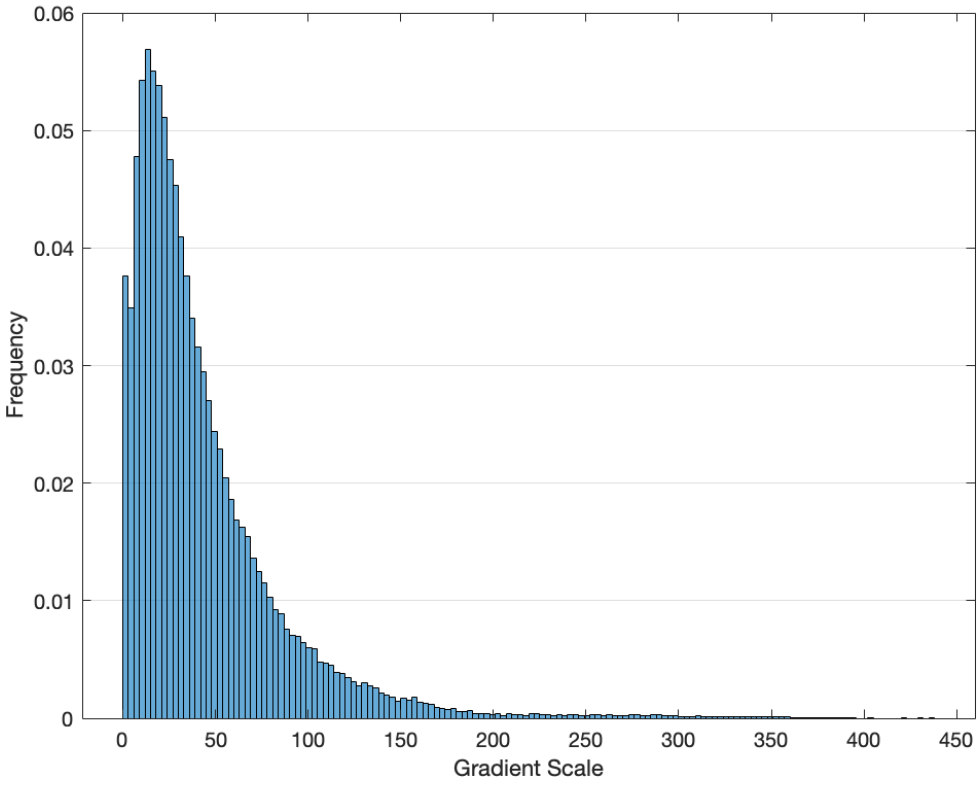

Figure 23: Gradient Probabilty of *Pig.raw*

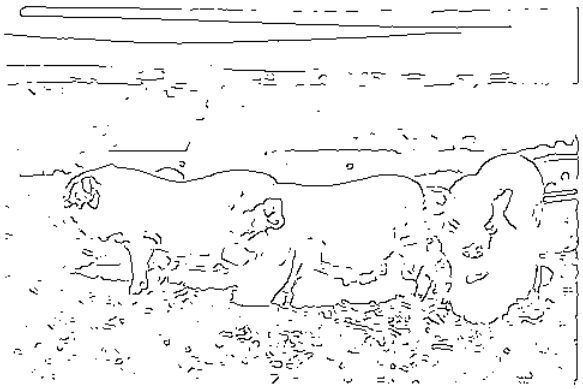

Figure 24: *Pig_canny.raw*, low = 0.22, high = 0.66

Figure 25: *Pig_canny.raw*, low = 0.25, high = 0.66

1.2.4 Discussion

Figures above show the edge map is quite clear than the sobel detector.

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, **minVal and maxVal**. Since image formed with maxVal will contain fewer false edges. but T2 may have gaps in the contours because it makes too many false judgements. Thus, we need the minVal. Those below minVal are sure to be non-edges, so discarded. The algorithm continues to gather edges from minVal until the gap has been bridged to an edge. Usually maxVal is between $2 \times minVal$ to $3 \times minVal$.

1.3 Structured Edge

1.3.1 Abstract and Motivation

Structured Edge was proposed by Piotr Dollár and C. Lawrence Zitnick in 2013. This approach allows us to take advantage of the inherent structure in edge patches, while being surprisingly computationally efficient.

This algorithm is trained by Random forests. The Random Forest is consist by the decision trees. Every dicision tree is trained in a recursive way. First compute the information gain for each feature to dicide the root. Choose the feature with the most infomation gain and divide the dataset into left leaf and right leaf. However, if we only train one tree in the model, it is easy to overfit. Therefore, the problem of overfitting can be solved by training multiple irrelative trees and combining their outputs.

