ECE 637: Lab 7

Rahul Deshmukh
deshmuk5@purdue.edu

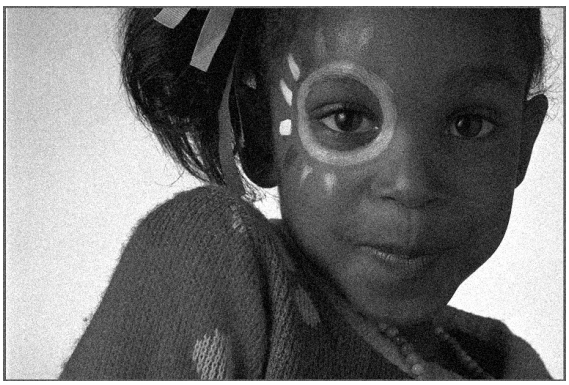
April 2, 2021

Section 1 Report

1. Plot of four original images:

(a) Plot of img14g.tif

(b) Plot of img14bl.tif



(c) Plot of img14gn.tif

(d) Plot of img14sp.tif

2. Output of filtering of blurred and noisy image

(a) Plot of restored img14bl.tif

(b) Plot of restored img14gn.tif

(c) Plot of restored img14sp.tif

3. Computed MSME filter log files:

Listing 1: output log for img14bl

```
img14bl_predicted.pdf
MSME filter:
[[ 1.4921875  -1.6601562   0.0625      0.9140625  -0.1171875   0.0546875
   0.5625    ]
 [-0.15625    -1.359375   -0.4296875  -0.4375     -1.21875    -0.390625
  -0.421875  ]
 [ 1.3046875  -0.953125    0.75        2.40625     0.765625   -1.09375
   0.265625  ]
 [-1.15625     0.0625      1.109375    1.984375    0.4296875  -0.609375
  -0.2421875 ]
 [ 0.3828125  -0.8203125   0.078125    1.0703125   0.625      -1.28125
   0.71875   ]
 [-0.703125   -0.59375     0.4609375   0.046875   -0.78125    -2.0859375
   1.296875  ]
 [ 1.15625    -0.8828125   0.33203125 -1.015625    1.1484375  -0.0625
  -0.01367188]]
```

Listing 2: output log for img14gn

```
img14gn_predicted.pdf
MSME filter:
[[-0.01063347 -0.00878716 -0.00785637  0.03654099  0.03588104  0.01151276
   0.003479  ]
 [ 0.03536224  0.0034523   0.01105881  0.04052734  0.04608154 -0.01688004
  -0.00157547]
 [-0.0184288   0.02134514  0.06572723  0.09631729  0.03370667 -0.01888657
  -0.02017593]
 [-0.01058197  0.00429344  0.1015377   0.19906235  0.07169342  0.015522
  -0.00151062]
 [-0.00034142  0.04063416  0.03710556  0.08506012  0.03329086  0.00695038
   0.01192093]
 [-0.02857971  0.01278305  0.02582932  0.01792717  0.00188446  0.02498627
  -0.01054001]
 [-0.0283699   0.0007782   0.01643181  0.01193619  0.01169395  0.00658798
   0.01018143]]
```

