HERLEEYANDI MARKONI
M10407812

Homework 2 Report

Theory

1) Flow Of image compression

   The image compression using vector quantization begins with divide the image into non-overlapping patch. For every patch, we reshape the block into a vector. This vector then send into the encoder. In the encode phase we will have 1 codebook which have been generate before. Then the vector will be compared using Mean Squared Error(MSE) or Weighted MSE with every vector in the codebook to see the closest match. After the closest match found, then the index of the matched codeword will be used as compressed data. Same things also happen in decode the image. For every value in the compressed data, we send it as index to the codebook. The vector which associate with the index will be convert into matrix with the same block size. The matrix value will replace the non-overlapped value as decoded image.

2) Codebook Generation using LBG

   A. Form a training vector from the image

   B. Let N = 1 then get the average of all vector and calculate the distortion.

   $$\mathbf{c}_1^* = \frac{1}{M}\sum_{m=1}^{M}\mathbf{x}_m.$$

   $$D_{ave}^* = \frac{1}{Mk}\sum_{m=1}^{M}||\mathbf{x}_m - \mathbf{c}_1^*||^2.$$

   C. Split to form a new codevector.

   $$\begin{aligned}\mathbf{c}_i^{(0)} &= (1+\epsilon)\mathbf{c}_i^*,\\ \mathbf{c}_{N+i}^{(0)} &= (1-\epsilon)\mathbf{c}_i^*.\end{aligned}$$

   D. Do the iteration.

      i. Find the minimum value

      $$||\mathbf{x}_m - \mathbf{c}_n^{(i)}||^2,$$

      ii. Update the codevector

      $$\mathbf{c}_n^{(i+1)} = \frac{\sum_{Q(\mathbf{x}_m)=\mathbf{c}_n^{(i)}}\mathbf{x}_m}{\sum_{Q(\mathbf{x}_m)=\mathbf{c}_n^{(i)}}1}$$

      iii. Calculate average distortion

      $$D_{ave}^{(i)} = \frac{1}{Mk}\sum_{m=1}^{M}||\mathbf{x}_m - Q(\mathbf{x}_m)||^2.$$

      $$(D_{ave}^{(i-1)} - D_{ave}^{(i)})/D_{ave}^{(i-1)} > \epsilon$$

      iv. Set current average distortion for the next step.

      $$D_{ave}^* = D_{ave}^{(i)}$$

E. Repeat C and D process to get the exact amount of codevector.

Implementation

1) Generate codebook using LBG

A. Whole Process Generate codebook

```
def generate_codebook(data, size_codebook, epsilon=0.00005):
    global _size_data, _dim

    _size_data = len(data)
    assert _size_data > 0

    _dim = len(data[0])
    assert _dim > 0

    codebook = []
    codebook_abs = [_size_data]
    codebook_rel = [1.0]

    c0 = avg_all_vectors(data, _dim, _size_data)
    codebook.append(c0)

    avg_dist = initial_avg_distortion(c0, data)

    while len(codebook) < size_codebook:
        codebook, codebook_abs, codebook_rel, avg_dist =
split_codebook(data, codebook,

epsilon, avg_dist)

    return codebook, codebook_abs, codebook_rel
```

B. Split Process

```
def split_codebook(data, codebook, epsilon, initial_avg_dist):
    # split into 2
    new_cv = []
    for c in codebook:
        # plus and minus epsilon for the new codebook
        c1 = new_codevector(c, epsilon)
        c2 = new_codevector(c, -epsilon)
        new_cv.extend((c1, c2))

    codebook = new_cv
    len_codebook = len(codebook)
    abs_weights = [0] * len_codebook
    rel_weights = [0.0] * len_codebook

    # Get the best centroid by taking average distortion as cost
function. This problems mimic K-Means.
    avg_dist = 0
    err = epsilon + 1
    num_iter = 0
    while err > epsilon:
        # Get nearest codevector.
        closest_c_list = [None] * _size_data    # nearest codevector
        vecs_near_c = defaultdict(list)         # input data vector
mapping
        vec_idxs_near_c = defaultdict(list)     # input data index
mapping
```

```
        for i, vec in enumerate(data):  # for each input vector
            min_dist = None
            closest_c_index = None
            for i_c, c in enumerate(codebook):
                d = get_mse(vec, c)
                # Get the nearest ones.
                if min_dist is None or d < min_dist:
                    min_dist = d
                    closest_c_list[i] = c
                    closest_c_index = i_c
            vecs_near_c[closest_c_index].append(vec)
            vec_idxs_near_c[closest_c_index].append(i)

        # Update the codebook
        for i_c in range(len_codebook):
            vecs = vecs_near_c.get(i_c) or []
            num_vecs_near_c = len(vecs)
            if num_vecs_near_c > 0:
                # assign as new center
                new_c = avg_all_vectors(vecs, _dim)
                codebook[i_c] = new_c
                for i in vec_idxs_near_c[i_c]:
                    closest_c_list[i] = new_c

                # update the weights
                abs_weights[i_c] = num_vecs_near_c
                rel_weights[i_c] = num_vecs_near_c / _size_data

        # Recalculate average distortion
        prev_avg_dist = avg_dist if avg_dist > 0 else initial_avg_dist
        avg_dist = avg_codevector_dist(closest_c_list, data)

        # Recalculate the new error value
        err = (prev_avg_dist - avg_dist) / prev_avg_dist
        num_iter += 1

    return codebook, abs_weights, rel_weights, avg_dist
```