For each tree, the training set they use is randomly sampled from the total training set with replacing while the features used to train the nodes of each tree are randomly selected from all the features in a certain proportion with no replacement. For the given train set S, test set T and F demention feature, the training process is as follows. 1) Sample the traing set $S(j)$ from S with same size as the sample of the root. 2) Train the decision tree. Sample f demension features from F with out replacing. Train the tree by finding the best threshold for each node. If it reach the terminal condition, return the mojority of the class. 3) Repeat the 1) and 2) until all the trees have been evaluated.

1.3.2 Approach and Procedures

The process of Structured Detection are as follows. First, Dividing all images in the segmentation-based dataset into multiple 32×32 patches that may overlapping. A 16×16 size mask located at the center is obtained in each patch. Define the mapping of the form:

$$\Pi : Y \to Z \tag{12}$$

There are $\binom{16 \cdot 16}{2} = 32640$ dimensions. To enhance the efficiency, the demension is reduced by PCA method. Use PCA quantization to obtain $k = 2$ labels (binary classification). Label all patches 0 or 1. 1 stands for edges and 0 stands for non-edge.

Then, input features. First, set 13 channels for each patches as features, including 3 color, 2 magnitude and 8 orientation channels. After that, downsample the patches by a factor of 2. resulting in $32 \times 32 \times 13/4 = 3328$ candidate features x. Sampling all candidate pairs and computing their differences yields an additional 300 candidate features per channel, resulting in 7228 in total. So far, each feature corresponds to an patch and a binary label.

Thirdly, train the decision trees. Choose several patches and obtain the features as described above. Then compute the information gain. Choose the most information gain as the root and split the patches into two class. When it comes to the depth of the tree reach 64, or the datasets belongs to one class, or the number of patches is less than 8, split terminate. Finally, set the random forest. Use 4 trees to train the patches.

The prediction part is as follows. First, input the prediction image. Cut it into 32×32 patches with overlapping can compute the 7728 candidate features for each patch. Then predict by Random forest and get the 16×16 edge patches. Average the edge patches and merge the classification result. Finaly get the output edge map.

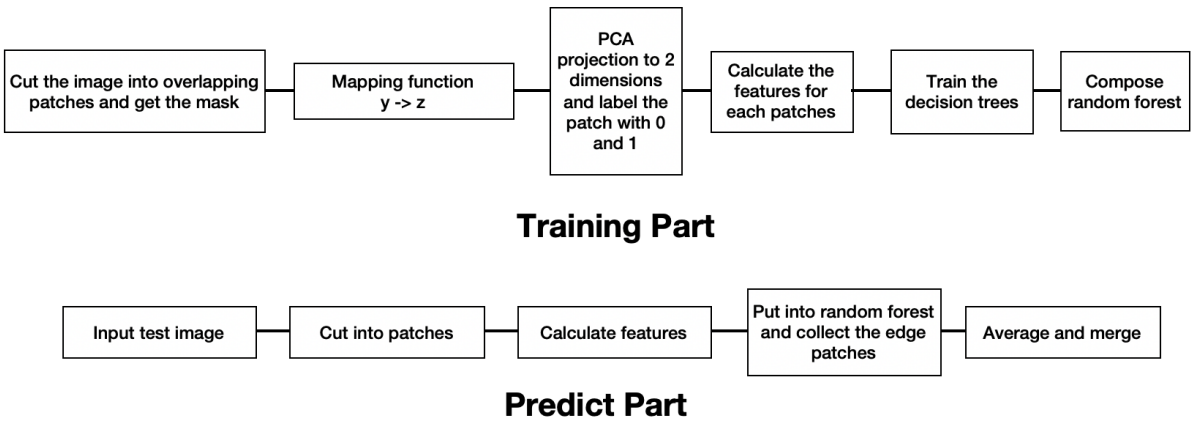

Figure 26: FlowChart of SE

1.3.3 Experimental Result

Figure 27 and 28 are the best probability edge maps for *tiger.raw* and *pig.raw* obtained by SE (Set the parameter as: NMS = Fales, Multiscale = 0, Number of Trees = 4, nTread = 4, Sharpen =

2). Figure 29 to 37 is the probability map with different parameters. And Figure 39 to **??** is the binary edge image result from the best probability map.

