ECG 782 Homework #1

Minsung Cho

2021/01/20

1. Getting started
   (a) Determine how you will use LaTeX:
      - I will be using Overleaf.
   (b) Download the "standard" test images from the Gonzalez and Woods website. $\checkmark$
   (c) Download the sample images fro the class website. $\checkmark$
   (d) Indicate the method you have selected for LaTeX use. $\checkmark$
   (e) Generate your report using the article class. $\checkmark$
2. Histogram Equalization
   (a) Write a function hist_eq.m that performs histogram equalization on an intensity image. The function should take as inputs an intensity image and the number of gray level value bins. Create a separate m-file for this function.

```python
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import math

# Setting the input and output files.
IN_FILE = 'sample_images/jetplane.png'
OUT_FILE = 'sample_images/jetplane_64.png'

# # Set the bin size. You can comment out the ones that you don't want.
```

```
# bin = 256
# bin = 128
bin = 64

# Load image, store width and height into constants
pillow_img = Image.open(IN_FILE)
IMG_W, IMG_H = pillow_img.size
MN = IMG_H*IMG_W
img = np.array(pillow_img).flatten()

# Make a histogram
histogram = np.zeros(256, dtype=int)
for i in range(img.size):
    histogram[img[i]] += 1

# Divide the histogram into bin sizes and get the intensity distribution and transformation function.
if bin == 256:
    p256 = np.zeros(bin)
    hist_out = np.zeros(bin)
    for i in range(bin):
        p256[i] = histogram[i]/MN
        if i == 0:
            hist_out[i] = bin * p256[i]
        else:
            hist_out[i] = bin * p256.sum()
    hist_out = np.round(hist_out, 0)
elif bin == 128:
    n128 = np.zeros(bin, dtype=int)
    p128 = np.zeros(bin)
    s128 = np.zeros(bin)
    for i in range(bin):
        for j in range(int(i*256/bin), int((i+1)*256/bin)):
            n128[i] += histogram[j]
        p128[i] = n128[i]/MN
        if i == 0:
            s128[i] = bin * p128[i]
        else:
            s128[i] = bin * p128.sum()
    s128 = np.round(s128, 0)
elif bin == 64:
    n64 = np.zeros(bin, dtype=int)
    p64 = np.zeros(bin)
```

```
    s64 = np.zeros(bin)
    for i in range(bin):
        for j in range(int(i*256/bin), int((i+1)*256/bin)):
            n64[i] += histogram[j]
        p64[i] = n64[i]/MN
        if i == 0:
            s64[i] = bin * p64[i]
        else:
            s64[i] = bin * p64.sum()
    s64 = np.round(s64, 0)

# Make the new histogram for bin size 128 and 64
if bin == 128:
    hist_out = np.zeros(256, dtype = int)
    for i in range(256):
        hist_out[i]=s128[math.floor(i/2)]
elif bin == 64:
    hist_out = np.zeros(256, dtype = int)
    for i in range(256):
        hist_out[i]=s64[math.floor(i/4)]

# Make the new output image
new_img = np.zeros(img.size, dtype=int)
for i in range(img.size):
    new_img[i] = hist_out[img[i]]

# Save the image
output_file = Image.fromarray(np.uint8(new_img.reshape((IMG_H, IMG_W))))
output_file.save(OUT_FILE)

# Display the old (blue) and new (orange) histograms next to eachother
x_axis = np.arange(256)
fig = plt.figure()
fig.add_subplot(2, 2, 1)
a = plt.bar(x_axis, histogram)
a = plt.title('histogram before')
fig.add_subplot(2, 2, 2)
b = plt.bar(x_axis, hist_out, color="orange")
b = plt.title('histogram with bin size 64')
fig.add_subplot(2, 2, 3)
orig_img = mpimg.imread(IN_FILE)
c = plt.imshow(orig_img, cmap='gray')
```

```
c = plt.title('Original Image')
fig.add_subplot(2, 2, 4)
eq_img = mpimg.imread(OUT_FILE)
d = plt.imshow(eq_img, cmap='gray')
d = plt.title('Equalized image with bin size 64')
plt.show()

```

Listing 1: Histogram equalization code

(b) Perform histogram equalization on the jetplane image using 256, 128, and 64 bins. Compare the original image and the histogram equalized images by plotting the corresponding histograms and images side-by-side in a 2 × 2 subplot matrix.