C. Get Average All Vector

```
def avg_all_vectors(vecs, dim=None, size=None):
    size = size or len(vecs)
    nvec = np.array(vecs)
    nvec = nvec / size
    navg = np.sum(nvec, axis=0)
    return navg.tolist()
```

D. Create New Codevector

```
def new_codevector(c, e):
    nc = np.array(c)
    return (nc * (1.0 + e)).tolist()
```

E. Get Initial Average Distortion

```
def initial_avg_distortion(c0, data, size=None):
    size = size or _size_data
    nc = np.array(c0)
    nd = np.array(data)
    f = np.sum(((nc-nd)**2)/size)
    return f
```

F. Get Average of Codevector Distance

```
def avg_codevector_dist(c_list, data, size=None):
    size = size or _size_data
    nc = np.array(c_list)
    nd = np.array(data)
    f = np.sum(((nc-nd)**2)/size)
    return f
```

G. Get The Squared Error

```
def get_mse(a, b):
    na = np.array(a)
    nb = np.array(b)
    return np.sum((na-nb)**2)
```

H. Generate Training

```
def generate_training(img, block):
    train_vec = []
    x = block[0]
    y = block[1]
    for i in range(0, img.shape[0], x):
        for j in range(0, img.shape[1], y):
            train_vec.append(img[i:i + x, j:j + y].reshape((x * y)))
    return (np.array(train_vec))

def generate_multi_training(path_list, block):
    img_list = []
    for path in path_list:
        img_list.append(cv2.imread(path, cv2.IMREAD_GRAYSCALE))
    train_vec = []
    x = block[0]
    y = block[1]
    for img in img_list:
        for i in range(0, img.shape[0], x):
            for j in range(0, img.shape[1], y):
                train_vec.append(img[i:i + x, j:j + y].reshape((x * y)))
    return (np.array(train_vec))
```

2) Encode image

```
def encode_image(img, cb, block):
    x = block[0]
    y = block[1]
    compressed = np.zeros((img.shape[0] // y, img.shape[1] // x))
    ix = 0
    for i in range(0, img.shape[0], x):
        iy = 0
        for j in range(0, img.shape[1], y):
            src = img[i:i + x, j:j + y].reshape((x * y)).copy()
            k = closest_match(src, cb)
            compressed[ix, iy] = k
            iy += 1
        ix += 1
    return compressed
```

3) Decode Image

```
def decode_image(cb, compressed, block):
    x = block[0]
    y = block[1]
    original = np.zeros((compressed.shape[0] * y, compressed.shape[1] * x))
    ix = 0
    for i in range(0, compressed.shape[0]):
        iy = 0
        for j in range(0, compressed.shape[1]):
            original[ix:ix + x, iy:iy + y] = cb[int(compressed[i,
j])].reshape(block)
            iy += y
        ix += x
    return original
```

4) Calculate PSNR

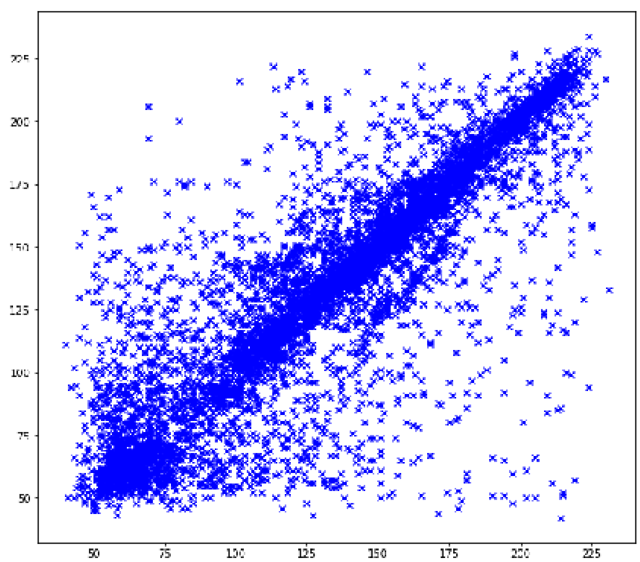
```
def psnr(img1, img2):
    mse = np.mean( (img1 - img2) ** 2 )
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
```