Figure 27: Best Probability Edge Map

Figure 28: Best Probability Edge Map

Figure 29: multiscale = 1

Figure 30: multiscale = 1

Figure 31: NMS = True

Figure 32: NMS = True

Figure 33: Number of Trees = 1

Figure 34: Number of Trees = 1

Figure 35: Number of Thread = 2

Figure 36: Number of Thread = 2

Figure 37: Sharpen = 0

Figure 38: Sharpen = 0

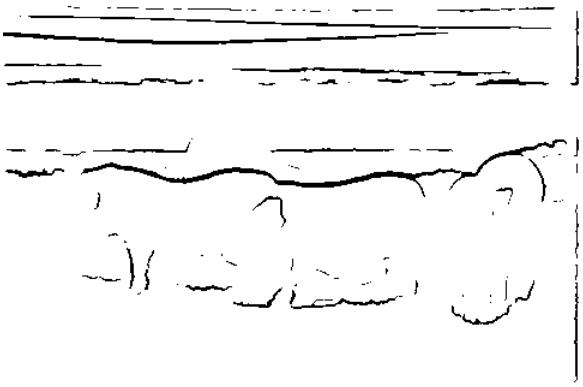

Figure 39: Threshold = 0.75

Figure 40: Threshold = 0.75

Figure 41: Threshold = 0.85

Figure 42: Threshold = 0.85

Figure 43: Threshold = 0.90

1.3.4 Discussion

For each imgae *tiger.raw* and *pig.raw* I choose the parameter to predict the edge as follow: multi-scale = 0, nms = False, number of tree evaluation = 4, nThread = 5 shapen = 2. Because the nms will thin the edges, which could result in broken edges, so I mute the NMS. Choose the Number of tree of evaluation and Thread to maximum to get the accurate result. Choose shapen = 2 to make the edge clear. Choose the multistage = 0 to make this algorithm efficient, because it does not differ much by adding up the multiscale in visual quality.

The result of SE has less fake edge compared with canny detector and sobel detector because it has a robust machine learning system, which can learn the edges and predict with higher accuracy. However, the operation speed is much slower. The complixity of the algorithm is pretty high because it include the process of training random forest. Besides, the edge obtained by canny detector is much thinner than SE because it include a NMS process.

1.4 Performance Evaluation

1.4.1 Abstract and Motivation

In pattern recognition, information retrieval and binary classification, while recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant. Both precision and recall are therefore based on an understanding and measure of relevance. Then, input the patches into Random forest and the output is 16×16 edge image patches independently. Merge overlapping predections by averaging.

$$precision = \frac{true \quad positive}{true \quad positive + false \quad positive} \tag{13}$$

$$recall = \frac{true \quad positive}{true \quad positive + false \quad negative} \tag{14}$$

However, in cases where we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F measure.

$$F = 2 \times \frac{precision \times recall}{precision + recall} \tag{15}$$

1.4.2 Approach and Procedures

Use MATLAB 2018b to perform the evaluation process. First import the ground truth dataset of *tiger.raw* and *pig.raw*. Then read the edge map generate with differen probability. Compare the result of precusuib, recall and F score.

1. First, select some threshold and generate some result image for each edge detector. Caculate the precision and recall of each ground truth and each threshold. Then average the precision and recall over threshold for each ground truth. For sobel and SE detector, The threshold are sampled evenly between 0 to 0.5 with stride = 0.05. For canny detector, both low and high threshold are sampled from 0 to 0.5 with stride = 0.05, and the low threshold should not go over the high threshold. Then caculate the F score of each ground truth. We can use this result to approximate area under ROC curve (AUC).

2. Use the gradient map for each edge detector. caculate the mean precision and recall over the ground truth for each threshold. I use different threshold and caculate the F score for each threshold. For sobel and SE detector, The threshold are sampled evenly between 0 to 0.5 with stride = 0.01. For canny detector, both low and high threshold are sampled from 0 to 1 with stride = 0.01, and the low threshold should not go over the high threshold. Plot the F score function verses the threshold.

1.4.3 Experimental Result

1. Structured Edge

Table 1: Average Precision, Recall and F Score of *tiger.raw* over threshold of SE detector

	GT1	GT2	GT3	GT4	GT5
F score	0.4778	0.4868	0.4677	0.3225	0.4083
precision	0.6725	0.6900	0.6999	0.9408	0.6729
recall	0.3705	0.3761	0.3512	0.1946	0.2931

Table 2: Average Precision, Recall and F Score of *pig.raw* over threshold of SE detector

	GT1	GT2	GT3	GT4	GT5
F score	0.3912	0.4061	0.4946	0.6075	0.4896
precision	0.3406	0.3629	0.5353	0.7780	0.5288
recall	0.4594	0.4610	0.4596	0.4983	0.4558

For each groud truth, we can observe that the precision of SE is high while the recall is low. That is, the prediction of SE detector is quite accurate, while there is a lot of edges unfound. The large gap between recall and precision results in low F score. Maximum for *pig.raw* is 0.5707, when threshold = 0.1. Maximum for *tiger.raw* is 0.5263, when threshold = 0.1.

Stuctured method can get the best result among the three edge detection method. The highest F score can reach over 0.5. The other two methods can reach that high no matter how to adjust the parameters. **Pros** : It can provide a good and clean result. **Cons** : The algorithm is too complicate and the compute speed is quite low.

