EE 569: Homework #2 Issued: 1/20/2019 Due: 11:59PM, 2/12/2019

Professor C.-C. Jay Kuo

Problem 1: Edge Detection (50 %)

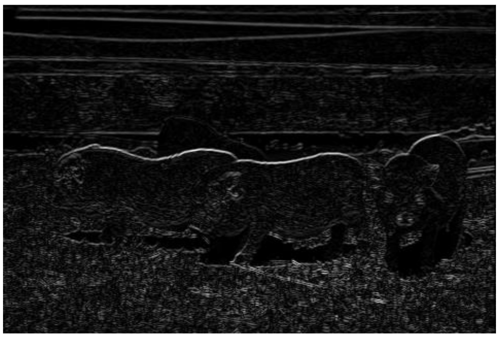
a) Sobel Edge Detector (Basic: 10%) (MATLAB)

- **Convert RGB to gray-level, use the convert formula:**

$Y_{\text{linear}} = 0.2126R_{\text{linear}} + 0.7152G_{\text{linear}} + 0.0722B_{\text{linear}}$, which Y is the gray value.

- **Normalize the x-gradient and the y-gradient values to 0-255 and show the results:**



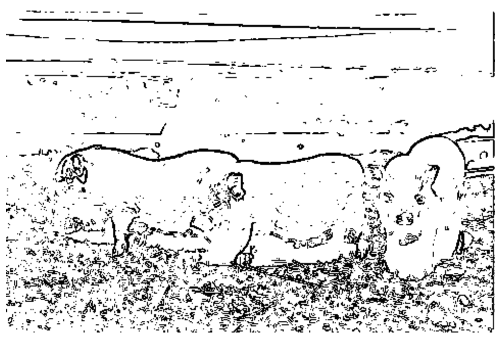



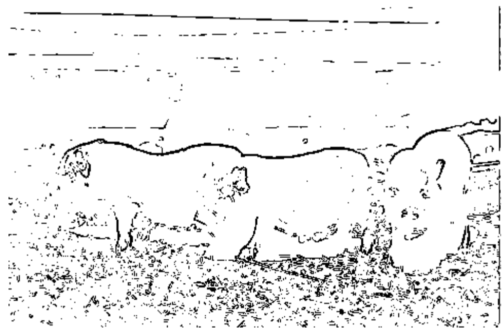
x-gradient y-gradient

- **Use the formula:**

$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$ **, to get the gray value, then compare to the threshold.**

