Coursework on Texture Segmentation

1. Methods

The k-means algorithm is used to cluster different features in this experiment. The main principle is that first randomly pick k prototypes from the dataset, then cluster data by proximity to nearest prototype, next update each prototype with the mean of the set of data points in its cluster, repeat last two steps until the prototype does not change.

This experiment contains two layers. In the first layer, if natural images are RGB or RGBA images, they would be processed to create a 2D grayscale images by finding mean of RGB('A' channel would be ignored if it is the RGBA image). Each natural image is processed normalisation by Local Contrast Normalisation before extracting patches in order to get better results. After converting to grayscale and normalization, randomly extract P number of N*N patches in each image, rearrange patches into vectors with N*N components (2D to 1D) and store P number of patches into an array X with P rows and N*N columns. Use method *vq.kmeans* to find K clusters in array X. These clusters define distinct features which are returned as variable 'codebook'. Then, extract every patch about each pixel in each image and store all patches into an array, then use method *vq.vq* to label each pixel with the nearest prototype which is the variable 'codebook' returned from method *vq.kmeans.*

The second layer is similar to the first layer, but use k-means algorithm for histograms instead of patches. Generally speaking, randomly extract P numbers of M*M patches in each image which is generated in the first layer, then, generate histograms of the labels from extracted patches, next, get variable 'codebook' by clustering histograms, then, extract every patch about each pixel in each image which is generated in the first layer and generate histograms of the labels from extracted patches again, last, label each pixel with nearest prototype. Explain in more detail, first, randomly extract P number of M*M patches in each image which is generated in the first layer, rearrange patches into vectors with M*M components (2D to 1D) and store P number of patches into an array X with P rows and M*M columns. Then, use method *np.bincount* for each row of X to generate histograms which represent the occurrence of each prototype generated in the last layer. Cluster the histograms using k-means, where K is the number of distinct textures observed in the source image. The variable 'codebook' would be generated by clustering histograms. Next, extract every patch about each pixel in each image and use method *np.bincount* again for each patch to generate histograms. At last, use method *vq.vq* to label each pixel with the nearest prototype which is the variable 'codebook'.