2. Canny edge detector

The canny detector can obtain the precision and recall evenly. And the F score is low in *tiger.raw*, because the precision much is lower than recall.

Set that the high threshold can only exist between 2 * low threshold to 3 * high threshold. The maximum F score of *tiger.raw* reached by Canny detecor is 0.4266, while low threshold =

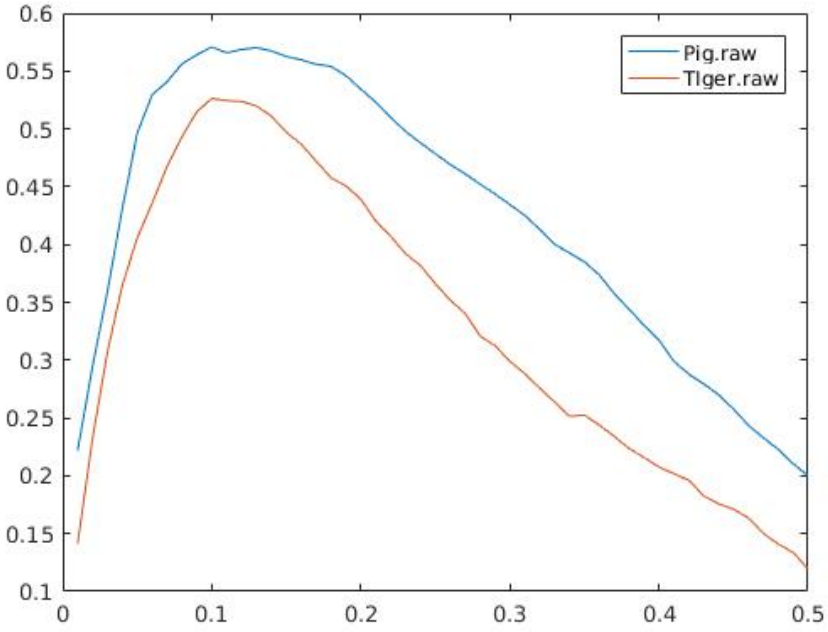

Figure 44: SE detector F score verses threshold(normalized)

Table 3: Average Precision, Recall and F Score of *tiger.raw* over threshold of Canny detector

	GT1	GT2	GT3	GT4	GT5
F score	0.2278	0.2463	0.2633	0.6865	0.2570
precision	0.1428	0.1564	0.1711	0.7829	0.1735
recall	0.5624	0.5793	0.5711	0.6111	0.4950

Table 4: Average Precision, Recall and F Score of *pig.raw* over threshold

	GT1	GT2	GT3	GT4	GT5
F score	0.4484	0.4472	0.4198	0.4569	0.4460
precision	0.4515	0.4627	0.5164	0.6088	0.5449
recall	0.4453	0.4327	0.3537	0.3657	0.3774