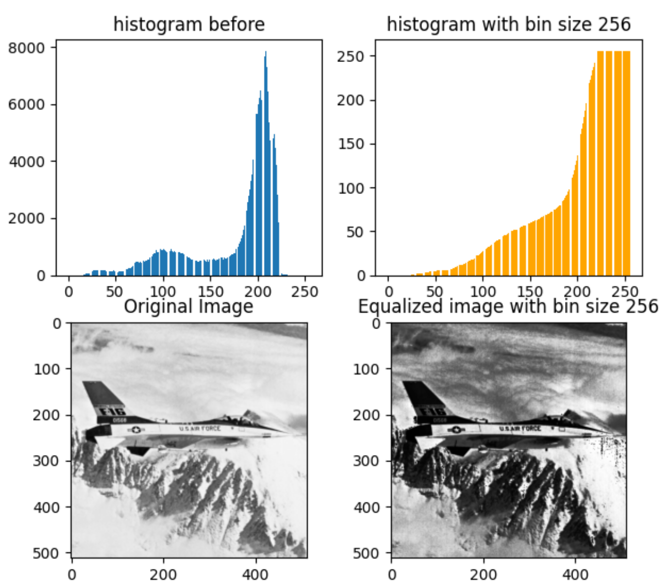

Figure 1: Bin sizes 256 histogram equalization

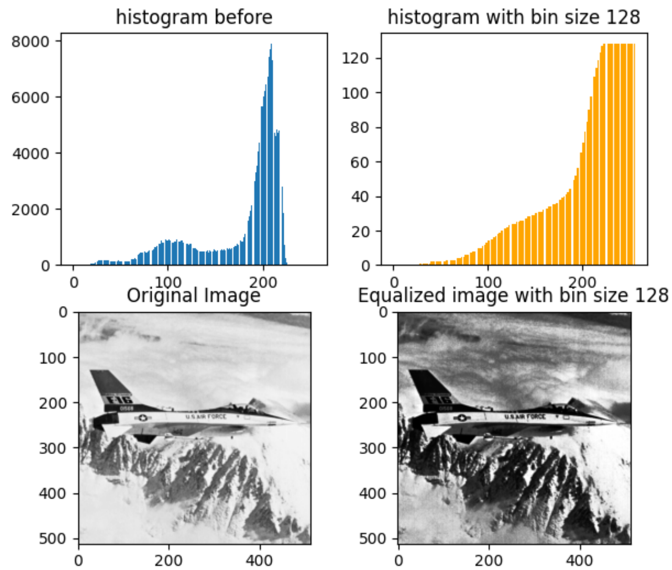

Figure 2: Bin sizes 128 histogram equalization

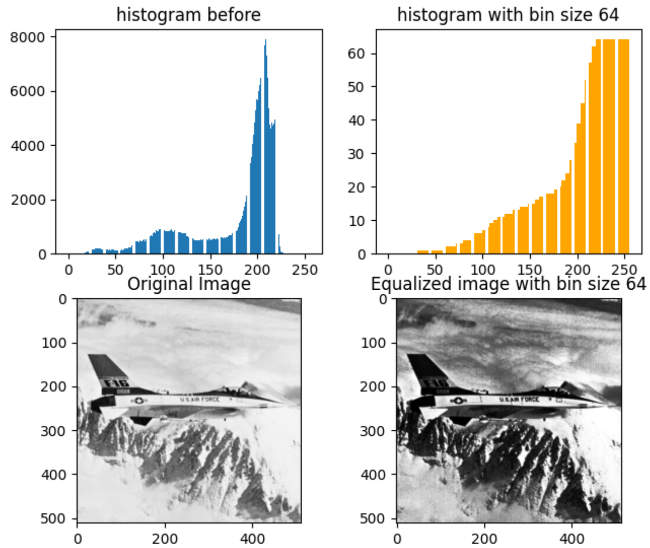

Figure 3: Bin sizes 64 histogram equalization

(c) Perform the equalization in 32 x 32 blocks. Display the output image. You will find blockproc.m useful.

```

  from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import math

# Setting the input and output files.
IN_FILE = 'hw1/sample_images/jetplane.png'
OUT_FILE = 'hw1/sample_images/jetplane_32by32.png'

# Load image, store width and height into constants
pillow_img = Image.open(IN_FILE)
IMG_W, IMG_H = pillow_img.size
img = np.array(pillow_img)
new_img = np.zeros((IMG_H,IMG_W), dtype=int)

# Make a histogram for 32 by 32 blocks
for j in range(16):
```

```
    for k in range(16):
        # filling up 32x32 histogram
        histogram = np.zeros(256, dtype=int)
        for a in range(32):
            for b in range(32):
                histogram[img[a+k*32,b+j*32]] += 1
        print('histogram:',histogram)
        p256 = np.zeros(256)
        hist_out = np.zeros(256)
        for i in range(256):
            p256[i] = histogram[i]/1024
            if i == 0:
                hist_out[i] = 256 * p256[i]
            else:
                hist_out[i] = 256 * p256.sum()
        hist_out = np.round(hist_out, 0)
        print('pdf: ',p256)
        print('hist_out:',hist_out)
        for a in range(32):
            for b in range(32):
                new_img[a+k*32,b+j*32] = hist_out[img[a+k*32,b+j*32]]

output_file = Image.fromarray(np.uint8(new_img.reshape((IMG_H, IMG_W))))
output_file.save(OUT_FILE)
```