2. Experimentation

1. Change N

Patchwork1

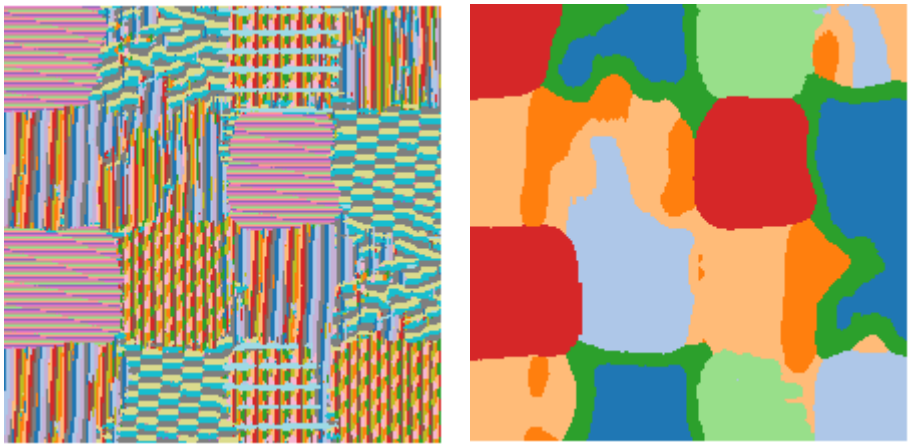

Figure 1: N = 20, M = 20, K = 20

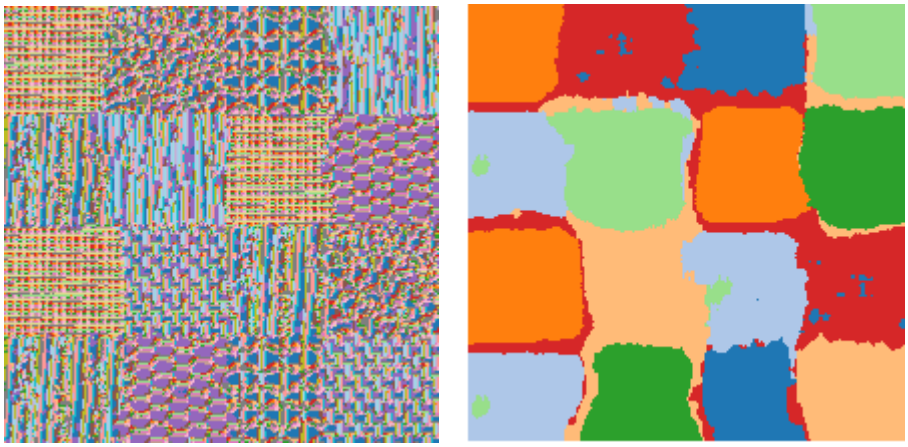

Figure 2: N = 4, M = 20, K = 20

Patchwork2

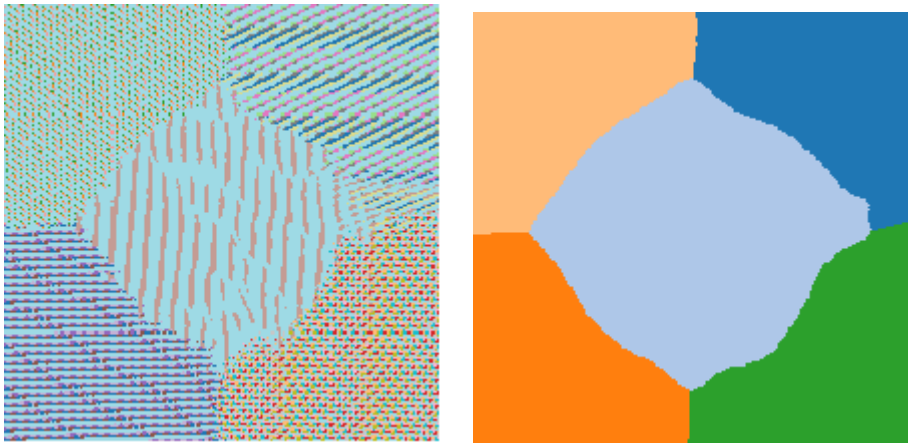

Figure 3: N = 40, M = 22, K = 40

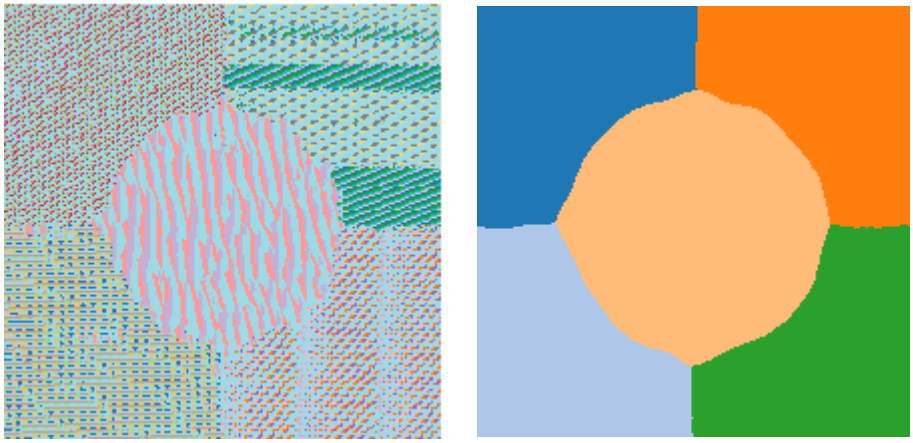

Figure 4: N = 20, M = 22, K = 40

If we change the value of N and keep the value of M and K the same, we can find that the larger the value of N, the more details the image loses both in the first and second layer. The smaller N means that the patch size is small which can learn features in small granularity. So that it can get better clustering both in the first and second layer. At the same time, the clustering is more efficient, because the size of patches is smaller.