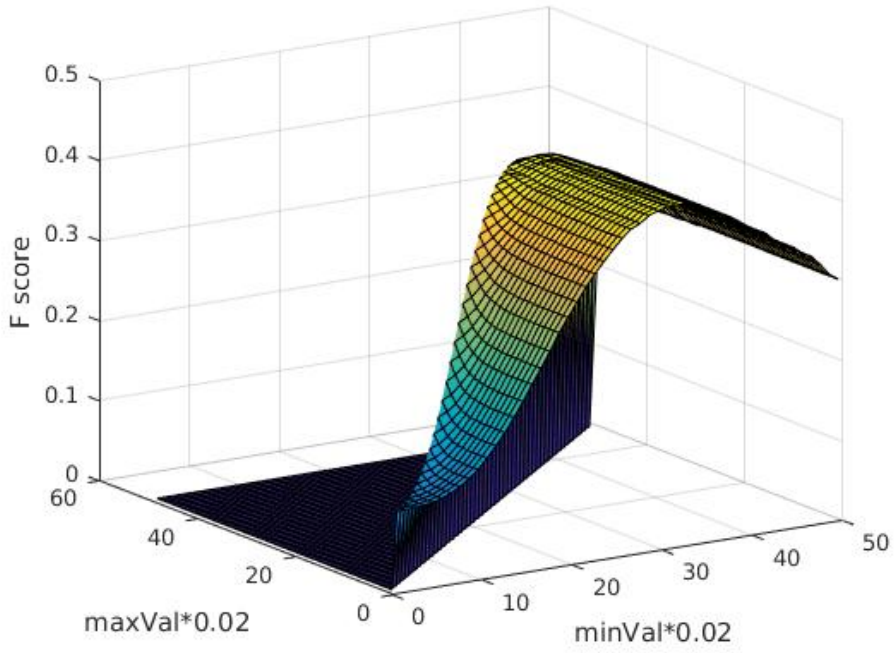

Figure 45: canny detector F score verses threshold(normalized) of *tiger.raw*

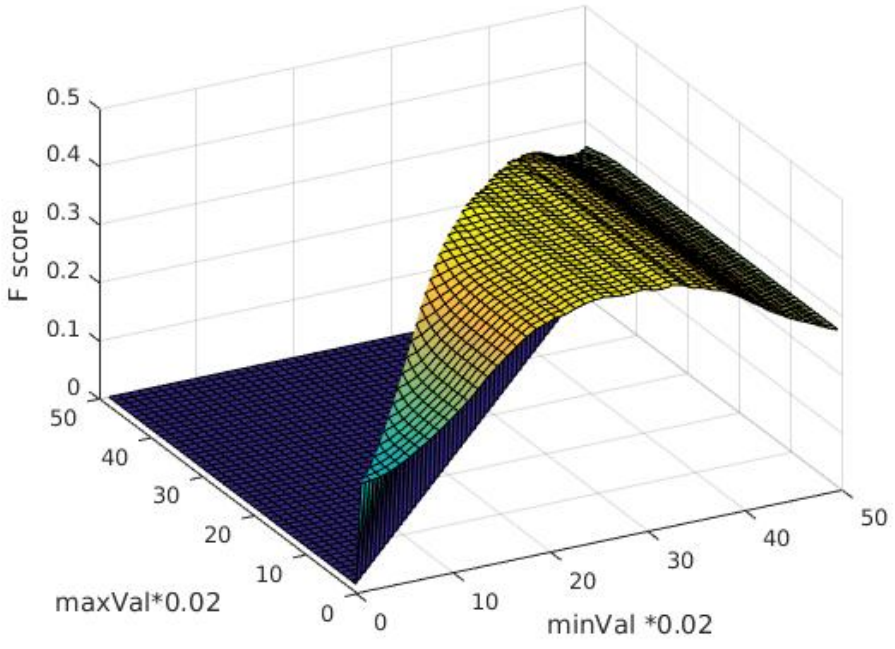

Figure 46: canny detector F score verses threshold(normalized) of *pig.raw*

0.18, high threshold = 0.33. The maximum F score of *pig.raw* is 0.4246, while low threshold = 0.24, high threshold = 0.25.

The basic idea of Canny detector uses a Gaussian function to smooth image firstly. Then the maximum value of first derivative also corresponds to the minimum of the first derivative. Thus these two thresholds are used to detect strong edges and weak edges. Considered good so far due to non-maximal suppression and two thresholding which extracts most of the edges. **Pros** : Detect the most of the edge, both strong and weak, and the edge is thin and accurate due to NMS. **Cons** : The parametors need to be manully tuned.

3. Sobel Detection

Table 5: Average Precision, Recall and F Score of *tiger.raw* over threshold of Sobel detector

	GT1	GT2	GT3	GT4	GT5
F score	0.1599	0.1676	0.1868	0.6222	0.1796
precision	0.0896	0.0944	0.1067	0.5081	0.1039
recall	0.7414	0.7444	0.7474	0.8025	0.6608

Table 6: Average Precision, Recall and F Score of *pig.raw* over threshold of Sobel detector

	GT1	GT2	GT3	GT4	GT5
F score	0.3521	0.3504	0.3509	0.3642	0.3555
precision	0.2541	0.2558	0.2807	0.3014	0.2796
recall	0.5732	0.5558	0.4677	0.4602	0.4881

From the table above, we can observe that the recall of sobel detecotor is much higher that precision, which means, the sobel detector can found the mojority of edges, however, there is lots of fake edges remains in the result image. Thus, the F score is lower compared with other detector. When threshold = 0.34, the F score of *tiger.raw* reach the highest 0.3686. When threshold = 0.28, the F score of *pig.raw* reach the highest 0.3063.

Sobel operator is easy to achieve in space, has a smoothing effect on the noise, is nearly affected by noise, can provide more accurate edge direction information but it will also detect many false edges with coarse edge width. Sobel detector operate quick but usually it fails to detect weak edges. **Pros** : Quick, deal good with noise. **Cons** : Fail to detect weak edges and predent many false edges.