Listing 2: 32 by 32 equalization code

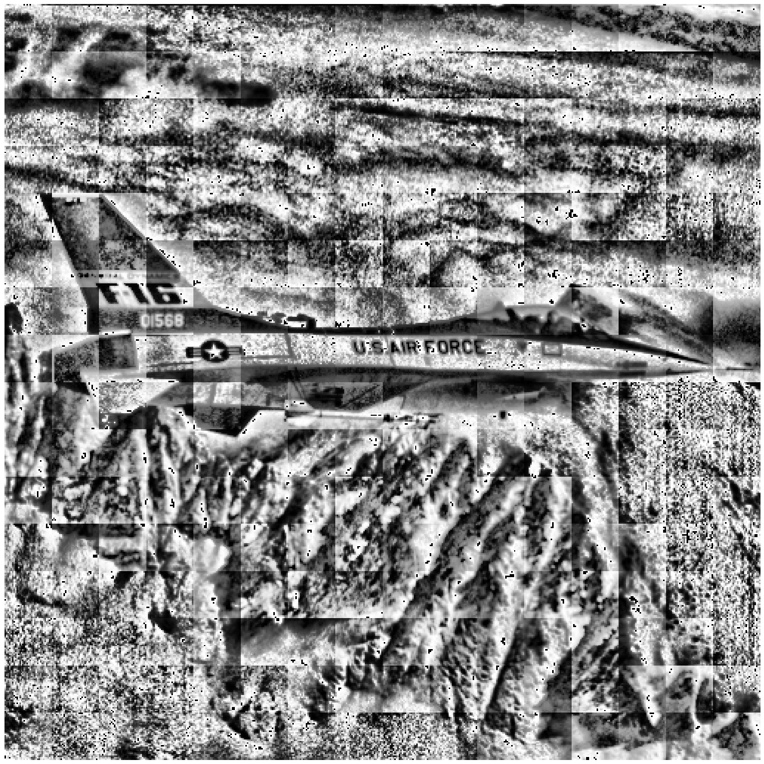

Figure 4: 32x32 equalization image

3. Basic Morphology

```
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

# Make histogram from image
def make_histogram(img):
    histogram = np.zeros(np.amax(img))
    for i in range(img.size):
        histogram[img[i]] += 1
    return histogram

# Make a box
def make_box(i, j, box):
    box[i:i+60, j:j+5, 0] = 255
    box[i:i+60, j+45:j+50, 0] = 255
    box[i:i+5, j:j+50, 0] = 255
```

```python
    box[i+60:i+65, j:j+50, 0] = 255
    return box

# Setting the input and output files.
IN_FILE = 'hw1/sample_images/SJEarthquakesteampic.jpg'
THRESHOLD_OUT_FILE = 'hw1/sample_images/SJEarthquakesteampic_threshold.jpg'
EROSION_OUT_FILE = 'hw1/sample_images/SJEarthquakesteampic_erosion.jpg'
DILATION_OUT_FILE = 'hw1/sample_images/SJEarthquakesteampic_dilation.jpg'
MORPHOLOGY_OUT_FILE = 'hw1/sample_images/SJEarthquakesteampic_morphology.jpg'
BOXED_OUT_FILE = 'hw1/sample_images/SJEarthquakesteampic_boxed.jpg'

# Loading the image and converting to array
pillow_img = Image.open(IN_FILE)
IMG_W, IMG_H = pillow_img.size
hsv_img = np.array(pillow_img.convert('HSV'))
img = np.array(pillow_img)

# Thresholding with the rule of Hue > 30 is skin
# I got this by looking at the results.
threshold_img = np.zeros((IMG_H, IMG_W), dtype=int)
for i in range(IMG_H):
    for j in range(IMG_W):
        if hsv_img[i, j, 0] > 30:
            threshold_img[i, j] = 255
        else:
            threshold_img[i, j] = 0

# Saving the threshold image as .jpg
output_file = Image.fromarray(np.uint8(threshold_img.reshape((IMG_H, IMG_W))))
output_file.save(THRESHOLD_OUT_FILE)

# Morphology method: erosion then dilation.
# Setting the borders as is. No morphing performed with 5x5 on the borders for simplicity.
erosion_img = np.zeros((IMG_H, IMG_W), dtype=int)
erosion_img[0:1,:] = threshold_img[0:1,:]