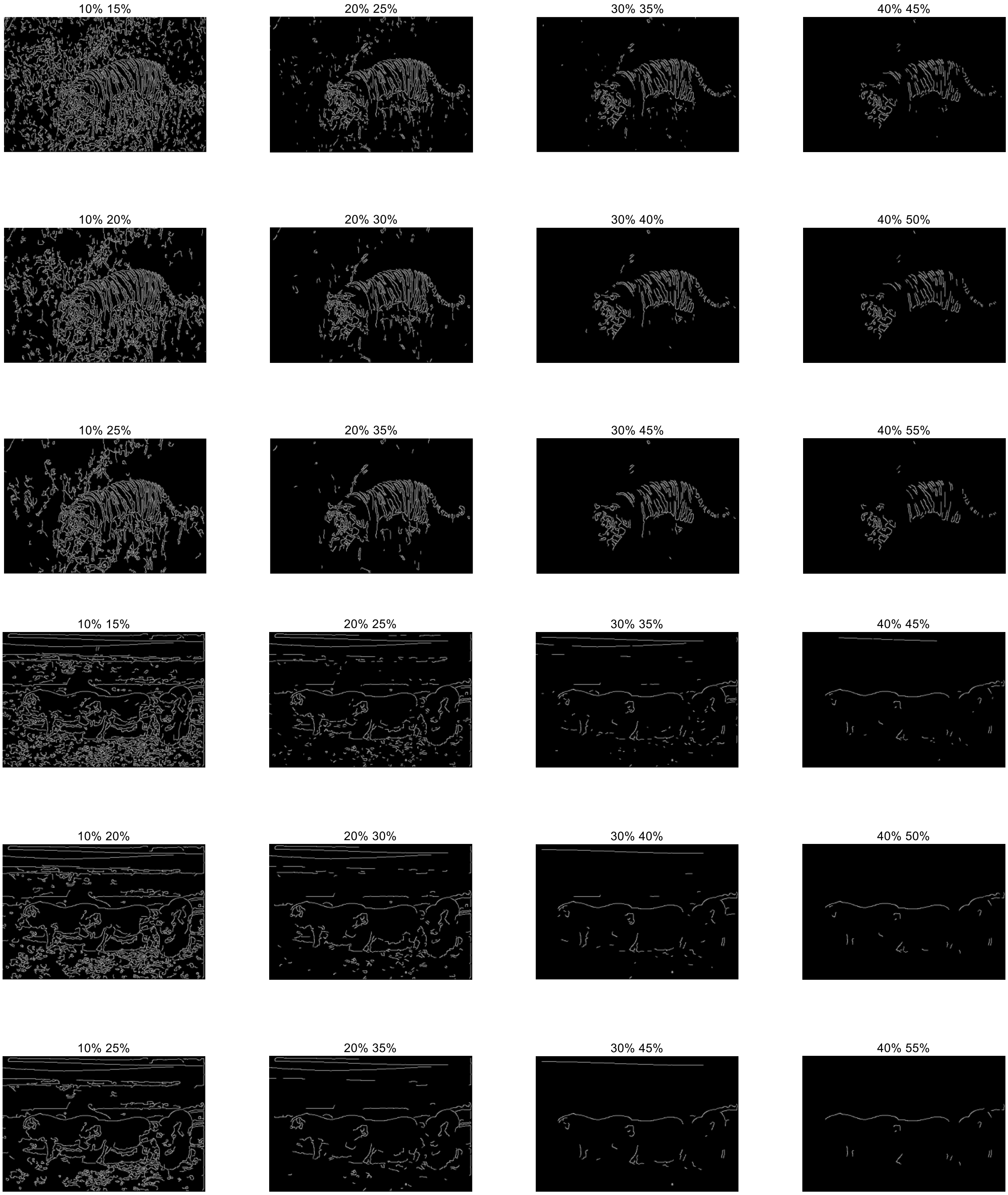
- **Tune the thresholds:**

Threashold(%)	tiger edge map	Pig
20		

25(BEST)		
30		
40		

- **Disscuss:**
  From the table above, we can see about 20% is the best edge result.
  Set the threshold too high -> useful edges and useless edges will both decrease.
  Set the threshold too low ->more useful edges, but noise edge will increase.
  Sobel has a smoothing effect on noise and provides more accurate edge direction information. But, the edge accuracy is not high enough.

b) Canny Edge Detector (Basic: 10%) (MATLAB)



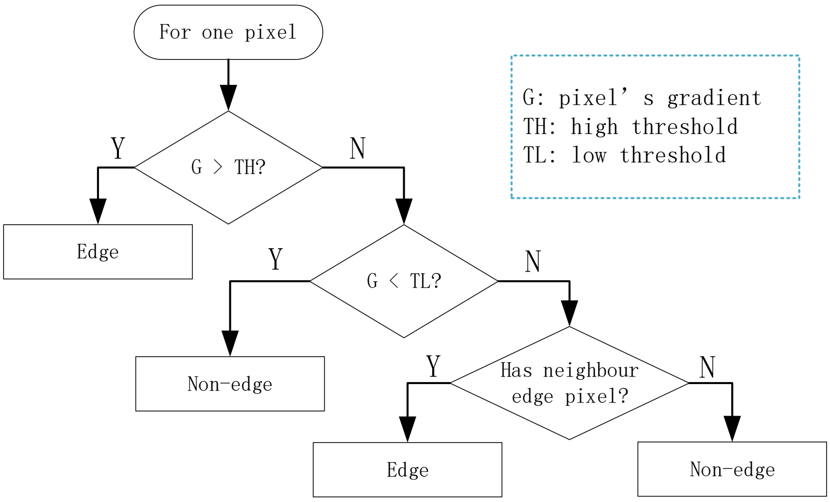
- **Non-maximum suppression explaination**

  As the name suggest, non-maximun suppression means suppress the value which are not the maximun value.