1.4.4 Discussion

Compare the two image, *pig.raw* is easy to get better result by SE detecor. Because the line in image *pig.raw* is much "clear" than *tiger.raw*. SE use random forest to learn the edge. There is lots of grass in the *tiger.raw*, which could be detect as fake edges. And the pig has no streak like

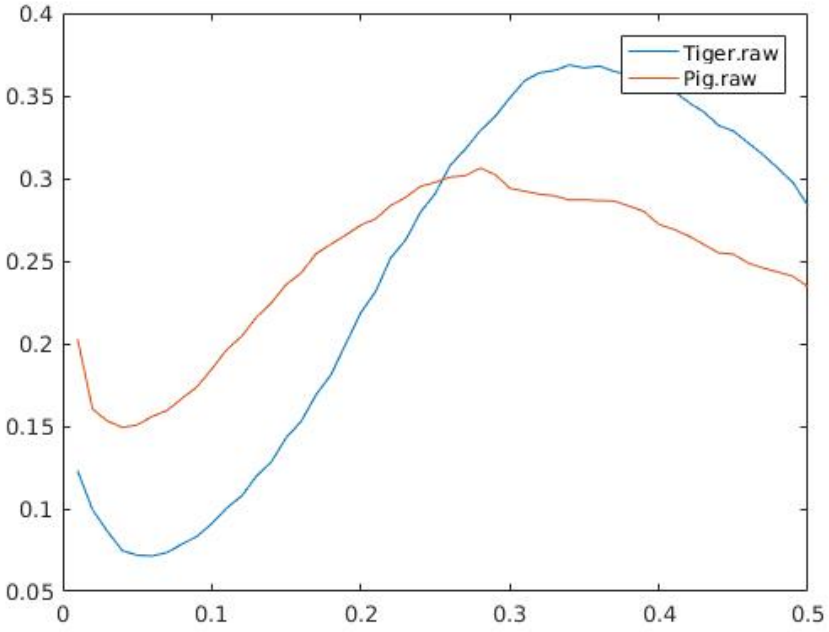

Figure 47: F Score verses Threshold of Sobel Detector

tiger does by the way:). Thus, by perform SE edge detection method, the edge of *pig.raw* can be detected accurately and usually get higher F score.

However, the F score of *tiger.raw* is higher than *pig.raw* by canny detector and sobel detector. Maybe the contrast of background and the object is more obvious. For the traditional edge detector, the gradient is higher and the figure is easy to be found.

There is a trade-off between precision and recall. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the all observations in actual class. F1 Score is the weighted average of Precision and Recall. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

To get higher F score, the precision and recall value can not differ too much. With high precision but low recall, you classifier is extremely accurate, but it misses a significant number of instances that are difficult to classify. This is not very useful. With high recall and low precision, your classifier may found much positive label, but the accuracy is quiet low because it makes many false judgement.

If the sum of precision and recall are constant C. According to *Cauchy–Schwarz inequality*,

$$F = \frac{2PR}{P+R} = \frac{2P(C-P)}{C} \leq \frac{C}{2} \tag{16}$$

When $C - P = R = P$, the F score reach the maximum.

2 Problem 2: Digital Half-toning

2.1 Dithering

2.1.1 Abstract and Motivation

Dithering is a type of half tone thresholding where greyscale (or RGB channel) intensity is converted into a local density of binary pixels. Originally, Dithering has been developed for colour reduction in two dimensional pictures. Because human eyes got low-pass filter property. This method of "creating" a large color palette with a limited set of colors is often used in computer images, television and the printing industry. In this section, I use two method to dithering the grayscale image.

2.1.2 Approach and Procedures

1. Random thresholding
   In order to break the monotones in the result from fixed thresholding, we may use a "random" threshold. The algorithm can be described as: General a random number from 0 to 255 for each pixel so called $R(i,j)$. Then determined the half-toned pixel value by:

$$G(i,j) = \begin{matrix} 255 & R(i,j) \leq F(i,j) \\ 0 & else \end{matrix} \tag{17}$$

2. Dithering Matrix
   Dithering parameters are specified by an index matrix. The values in an index matrix indicate how likely a dot will be turned on. For example, an index matrix is given by

$$I_2(i,j) = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \tag{18}$$

   where 0 indicates the pixel most likely to be turned on, and 3 is the least likely one. The Bayer index matrices are defined recursively using the formula:

$$I_{2n} = \begin{pmatrix} 4 \times I_2(i,j) + 1 & 4 \times I_2(i,j) + 2 \\ 4 \times I_2(i,j) + 3 & 4 \times I_2(i,j) \end{pmatrix} \tag{19}$$

   The index matrix can then be transformed into a threshold matrix T for an input gray-level image with normalized pixel values (i.e. with its dynamic range between 0 and 255) by the following formula:

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2} \times 255 \tag{20}$$

   where N^2 denotes the number of entries in the matrix. Since the image is usually much larger than the threshold matrix, the matrix is repeated periodically across the full image. This is done by using the following formula:

$$G(i,j) = \begin{matrix} 255 & F(i,j) \geq T(i \mod N, \; j \mod N) \\ 0 & else \end{matrix} \tag{21}$$

The input image *bridge.raw* is modified by both two methods.

