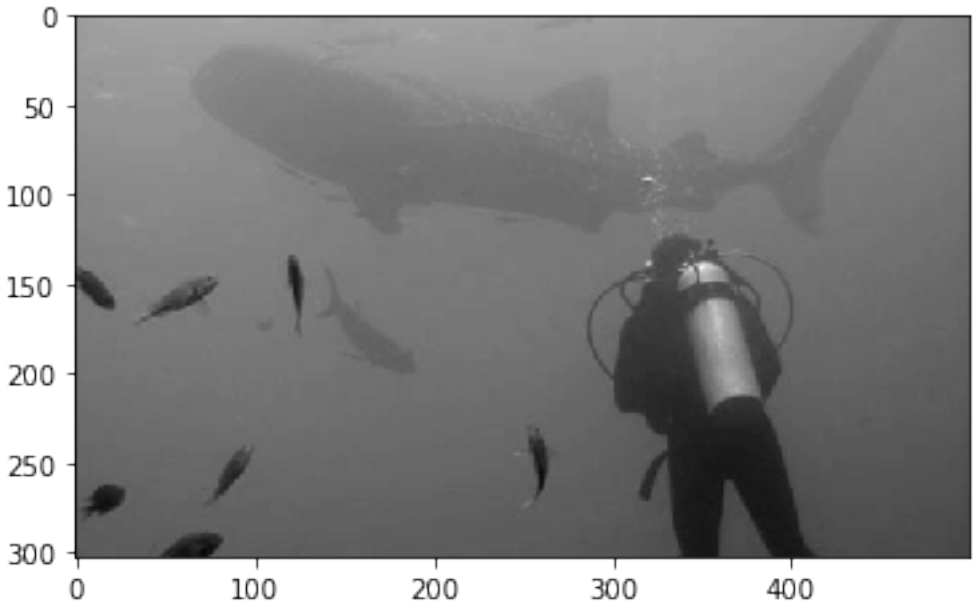
HW2.3

November 8, 2019

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

[2]:
```
# imgs = cv2.imread("data/*.tif", 0)
path = "data/*.tif"
imgs = []
for file in glob.glob(path):
    imgs.append(cv2.imread(file, 0))
plt.imshow(imgs[1], cmap = 'gray')
plt.show()
```



[3]:
```
def bit_change(img, ori, out):
    change = 2 ** (ori - out)
    img_new = img // change * change
```

```
    return img_new
```

(i) The function quantilize the 8-bit image to s-bit image is shown as above.

[4]:
```
def lloyds_img(img, ori, out):
    [M, N] = img.shape;
    img = img.astype(float)
    img_new_l = np.zeros(img.shape)
    training_set = np.reshape(img,N*M,1);
    partition, codebook = lloyds(training_set, [2**out])
    for i in range(M):
        for j in range(N):
            if (img[i][j] < partition[0]):
                img_new_l[i][j] = codebook[0];
            else:
                curdist = [img[i][j] - p for p in partition]
                curmin = min(i for i in curdist if i >= 0)
                minIndex = curdist.index(curmin)
                img_new_l[i][j] = codebook[minIndex + 1]
    return img_new_l
```

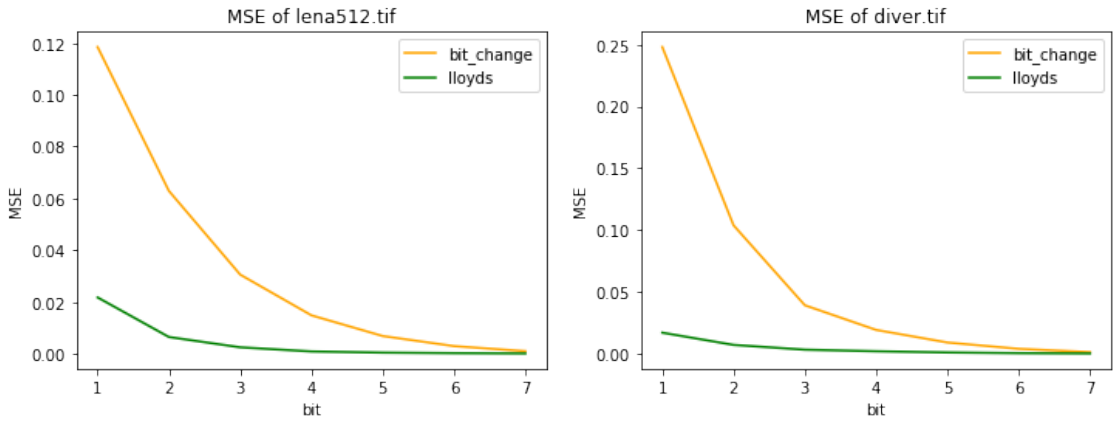
[5]:
```
def MSE(img, img_new):
    size = img.shape[0] * img.shape[1]
    return np.linalg.norm(img_new.astype(float) - img.astype(float), 2)  / size
```

[6]:
```
def mses_compute(img):
#     plt.imshow(img, cmap = 'gray')
#     plt.show()
    ss = np.arange(1, 8, 1)
    mses_bit_change = []
    mses_lloyds = []

    for s in ss:
        img_new_bit_change = bit_change(img, 8, s)
        mse_bit_change = MSE(img, img_new_bit_change)
        mses_bit_change.append(mse_bit_change)

        img_new_l = lloyds_img(img, 8, s)
        mse_lloyds = MSE(img, img_new_l)
        mses_lloyds.append(mse_lloyds)

#         f = plt.figure(figsize=(8,4))
#         ax1 = f.add_subplot(1,2,1)
#         ax1.imshow(img_new_bit_change, cmap = 'gray')
#         ax2 = f.add_subplot(1,2,2)
#         ax2.imshow(img_new_l, cmap = 'gray')
#         plt.show()
```

```
    return mses_bit_change, mses_lloyds
```