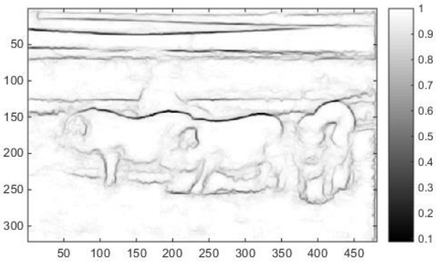
  When we look along the gradient direction at the edge, we will set the gradient to zero if this gradient is not bigger than both its neighbours' gradient(along the gradient direction)(because these location is just the transition zone between the non-edge area and the edge), otherwise, we preserve the gradient value(these locations are the true thin edge).

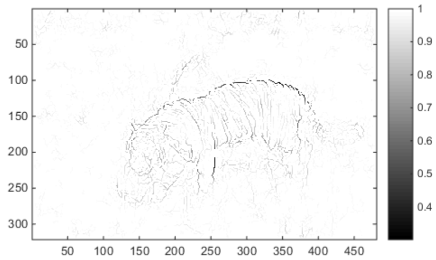
  After this process, our edges will be much thinner and clearer.

- **High and low thresholds**

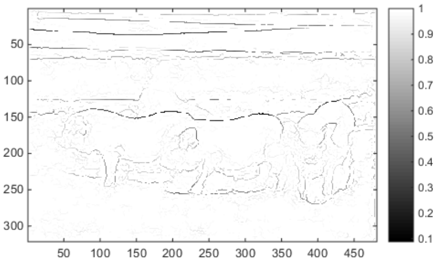


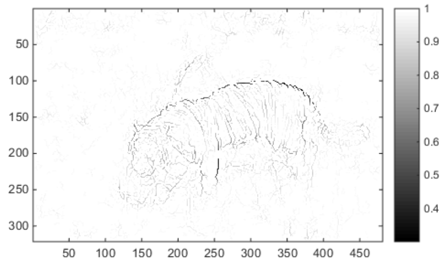
c) Structured edge (advanced: 15%)



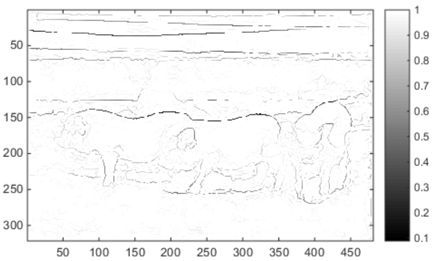


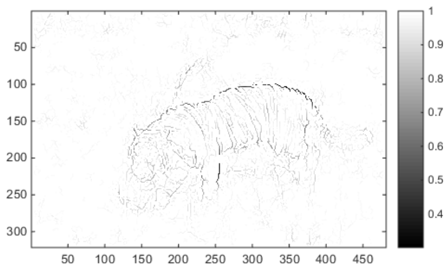
```
model.opts.multiscale=0;
model.opts.sharpen=2;
model.opts.nTreesEval=4;
model.opts.nThreads=4;
model.opts.nms=0;
```



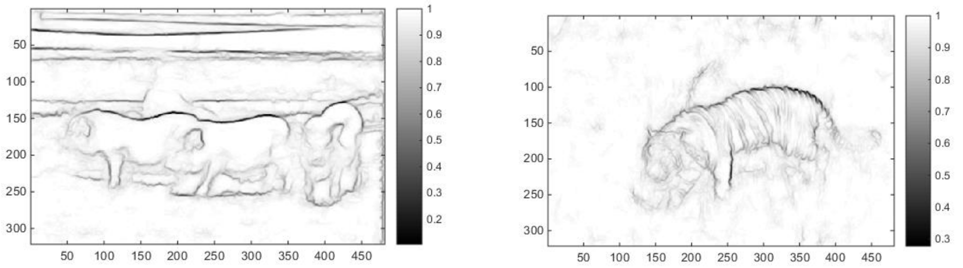


```
model.opts.multiscale=0;
model.opts.sharpen=2;
model.opts.nTreesEval=4;
model.opts.nThreads=4;
model.opts.nms=1;
```