erosion_img[IMG_H-2:IMG_H-1:] = threshold_img[IMG_H-2:IMG_H-1:]
erosion_img[:,0:1] = threshold_img[:,0:1]
```

```python
erosion_img[:,IMG_W-2:IMG_W-1] = threshold_img[:,IMG_W-2:
    IMG_W-1]
# Performing erosion
for i in range(2, IMG_H-2):
    for j in range(2, IMG_W-2):
        erase_flag = 0
        for k in range(-2,3):
            for l in range(-2,3):
                if threshold_img[i-k, j-l] != 0 :
    erase_flag = 1
        if erase_flag == 1 : erosion_img[i, j] = 255
        else: erosion_img[i, j] = 0

output_file = Image.fromarray(np.uint8(erosion_img.
    reshape((IMG_H, IMG_W))))
output_file.save(EROSION_OUT_FILE)

#performing dilation
dilation_img = np.zeros((IMG_H, IMG_W), dtype=int)
dilation_img[0,:] = erosion_img[0,:]
dilation_img[IMG_H-1:] = erosion_img[IMG_H-1:]
dilation_img[:,0] = erosion_img[:,0]
dilation_img[:,IMG_W-1] = erosion_img[:,IMG_W-1]
for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
        create_flag = 0
        for k in range(-1,2):
            for l in range(-1,2):
                if erosion_img[i-k, j-l] == 0 :
    create_flag = 1
        if create_flag == 1 : dilation_img[i, j] = 0
        else: dilation_img[i, j] = 255

output_file = Image.fromarray(np.uint8(dilation_img.
    reshape((IMG_H, IMG_W))))
output_file.save(DILATION_OUT_FILE)

# Making the boxes using the function above.
make_box(60, 60, img)
make_box(60, 150, img)
make_box(60, 230, img)
make_box(40, 310, img)
make_box(40, 400, img)
make_box(40, 480, img)
make_box(180, 70, img)
```

```
make_box(180, 160, img)
make_box(180, 260, img)
make_box(180, 360, img)
make_box(180, 470, img)

output_file = Image.fromarray(img)
output_file.save(BOXED_OUT_FILE)

```

Listing 3: Morphology code

(a) Threshold the image SJEarthquakesteampic.jpg to detect faces. Be sure to describe how you obtained your threshold. You may find this is easier in another colorspace such as HSV.