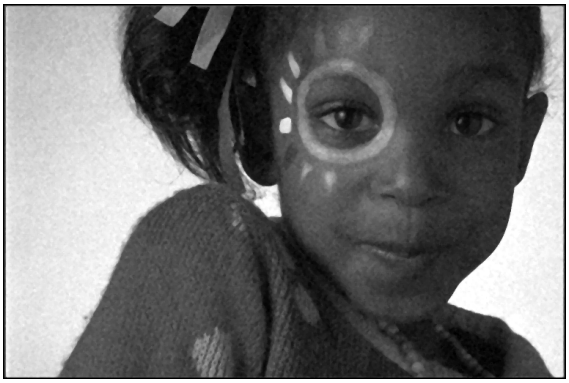
Listing 3: output log for img14sp

```
img14sp_predicted.pdf
MSME filter:
[[ 2.55813599e-02 -3.52668762e-02  2.54096985e-02  4.34341431e-02
   4.62589264e-02  5.03158569e-03 -1.52587891e-04]
 [ 2.32276917e-02  4.73403931e-03 -2.82287598e-03  2.36091614e-02
   3.70140076e-02 -1.42402649e-02 -2.33078003e-03]
 [-1.43814087e-03 -1.71585083e-02  6.91909790e-02  1.28982544e-01
   5.95092773e-03 -1.69448853e-02 -7.74383545e-03]
 [ 1.77001953e-02 -1.22451782e-02  7.86056519e-02  1.83246613e-01
   8.58840942e-02 -6.29043579e-03  7.38525391e-03]
 [-1.05819702e-02  2.92892456e-02  5.72891235e-02  1.10614777e-01
   6.13117218e-02 -1.10015869e-02  1.42440796e-02]
 [-2.28233337e-02  1.28097534e-02  1.84783936e-02  2.27775574e-02
   2.81219482e-02  1.04141235e-03 -1.97486877e-02]
 [-3.70254517e-02  1.55181885e-02  1.83067322e-02 -1.18026733e-02
  -1.48773193e-03  1.66130066e-02  1.60026550e-02]]
```

For python code refer to Listing 4 at page 6 and Listing 6 at page 11.

Section 2 Report

Results of median filtering:



(a) median filtered img14gn.tif

(b) median filtered img14sp.tif

For C-code refer to Listing 5 at page 7 and Listing 7 at page 11.

Appendix

Got to git repo for complete code.

Listing 4: Python code for section 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 30 13:58:00 2021

@author: rahul
lab 7: Image restoration Section-1
ece637-DIP-1
"""
import sys, os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib import cm
#from scipy.signal import convolve2d

filter_size = 7
sampling_frequency = 20

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

def conv2d(X_img, conv_filter):
    "apply conv with zero padding"
    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 main(X_img,Y_img, filter_size):
    filter_half_wd = filter_size//2
    v = np.arange(filter_size) - filter_half_wd
    ht, wd = X_img.shape
    assert (ht,wd) == Y_img.shape
    #make Z
    Z_sparse=[]; Y_sparse=[]; N=0
    for i in range(ht//sampling_frequency):
        for j in range(wd//sampling_frequency):
            row = (i + 1)*sampling_frequency
            col = (j + 1)*sampling_frequency
            #zs = X_img[row - filter_half_wd : row + filter_half_wd + 1,
            #           col - filter_half_wd : col + filter_half_wd + 1]
            zs = X_img[row + v[0] : row + v[-1] + 1, col + v[0] : col + v[-1] + 1]
```

```
                zs = X_img[row, col]
                ys = Y_img[row, col]
                Z_sparse.append(list(zs.reshape(filter_size*filter_size)))
                Y_sparse.append(ys)
                N+=1
    Z_sparse = np.array(Z_sparse)
    Y_sparse = np.array(Y_sparse)

    #compute R_zz, r_zy, theta
    R_zz = np.dot(Z_sparse.T, Z_sparse)/N
    r_zy = np.dot(Z_sparse.T, Y_sparse)/N
    theta = np.dot(np.linalg.inv(R_zz), r_zy)

    #apply theta filter to X to get Y_hat
    theta = np.reshape(theta, (filter_size, filter_size))
    print('MSME filter: ')
    print(theta)
    #Y_hat_img = convolve2d(X_img, theta)
    Y_hat_img = conv2d(X_img, theta)

    #rescale to 0-255
    Y_hat_img -= Y_hat_img.min()
    Y_hat_img *= (255.0/Y_hat_img.max())
    return Y_hat_img.astype(np.uint8)

if __name__=="__main__":
    original_img_name = sys.argv[1]
    noisy_img_name = sys.argv[2]
    output_name = get_base(noisy_img_name) + '_predicted.pdf'
    print(output_name)

    X_img = np.array(Image.open(noisy_img_name)).astype(np.float32)
    Y_img = np.array(Image.open(original_img_name)).astype(np.float32)

    Y_hat_img = main(X_img, Y_img, filter_size)

    gray = cm.get_cmap('gray',256)
    plt.figure(frameon=False)
    plt.imshow(Y_hat_img, cmap=gray)
    plt.axis('off')
    plt.savefig(output_name, bbox_inches='tight', pad_inches=0)