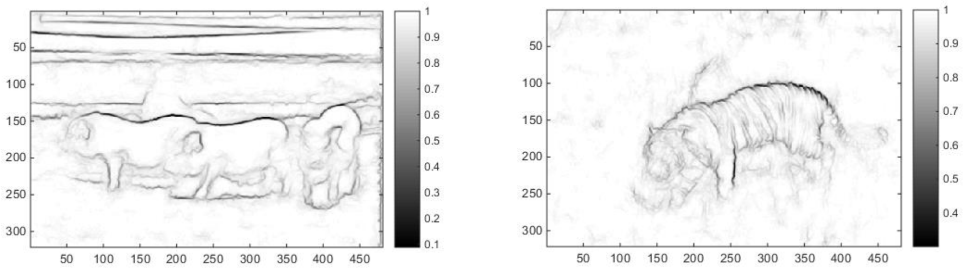




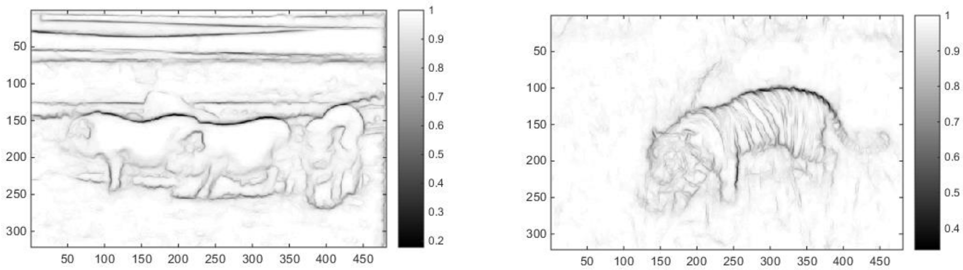
```
model.opts.multiscale=0;
model.opts.sharpen=4;
model.opts.nTreesEval=4;
model.opts.nThreads=4;
model.opts.nms=1;
```



```
model.opts.multiscale=0;
model.opts.sharpen=4;
model.opts.nTreesEval=10;
model.opts.nThreads=4;
model.opts.nms=0;
```



```
model.opts.multiscale=0;
model.opts.sharpen=4;
model.opts.nTreesEval=4;
model.opts.nThreads=10;
model.opts.nms=0;
```



```
model.opts.multiscale=1;
model.opts.sharpen=2;
model.opts.nTreesEval=4;
model.opts.nThreads=4;
model.opts.nms=0;
```

Sharpen: when non-zero, the differences are not very big.
Nms: will make the edges much thiner and clear
Multiscale: big -> accurate
nTreeseval & nThreads: not much effect

d) Performance evaluation
1.
2. tiger. Because pig image has some area hard to recognize the egde.
3.

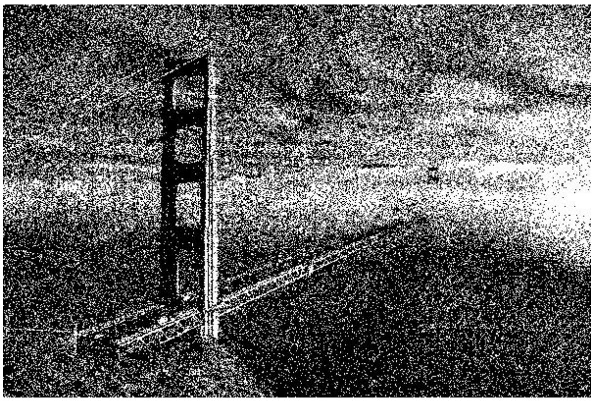
Problem 2: Digital Half-toning (50%)

a) Dithering (Basic: 15%)
1. Random thresholding
Random thresholding will break the monotones in the result from fixed thresholding, and this will make the output image more natural.