Patchwork6 **Patchwork7**

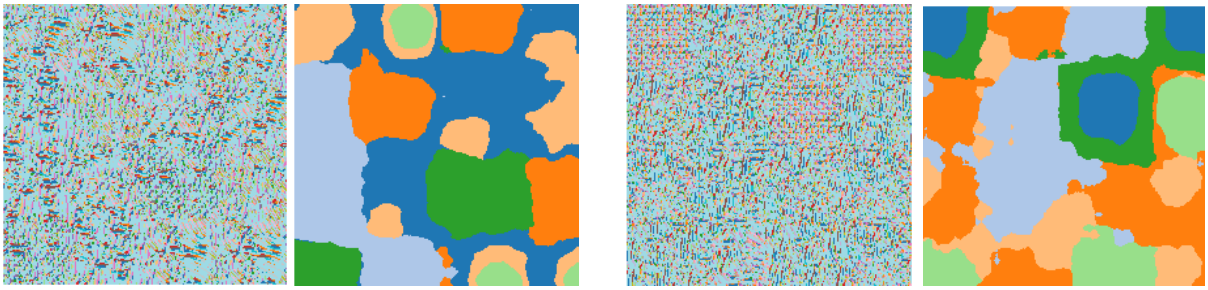

Figure 5: N = 20, M = 22, K = 40

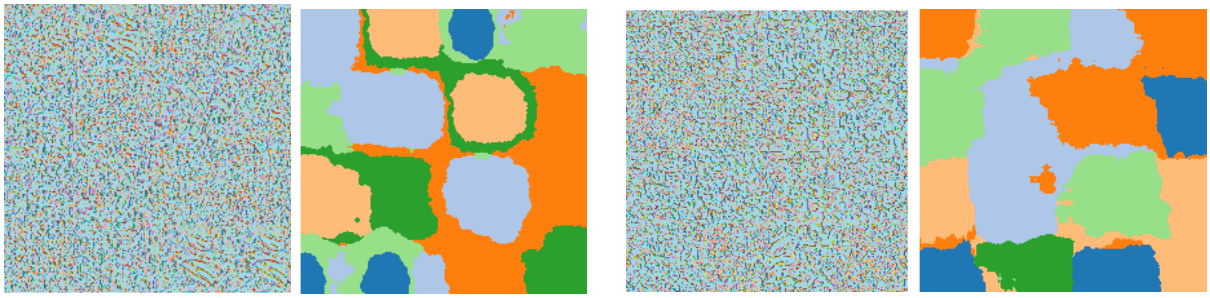

Figure 6: N = 6, M = 70, K = 40

Patchwork6 is similar to patchwork7. They have the same size and very similar textures. Those two images are larger and have more textures than the first and second images. In addition, those textures are similar. So, it's difficult to segment each texture clearly. But we still can see that Figure 6 is better than Figure 5 because it has more details. Obviously, the textures bleed colours more into each other in Figure 5.

2. Change M

Patchwork1

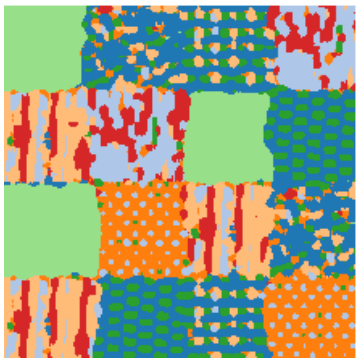

Figure 7:N=8,M=5,K=70 *N=8,M=25,K=70* *N=8,M=50,K=70*

M is tested by a range of value. Figure 7 shows that the smaller the value of N, the more details the image clusters. Especially, each texture area is not homogenous in the first image of Figure 7. It clusters more details so that it considers that each texture area contains more than one prototype. By contrast, in the last image of Figure 7 textures bleed colours into each other. As the second image displaying, the appropriate value of M is 25 which almost segments the different texture areas as homogenous areas.

Patchwork2



Figure 8:N=8,M=5,K=40 *N=8,M=25,K=40* *N=8,M=50,K=40*

Patchwork2 is similar to patchwork1. When the value of M equals 25, the image would be segmented appropriate.

Patchwork3

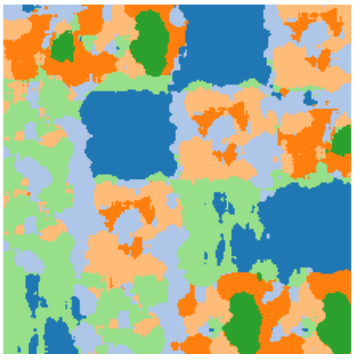

Figure 9:N=6,M=30,K=40 *N=6,M=70,K=40* *N=6,M=130,K=40*

Patchwork4

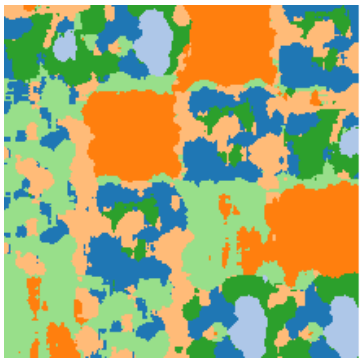
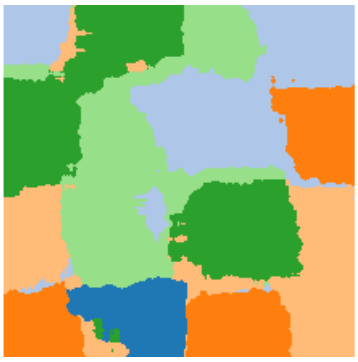


Figure 10:N=6,M=30,K=40 *N=6,M=70,K=40* *N=6,M=130,K=40*

Patchwork6 is similar to patchwork7. They have the same size and very similar textures. Those two images are difficult to be segmented clearly because their textures are similar to each other and contain more different texture areas. So, the value of M should be much larger.