5) Utility function to run simulations

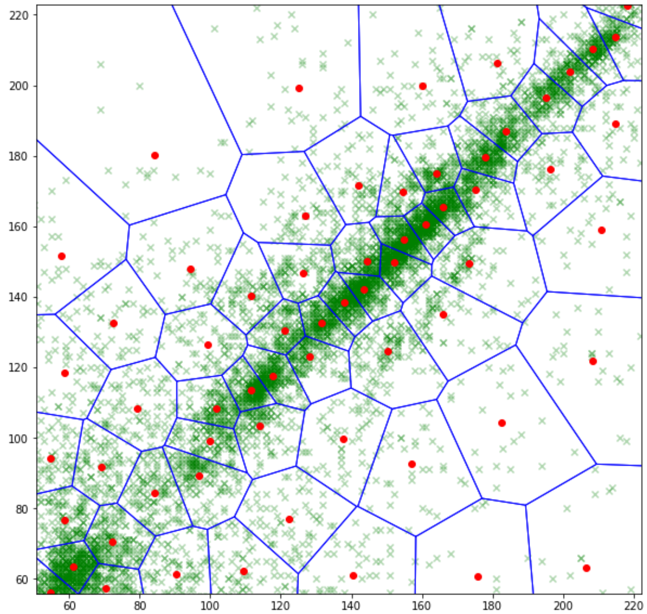
```
def sim_protocol(img, cb_size, epsilon, block, root, outpng):
    train_X = generate_training(img, block)
    cb, cb_abs_w, cb_rel_w = lbg.generate_codebook(train_X, cb_size, epsilon)
    cb_n = np.array(cb)
    cb_abs_w_n = np.array(cb_abs_w)
    cb_rel_w_n = np.array(cb_rel_w)
    result = encode_image(img, cb_n, block)
    final_result = decode_image(cb_n, result, block)
    fig = plt.gcf()
    fig.set_figheight(6)
    fig.set_figwidth(6)
    plt.imshow(final_result, cmap='gray')
    cv2.imwrite(root + outpng + '.png', final_result)
    save_csv(root, outpng, cb_n, cb_abs_w_n, cb_rel_w_n)

def sim_multi_protocol(path_list, cb_size, epsilon, block, root, outpng):
    train_X = generate_multi_training(path_list, block)
    cb, cb_abs_w, cb_rel_w = lbg.generate_codebook(train_X, cb_size, epsilon)
    cb_n = np.array(cb)
    cb_abs_w_n = np.array(cb_abs_w)
    cb_rel_w_n = np.array(cb_rel_w)
    save_csv(root, outpng, cb_n, cb_abs_w_n, cb_rel_w_n)
    print('Weight Saved as: '+outpng)

def sim_testing_protocol(inpath_list, weight, block, outpng):
    fig, ax = plt.subplots(nrows=1, ncols=4)
    idx = 1
    for inpath in inpath_list:
        img = cv2.imread(inpath, cv2.IMREAD_GRAYSCALE)
        cb = pd.read_csv(weight, header=None).as_matrix().astype('int')
        cb = cb[:,0:cb.shape[1]-1]
        result = encode_image(img, cb, block)
        final_result = decode_image(cb, result, block)
        rem = inpath.replace('./images/', '')
        cv2.imwrite(outpng + rem.replace('.csv',''), final_result)
        psnr_value = psnr(img, final_result)
```