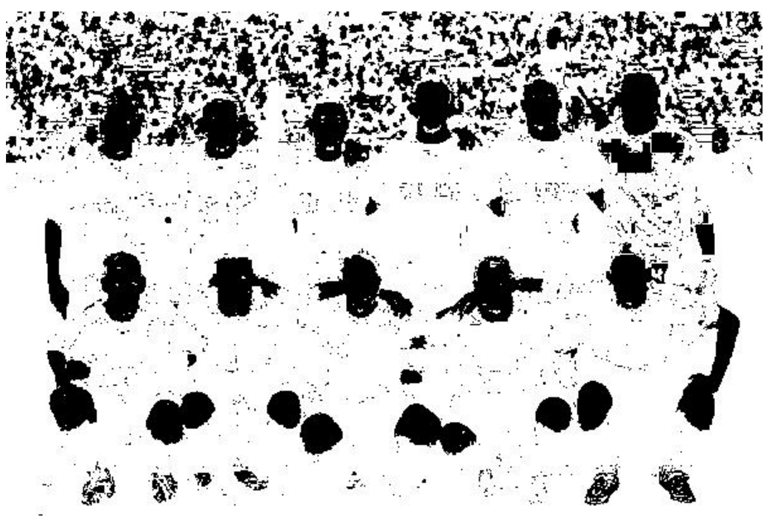

Figure 5: Thresholded image

I first converted the image into HSV color code. Then the hue of over 30 became white and the rest became black.

(b) Use morphological operations to clean the image. Count the number of players in the cleaned threshold image.

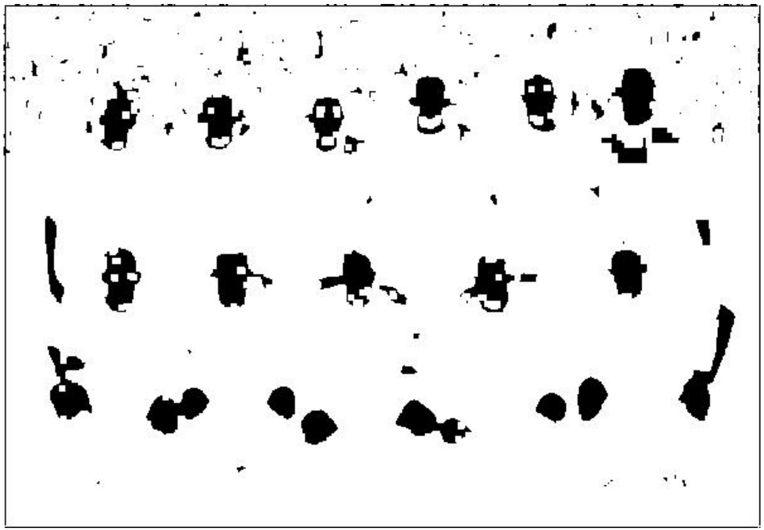

Figure 6: Erosion by 5x5 box image

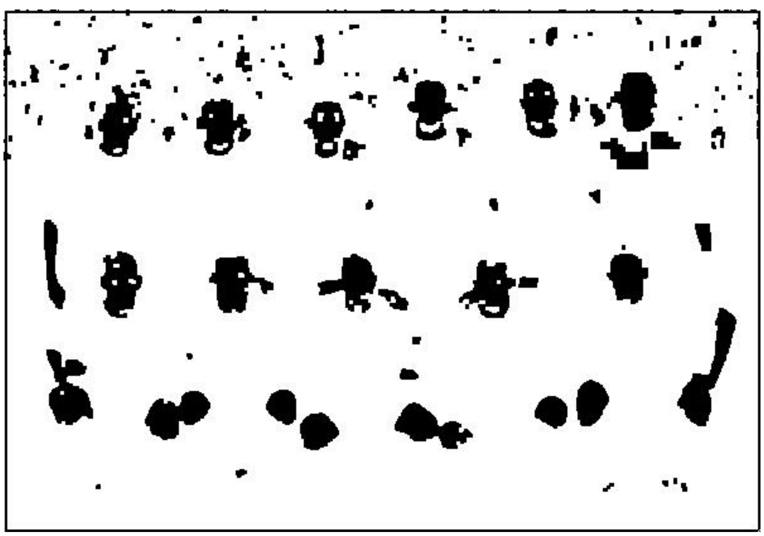

Figure 7: Dilation by 3x3 box image

I used erosion then dilation to get rid of most of the speckles. The reason for using different box size is to not make the surviving speckles bigger. The face count is 11.