3. Change K

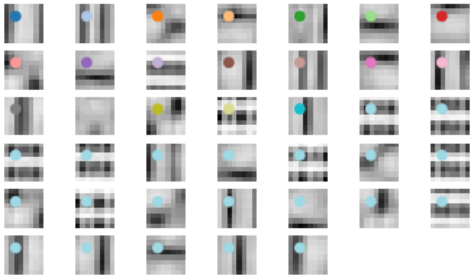
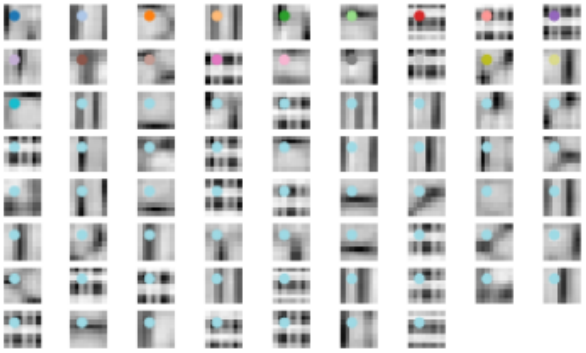

Figure 11: K=40 *K=70*

The K is the number of prototypes generated by method *vq.kmeans*. K cannot be too small or too big. If K is too small, it cannot learn efficient features. The image would lose some details. If K is too big, it would generate some similar prototypes. Those similar prototypes are useless and led that the

programme becomes inefficient. The value K is also the column number of histogram array in the second layer. When the histogram array is initiated, its column should be set as the value of K.

4. Final result

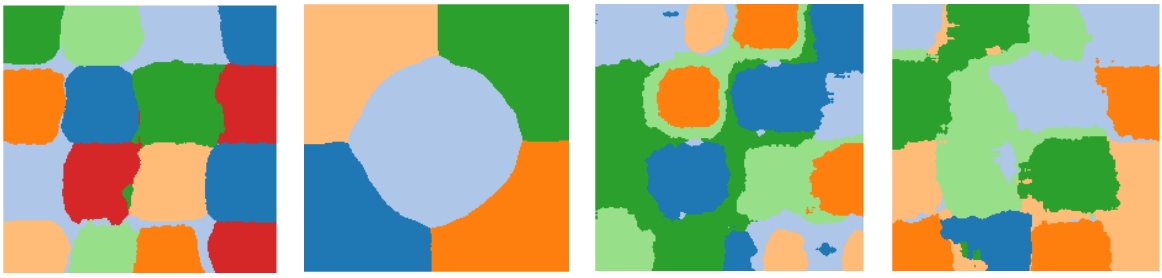

Figure 12: patchwork1 *patchwork2* *patchwork6* *patchwork7*

3. Improvement

If I have more time, I would try a different set of filters to preprocess image to improve the method. Image filtering can suppress the noise of the target image under the condition of retaining the details of the image as much as possible, and the effect of the processing will directly affect the effectiveness and reliability of subsequent image processing and analysis. The different set of filters can affect the segmentation accuracy of images with different textures. The filters can be intensity filters, gradient filters, Gabor filters or LoG filters.