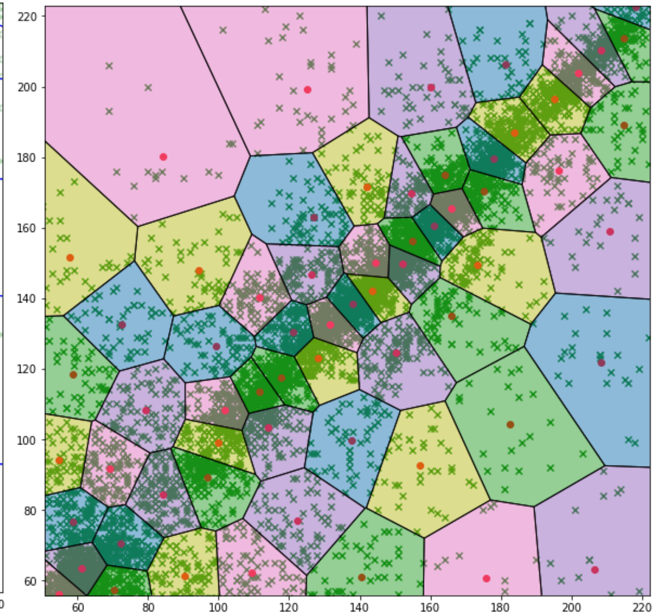
```
        ax = plt.subplot(1, 4, idx)
        ax.set_title('PSNR = {}'.format(psnr_value))
        ax.imshow(final_result, cmap='gray')
        idx+=1
    fig.set_figheight(6)
    fig.set_figwidth(24)
    plt.show()

def measure_psnr(apath, bpath):
    img1 = cv2.imread(apath, cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread(bpath, cv2.IMREAD_GRAYSCALE)
    print('PSNR: {}'.format(psnr(img1, img2)))

    fig, ax = plt.subplots(nrows=1, ncols=2)
    ax1 = plt.subplot(1, 2, 1)
    ax1.set_title("Original")
    ax1.imshow(img1, cmap='gray')

    ax2 = plt.subplot(1, 2, 2)
    ax2.set_title("Result")
    ax2.imshow(img2, cmap='gray')

    fig.set_figheight(7)
    fig.set_figwidth(14)
    plt.show()
```

6) Trying to use block (1,2) and plot voronoi diagram



HERLEEYANDI MARKONI
M10407812

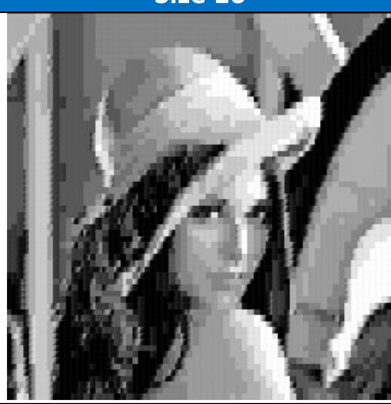
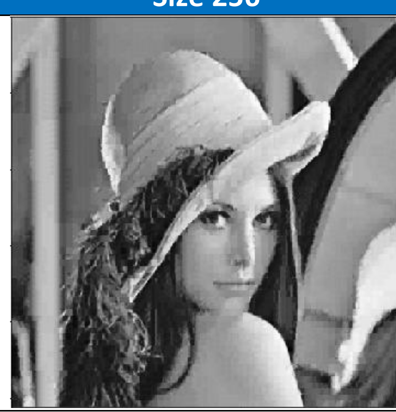




Experiment

In this part some experiments have been done with different scenarios.

1) Effect on the codebook size

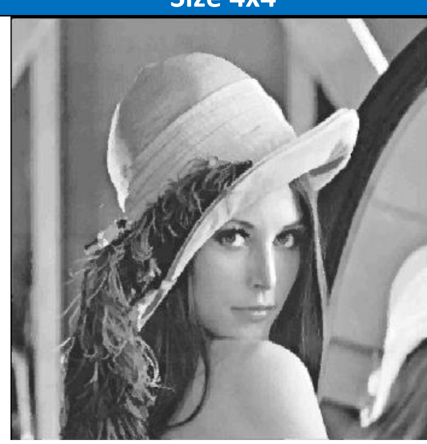
Size 16	Size 32	Size 64
		
PSNR: 31.160	PSNR: 31.715	PSNR: 32.264
Size 128	**Size 256**	**Size 512**
		
PSNR: 32.644	PSNR: 33.179	PSNR: 33.940

HERLEEYANDI MARKONI
M10407812

2) Effect on different epsilon (threshold)