2.1.3 Experimental Result

Figure 48 shows the original image. Figure 49 shows the random dithered image. Figure 50 to 52 shows the image dithered by dithering matrix.

Figure 48: *bridge.raw*

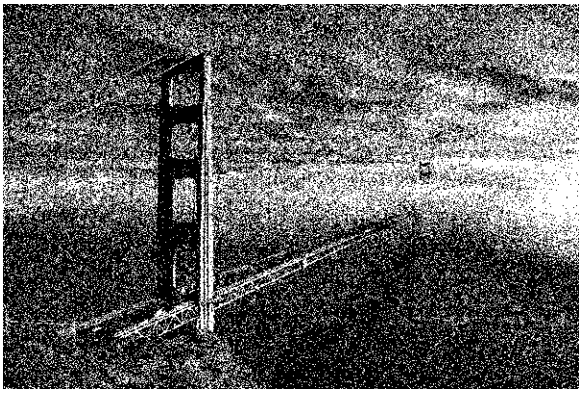

Figure 49: *bridge.raw* dethiered randomly

Figure 50: *bridge.raw* dithered by I_2

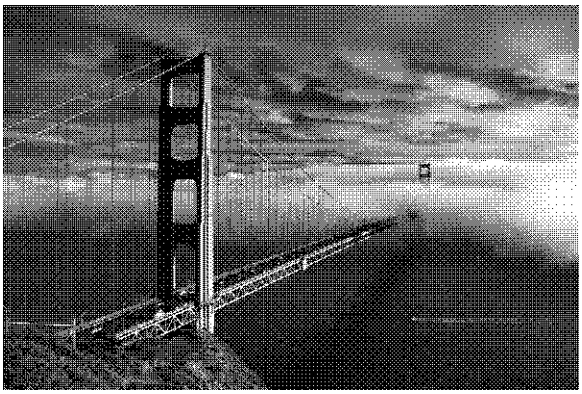

Figure 51: *bridge.raw* dithered by I_8

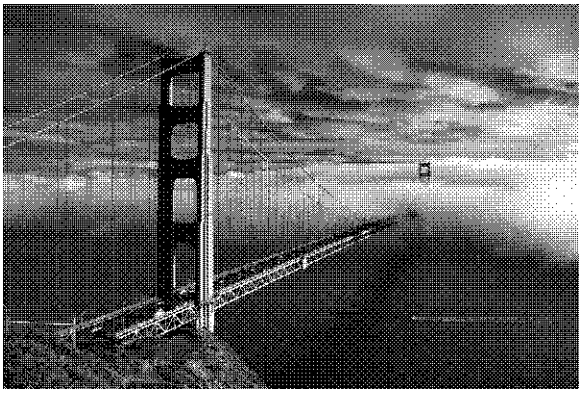

Figure 52: *bridge.raw* dithered by I_{32}

2.1.4 Discussion

From figure 49 we can observe white noise appears in the entire image, which looks like TV picture "snow". Though the image produced is very inaccurate and noisy, it is free from "artifacts" which are phenomena produced by digital signal processing. Besides, the algorithm operation is quite simple, it takes only $O(mn)$ times.

Figure 50 to 52 indicate that the image quality has been siginaficantly improved with the implementation of dithering matrix. From I_2 to I_{32}, the image quality is enhanced. However, by zooming the the piture we can clearly observe some cross-hatch pattern artifacts appears in the image. That makes the image looks like a "Cross Stitch", expecially in figure 50. Figure 50 dithered by I_2 looks like anime movie made by Shinkai Makoto. With the increasing of the size of diffusion matrix, the atrifacts reduce. The algorithm complixity is the same as random dithering.

2.2 Error Diffusion

2.2.1 Abstract and Motivation

Error Diffusion method diffuses the quantization error of a pixel to its neighboring pixels. The output value is determined by comparing with the thresholding and drawing the least error output value. By scanning the image in serpentine order, diffusion matrix diffuses the error to the adjacent, unwritten pixels.

2.2.2 Approach and Procedures

The error dispersion technique is very simple to describe: for each point in the image, first find the closest color available. Calculate the difference between the value in the image and the color you have. Now divide up these error values and distribute them over the neighboring pixels which you have not visited yet. When you get to these later pixels, just add the errors distributed from the earlier ones, clip the values to the allowed range if needed, then continue as above.

There are many way to distribute the error. I use three diffusion matrix to half-tone the input image, that is the Floyd-Steinberg's, Jarvis's and Stucki's. As usual, the original image should be extented by reflection in order to get the result image with same size as input.

2.2.3 Experimental Result

Figure 53 is half-toned by Floyd-Steinberg's error diffusion. Figure 54 is half-toned by Jarvis, Judice, and Ninke's error diffusion. Figure 55 is half-toned by Stucki's error diffusion.

