دانشکدگان فنی دانشگاه تهران
دانشکده مهندسی نقشه برداری و اطلاعات مکانی

تمرین segmentation

دانشجو:
علیرضا براهیمی

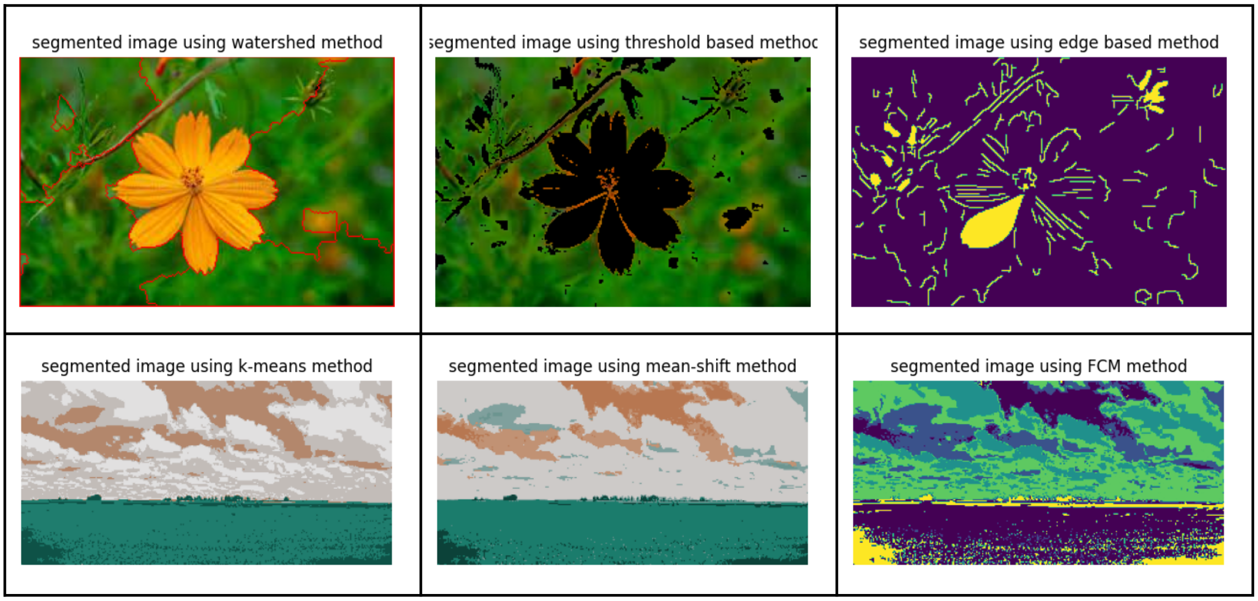
شماره دانشجویی:
810301017

استاد:
دکتر حسنلو

نیمسال اول سال تحصیلی 1401 – 1402

مقدمه :

تقسیم بندی تصویر روشی برای تقسیم یک تصویر دیجیتال به زیر گروه هایی به نام segment است که پیچیدگی تصویر را کاهش می دهد و امکان پردازش یا تجزیه و تحلیل بیشتر هر بخش تصویر را فراهم می کند.



