# Srinidhi-HW2

November 8, 2021

# 1 Srinidhi Bharadwaj Kalgundi Srinivas

# 2 A59010584

Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind. By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

### 2.0.1 Problem 1: Adaptive Histogram Equalization

[1]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```
def Adaptive_HE(image, win_size):
    pad_size = win_size//2
    out_image = np.zeros((image.shape[0], image.shape[1]), dtype=np.uint8)
    image = np.pad(image, (pad_size, pad_size), 'symmetric')
    rows = image.shape[0]
    cols = image.shape[1]
    i = 0
    j = 0
    for x in range(pad_size, rows-pad_size):
        for y in range(pad_size, cols-pad_size):
            rank = 0
            contextual_region = image[i:i+(pad_size*2)+1, j:j+(pad_size*2)+1]
            #print(contextual_region.shape)
            binary_mask = (image[x][y] > contextual_region)
            int_mask = binary_mask.astype(int)
            rank = np.sum(int_mask)
            out_image[i][j] = rank * 255/(win_size*win_size)
            j = j+1
        i = i+1
        j = 0
    return out_image
```

[3]:
```
A = cv2.imread("beach.png")
A = cv2.cvtColor(A, cv2.COLOR_BGR2RGB) #OpenCV reads in BGR order hence the␣
↪conversion
print("Input image is a color image of dimensions: ",A.shape)

out_image_33 = np.zeros((A.shape[0], A.shape[1], 3), dtype=np.uint8)
out_image_33[:,:,0] = Adaptive_HE(A[:,:,0], 33)
out_image_33[:,:,1] = Adaptive_HE(A[:,:,1], 33)
out_image_33[:,:,2] = Adaptive_HE(A[:,:,2], 33)

out_image_65 = np.zeros((A.shape[0], A.shape[1], 3), dtype=np.uint8)
out_image_65[:,:,0] = Adaptive_HE(A[:,:,0], 65)
out_image_65[:,:,1] = Adaptive_HE(A[:,:,1], 65)
out_image_65[:,:,2] = Adaptive_HE(A[:,:,2], 65)

out_image_129 = np.zeros((A.shape[0], A.shape[1], 3), dtype=np.uint8)
out_image_129[:,:,0] = Adaptive_HE(A[:,:,0], 129)
out_image_129[:,:,1] = Adaptive_HE(A[:,:,1], 129)
out_image_129[:,:,2] = Adaptive_HE(A[:,:,2], 129)
```

```
Input image is a color image of dimensions:  (600, 800, 3)
```

[4]:
```
#Histogram equalization for color image
#https://www.opencv-srf.com/2018/02/histogram-equalization.html - Document I␣
↪read for understanding HE for color images

A = cv2.imread("beach.png")
img = cv2.cvtColor(A, cv2.COLOR_BGR2YUV)
# Equalize the histogram of the Y channel as it contains all intensity values
img[:,:,0] = cv2.equalizeHist(img[:,:,0])
image_HE = cv2.cvtColor(img, cv2.COLOR_YUV2BGR)
```

[5]:
```
fig, axs = plt.subplots(3,2, figsize=(20, 20))
ax1= fig.add_subplot(3,2,1)
ax1.title.set_text("Original Image")
ax1.axis('off')
ax1.imshow(A)
axs[0, 0].axis('off')

ax1= fig.add_subplot(3,2,2)
ax1.title.set_text("AHE with window size 33")
ax1.axis('off')
ax1.imshow(out_image_33)
axs[0, 1].axis('off')

ax1= fig.add_subplot(3,2,3)
ax1.title.set_text("AHE with window size 65")
ax1.axis('off')
```

```
ax1.imshow(out_image_65)
axs[1, 0].axis('off')

ax1= fig.add_subplot(3,2,4)
ax1.title.set_text("AHE with window size 129")
ax1.axis('off')
ax1.imshow(out_image_129)
axs[1, 1].axis('off')

ax1= fig.add_subplot(3,2,5)
ax1.title.set_text("Histogram Equalized image using OpenCV")
ax1.axis('off')
ax1.imshow(image_HE)
axs[2, 0].axis('off')
axs[2, 1].axis('off')
```

[5]: (0.0, 1.0, 0.0, 1.0)

Original Image

AHE with window size 33

AHE with window size 65

AHE with window size 129

Histogram Equalized image using OpenCV

Solution for problem 1:

i) Original image has unbalanced histogram where background or the sky is completely dark and the boys in the window are completely darkened out. Adaptive Histogram Equalization tries to spread out the histogram based on the local neighborhood information resulting in more brighter spots becoming slightly darker and darker spots becoming slightly brighter as a result of which faces of the people in the darker part of the image are better visible. Histogram equalization performs histogram spread based on global information that results in darker part of the images remain on the darker side of the histogram. However, overall histogram is more spread out.

ii) For the given beach.png image, AHE with a window size of 129 works the best as it results in faces becoming more visible. No, it is not true for any image, it depends on the type of task we are trying to achieve and the histogram of the image.

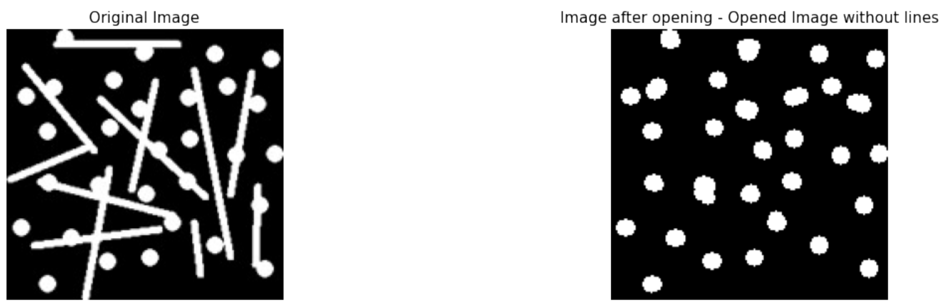
### 2.0.2 Problem 2: Binary Morphology

[6]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import skimage.morphology as sm
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

**Problem 2 - i)**

[7]:
```
circles_lines = cv2.imread("circles_lines.jpg", 0)
kernelSize = (9,9)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, kernelSize)
opened_image = cv2.morphologyEx(circles_lines, cv2.MORPH_OPEN, kernel)
(thresh, opened_image) = cv2.threshold(opened_image, 10, 255, cv2.THRESH_BINARY)