(c) Create an output image that has a bounding box around each face. Use regionprops.m. In your report, create an output figure with three images in a row. (a) is the face threshold image, (b) morphologically cleaned image, and (c) the color image with bounding box around face areas.

Figure 8: Boxed output color image

(d) Repeat for barcelona-team.jpg. Explain the differences you found.

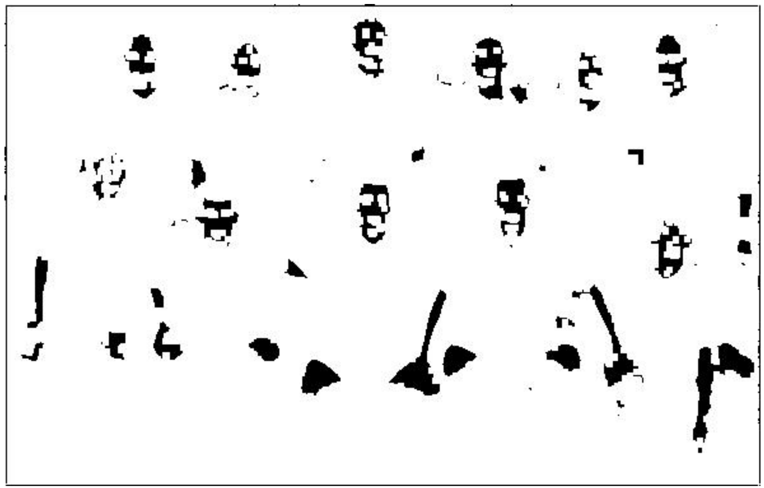

Figure 9: Barcelona team's morphology image

This team had a slightly different result than the other team. The team members are whiter than the other team, giving it more of a hue. That led to their skins being overly morphed during the noise reduction stages. However, because there aren't many crowds in the back, the image is cleaner. You can see the eye holes and mouth holes clearly in Barcelona team.