- **Generate random integer number form 0 to 255**
- **Use the mapping rule:**

$$G(i,j) = \begin{cases} 0 & \text{if } 0 \le F(i,j) < rand(i,j) \\ 255 & \text{if } rand(i,j) \le F(i,j) < 256 \end{cases}$$

- **Result**



2. Dithering Matrix

- **Use the following formulas:**

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}, \quad I_{2n}(i,j) = \begin{bmatrix} 4\times I_n(i,j)+1 & 4\times I_n(i,j)+2 \\ 4\times I_n(i,j)+3 & 4\times I_n(i,j) \end{bmatrix}, \quad T(x,y) = \frac{I(x,y)+0.5}{N^2}\times 255$$

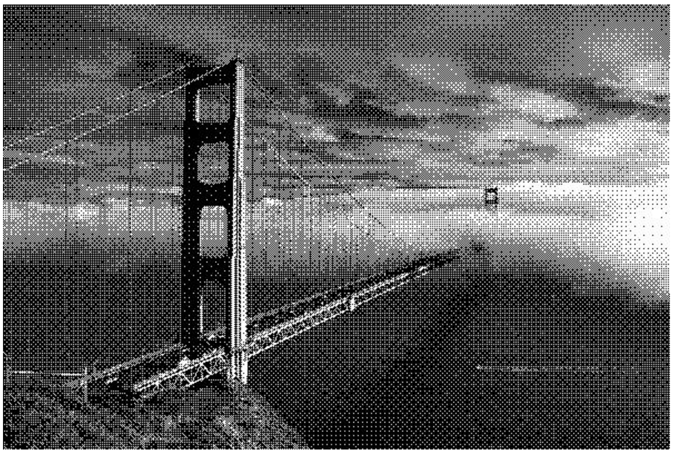
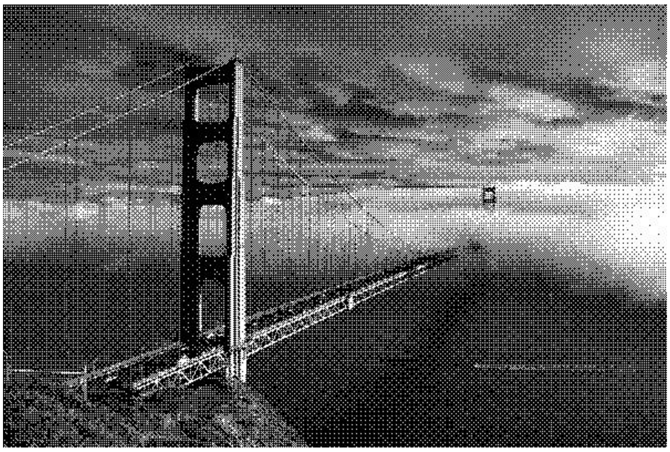
Write a function to generate the target Dithering matrix, input: empty Dithering matrix and matrix size, output: final Dithering matrix.

```
// ex.: if the x size of dithering matrix is 8, LoopNum should be 2
int DitherMat(int *Dither, int LoopNum) {⋯}
```

Target matrix size	Input: LoopNum	Before normalize	Output
2	0	1 2 3 0	95 159 223 31
4	1	5 9 6 10 13 1 14 2 7 11 4 8 15 3 12 0	87 151 103 167 215 23 231 39 119 183 71 135 247 55 199 7
8	2	21 37 25 41 22 38 26 42 53 5 57 9 54 6 58 10 29 45 17 33 30 46 18 34 61 13 49 1 62 14 50 2 23 39 27 43 20 36 24 40 55 7 59 11 52 4 56 8 31 47 19 35 28 44 16 32 63 15 51 3 60 12 48 0	85 149 101 165 89 153 105 169 213 21 229 37 217 25 233 41 117 181 69 133 121 185 73 137 245 53 197 5 249 57 201 9 93 157 109 173 81 145 97 161 221 29 237 45 209 17 225 33 125 189 77 141 113 177 65 129 253 61 205 13 241 49 193 1
32	4	[illegible]	[illegible]