fig, axs = plt.subplots(1,2, figsize=(20, 5))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("Original Image")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(circles_lines, cmap='gray')
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("Image after opening - Opened Image without lines")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(opened_image, cmap='gray')
axs[1].axis('off')
```

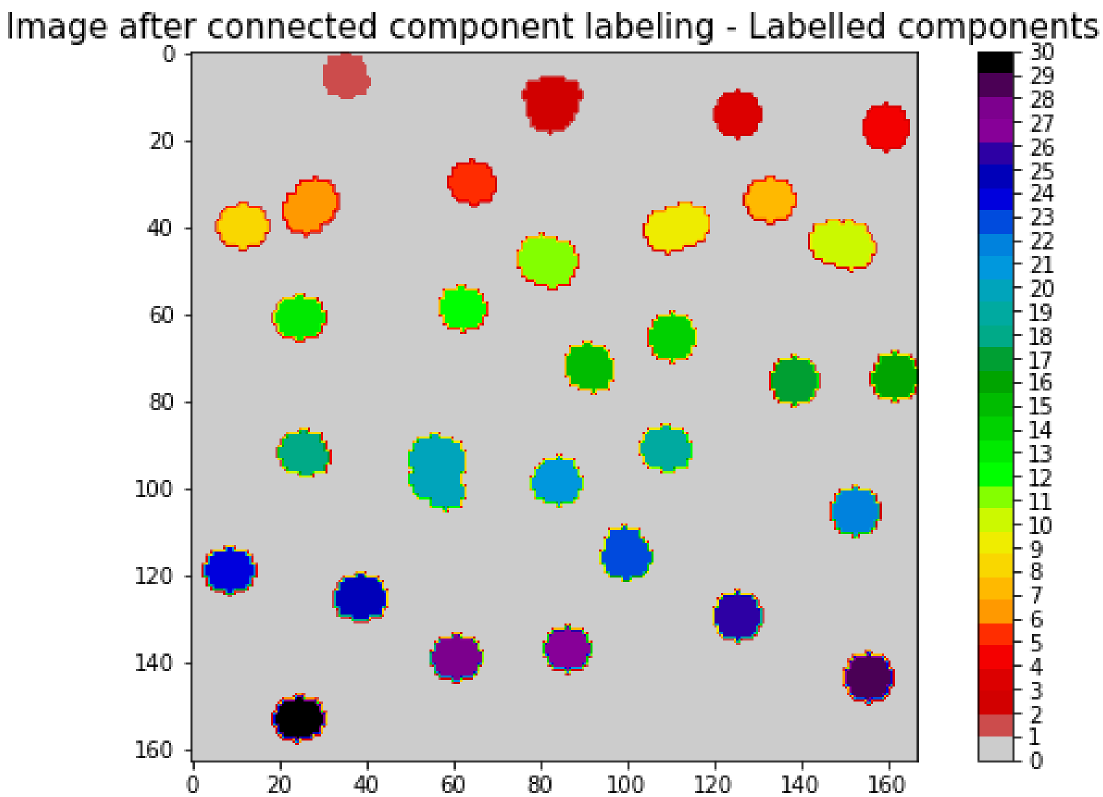
[7]:
```
(0.0, 1.0, 0.0, 1.0)
```




[8]:
```
def discrete_cmap(N, base_cmap=None, linspaceFlip=False):
    base = plt.cm.get_cmap(base_cmap)
    color_list = base(np.linspace(1, 0, N))
```

```
    if linspaceFlip == True:
        color_list = base(np.linspace(0, 1, N))
    cmap_name = base.name + str(N)
    return base.from_list(cmap_name, color_list, N)
```

[9]:
```
num_labels, labels_im = cv2.connectedComponents(opened_image)
N = num_labels
plt.figure(figsize=(10,6))
plt.imshow(labels_im, cmap=discrete_cmap(N, 'nipy_spectral'))
plt.colorbar(ticks=range(N))
plt.title('Image after connected component labeling - Labelled components',␣
 ↪size=15)

plt.show()
```




[10]:
```
area_dictionary = {}
for i in range(1,num_labels):
    area_dictionary[i] = np.size(np.where(labels_im==i)[0])
print(area_dictionary)
```

```
{1: 114, 2: 146, 3: 96, 4: 96, 5: 88, 6: 126, 7: 96, 8: 96, 9: 128, 10: 133, 11:
130, 12: 88, 13: 96, 14: 96, 15: 100, 16: 94, 17: 96, 18: 97, 19: 99, 20: 182,
```