    #im = Image.fromarray(Y_hat_img)
    #im.save(temp_name)
```

Listing 5: C-code for section 2

```

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "iostream"
using namespace std;

void error(char *name);
int median_filter(uint8_t **X, char **wts,int filter_wd, int row, int col);
int* Sort(int *A, int *sorted_ids, int size);
void merge_sort(int *A, int *sorted_ids, int start, int end);
```

```
14 void merge(int *A,int *sorted_ids,int start,int mid,int end);
15 void print_array(int *A, int size);
16
17 int32_t filter_size=5;
18
19 int main (int argc, char **argv)
20 {
21   FILE *fp;
22   struct TIFF_img input_img, out_img;
23   char **wts;
24   int32_t i,j;
25
26   if ( argc != 3 ) error( argv[0] );
27
28   /* open image file */
29   if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
30     fprintf ( stderr, "cannot open file %s\n", argv[1] );
31     exit ( 1 );
32   }
33
34   /* read image */
35   if ( read_TIFF ( fp, &input_img ) ) {
36     fprintf ( stderr, "error reading file %s\n", argv[1] );
37     exit ( 1 );
38   }
39
40   /* close image file */
41   fclose ( fp );
42
43   /* check the type of image data */
44   if ( input_img.TIFF_type != 'g' ) {
45     fprintf ( stderr, "error:  image must be 8-bit color\n" );
46     exit ( 1 );
47   }
48
49   /* Allocate image of double precision floats */
50   wts = (char **)get_img(filter_size, filter_size, sizeof(char));
51
52   /* create wts array*/
53   for ( i = 0; i < filter_size; i++ ){
54   for ( j = 0; j < filter_size; j++ ) {
55     if((i==0)||(i==filter_size-1)||(j==0)||(j==filter_size-1)){
56       wts[i][j] =1;}
57     else{wts[i][j] =2;}
58   }
59   }
60
61   /* set up structure for output achromatic image */
62   /* to allocate a full color image use type 'c' */
63   get_TIFF ( &out_img, input_img.height, input_img.width, 'g' );
64
65   /* median filtering of input image */
66   for ( i = 0; i < input_img.height; i++ )
67   for ( j = 0; j < input_img.width; j++ ) {
68     // zero for boundary pixels
69     if( (i<(filter_size/2))||(j<(filter_size/2))||\
70       (i > input_img.height-(filter_size/2)-1)||\
71       (j>input_img.width-(filter_size/2)-1) ){
72       out_img.mono[i][j] = 0;
```

```
73        continue;
74      }
75      //median filtering
76      out_img.mono[i][j] = (uint8_t)median_filter(input_img.mono, wts, filter_size, i,
         j);
77    }
78    fprintf(stdout,"filtering done\n");
79    /* open output image file */
80    if ( ( fp = fopen ( argv[2], "wb" ) ) == NULL ) {
81      fprintf ( stderr, "cannot open file %s\n",argv[2]);
82      exit ( 1 );
83    }
84
85    /* write output image */
86    if ( write_TIFF ( fp, &out_img ) ) {
87      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
88      exit ( 1 );
89    }
90
91    /* close output image file */
92    fclose ( fp );
93
94    /* de-allocate space which was used for the images */
95    free_TIFF ( &(input_img) );
96    free_TIFF ( &(out_img) );
97
98    free_img( (void**)wts );
99
100   return(0);
101 }
102
103 void error(char *name)
104 {
105     printf("usage:  %s  image.tiff \n\n",name);
106     printf("this program reads in a 8-bit grayscale TIFF image.\n");
107     printf("it then restores the image by reducing noise using median filtering.\n")
       ;
108     printf("finally it saves an 8-bit grayscale image for the restored image.\n");
109     exit(1);
110 }
111
112 int median_filter(uint8_t **X, char **wts, int filter_wd, int row, int col){
113   int i=0, j, r, c, size=filter_wd*filter_wd;
114   int sum1, sum2;
115   int *X_array, *a_array;
116   int *sorted_ids=NULL;
117   //populate array for sorting
118   X_array = new int[size];
119   a_array = new int[size];
120   for(i=0; i< size; i++){
121     r = i/filter_wd;
122     c = i%filter_wd;
123     X_array[i] = X[row + r -filter_wd/2][col+ c - filter_wd/2];
124     a_array[i] = wts[r][c];
125   }
126   //sort the arrays using X_array values
127   sorted_ids=Sort(X_array, sorted_ids, size);
128   sorted_ids=Sort(a_array, sorted_ids, size);//sort using ids
129
```

