ECE 637: Lab 8

Rahul Deshmukh
deshmuk5@purdue.edu

April 12, 2021

Section 3.1 Report

1. Original and thresholded image:

(a) Original image

(b) Thresholded image

2. Computed RMSE and fidelity values:

Listing 1: RMSE and fidelity for section 3.1

```
1 house_threshold
2 RMSE:87.393       Fidelity:75.420
```

3. For python code for fidelity function refer to Listing 8 at page 4

   For python code refer to Listing 7 at page 4 and Listing 12 at page 9.

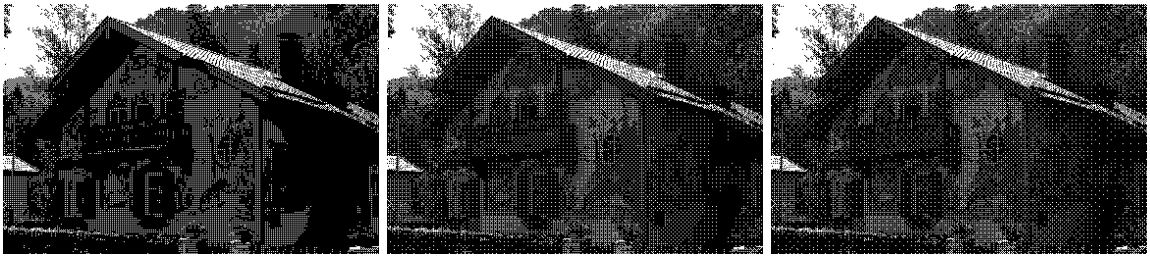
Section 4.2 Report

- The three Bayer index matrices of sizes 2x2, 4x4, and 8x8:

Listing 2: Bayer index matrices

```
Bayer index matrix: size 2x2
[[1  2]
 [3  0]]
Bayer index matrix: size 4x4
[[ 5  9  6 10]
 [13  1 14  2]
 [ 7 11  4  8]
 [15  3 12  0]]
Bayer index matrix: size 8x8
[[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]]
```

- The three halftoned images:



(a) Halftoned image for 2x2 (b) Halftoned image for 4x4 (c) Halftoned image for 8x8

- RMSE and Fidelity for each of the three halftoned images:

Listing 3: RMSE and fidelity for 2x2

```
house_dither2
RMSE:97.669     Fidelity:48.217
```

Listing 4: RMSE and fidelity for 4x4

```
house_dither4
RMSE:101.007    Fidelity:15.524
```

Listing 5: RMSE and fidelity for 8x8

```
house_dither8
RMSE:100.915    Fidelity:15.087
```

For python refer to Listing 9 at page 6, Listing 10 at page 7 and Listing 12 at page 9.

Section 5 Report

- For python code refer to Listing 11 at page 7
- The error diffusion result:

Figure 3: Error diffusion result

- RMSE and fidelity results:

Listing 6: RMSE and fidelity

```
1 house_error_diff
2 RMSE: 98.847      Fidelity :13.640
```

- Table of RMSE and fidelity

Method	RMSE	Fidelity
Simple thresholding	87.393	75.420
Order dithering 2x2	97.669	48.217
Order dithering 4x4	101.007	15.524
Order dithering 8x8	100.915	15.087
Error diffusion	98.847	13.640

As we can observe visually from the halftoned images produced using different methods, the image contours improved when we used order dithering. However, the RMSE metric increases in this case which indicates that RMSE is not an appropriate metric. For the case of Fidelity the metric score decreases as we increase the size of dithering and visually the image contours improve and we can see smoother grayscale colors. Therefore, fidelity is a good metric in this case. Fidelity score is the lowest for Error diffusion method and visually the image also looks the closest to the input image out of all the halftoned images.

Appendix

Got to git repo for complete code.

Listing 7: Python code for section 3

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: rahul
ECE637 DIP-1: lab 8 Halftoning
section 3
"""

import numpy as np
from PIL import Image
import os, sys
import matplotlib.pyplot as plt
from matplotlib import cm

from utils import RMSE, Fidelity

def get_base(filename):
    base = os.path.basename(filename).split('.')[0]
    return base

def main(input_img, Threshold):
    #compute binary image using threshold
    out_img = (255*(input_img>Threshold).astype(np.int)).astype(np.uint8)
    #compute rmse
    rmse = RMSE(input_img, out_img)
    fidelity = Fidelity(input_img, out_img)
    return out_img, rmse, fidelity

if __name__=="__main__":
    input_img_name = sys.argv[1]
    output_img_name = get_base(input_img_name)+ '_threshold'
    Threshold = 127.0

    im = Image.open(input_img_name)
    input_img = np.array(im)

    out_img, rmse, fidelity = main(input_img, Threshold)