```
21: 100, 22: 96, 23: 110, 24: 96, 25: 102, 26: 96, 27: 85, 28: 96, 29: 96, 30:
94}
```

[11]:
```
def calc_centroid(num_labels, labels_im):
    centroid_x = {}
    centroid_y={}
    for i in range(1, num_labels):
        moment_0_0 = np.sum(np.size(np.where(labels_im==i)))
        moment_0_1 = np.sum(np.where(labels_im==i)[1])
        moment_1_0 = np.sum(np.where(labels_im==i)[0])
        centroid_x[i] = 2 * round((moment_1_0/moment_0_0), 3)
        centroid_y[i] = 2 * round(moment_0_1/moment_0_0, 3)

    centroid_cooridinates = list(zip(centroid_x.values(), centroid_y.values()))
    return centroid_cooridinates
```

[12]:
```
centroid_cooridinates = calc_centroid(num_labels, labels_im)
area_centroid_coordinates = list(zip(area_dictionary.
 ↪values(),centroid_cooridinates))
area_centroid_table = {}
print("Table of area and centroids")
print("Note: Format is (area, (centroid_rows, centroid_cols))\n")
for i in range(num_labels-1):
    area_centroid_table[i+1] = area_centroid_coordinates[i]
    print("Area and centroid of {0} component is : {1}".format(i+1,␣
 ↪area_centroid_table[i+1]))
```

```
Table of area and centroids
Note: Format is (area, (centroid_rows, centroid_cols))

Area and centroid of 1 component is : (114, (5.658, 35.264))
Area and centroid of 2 component is : (146, (11.828, 82.404))
Area and centroid of 3 component is : (96, (14.5, 125.0))
Area and centroid of 4 component is : (96, (17.5, 159.0))
Area and centroid of 5 component is : (88, (30.124, 64.114))
Area and centroid of 6 component is : (126, (35.492, 27.008))
Area and centroid of 7 component is : (96, (34.0, 132.5))
Area and centroid of 8 component is : (96, (40.0, 11.5))
Area and centroid of 9 component is : (128, (40.296, 111.078))
Area and centroid of 10 component is : (133, (44.12, 149.038))
Area and centroid of 11 component is : (130, (48.038, 81.392))
Area and centroid of 12 component is : (88, (58.876, 61.886))
Area and centroid of 13 component is : (96, (61.0, 24.5))
Area and centroid of 14 component is : (96, (65.5, 110.0))
Area and centroid of 15 component is : (100, (72.4, 91.03))
Area and centroid of 16 component is : (94, (74.554, 161.096))
Area and centroid of 17 component is : (96, (75.5, 138.0))
Area and centroid of 18 component is : (97, (92.154, 25.516))
```

```
Area and centroid of 19 component is : (99, (91.112, 108.616))
Area and centroid of 20 component is : (182, (96.204, 56.198))
Area and centroid of 21 component is : (100, (98.6, 83.58))
Area and centroid of 22 component is : (96, (105.5, 152.0))
Area and centroid of 23 component is : (110, (115.29, 99.39))
Area and centroid of 24 component is : (96, (119.0, 8.5))
Area and centroid of 25 component is : (102, (125.216, 38.5))
Area and centroid of 26 component is : (96, (129.5, 125.0))
Area and centroid of 27 component is : (85, (137.0, 86.0))
Area and centroid of 28 component is : (96, (139.0, 60.5))
Area and centroid of 29 component is : (96, (143.5, 155.0))
Area and centroid of 30 component is : (94, (152.904, 24.458))
```

### 2.1 Structuring element used for this morphological operation is:

[13]:
```
print(kernel)
print("Type of kernerl used is: "+str(kernel.dtype) +" and the shape is an␣
↪ellipse")
print("Size of kernerl used is {0} and shape of the kernel is {1}".
↪format(kernel.size, kernel.shape))
```