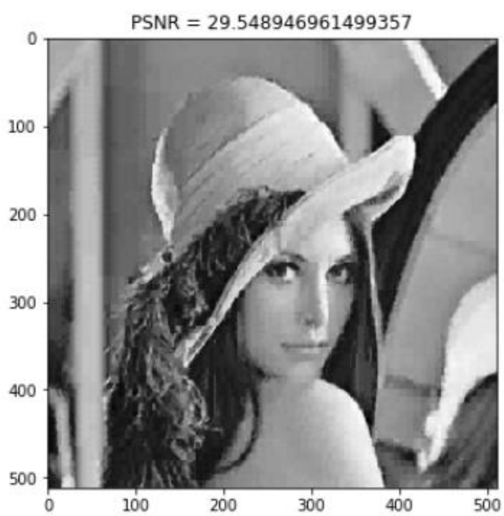
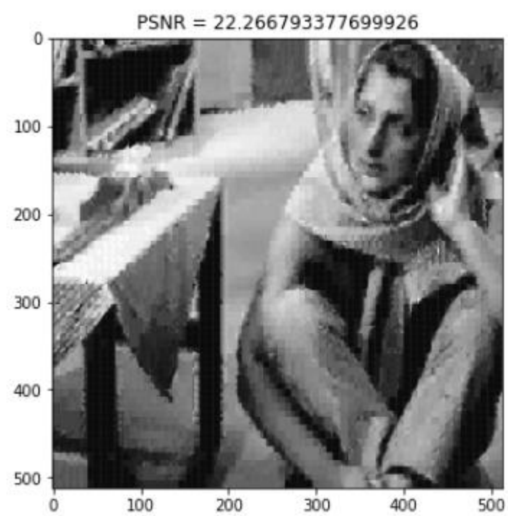
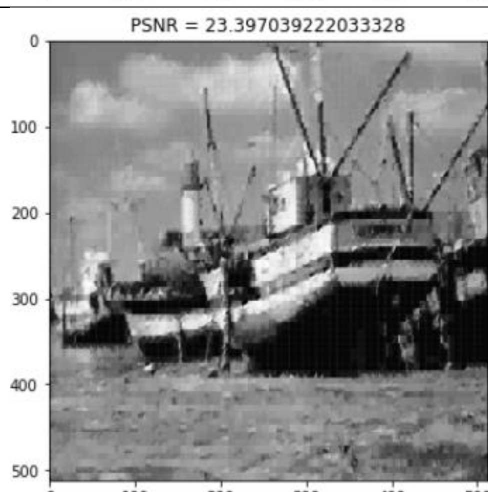
e = 0.005	**e = 0.0005**
	