4. Filtering

```
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from numpy import random
import math

# Setting the input and output files.
IN_FILE = 'hw1/sample_images/DSCN0479-001.jfif'
JPG_OUT_FILE = 'hw1/sample_images/DSCN0479-001.jpg'
NOISY_OUT_FILE = 'hw1/sample_images/DSCN0479-001_noisy.
    jpg'
SEASONED_OUT_FILE = 'hw1/sample_images/DSCN0479-001
    _seasoned.jpg'
THREEBOX_OUT_FILE = 'hw1/sample_images/DSCN0479-001
    _threebox.jpg'
```

```
SEVENBOX_OUT_FILE = 'hw1/sample_images/DSCN0479-001_sevenbox.jpg'
MEDIAN_OUT_FILE = 'hw1/sample_images/DSCN0479-001_median.jpg'


# Load image, store width and height into constants
pillow_img = Image.open(IN_FILE)
IMG_W, IMG_H = pillow_img.size
img = np.array(pillow_img)

# Make Gaussian noise with the variant of 0.005
noise = np.random.randint(255,size=(IMG_H, IMG_W, 3))
noisy_img = np.array(img + math.sqrt(0.005)*noise, dtype = int)

# # Save the images to compare
output_file = Image.fromarray(np.uint8(img))
output_file.save(JPG_OUT_FILE)
output_file = Image.fromarray(np.uint8(noisy_img))
output_file.save(NOISY_OUT_FILE)

# Make salt and pepper noise
seasoned_img = img
for i in range(1, IMG_H):
    for j in range(1, IMG_W):
        if np.random.randint(20) == 1:
            seasoned_img[i, j] = [255,255,255]
        elif np.random.randint(20) == 2:
            seasoned_img[i, j] = [0,0,0]
output_file = Image.fromarray(np.uint8(seasoned_img))
output_file.save(SEASONED_OUT_FILE)

# Smoothing with a 3x3 box filter
threebythree_box_img = np.zeros((IMG_H, IMG_W, 3), dtype=int)
threebythree_box_img[0,:] = noisy_img[0,:]
threebythree_box_img[IMG_H-1:] = noisy_img[IMG_H-1:]
threebythree_box_img[:,0] = noisy_img[:,0]
threebythree_box_img[:,IMG_W-1] = noisy_img[:,IMG_W-1]
for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
        for k in range(3):
            sum = 0
            for l in range(-1,2):
```

```
                for m in range(-1,2):
                    sum += noisy_img[i+l, j+m, k]
            threebythree_box_img[i, j, k] = sum / 9
output_file = Image.fromarray(np.uint8(threebythree_box_img))
output_file.save(THREEBOX_OUT_FILE)

# Smoothing with a 7x7 box filter
sevenbyseven_box_img = np.zeros((IMG_H, IMG_W, 3), dtype=int)
sevenbyseven_box_img[0:2,:] = noisy_img[0:2,:]
sevenbyseven_box_img[IMG_H-3:IMG_H-1:] = noisy_img[IMG_H-3:IMG_H-1:]
sevenbyseven_box_img[:,0:2] = noisy_img[:,0:2]
sevenbyseven_box_img[:,IMG_W-3:IMG_W-1] = noisy_img[:,IMG_W-3:IMG_W-1]
for i in range(3, IMG_H-3):
    for j in range(3, IMG_W-3):
        for k in range(3):
            sum = 0
            for l in range(-3,4):
                for m in range(-3,4):
                    sum += noisy_img[i-l, j-m, k]
            sevenbyseven_box_img[i, j, k] = sum/49
        sum = 0
        sum += noisy_img[i, j]

output_file = Image.fromarray(np.uint8(sevenbyseven_box_img))
output_file.save(SEVENBOX_OUT_FILE)

# Smoothing with a median filter
median_img = np.zeros((IMG_H, IMG_W, 3), dtype=int)
median_img[0,:] = noisy_img[0,:]
median_img[IMG_H-1:] = noisy_img[IMG_H-1:]
median_img[:,0] = noisy_img[:,0]
median_img[:,IMG_W-1] = noisy_img[:,IMG_W-1]
for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
        for k in range(3):
            median_array = np.zeros(9)
            count = 0
            for l in range(-1,2):
                for m in range(-1,2):
                    median_array[count] = noisy_img[i-l,
```

```
   j-m, k]
                        count += 1
                median_array = sorted(median_array)
                median_img[i, j, k] = median_array[4]
output_file = Image.fromarray(np.uint8(median_img))
output_file.save(MEDIAN_OUT_FILE)

# Calculate the MSE for 3x3
MSE = 0
img1 = np.array(noisy_img).flatten()
img2 = np.array(threebythree_box_img).flatten()
for i in range(noisy_img.size):
    MSE += (img1[i] - img2[i])**2/(IMG_H*IMG_W*3)
print('MSE of 3x3:',MSE)
PSNR = 20*math.log(255/math.sqrt(MSE),10)
print('PSNR of 3x3:',PSNR)

# Calculate the MSE for 7x7
MSE = 0
img3 = np.array(sevenbyseven_box_img).flatten()
for i in range(noisy_img.size):
    MSE += (img1[i] - img3[i])**2/(IMG_H*IMG_W*3)
print('MSE of 7x7:',MSE)
PSNR = 20*math.log(255/math.sqrt(MSE),10)
print('PSNR of 7x7:',PSNR)

# Calculate the MSE for median
MSE = 0
img4 = np.array(median_img).flatten()
for i in range(noisy_img.size):
    MSE += (img1[i] - img4[i])**2/(IMG_H*IMG_W*3)
print('MSE of median:',MSE)
PSNR = 20*math.log(255/math.sqrt(MSE),10)
print('PSNR of median:',PSNR)
```

Listing 4: Histogram equalization code

(a) Consider image DSCN0479-001.JPG as a perfect image. Add white Gaussian noise with variance 0.005. Smooth with a 3×3 and 7×7 box filter and a median filter. Compute the mean squared error (MSE)

$$MSE = \frac{1}{MN}\sum_m \sum_n (I_1(m,n) - I_2(m,n))^2$$

and the peak signal-to-noise ratio (PSNR)

$$PSNR = 20 \times log_{10}(255/\sqrt{MSE})$$

for the noise reduced images. Compile results using a LaTeX Table. Which filter has the best results based on the error measures? How do the results compare visually?