2.2.4 Discussion

The result of error diffusing method is much better than the dithering method. It generates fewer artifacts and enhance the edge. However, the compute complexity is greatly increased.

Besides, the output result of Jarvis, Judice, and Ninke Dithering is slightly better than Floyd-Steinberg Dithering. With this algorithm, the error is distributed to three times as many pixels as in Floyd-Steinberg, leading to much smoother – and more subtle – output. It preserve the stereoscopy of the objects, such as the slings and the rocks. But it still have shortcomings. It pushes the error down to the follwing two rows rather than one row. That means we have to keep

Figure 53: Floyd-Steinberg

Figure 54: Jarvis

Figure 55: Stucki

two forward arrays – one for the next row, and another for the row after that. Stucki Dithering choose the entry of the matrix as power of two. Thus, bit-shifting can be used to derive the specific values to propagate. That speed up the computing process a lot and the output image has minimal difference compared with JJN Dithering.

Thus, I prefer the Error Diffusion method with Stucki diffusion matrix because it provide good visual quality and the algorithm speed has been enhanced slightly.

To improve the result, I suggest that not only arrange the entries of the dithering matrix equal to 2^k but adjust the sum of the entries equal to 2^k as well. Thus, bit-shifting can be used to derive the specific values to propagate. Besides, the size of the dithering matrix could be decrese to 3×3 matrix. This would definitely speed up this algorithm.

2.3 Color Halftoning with Error Diffusion

2.3.1 Abstract and Motivation

In theory, all colors should be printable using just a combination of yellow, magenta, and cyan inks, which is called CMYK (K, meaning key)—usually but not always overprinting—system. Unlike RGB system, CMYK system is reflection system. It reflect the color that we actually observe. Color image halftoning is similar to the grayscale image. When properly printed using a quality high-resolution halftone screen, the dot structure of the printed image should not be visible. This chapter describes two color dithering techniques, an error-diffusion dither based on the Floyd–Steinberg method and a MBVQ-based Error diffusion method.

2.3.2 Approach and Procedures

1. Separable Error Diffusion
   This method is to dither RGB channels separately with Floyd-Steinberg. First of all, the image should be separated into CMY three channels.

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{22}$$

   where the R, G, B are the normalized pixel value of input image.

   Then apply the Floyd-Steinberg error diffusion algorithm to quantize each channel separately as Section 2.2.

2. MBVQ-based Error diffusion
   It is known that given a color in the RGB cube, it maybe rendered in 8 basic color located at the vertices of the cube. Relevant to the problem at hand is the fact that the human visual system is more sensitive to changes in brightness than to changes in the chrominance. Thus we arrive at the Minimal Brightness Variation Quadruples (MBVQ) for halftoning of solid color patches. The main idea of MBVQ is to reduce halftone noise select from within all halftone sets by which the desired color maybe rendered the one whose brightness variation is minimal.

   This method consider that the input image pixel value could be arranged in one of the six half toning quadruples, RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB. Each with obviously

minimal brightness variation. Then use the pixel value with diffused error from privious pixels to find the nearest vertex (with minimal quantization error). Finally distribute the error to adjacent pixels.

According to the MBVQ, colors have to be rendered using the halftone set that not only preserve the average color but also the pixels bright variation is minimal. There will be some Chrominance difference by implementing MBVQ method, due to the low-pass property of human eyes, it matters much less in large solid color patches.

2.3.3 Experimental Result

Figure 56 shows the origianl image *bird.raw*. Figure 57 shows the image toned by Separable Error Diffusion. Figure 58 and 59 shows the image toned by MBVQ-based Error diffusion.

Figure 56: *bird.raw*

Figure 57: Separable Error Diffusion

Figure 58: MBVQ-based Error diffusion with FS error diffusion

Figure 59: MBVQ-based Error diffusion with JJN error diffusion

2.3.4 Discussion

Separable Error Diffusion could reach a good result. However, there is no constrain imposed to the error diffusion process so the luminance are various in the image. Because that the error will add up by using error diffusion method. While pixel values going up, the brightness increase. There is a relationship between luminanance and chrominance,

$$\mu = \frac{R + G + B}{3} \tag{23}$$

Such an approcah implicity assume there is no correlationship between luminance and chrominance. Thus, this algorithm creates a white noise among the picture.

Comparing the visual quality with the output image of Separable Error Diffusion, we can observe that the white noise reduces and the contrast of the image has been enhanced. This algorithm effectively reduces the number of pixel colors visible in a given solid area. It does not modify the color while you observe the picture from a distance because it major color should remain the same. However, this method takes more compute time compared with separable error diffusion method.