PSNR: 32.228	PSNR: 32.264
e = 0.0005	**e = 0.00001**
	
PSNR: 32.251	PSNR: 32.228

3) Effect on the block size

Size 2x2	**Size 4x4**
	
PSNR: 36.282	PSNR: 34.329

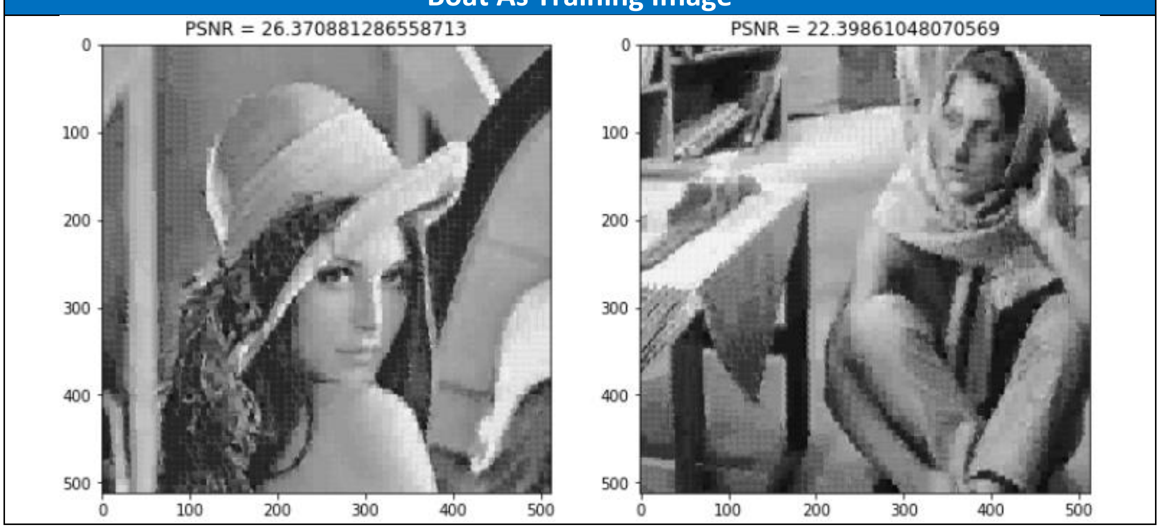
Size 8x8	Size 16x16
	
PSNR: 32.907	PSNR: 31.888

4) Using other image as Training and use the codebook for different image

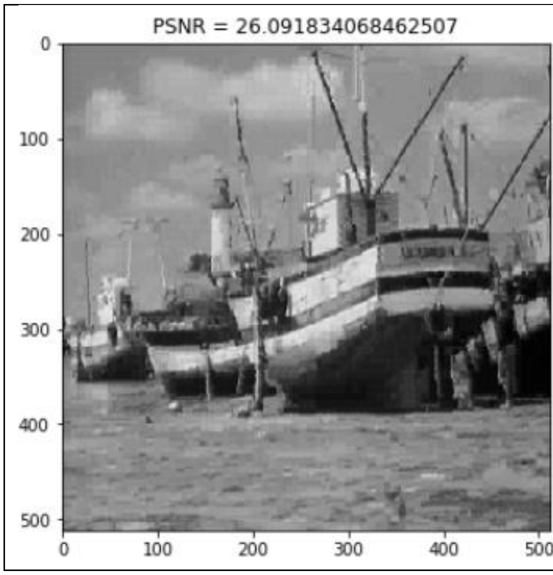
Lenna As Training Image	
	
	