Figure 10: Gaussian noise image with variance 0.005

Figure 11: 3x3 box filtering image

Figure 12: 7x7 box filtering image

Figure 13: median filtering image

	3x3	7x7	median
MSE	116.93	575.58	98.68
PSNR	27.45	20.53	28.19

(b) Repeat (a) with salt and pepper noise with noise density 0.05. Compile results using a LaTeX Table.

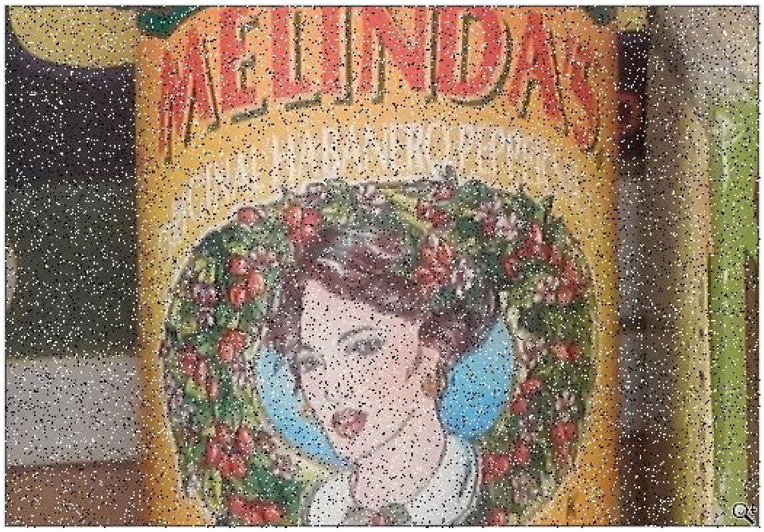

Figure 14: Salt Pepper noise density 0.05 image

Figure 15: 3x3 box filtering image

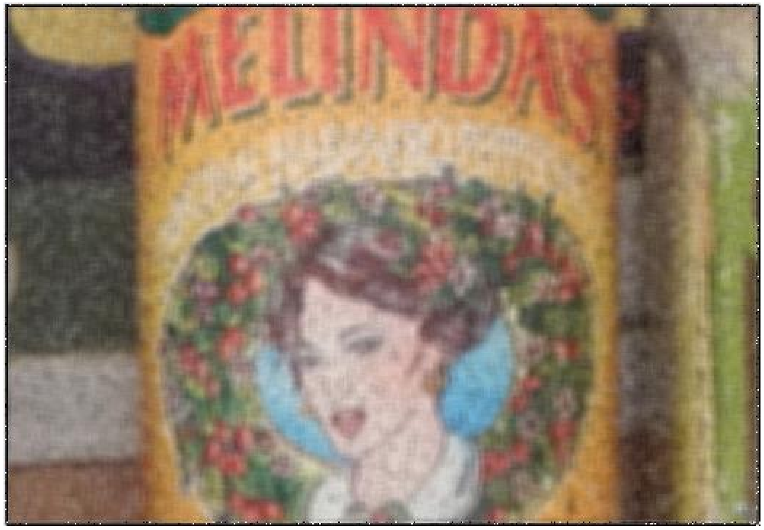

Figure 16: 7x7 box filtering image

Figure 17: median filtering image

	3x3	7x7	median
MSE	1695.65	2230.25	1869.64
PSNR	15.84	14.63	15.41

(c) Do the filtering again but this time on a real noisy image DSCN0482-001.JPG obtained at higher ISO. Compare the results visually only this time. Which filter works best for "real" noise? How much time does each type of filter require (use tick.m and toc.m)?

Figure 18: Real noisy image

Figure 19: 3x3 box filter image

Figure 20: 7x7 box filter image

Figure 21: median filter image

	3x3	7x7	median
MSE	76.96	419.33	66.95
PSNR	29.27	21.91	29.87

Numerically and visually, median filter worked the best. For 3x3, it took 3.5 seconds. For 7x7 it took 12.3 seconds. For median, it took 3.6 seconds.