```
[[0 0 0 0 1 0 0 0 0]
 [0 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 0]
 [1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1]
 [0 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 0]
 [0 0 0 0 1 0 0 0 0]]
Type of kernerl used is: uint8 and the shape is an ellipse
Size of kernerl used is 81 and shape of the kernel is (9, 9)
```

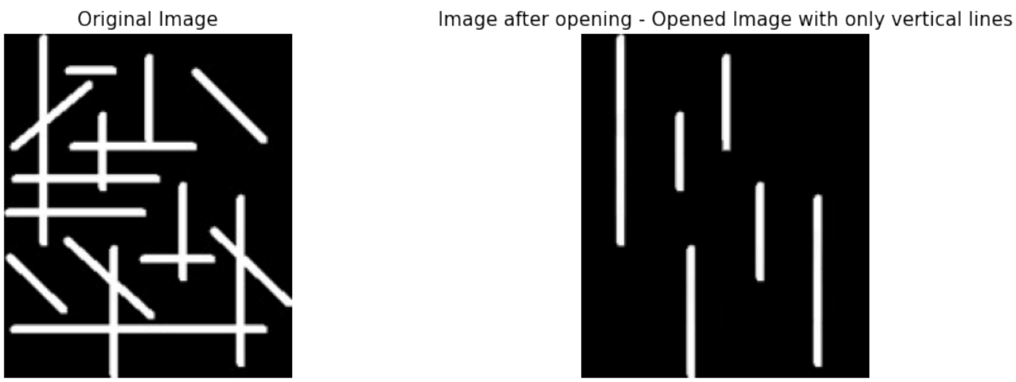
**Problem 2: ii)**

[14]:
```
lines = cv2.imread("lines.jpg", 0)
```

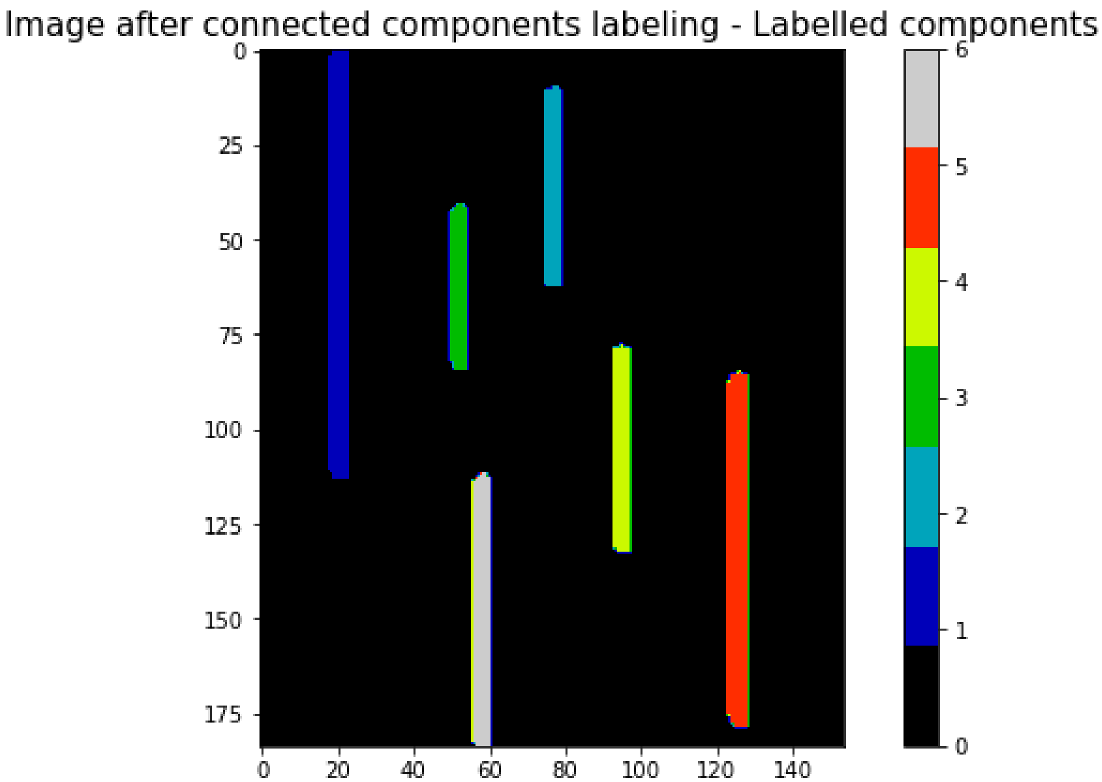
[15]:
```
kernelSize = (1,15)

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernelSize)
opened_image = cv2.morphologyEx(lines, cv2.MORPH_OPEN, kernel)
(thresh, vertical_lines) = cv2.threshold(opened_image, 10, 255, cv2.
↪THRESH_BINARY)

fig, axs = plt.subplots(1,2, figsize=(15, 5))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("Original Image")
ax1.title.set_size(15)
```

```
ax1.axis('off')
ax1.imshow(lines, cmap='gray')
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("Image after opening - Opened Image with only vertical␣
↪lines")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(opened_image, cmap='gray')
axs[1].axis('off')
```

[15]: (0.0, 1.0, 0.0, 1.0)




[16]:
```
num_labels_lines, labels_im_lines = cv2.connectedComponents(vertical_lines)
plt.figure(figsize=(10,6))
plt.imshow(labels_im_lines, cmap=discrete_cmap(num_labels_lines,␣
↪'nipy_spectral', linspaceFlip=True))
plt.colorbar(ticks=range(num_labels_lines))
plt.title('Image after connected components labeling - Labelled components',␣
↪size=15)
plt.show()
```




[17]:
```
length_dict = {}
for i in range(1,num_labels_lines):
    #list_co.append(np.where(labels_im_lines==i)[0].max())
    max_val = np.where(labels_im_lines==i)[0].max()
    min_val = np.where(labels_im_lines==i)[0].min()
    length_dict[i] = max_val - min_val
print(length_dict)
```

```
{1: 113, 2: 52, 3: 43, 4: 54, 5: 93, 6: 71}
```

[18]:
```
centroid_coordinates = calc_centroid(num_labels_lines, labels_im_lines)

length_centroid_coordinates = list(zip(length_dict.
 ↪values(),centroid_coordinates))
length_centroid_table = {}
print("Table of length and centroids")
print("Note: Format is (length, (centroid_rows, centroid_cols))\n")
for i in range(num_labels_lines-1):
    length_centroid_table[i+1] = length_centroid_coordinates[i]
    print("Length and centroid of {0} component is : {1}".format(i+1,␣
 ↪length_centroid_table[i+1]))
```

```
Table of length and centroids
Note: Format is (length, (centroid_rows, centroid_cols))

Length and centroid of 1 component is : (113, (56.5, 20.514))
Length and centroid of 2 component is : (52, (36.298, 77.004))
Length and centroid of 3 component is : (43, (62.7, 52.032))
Length and centroid of 4 component is : (54, (105.3, 95.008))
Length and centroid of 5 component is : (93, (131.752, 125.526))
Length and centroid of 6 component is : (71, (147.798, 58.014))
```

### 2.2 Structuring element used for this morphological operation is:

[19]:
```python
print(kernel)
print("Type of kernerl used is: "+str(kernel.dtype) +" and the shape is a␣
↪rectangle")
print("Size of kernerl used is {0} and shape of the kernel is {1}".
↪format(kernel.size, kernel.shape))
```

```
[[1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]]
Type of kernerl used is: uint8 and the shape is a rectangle
Size of kernerl used is 15 and shape of the kernel is (15, 1)
```

#### 2.2.1 Problem 3: Lloyd-Max Quantizer

[20]:
```python
#Imported from the provided code - lloyd_python.py
import numpy as np
from numpy.linalg import norm
import math

def lloyds(training_set, ini_codebook, tol=1e-7, plot_flag=False):
    """
    training_set: np [N, 1]
    ini_codebook: int (number of partitions)
```