Barbara As Training Image

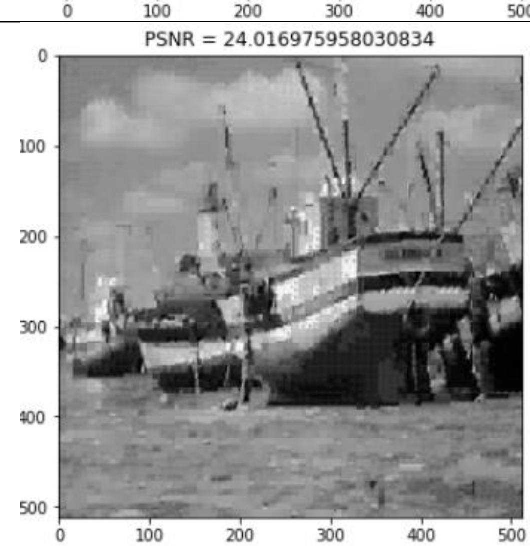
Boat As Training Image



HERLEEYANDI MARKONI
M10407812



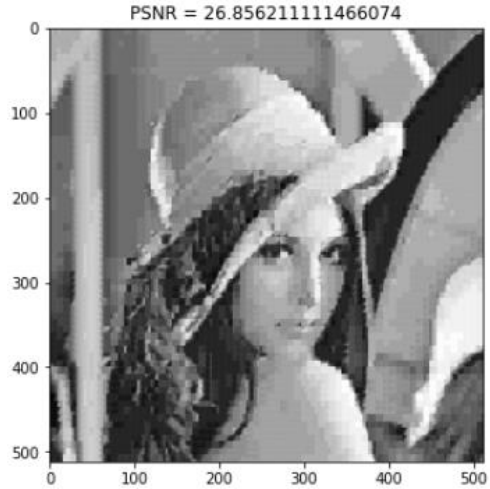
Cameraman As Training Image

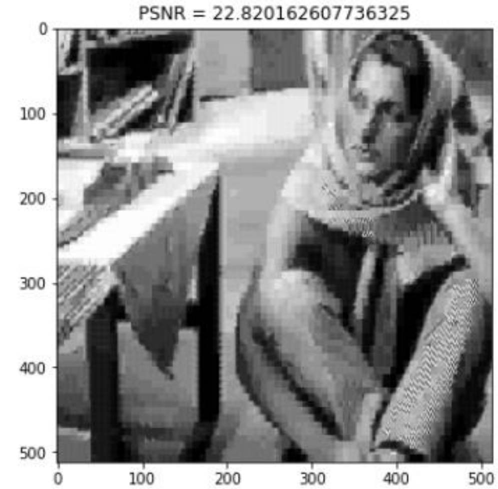


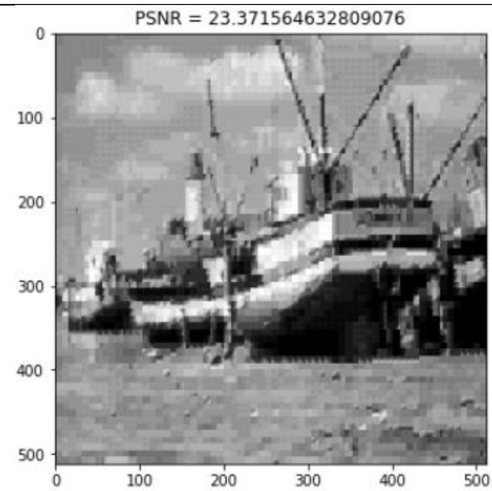
HERLEEYANDI MARKONI
M10407812

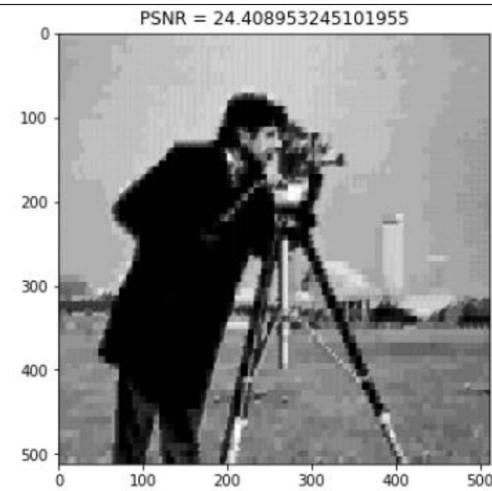
5) Use more than 1 image to Train

Lenna As Training Image

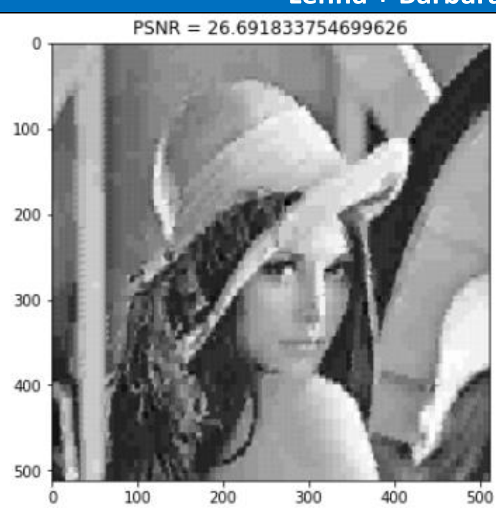


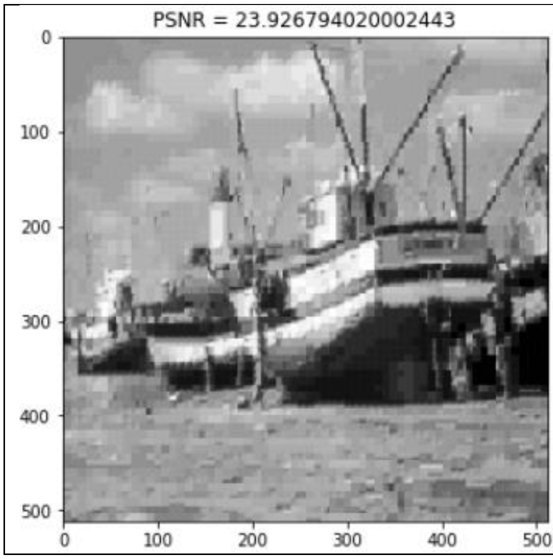




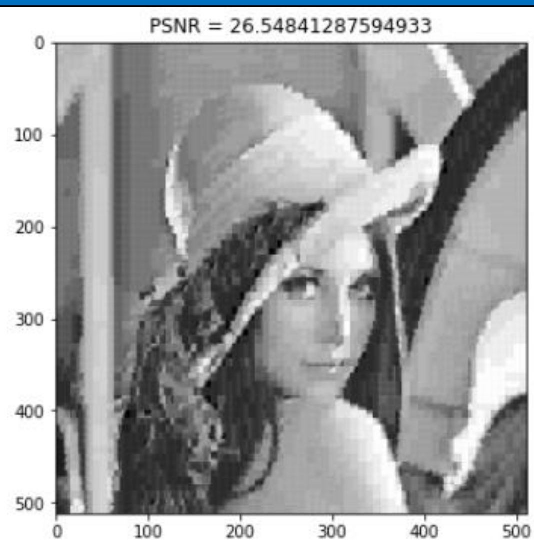


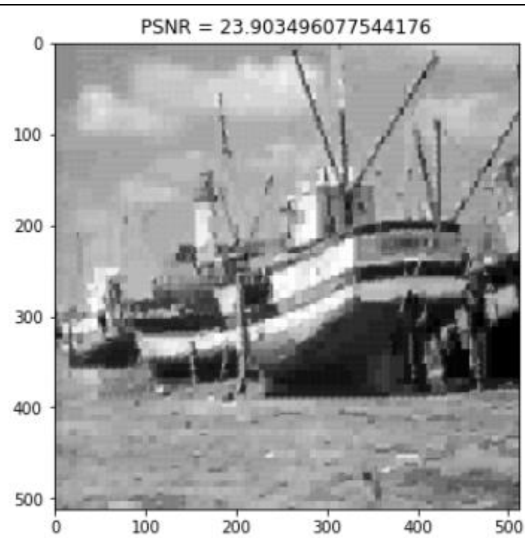
Lenna + Barbara As Training Image





Lenna + Barbara + Boat As Training Image





6) Best image as training image
   - All average.
     - ✓ Lenna got : 24.325
     - ✓ Barbara got : 25.378
     - ✓ Boat got : 25.335
     - ✓ Cameraman got : 25.672
   - Not Counted with train data.
     - ✓ Lenna got : 22.589
     - ✓ Barbara got : 25.491
     - ✓ Boat got : 25.088
     - ✓ Cameraman got : 24.032

Discussion

From the experiment we can discuss every section as follows.