[7]:
```
msess_bit_change = []
msess_lloyds = []

for img in imgs:
    mses_bit_change, mses_lloyds = mses_compute(img)
    msess_bit_change.append(mses_bit_change)
    msess_lloyds.append(mses_lloyds)
```

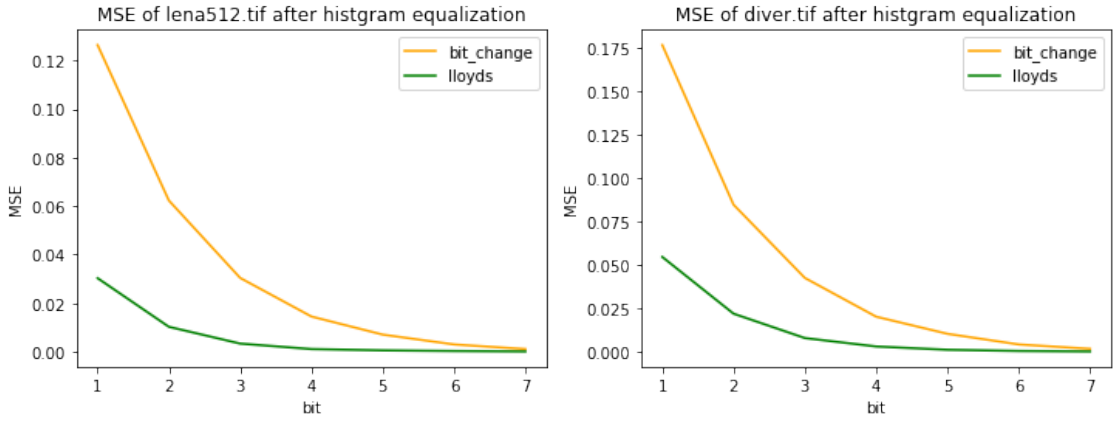
[8]:
```
f = plt.figure(figsize = (12, 4))
ss = np.arange(1, 8, 1)

ax1 = f.add_subplot(121)
ax1.title.set_text("MSE of lena512.tif")
ax1.plot(ss, msess_bit_change[0], color='orange', label='bit_change')
ax1.plot(ss, msess_lloyds[0], color='green', label='lloyds')
ax1.legend() #
ax1.set_xlabel('bit')
ax1.set_ylabel('MSE')

ax2 = f.add_subplot(122)
ax2.title.set_text("MSE of diver.tif")
ax2.plot(ss, msess_bit_change[1], color='orange', label='bit_change')
ax2.plot(ss, msess_lloyds[1], color='green', label='lloyds')
ax2.legend() #
ax2.set_xlabel('bit')
ax2.set_ylabel('MSE')
plt.savefig("p3_noHE.jpg")

plt.show()
```



(ii) The results of lena512.tif and diver.tif are shown above which shows that the lloyd-max quantizer outperform the bit-change method as the lloyd-max quantizer will get much smaller MSE than bit-change quantizer and the lloyd-max quantizer's performace gap is much smaller than the bit-change method. The reason why lloyd-max quantizer is outperformed than bit-change method is that it will consider the detail situation of this whole image and decide the specific partition and codebook value for this specific image to be quantized. However, the bit-change method will not consider the specific situation for this image and will simply downsampled the value. So as the bit become smaller, the MSE for the bit-change method will become larger.

[9]:
```
imgs_histequs = []
for img in imgs:
    imgs_histequs.append(cv2.equalizeHist(img))

msess_histequ_bit_change = []
msess_histequ_lloyds = []

for img in imgs_histequs:
    mses_bit_change, mses_lloyds = mses_compute(img)
    msess_histequ_bit_change.append(mses_bit_change)
    msess_histequ_lloyds.append(mses_lloyds)
```

[10]:
```
f = plt.figure(figsize = (12, 4))
ss = np.arange(1, 8, 1)

ax1 = f.add_subplot(121)
ax1.title.set_text("MSE of lena512.tif after histgram equalization")
ax1.plot(ss, msess_histequ_bit_change[0], color='orange', label='bit_change')
ax1.plot(ss, msess_histequ_lloyds[0], color='green', label='lloyds')
ax1.legend() #
ax1.set_xlabel('bit')
ax1.set_ylabel('MSE')

ax2 = f.add_subplot(122)
ax2.title.set_text("MSE of diver.tif after histgram equalization")
ax2.plot(ss, msess_histequ_bit_change[1], color='orange', label='bit_change')
ax2.plot(ss, msess_histequ_lloyds[1], color='green', label='lloyds')
ax2.legend() #
ax2.set_xlabel('bit')
ax2.set_ylabel('MSE')
plt.savefig("p3_HE.jpg")

plt.show()
```



(iii) The gap in MSE between the two quantization approaches become smaller. This is because that for when using histogram equalization, the histogram will be flattened. And thus the MSE between this two method will be smaller as the MSE of lloyds-max method will be larger than it used to be.

(iv) Though the equalization will flatten the distribution, the differece between all the pixels will become larger and thus will make the lloyds-max quantizer performs well for 7-bit situation.