```
    """
    ## init codebook
    assert len(ini_codebook)==1 # only support that
    if len(ini_codebook) == 1:
        ini_codebook = ini_codebook[0]
        assert ini_codebook >= 1, "invalid ini_codebook"
        min_training = training_set.min()
        max_training = training_set.max()
        ter_condition_2 = np.spacing(1) * max_training

        int_training = (max_training - min_training)/ini_codebook
        if int_training <= 0:
            print('The input training set is not valid because it has only one value.')
        codebook = np.linspace(min_training+int_training/2, max_training-int_training/2, ini_codebook);
    # print(f"codebook: {codebook}")

    ## init partition
    partition = (codebook[1 : ini_codebook] + codebook[0 : ini_codebook-1]) / 2
    def quantiz(training_set, partition, codebook):
        # print(f"quan - partition: {partition}, codebook: {codebook}")
        ## compute index
        indx = np.zeros((training_set.shape[0], 1))
        # print(f"indx: {indx.shape}")
        for i in range(len(partition)):
            indx = indx + (training_set > partition[i])
        ## compute distor
        distor = 0
        for i in range(len(codebook)):
            distor += norm(training_set[indx == i] - codebook[i])**2
        distor = distor / training_set.shape[0]
        # print(f"quantiz-distor {distor}")
        return indx, distor

    def get_rel_distortion(distor, last_distor, ter_condition_2):
        if distor > ter_condition_2:
            rel_distor = abs(distor - last_distor)/distor
        else:
            rel_distor = distor
        return rel_distor

    index, distor = quantiz(training_set, partition, codebook)
    last_distor = 0
    # rel_distor = abs(distor - last_distor)/distor
    rel_distor = get_rel_distortion(distor, last_distor, ter_condition_2)
    count = 0
```

```
    while (rel_distor > tol) and (rel_distor > ter_condition_2):
        # computer x_hat
        ## handle boundary condition
        partition_aug = np.concatenate((np.array([min_training]), partition, np.
 ↪array([max_training])))
        # print(f"partition: {partition}")
        # print(f"partition_aug: {partition_aug}")
        for i in range(len(partition_aug)-1):
            part_set = training_set[np.
 ↪logical_and(training_set>=partition_aug[i], training_set<partition_aug[i+1])␣
 ↪]
            if len(part_set) > 0:
                codebook[i] = part_set.mean()
            else:
                codebook[i] = (partition_aug[i] + partition_aug[i+1])/2

        # update t_hat: codebook
        partition = (codebook[1 : ini_codebook] + codebook[0 : ini_codebook-1])␣
 ↪/ 2
        # print(f"count: {count}, partition: {partition}, codebook:␣
 ↪{codebook}")
        # quantize again
        last_distor = 0 + distor
        index, distor = quantiz(training_set, partition, codebook)

        # get distortion

        # print(f"distor: {distor}, last_distor, {last_distor}, rel_distor:␣
 ↪{rel_distor}")
        rel_distor = get_rel_distortion(distor, last_distor, ter_condition_2)
        count += 1
        # print(f"distor: {distor}, rel_distor: {rel_distor}")

    return partition, codebook
```

[21]:
```
lena = cv2.imread("lena512.tif", 0)
diver = cv2.imread("diver.tif", 0)
scalars = np.arange(1, 8, 1)
```

[22]:
```
# Helper functions for problem 2

#Below function performs uniform quantization
#Takes image and number of bits as input and return quantized image
#L_passed argument added to allow the funciton caller to pass number of levels␣
 ↪directly instead of number of bits

# Problem 3 - i)
```

```python
def unifrom_quantizer(image, scalar_bit, L_passed=False):
    rows, cols = image.shape
    f_min = 0
    f_max = 256
    #Bypassing level calculation
    if L_passed:
        L = scalar_bit
    else:
        L = 2**scalar_bit
    q = (f_max-f_min)/L
    q_2 = q/2
    quantized_image = np.copy(image)
    quantized_image = quantized_image//q * q + q_2
    #print(mse_calc(quantized_image_cp, quantized_image))
    return quantized_image

#Below method calculates the mean squared error between 2 images
#2 input images are subtracted and squared and normalizes the squared error
def mse_calc(original, quantized):
    original = np.uint8(original)
    quantized = np.uint8(quantized)
    mse = 0
    rows = original.shape[0]
    cols = original.shape[1]
    error = np.square(original - quantized)
    mean_error = float(np.sum(error)/(rows*cols))
    return mean_error

#Below method is for converting partitiona and codebook to quantized image
#Uses vectorized implementation for comparision(faster)

#Problem 3 - ii)
def convert_image_from_codebook(partition, codebook, img):
    quantized_image = np.copy(img)
    partition = np.insert(partition, 0, 0)
    partition = np.insert(partition, len(partition), 255)
    for i in range(len(partition)-1):
        condition_low = (quantized_image > partition[i])
        condition_high  = (quantized_image <= partition[i+1])
        condition = condition_low & condition_high
        quantized_image[condition]= codebook[i]
    quantized_image = quantized_image.reshape(img.shape[0], img.shape[1])
    return quantized_image