1) Effect on the codebook size

   From the codebook size experiment we can see that when we increase the size of the codebook, the PSNR will be higher. This means that increasing the codebook will leads into better image quality. The factor that affecting this condition is when we have large codebook means that we have more codevector so every block will have more choice to choose the best match of the vector.
2) Effect on different epsilon (threshold)

   Different epsilon can give different result but can't give some certain effect. We can see from the experiment when we compared with epsilon 0.005 with 0.0005, the 0.0005 give better image quality. However very contrasted when we compared with 0.00005. The 0.00005 should give better PSNR however the 0.0005 beaten 0.00005. Remember that in the algorithm the epsilon is used to stop the iteration when the error is produce and also for splitting process. I believe maybe because not in all condition smaller epsilon will be better, but in some cases smaller will be better like we compare 0.005 with 0.0005. We can said that every image has their matched threshold. So to avoid this cases I think we should use more data to train and choose smaller epsilon.
3) Effect on the block size

   The effect of the block size is very clearer that when we reduce the block size, the feature will be smaller so every block comparison will have higher match probability rather than bigger feature vector. However in the case like (2x2) we can see that it's PSNR is really higher compared with others. I found that this is because block effect. When we giving bigger block size but we doesn't give bigger image size, blocky pattern will be introduce which can reduce the PSNR. So I can said that when we choose higher block, we also must consider the image size. Choosing the smaller block will be good choice but remember when we choose to reduce the block also remember to increase the codebook size. Why? Because when we have smaller feature vector, we have higher probability that the image block matched with vector in codebook. So by increasing the codebook size it will have more variation of the block which can improve performance.

4) Using other image as Training and use the codebook for different image

   In this result we can’t conclude that only one type of the image is the best for other images. However I can said that some characteristic of the images will give better codebook. For example when the images have variety structure inside the block will lead to the variety training vector which easily attain the better generalized codebook.

5) Use more than 1 image to Train

   In this scenario I have tested one by one of the image added as training data. Pay attention in here why I didn’t add last image (cameraman.png) as training image is because I will let that image as testing data. So suppose if we train by generating codebook, it is better when our codebook can giving better generalization to another image. From the experiment by adding the number of image as training data can give better result for the last image. This result shows us that when we adding more training image the codebook will give better generalization compared with just one or smaller number of training image. Now the question is what type of the image is the best for training image.

6) Best image as training image

   From the result we can see that Barbara image is the best type image for training because in the scenario it will give better generalization. I just define not counted means that the average taken from only test data not train data (image itself). This because when we average all result maybe the result is higher because the training data PSNR is higher. I think why Barbara image is better because in the image contains different type of illumination, details, and structure every block. While in the boat and cameraman maybe the sand and grass part will introduce the structure. In Lenna I believe that the Lenna’s hair give variety type of block but its not enough compared with Boat and Cameraman, especially with Barbara.

Conclusion

From all experiment and discussion we can conclude that the Vector Quantization especially Linde-Buzo-grey algorithm is powerful enough to compress the image data. Some things that we need to consider is the first large codebook size to get better image quality but slower process because we must search the best matched vector inside codebook. The second is block size, if we want to use larger block size we must consider the image size to avoiding blocky effect. The third is we can adding more data to train. This will leads to better codebook generalization, and we rally must care about the threshold because every image has different optimum threshold. However when we use more image it will be better to set threshold with smaller value. The forth that we must choose a better image type as training image especially the image which has a higher block variety because it will can forming better codebook compared with the homogeneous structure inside the image block.

Full code: https://github.com/herleeyandi/Image-Compression-with-Vector-Quantization

References

[1] http://www.data-compression.com/vq.shtml
[2] https://mkonrad.net/projects/gen_lloyd.html
[3] Multimedia Signal Processing Class, fall 2017, Prof. Jing Ming Guo
[4] http://mathworld.wolfram.com/VoronoiDiagram.html