segmented image using watershed method

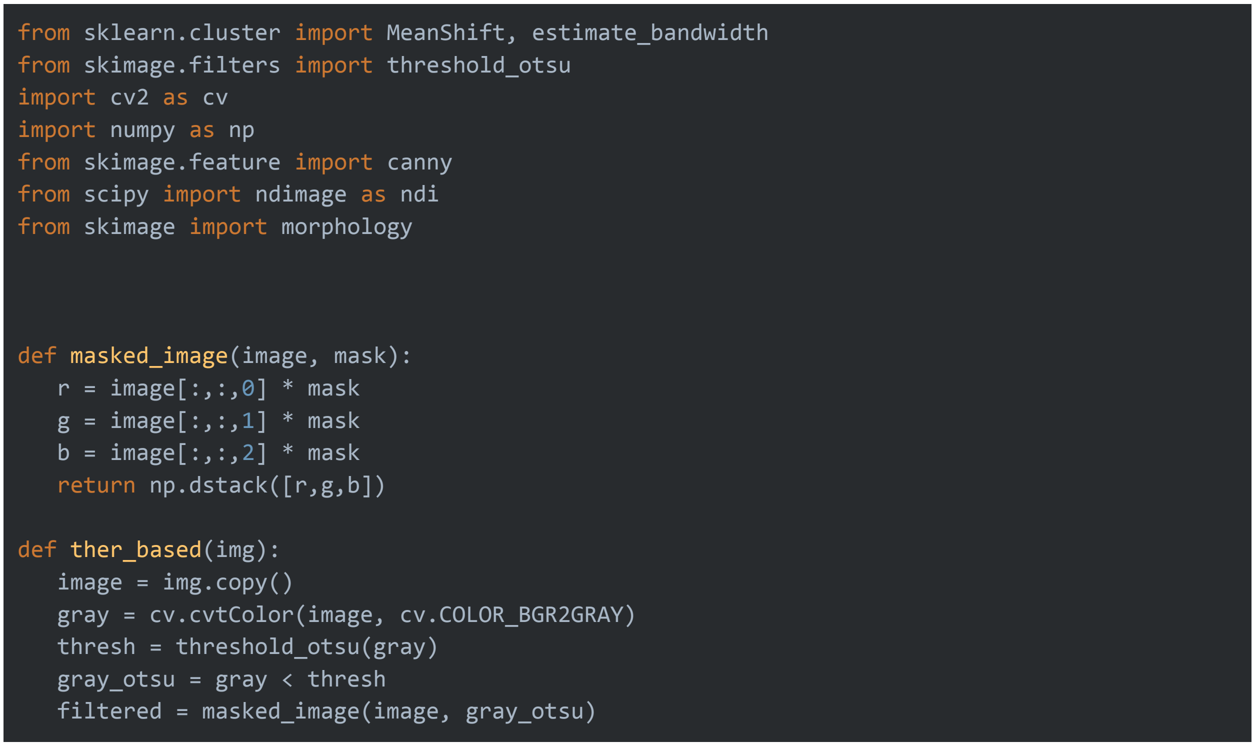
egmented image using threshold based methoc

segmented image using edge based method

segmented image using k-means method

segmented image using mean-shift method

segmented image using FCM method



```
from sklearn.cluster import MeanShift, estimate_bandwidth
from skimage.filters import threshold_otsu
import cv2 as cv
import numpy as np
from skimage.feature import canny
from scipy import ndimage as ndi
from skimage import morphology



def masked_image(image, mask):
   r = image[:,:,0] * mask
   g = image[:,:,1] * mask
   b = image[:,:,2] * mask
   return np.dstack([r,g,b])

def ther_based(img):
   image = img.copy()
   gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
   thresh = threshold_otsu(gray)
   gray_otsu = gray < thresh
   filtered = masked_image(image, gray_otsu)
```