#Helper method to quantize the given image using both uniform quantization
#and Lloyd-Max quantization
#Returns a tuple that contains:
```

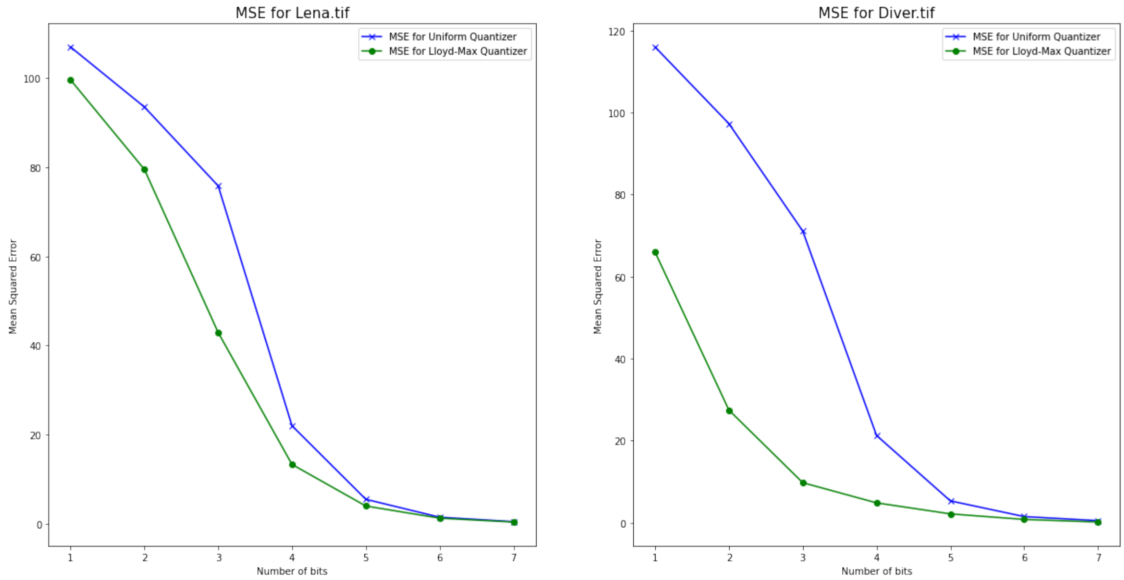
```
#  - List of MSE for uniform quantization using 1-7 bits
#  - List of quantized images for uniform quantization using 1-7 bits
#  - List of MSE for Lloyd-Max quantization using 1-7 bits
#  - List of quantized for Lloyd-Max quantization using 1-7 bits
def quantizer(img):
    #Uniform quantizer
    uniform_mse = []
    quantized_img = []
    idx = 0
    for scalar in scalars:
        quantized_img.append(unifrom_quantizer(img, scalar))
        uniform_mse.append(mse_calc(img, quantized_img[idx]))
        idx+=1

    #Lloyd max quantizer
    img_input = img.reshape(-1, 1)
    lloyd_quantized_img = []
    lloyd_mse = []
    idx = 0
    #print(img_input.shape)
    for scalar in scalars:
        initcodebook = [2**scalar]
        partition, codebook = lloyds(img_input, initcodebook)
        lloyd_quantized_img.append(convert_image_from_codebook(partition,␣
↪codebook, img))
        lloyd_mse.append(mse_calc(img, lloyd_quantized_img[idx]))
        #lloyd_mse.append(np.square(img-lloyd_quantized_img[idx]).
↪mean(axis=None))
        idx+=1
    return uniform_mse, quantized_img, lloyd_mse, lloyd_quantized_img

#Helper method for plotting the MSE as a function of number of bits
def plot_error_func(x, y1, y2):
    plt.xlabel("Number of bits")
    plt.ylabel("Mean Squared Error")
    plt.plot(x, y1, color='b', marker='x', label='MSE for Uniform Quantizer')
    plt.plot(x, y2, color='g', marker='o', label='MSE for Lloyd-Max Quantizer')
    plt.legend()
```

[23]:
```
# Quantize the given images
lena_mse, lena_quant, lloyd_lena_mse, lloyd_lena_quant = quantizer(lena)
diver_mse, diver_quant, lloyd_diver_mse, lloyd_diver_quant = quantizer(diver)
```

[24]:
```
fig, axs = plt.subplots(1,2, figsize=(20, 10))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("MSE for Lena.tif")
ax1.title.set_size(15)
plot_error_func(scalars, lena_mse,  lloyd_lena_mse)
```

```
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("MSE for Diver.tif")
ax1.title.set_size(15)
plot_error_func(scalars, diver_mse, lloyd_diver_mse)
axs[1].axis('off')

print("Mean squared errors of HE Lena in increasing order of bits: ",lena_mse)
print("")
print("Mean squared errors of HE Lena increasing order of bits using Lloyd Max:␣
↪",lloyd_lena_mse)
print("")
print("Mean squared errors of HE Diver in increasing order of bits: ",diver_mse)
print("")
print("Mean squared errors of HE Diver in increasing order of bits using Lloyd␣
↪Max: ",lloyd_diver_mse)
```

```
Mean squared errors of HE Lena in increasing order of bits:
[106.88676834106445, 93.44853591918945, 75.75944900512695, 22.000843048095703,
5.517719268798828, 1.5023231506347656, 0.5004539489746094]

Mean squared errors of HE Lena increasing order of bits using Lloyd Max:
[99.51007080078125, 79.45072174072266, 42.88694763183594, 13.330535888671875,
4.012081146240234, 1.292999267578125, 0.421661376953125]

Mean squared errors of HE Diver in increasing order of bits:
[115.92547194719472, 97.25310231023103, 71.08174917491749, 21.27549174917492,
5.3293003300330035, 1.5062244224422443, 0.507003300330033]