    #print info
    print(output_img_name)
    print("RMSE:%0.3f \t Fidelity:%0.3f"%(rmse, fidelity))

    #save plots and images
    gray = cm.get_cmap('gray',256)
    plt.figure(frameon=False)
    plt.imshow(out_img, cmap=gray, interpolation='none')
    plt.axis('off')
    plt.savefig(output_img_name+'.pdf', bbox_inches='tight', pad_inches=0)

    out_im = Image.fromarray(out_img)
    out_im.save(output_img_name+'.tif')
```

Listing 8: Python code for RMSE and Fidelity

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: rahul
ECE637 DIP-1: lab 8 Halftoning
utility file for image metrics
"""

import numpy as np
from scipy.signal import convolve2d

def conv2d(X_img, conv_filter):
    "apply conv with zero padding"
    filter_size,_ = conv_filter.shape
    filter_half_wd = filter_size//2
    theta = conv_filter.flatten()
    ht,wd = X_img.shape
    padded_img = np.zeros((ht+2*filter_half_wd, wd+2*filter_half_wd))
    padded_img[filter_half_wd:-filter_half_wd, filter_half_wd:-filter_half_wd] =
        X_img
    out_img = np.zeros((ht,wd))
    for row in range(ht):
        for col in range(wd):
            x = padded_img[row : row + 2*filter_half_wd + 1,
                           col : col + 2*filter_half_wd + 1]
            out_img[row,col] = np.dot(x.flatten(), theta)
    return out_img

def Un_gamma_correct(img, gamma):
    lin =  255.*((img/255.)**gamma)
    return lin.astype(np.uint8)

def Scale_to_equal_brightness(img):
    return Un_gamma_correct(img, 1./3.)

def LPF(img, sigma=np.sqrt(2), size=7):
    "pass image through low-pass filter"
    #make the filter
    h = np.zeros((size,size))
    for r in range(size):
        for c in range(size):
            i = r - (size//2)
            j = c - (size//2)
            h[r,c] = np.exp(-1.0*(i**2+j**2)/(2.0*(sigma**2)))
    h /= np.sum(h) #normalize
    #convolve filter
    out_img = convolve2d(img, h, mode='same')
    #out_img = conv2d(img, h)
    #rescale to 0-255
    out_img -= out_img.min()
    out_img *= (255./out_img.max())
    return out_img.astype(np.uint8)

def RMSE(original_img, binary_image):
    "compute rmse error betweeen two images"
    h,w = original_img.shape
    assert (h,w)== binary_image.shape
    rmse = (1./(h*w))*np.sum((original_img.astype(np.float32) -
```

```
                                    binary_image.astype(np.float32))**2)
    return np.sqrt(rmse)

def Fidelity(original_img, binary_img):
    "Compute image fidelity metric"
    gamma = 2.2
    # un-gamma correct the images
    original_lin = Un_gamma_correct(original_img, gamma)
    binary_lin = Un_gamma_correct(binary_img, gamma)
    #print(np.linalg.norm(binary_lin-binary_img))
    # LPF
    original_lin_lpf = LPF(original_lin)
    binary_lin_lpf = LPF(binary_lin)
    assert original_lin_lpf.shape==original_img.shape
    # scale to equal brightness (cube-root)
    f_tilde = Scale_to_equal_brightness(original_lin_lpf)
    b_tilde = Scale_to_equal_brightness(binary_lin_lpf)
    #rmse
    fidelity = RMSE(f_tilde, b_tilde)
    return fidelity
```

Listing 9: Python code for section 4

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: rahul
ECE637 DIP-1: lab 8 Halftoning
section 4
"""

import numpy as np
from PIL import Image
import os, sys
import matplotlib.pyplot as plt
from matplotlib import cm

from utils import RMSE, Fidelity

def get_base(filename):
    base = os.path.basename(filename).split('.')[0]
    return base

def Dither_mat(N):
    if N==2:
        return np.array([[1,2],[3,0]])
    else:
        return np.block([[4*Dither_mat(N//2) + 1, 4*Dither_mat(N//2) + 2 ],
                         [4*Dither_mat(N//2) + 3, 4*Dither_mat(N//2)      ]])

def Threshold_mat(dither_mat):
    N,_=dither_mat.shape
    T = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            T[i,j] = 255.0*((dither_mat[i,j] + 0.5)/N**2)
    return T

def main(input_img, dither_size, gamma):
```

```
    h,w = input_img.shape
    linear_img = 255.*(input_img/255.)**gamma
    dither_mat = Dither_mat(dither_size)
    threshold_mat = Threshold_mat(dither_mat)