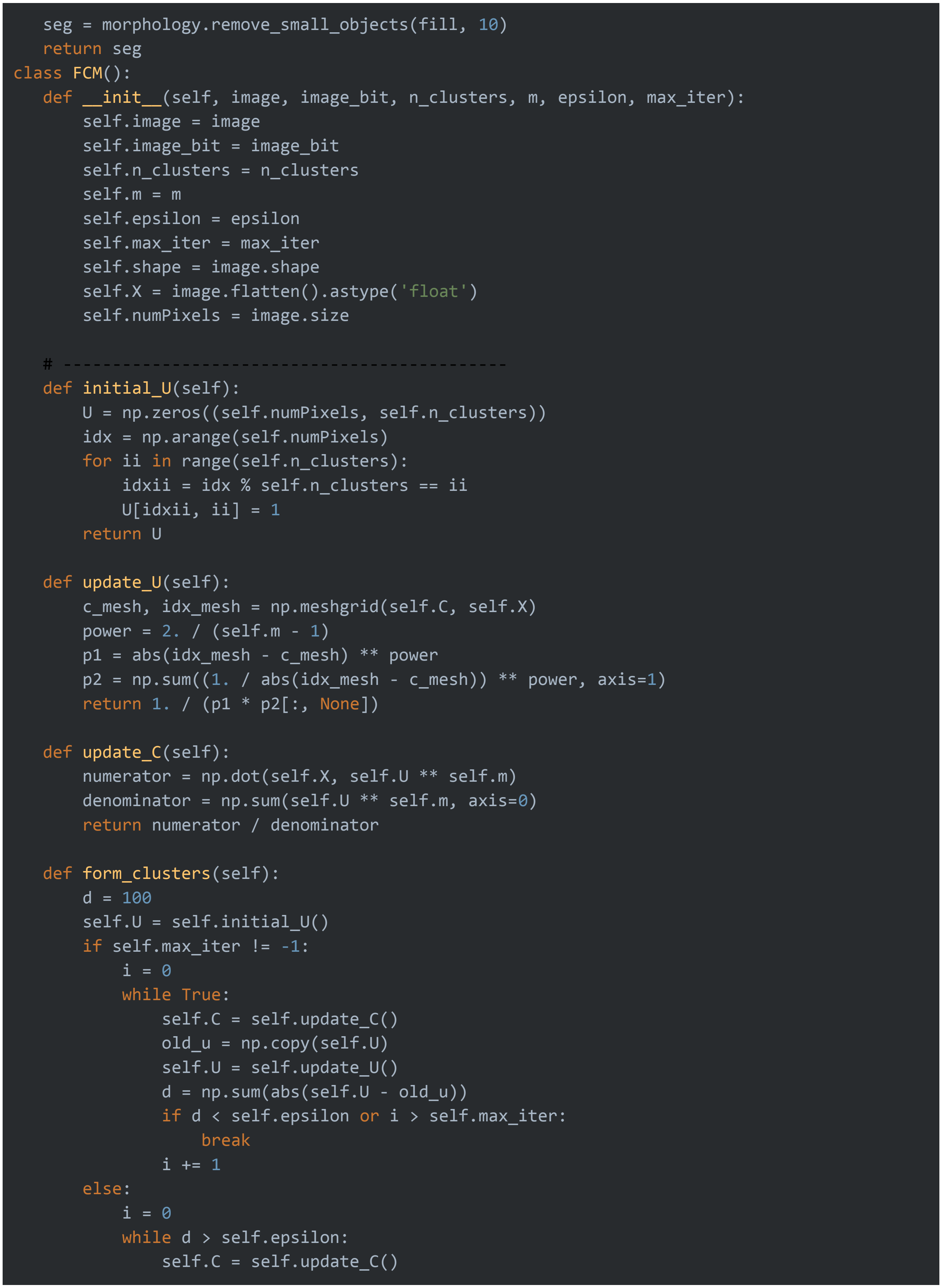
```python
   return filtered


def watershed(img):
   gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
   ret, thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU)
   kernel = np.ones((3, 3), np.uint8)
   opening = cv.morphologyEx(thresh, cv.MORPH_OPEN, kernel, iterations=2)
   sure_bg = cv.dilate(opening, kernel, iterations=3)
   dist_transform = cv.distanceTransform(opening, cv.DIST_L2, 5)
   ret, sure_fg = cv.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
   sure_fg = np.uint8(sure_fg)
   unknown = cv.subtract(sure_bg, sure_fg)
   ret, markers = cv.connectedComponents(sure_fg)
   markers = markers + 1
   markers[unknown == 255] = 0
   markers = cv.watershed(img, markers)
   img[markers == -1] = [255, 0, 0]
   return img


def k_means(img , numc):
   image = cv.cvtColor(img, cv.COLOR_BGR2RGB)
   pixel_values = image.reshape((-1, 3))
   pixel_values = np.float32(pixel_values)
   criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 0.2)
   _, labels, (centers) = cv.kmeans(pixel_values, numc, None, criteria, 10,
cv.KMEANS_RANDOM_CENTERS)
   centers = np.uint8(centers)
   labels = labels.flatten()
   segmented_image = centers[labels.flatten()]
   mask = segmented_image.reshape(image.shape)
   return mask

def mean_shift(img):
   img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
   originShape = img.shape
   flatimg = np.reshape(img, [-1, 3])
   bandwidth = estimate_bandwidth(flatimg, quantile=0.1, n_samples=100)
   ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
   ms.fit(flatimg)
   labels = ms.labels_
   cluster_centers = ms.cluster_centers_
   mask = cluster_centers[np.reshape(labels, originShape[:2])]
   mask = np.uint8(mask)
   return mask

def edge_based(img):
   img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
   image = img.copy()
   edge = canny(image)
   fill = ndi.binary_fill_holes(edge)
```