```
130   //find i*
131   for(i=0; i<size; i++){
132     sum1=0; sum2=0;
133     //sum1 1-i
134     for(j=0;j<i;j++){sum1+=a_array[j];}
135     //sum2 i-size
136     for(j=i;j<size;j++){sum2+=a_array[j];}
137     //check
138     if(sum1>=sum2){break;}
139   }
140   return X_array[i-1];
141 }
142
143 int* Sort(int *A, int *sorted_ids, int size){
144   //sort an array
145   int i=0, *temp;
146   temp = new int[size];
147   if(sorted_ids!=NULL){//sort A using ids
148     for(i=0; i<size; i++){temp[i]=A[i];}
149     for(i=0; i<size; i++){A[i] = temp[sorted_ids[i]];}
150     return sorted_ids;
151   }
152   for(i=0; i<size; i++){temp[i]=i;}
153   //sort the array
154   merge_sort(A, temp, 0, size);
155   return temp;
156 }
157
158 void merge_sort(int *A, int *sorted_ids, int start, int end){
159   //merge sort
160   if((end-start)==1){//base case
161     return;}
162   int mid=(start+end)/2;
163   merge_sort(A, sorted_ids, start, mid);
164   merge_sort(A, sorted_ids, mid, end);
165   merge(A, sorted_ids, start, mid, end);
166   return;
167 }
168
169 void merge(int *A,int *sorted_ids,int start,int mid,int end){
170   //merge two sub arrays: A[start:mid], A[mid+1:end]
171   //sort in descending order
172   int i,j,k=0;
173   int *temp, *temp_ids;
174   temp = new int[end-start];
175   temp_ids= new int[end-start];
176
177   for(i=start, j=mid; (i<mid)||(j<end);){
178     if((i<mid)&&(j<end)){//both subarrays not exhauted
179       if(A[i]>A[j]){
180         temp[k]=A[i];
181         temp_ids[k] = sorted_ids[i];
182         i++;
183       }
184       else{
185         temp[k]=A[j];
186         temp_ids[k] = sorted_ids[j];
187         j++;}
188       }
```

```
    else{//one of the sub-array has exhausted
      if(i==mid){
        temp[k] = A[j];
        temp_ids[k] = sorted_ids[j];
        j++;
        }
      else{//j=end
        temp[k]=A[i];
        temp_ids[k] = sorted_ids[i];
        i++;
        }
      }
    k++;
  }
  //copy back to A
  k=0;
  for(i=start;i<end;i++){
    A[i] = temp[k];
    sorted_ids[i] = temp_ids[k];
    k++;
  }
  return;
}

void print_array(int *A, int size){
  int i=0;
  for(;i<size;i++){
  fprintf(stdout,"%d,\t",A[i]);}
  fprintf(stdout,"\n");
  return;
}
```

Listing 6: Bash code for running python code for section 1

```
#!/bin/bash

#section 1
python section1.py ../../img14g.tif ../../img14bl.tif | tee bl.log
python section1.py ../../img14g.tif ../../img14gn.tif | tee gn.log
python section1.py ../../img14g.tif ../../img14sp.tif | tee sp.log
mv ./*.pdf output/sec1/
mv ./*.log output/sec1/
echo 'section 1 done'
```

Listing 7: Bash code for running C-code for section 2

```
#!/bin/bash
#run file for C-code
program=../bin/MedianFiltering
img_dir=../../..
cd ../src/
make all
cd ../run/
$program "$img_dir"/img14gn.tif ./output/img14gn_medianfilter.tif
$program "$img_dir"/img14sp.tif ./output/img14sp_medianfilter.tif
convert output/img14gn_medianfilter.tif output/img14gn_medianfilter.png
convert output/img14sp_medianfilter.tif output/img14sp_medianfilter.png
echo 'section 2 done'
```