Mean squared errors of HE Diver in increasing order of bits using Lloyd Max:
[65.91867326732674, 27.452435643564357, 9.765392739273928, 4.848356435643565,
2.163993399339934, 0.8103432343234324, 0.176019801980198]
```

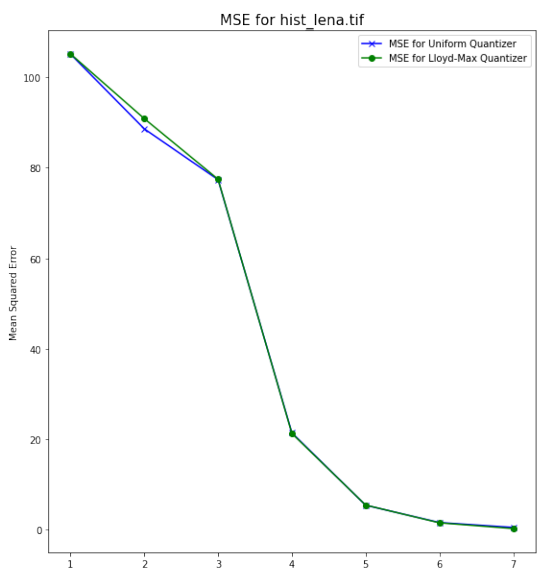


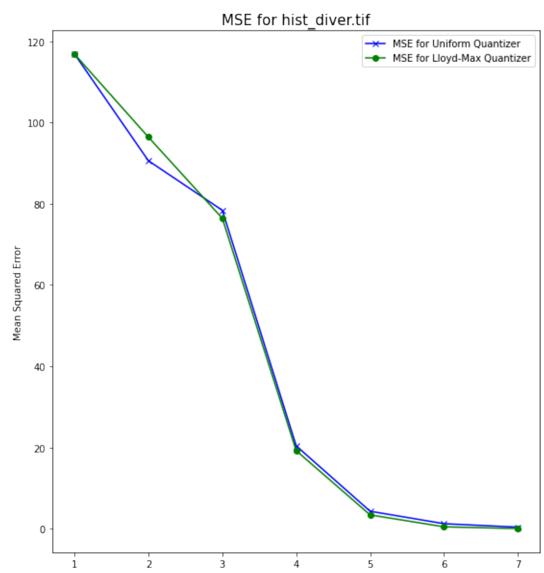

[26]:
```
# Problem 3 - iii)
# Global histogram equalization
hist_lena = cv2.equalizeHist(lena)
hist_diver = cv2.equalizeHist(diver)
# Quantize and calculate MSE
hist_lena_mse, hist_lena_quant, lloyd_hist_lena_mse, lloyd_hist_lena_quant =␣
↪quantizer(hist_lena)
hist_diver_mse, hist_diver_quant, lloyd_hist_diver_mse, lloyd_hist_diver_quant␣
↪= quantizer(hist_diver)
```

[27]:
```
fig, axs = plt.subplots(1,2, figsize=(20, 10))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("MSE for hist_lena.tif")
ax1.title.set_size(15)
plot_error_func(scalars, hist_lena_mse,  lloyd_hist_lena_mse)
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("MSE for hist_diver.tif")
ax1.title.set_size(15)
plot_error_func(scalars, hist_diver_mse, lloyd_hist_diver_mse)
axs[1].axis('off')

print("Mean squared errors of HE Lena in increasing order of bits:␣
↪",hist_lena_mse)
print("")
print("Mean squared errors of HE Lena increasing order of bits using Lloyd Max:␣
↪",lloyd_hist_lena_mse)
print("")
```

```python
print("Mean squared errors of HE Diver in increasing order of bits: ",hist_diver_mse)
print("")
print("Mean squared errors of HE Diver in increasing order of bits using Lloyd Max: ",lloyd_hist_diver_mse)
```

```
Mean squared errors of HE Lena in increasing order of bits:
[105.14220809936523, 88.60534286499023, 77.37133407592773, 21.397151947021484,
5.375759124755859, 1.5522880554199219, 0.4701423645019531]

Mean squared errors of HE Lena increasing order of bits using Lloyd Max:
[105.16981887817383, 90.86715316772461, 77.46942520141602, 21.258827209472656,
5.407634735107422, 1.4914627075195312, 0.2115478515625]

Mean squared errors of HE Diver in increasing order of bits:
[116.85354455445544, 90.57848184818482, 78.38046204620463, 20.408079207920792,
4.312607260726073, 1.2362244224422443, 0.3917623762376238]

Mean squared errors of HE Diver in increasing order of bits using Lloyd Max:
[116.82515511551155, 96.38768316831683, 76.4176897689769, 19.22161716171617,
3.4114983498349836, 0.4720726072607261, 0.040547854785478545]
```




#### 2.2.2 The gap between the MSE for uniform quantizer and Lloyd-Max quantizer reduced after quantizing the histogram equalized image. This is because the histogram equalized image has values more spread out and the 2 quantization methods yield nearly the same result as value ranges are more spread out.

#### 2.2.3 Problem 3 - iv)

**The MSE for 7-bit Lloyd-Max quantizer is nearly zero as the quantized image resembles the original image more so than performing Lloyd-Max quantization for an un-equalized image.**

### 2.2.4 Problem 4: Quantization with Dithering

Note: The uniform quantizer method used for problem 3 is reused

[28]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

[29]:
```
geisel = cv2.imread("geisel.jpg")
geisel = cv2.cvtColor(geisel, cv2.COLOR_BGR2RGB)
```

[30]:
```
quantized_output = np.zeros((geisel.shape), dtype=np.uint8)
quantized_output[:,:,0] = unifrom_quantizer(geisel[:,:,0] , 10, True)
quantized_output[:,:,1] = unifrom_quantizer(geisel[:,:,1] , 10, True)
quantized_output[:,:,2] = unifrom_quantizer(geisel[:,:,2] , 10, True)

fig, axs = plt.subplots(1,2, figsize=(20, 10))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("Original Image")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(geisel)
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("Uniformly quantized output")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(quantized_output)
axs[1].axis('off')
```

[30]:
```
(0.0, 1.0, 0.0, 1.0)
```



[31]:
```
#Helper functions