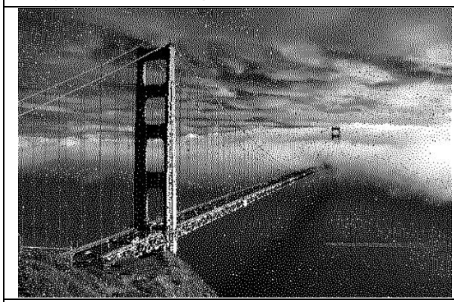
- **Output image**

I_2: lossing much details	I_4: more details than I_2, have similar small repeat pattern

	
I_8: compare to I_4, not improve too much	I_{32}: more details, especially in the cloud area. This is because taking more surrounding imformation into consideraion, monotones will be less.
	

b) Error Diffusion (Basic: 15%)

Threshold: 128

Floyd-Steinberg	JJN	Stucki
		
When comes to thin line, FS output is too blur.	JJN and Stucki's output is much better than FS. Showing more detail thin lines.	Seems like Sharpness is higher than JJN, because the matrix center weight is much higher than outter part.

- **Improve idea: use the dithering matrix as the threshold in replace of the fixed threshold. The fixed threshold will bring more monotones, so the details will not be shown very well.**

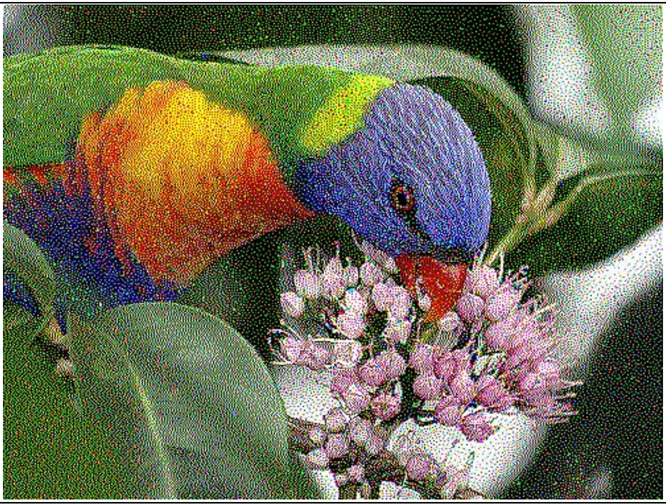
c) Color Halftoning with Error Diffusion (Advanced: 20%)

1. Separable

- **Additive color space to subtractive color space**

  (255, 255, 255) – (RGB color value) = (CMY color value)
- **Use the error diffusion method to each color channel seperately**

Output CMY image	Output RGB image

Use int 0 ~255 to calculate the color value, so the output image have some noise point caused by the loss of precission.

2. MBVQ

- MBVQ

  Key idea: keep the color coordinat transfermation in a **small area**, to reduce the risk of large change in **brightness**.

  In order to produce a good color halftone one has to place colored dots so that the following specifications are optimally met: The placement pattern is visually unnoticeable and the local average color is the desired color can easily be achieved. **But the colors used may reduce the notice-ability of the pattern using the separable error diffusion. Human visual system is more sensitive to changes in brightness than to changes in the chrominance, which average at much lower frequencies.** So in order to keep the notice-ability of the pattern we should restrict the color vertex move too far away, so we use the MBVC(To reduce halftone noise, select from within all halftone sets by which the desired color may be rendered, the one whose brightness variation is minimal) in MBVQ, so after the error diffusion, the brightness of each pixel will have a relatively small change. This will help the output have more details and the right color brightness
-

Output image	original
	

- Compare to the separable output, MBVQ has more details, and the brightness is more like the original one.
- MBVQ still have some noise because of the loss of precision. But still better than the separable method.