4. Annex

```
import numpy as np
import scipy as sp
from scipy import ndimage
import matplotlib.pyplot as plt  # for imshow
import matplotlib.colors as colors
import matplotlib.patches as patches
from scipy.cluster import vq     # for k-means and vq
import random


def frist_layer(im, rescale, number_0f_dimention, N, K, P):
    if rescale == True:
        im = im / 255.0  # convert to float in interval [0 1]
    if number_0f_dimention == 3:
        gim = np.mean(im[:, :, 0:3], axis=2)  # find mean of RGB to create a 2D
grayscale image (could be RGBA)
    elif number_0f_dimention == 2:
        gim = im

    # perform local contrast normlisation
    sgim = ndimage.gaussian_filter(gim, 4)  # smooth the intensity image
('reflect' at boundaries)
    dev = (gim - sgim)
    V = ndimage.gaussian_filter(dev * dev, 4)  # smooth the variance
    gim = dev / np.maximum(np.sqrt(V), 0.1)

    sgim = ndimage.gaussian_filter(gim, 4)  # smooth the intensity image
('reflect' at boundaries)
```

```
    dev = (gim - sgim)
    V = ndimage.gaussian_filter(dev * dev, 4)  # smooth the variance
    gim = dev / np.maximum(np.sqrt(V), 0.1)

    sgim = ndimage.gaussian_filter(gim, 4)  # smooth the intensity image
('reflect' at boundaries)
    dev = (gim - sgim)
    V = ndimage.gaussian_filter(dev * dev, 4)  # smooth the variance
    gim = dev / np.maximum(np.sqrt(V), 0.1)

    R = gim.shape[0]  # number of rows in image
    C = gim.shape[1]  # number of columns in image

    X = np.zeros((P, N * N), dtype=float)  # initialise array for random
patches

    for i in range(0, P):
        r = random.randint(0, R - N)
        c = random.randint(0, C - N)
        patch = gim[r:r + N, c:c + N]
        X[i, :] = np.reshape(patch, (-1))

    codebook, distortion = vq.kmeans(X, K)

    spn = np.ceil(np.sqrt(K))  # size of subplot display
    norm = colors.Normalize(vmin=codebook.min(), vmax=codebook.max())  # set
gray range from minimum to maximum
    for i in range(0, K):
        plt.subplot(spn, spn, i + 1)
        plt.imshow(np.reshape(codebook[i, :], (N, N)), cmap='gray', norm=norm)
        plt.gca().add_patch(patches.Circle((2, 2), radius=1,
color=plt.cm.tab20(i)))
        plt.axis('off')  # turn off the axes

    X = np.zeros(((R - N) * (C - N), N * N), dtype=float)  # initialise array
for all patches
    i = 0
    for r in range(0, R - N):
        for c in range(0, C - N):
            X[i, :] = np.reshape(gim[r:r + N, c:c + N], (-1))
            i = i + 1

    code, dist = vq.vq(X, codebook)
    code = np.reshape(code, (R - N, C - N))  # reshape the 1D code array into
the original 2D image shape

    return code

def second_layer(gim, M, K, P, last_K):

    R = gim.shape[0]    # number of rows in image
    C = gim.shape[1]    # number of columns in image

    X = np.zeros((P,M*M),dtype=int) # initialise array for random patches

    for i in range(0, P):
        r = random.randint(0,R-M)
        c = random.randint(0,C-M)
        patch = gim[r:r+M,c:c+M]
        X[i,:] = np.reshape(patch,(-1))

    Y = np.zeros((P,last_K),dtype=int) # initialise array for histogram of P
random patches
    for i in range(0, P):
        histogram = np.bincount(X[i].reshape(-1),minlength=last_K)
        Y[i] = histogram
```

```
    codebook, distortion = vq.kmeans(Y/1.0,K)

    Z = np.zeros(((R-M)*(C-M),M*M),dtype=int)   # initialise array for all
patches
    i=0
    for r in range(0,R-M):
        for c in range(0,C-M):
            Z[i,:] = np.reshape(gim[r:r+M,c:c+M],(-1))
            i=i+1

    Q = np.zeros(((R-M)*(C-M),last_K),dtype=int) # initialise array for
histogram of all patches
    for i in range(0, (R-M)*(C-M)):
        histogram = np.bincount(Z[i].reshape(-1),minlength=last_K)
        Q[i] = histogram

    code, dist = vq.vq(Q,codebook)
    code = np.reshape(code,(R-M,C-M))    # reshape the 1D code array into the
original 2D image shape

    return code

def show_image(im):
    plt.figure()
    norm = colors.NoNorm()
    plt.imshow(im, cmap='tab20', norm=norm)
    plt.axis('off')    # turn off the axes


im = plt.imread('patchwork1.jpg')
image_from_1st_layer = frist_layer(im, True, im.ndim, 8, 70, 20000)
show_image(image_from_1st_layer)
image_from_2st_layer = second_layer(image_from_1st_layer, 25, 7, 20000, 70)
show_image(image_from_2st_layer)

im = plt.imread('patchwork2.pbm')
image_from_1st_layer = frist_layer(im, True, im.ndim, 20, 40, 20000)
show_image(image_from_1st_layer)
image_from_2st_layer = second_layer(image_from_1st_layer, 25, 5, 20000, 40)
show_image(image_from_2st_layer)

im = plt.imread('patchwork6.png')
image_from_1st_layer = frist_layer(im, False, im.ndim, 6, 40, 20000)
show_image(image_from_1st_layer)
image_from_2st_layer = second_layer(image_from_1st_layer, 70, 6, 20000, 40)
show_image(image_from_2st_layer)

im = plt.imread('patchwork7.png')
image_from_1st_layer = frist_layer(im, False, im.ndim, 6, 40, 20000)
show_image(image_from_1st_layer)
image_from_2st_layer = second_layer(image_from_1st_layer, 70, 6, 20000, 40)
show_image(image_from_2st_layer)
```