    out_img = np.zeros((h,w))
    for i in range(h):
        for j in range(w):
            r = i%dither_size; c = j%dither_size
            if linear_img[i,j] > threshold_mat[r,c]:
                out_img[i,j] = 255
            else:
                out_img[i,j] = 0
    return out_img.astype(np.uint8)

if __name__=="__main__":
    input_img_name = sys.argv[1]
    dither_size = np.int(sys.argv[2])
    output_img_name = get_base(input_img_name)+ '_dither'+str(dither_size)
    gamma=2.2
    print(output_img_name)

    im = Image.open(input_img_name)
    input_img = np.array(im)

    out_img= main(input_img, dither_size, gamma)
    #compute rmse
    rmse = RMSE(input_img, out_img)
    fidelity = Fidelity(input_img, out_img)
    #print info
    print("RMSE:%0.3f \t Fidelity:%0.3f"%(rmse, fidelity))

    #save plots and images
    gray = cm.get_cmap('gray',256)
    plt.figure(frameon=False)
    plt.imshow(out_img, cmap=gray, interpolation='none')
    plt.axis('off')
    plt.savefig(output_img_name+'.pdf', bbox_inches='tight', pad_inches=0)

    out_im = Image.fromarray(out_img)
    out_im.save(output_img_name+'.tif')
```

Listing 10: Python code for printing bayer matrices

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  8 17:15:10 2021

@author: rahul
"""
from order_dithering import Dither_mat

for i in [2,4,8]:
    d = Dither_mat(i);
    print('Bayer index matrix: size %dx%d'%(i,i))
    print(d)
```

Listing 11: Python code for section 5

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: rahul
ECE637 DIP-1: lab 8 Halftoning
section 5
"""

import numpy as np
from PIL import Image
import os, sys
import matplotlib.pyplot as plt
from matplotlib import cm

from utils import RMSE, Fidelity

def get_base(filename):
    base = os.path.basename(filename).split('.')[0]
    return base

def main(input_img, gamma, Threshold):
    h,w = input_img.shape
    linear_img = 255.*(input_img/255.)**gamma
    out_img = np.zeros((h,w))
    H = np.array([7., 3., 5., 1.])/16.
    K = [0, 1, 1, 1]
    L = [1, -1, 0, 1]
    for i in range(h):
        for j in range(w):
            old_pxl = linear_img[i,j]
            q = 255. if old_pxl>Threshold else 0.
            out_img[i, j] = q
            e = old_pxl - q
            for h_kl,k,l in zip(H,K,L):
                r = i+k; c = j+l
                if r<h and r>=0 and c<w and c>=0:
                    linear_img[r,c]+= e*h_kl
    return out_img.astype(np.uint8)

if __name__=="__main__":
    input_img_name = sys.argv[1]
    output_img_name = get_base(input_img_name)+ '_error_diff'
    gamma=2.2
    Threshold = 127.
    print(output_img_name)

    im = Image.open(input_img_name)
    input_img = np.array(im)

    out_img= main(input_img, gamma, Threshold)
    #compute rmse
    rmse = RMSE(input_img, out_img)
    fidelity = Fidelity(input_img, out_img)
    #print info
    print("RMSE:%0.3f \t Fidelity:%0.3f"%(rmse, fidelity))

    #save plots and images
    gray = cm.get_cmap('gray',256)
```

```
    plt.figure(frameon=False)
    plt.imshow(out_img, cmap=gray, interpolation='none')
    plt.axis('off')
    plt.savefig(output_img_name+'.pdf', bbox_inches='tight', pad_inches=0)

    out_im = Image.fromarray(out_img)
    out_im.save(output_img_name+'.tif')
```

Listing 12: Bash code for running python code

```
#!/bin/bash

img_dir=../..

#section 3
python simple_thresholding.py "$img_dir"/house.tif | tee sec3.log
mv ./*.tif ./output/sec3/
mv ./*.pdf ./output/sec3/
mv ./*.log ./output/sec3/
echo 'section 3 done'

#section 4
python print_dither.py | tee dither_mats.log
for (( size=2; size<=8; size*=2 ))
do
python order_dithering.py "$img_dir"/house.tif $size | tee dither_"$size".log
done
mv ./*.tif output/sec4/
mv ./*.pdf output/sec4/
mv ./*.log output/sec4/
echo 'section 4 done'

#section 5
python error_diffusion.py "$img_dir"/house.tif | tee sec5.log
mv ./*.pdf output/sec5/
mv ./*.tif output/sec5/
mv ./*.log output/sec5
echo 'section 5 done'
```