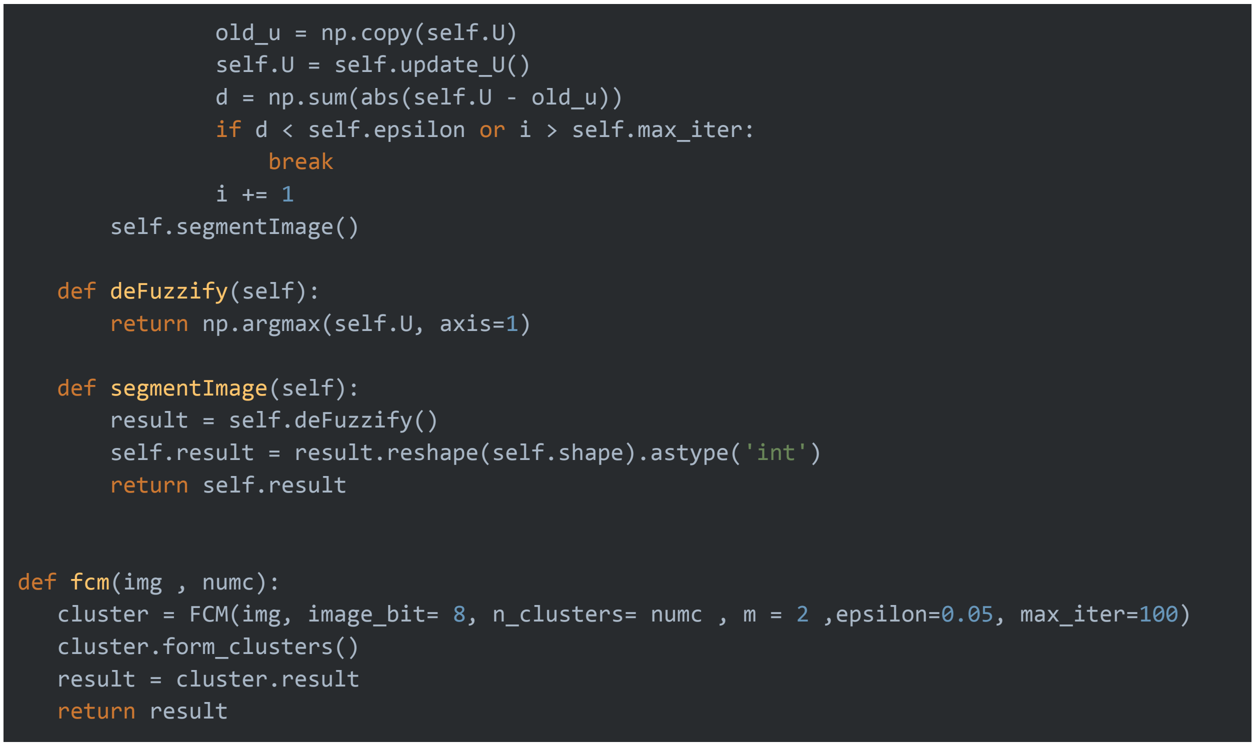
```
   seg = morphology.remove_small_objects(fill, 10)
   return seg
class FCM():
   def __init__(self, image, image_bit, n_clusters, m, epsilon, max_iter):
       self.image = image
       self.image_bit = image_bit
       self.n_clusters = n_clusters
       self.m = m
       self.epsilon = epsilon
       self.max_iter = max_iter
       self.shape = image.shape
       self.X = image.flatten().astype('float')
       self.numPixels = image.size

   # ---------------------------------------------
   def initial_U(self):
       U = np.zeros((self.numPixels, self.n_clusters))
       idx = np.arange(self.numPixels)
       for ii in range(self.n_clusters):
           idxii = idx % self.n_clusters == ii
           U[idxii, ii] = 1
       return U

   def update_U(self):
       c_mesh, idx_mesh = np.meshgrid(self.C, self.X)
       power = 2. / (self.m - 1)
       p1 = abs(idx_mesh - c_mesh) ** power
       p2 = np.sum((1. / abs(idx_mesh - c_mesh)) ** power, axis=1)
       return 1. / (p1 * p2[:, None])

   def update_C(self):
       numerator = np.dot(self.X, self.U ** self.m)
       denominator = np.sum(self.U ** self.m, axis=0)
       return numerator / denominator

   def form_clusters(self):
       d = 100
       self.U = self.initial_U()
       if self.max_iter != -1:
           i = 0
           while True:
               self.C = self.update_C()
               old_u = np.copy(self.U)
               self.U = self.update_U()
               d = np.sum(abs(self.U - old_u))
               if d < self.epsilon or i > self.max_iter:
                   break
               i += 1
       else:
           i = 0
           while d > self.epsilon:
               self.C = self.update_C()
```



```
               old_u = np.copy(self.U)
               self.U = self.update_U()
               d = np.sum(abs(self.U - old_u))
               if d < self.epsilon or i > self.max_iter:
                   break
               i += 1
       self.segmentImage()

   def deFuzzify(self):
       return np.argmax(self.U, axis=1)

   def segmentImage(self):
       result = self.deFuzzify()
       self.result = result.reshape(self.shape).astype('int')
       return self.result


def fcm(img , numc):
   cluster = FCM(img, image_bit= 8, n_clusters= numc , m = 2 ,epsilon=0.05, max_iter=100)
   cluster.form_clusters()
   result = cluster.result
   return result
```