#Function to clip and clamp the output to be 0 and 255
def clip_clamp(v):
    if v > 255:
        v = 255
    if v < 0:
        v = 0
    return v

#Quantize the given input values based on levels
#For the given example, levels = 10
def find_closest_color_palette(r, g, b, levels):
    b = clip_clamp(np.round(levels * b/255.0) * (255/levels))
    g = clip_clamp(np.round(levels * g/255.0) * (255/levels))
    r = clip_clamp(np.round(levels * r/255.0) * (255/levels))
    return r, g, b

#Floyd-Steinberg Dithering algorithm
#Inputs are image and number of levels
#Output is quantized image

def floyd_stein(inp, levels):
    image = np.copy(inp)
    rows, cols, channels = image.shape
    for i in range(rows-1):
        for j in range(1, cols-1):
            old_pix_r = image[i, j, 0]
            old_pix_g = image[i, j, 1]
```

```
            old_pix_b = image[i, j, 2]

            new_pix_r, new_pix_g, new_pix_b =␣
↪find_closest_color_palette(old_pix_r,
                                                                                    ␣
↪old_pix_g,
                                                                                    ␣
↪old_pix_b, levels)
            image[i, j, 0] = new_pix_r
            image[i, j, 1] = new_pix_g
            image[i, j, 2] = new_pix_b

            error_r = old_pix_r - new_pix_r
            error_g = old_pix_g - new_pix_g
            error_b = old_pix_b - new_pix_b

            image[i+1, j, 0] = clip_clamp(image[i+1, j, 0] + (error_r * 7/16.0))
            image[i-1, j+1, 0] = clip_clamp(image[i-1, j+1, 0] + (error_r * 3/
↪16.0))
            image[i, j+1, 0] = clip_clamp(image[i, j+1, 0] + (error_r * 5/16.0))
            image[i+1, j+1, 0] = clip_clamp(image[i+1, j+1, 0] + (error_r * 1/
↪16.0))

            image[i+1, j, 1] = clip_clamp(image[i+1, j, 1] + (error_g * 7/16.0))
            image[i-1, j+1, 1] = clip_clamp(image[i-1, j+1, 1] + (error_g * 3/
↪16.0))
            image[i, j+1, 1] = clip_clamp(image[i, j+1, 1] + (error_g * 5/16.0))
            image[i+1, j+1, 1] = clip_clamp(image[i+1, j+1, 1] + (error_g * 1/
↪16.0))

            image[i+1, j, 2] = clip_clamp(image[i+1, j, 2] + (error_b * 7/16.0))
            image[i-1, j+1, 2] = clip_clamp(image[i-1, j+1, 2] + (error_b * 3/
↪16.0))
            image[i, j+1, 2] = clip_clamp(image[i, j+1, 2] + (error_b * 5/16.0))
            image[i+1, j+1, 2] = clip_clamp(image[i+1, j+1, 2] + (error_b * 1/
↪16.0))
    #print("Processing one color done")
    return image
```

[32]:
```
geisel = cv2.imread("geisel.jpg")
geisel = cv2.cvtColor(geisel, cv2.COLOR_BGR2RGB)
fsd_out = floyd_stein(geisel.copy(), 10)
```

[33]:
```
fig, axs = plt.subplots(1,2, figsize=(20, 10))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("Uniformly quantized image")
ax1.title.set_size(15)
```

```
ax1.axis('off')
ax1.imshow(quantized_output)
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("Dithered Image")
ax1.title.set_size(15)
ax1.axis('off')
ax1.imshow(fsd_out)
axs[1].axis('off')
```

[33]: (0.0, 1.0, 0.0, 1.0)



**Problem 4: 1)** Uniform quantized image has contours near the sky as a result of reducing the number of grayscale levels in each fo the color channels where as the dithered image does not.

**Problem 4: 2)** Dithering is a process of diffusing the quantization error to the neighboring pixels. This causes reduction in the localized quantization error and causes small dots to appear in the quantized image. In the given image, as a result of the high resolution of the input image, the dots do not seem to appear and the resulting image is nearly the same as the original image. Reduction in space(10 levels instead of 256) is achieved without loss of clarity.

Note: Below is a piece of code added for debugging purposes. It can clearly be seen that the dithering algorithm is working from the downsampled image

[34]:
```
# Debug code to verify dithering
# Using downsampled image as dithered output is not noticeable when the␣
↪resolution is high
geisel = cv2.imread("geisel.jpg")
geisel = cv2.cvtColor(geisel, cv2.COLOR_BGR2RGB)
```

```
geisel = cv2.resize(geisel, None, fx=0.1, fy=0.1, interpolation=cv2.
    ↪INTER_LINEAR)
fsd_out_downsampled = floyd_stein(geisel.copy(), 10)

fig, axs = plt.subplots(1,2, figsize=(20, 10))
ax1= fig.add_subplot(1,2,1)
ax1.title.set_text("Downsampled and dithered Image")
ax1.axis('off')
ax1.imshow(fsd_out_downsampled)
axs[0].axis('off')

ax1= fig.add_subplot(1,2,2)
ax1.title.set_text("Dithered without downsampling")
ax1.axis('off')
ax1.imshow(fsd_out)
axs[1].axis('off')
```

[34]: (0.0, 1.0, 0.0, 1.0)


[ ]: