Optional Projects

Yining Wen

1. Project 1

For project 1, the code name is project 1, I download all picture that needed and store all images in the folder “project1”. I was planning to do this project as same as the homework 4. I assert three classes for those pictures, they are building, sculpture, door. And I want to use histograms to classifier those pictures. But I meet some problems to implement the mean shift algorithm. So this project is not finished.

1. Project 2

For this project, I want to do it as the homework 4. The code name is “project 3”. First, I store all images in the folder “castle\_dense”, including the input images and output images. I use the image “0008.jpg” as the train image and the mask image is showing below.

A large white building

Description automatically generated

Mask1: mask2:

A large brick building

Description automatically generatedA close up of a device

Description automatically generated

I resize the image first, because they are so large, then as the homework, I use k-means to separate the different part and obtain 10 visual words for them. And classify those parts by finding the nearest word.

Here are the sample output images for those two mask, you can see all the results in the “castle\_dense” folder.

Mask1: mask2:

A large white building

Description automatically generatedA picture containing outdoor, building, sky

Description automatically generated

A large white building

Description automatically generatedA picture containing outdoor, sky, building, road

Description automatically generated

A close up of a white building

Description automatically generatedA large white building

Description automatically generated

A large brick building with a clock on the side of a road

Description automatically generatedA building with snow on the ground

Description automatically generated

1. Project 3

For project 3, I searched some code, and look how they implement it and trying to implement it myself, but I’m not sure I understand it.

1. Project 4

For project 4, I implement the SLIC with matlab which is “SLIC.m”, and implement the SNIC and the original SLIC with python, and their name are “SNIC.py” and “project4.py”. I use the homework 3 images as the input images.

First is my SLIC’s output:

A picture containing tree, outdoor, sky, grass

Description automatically generated

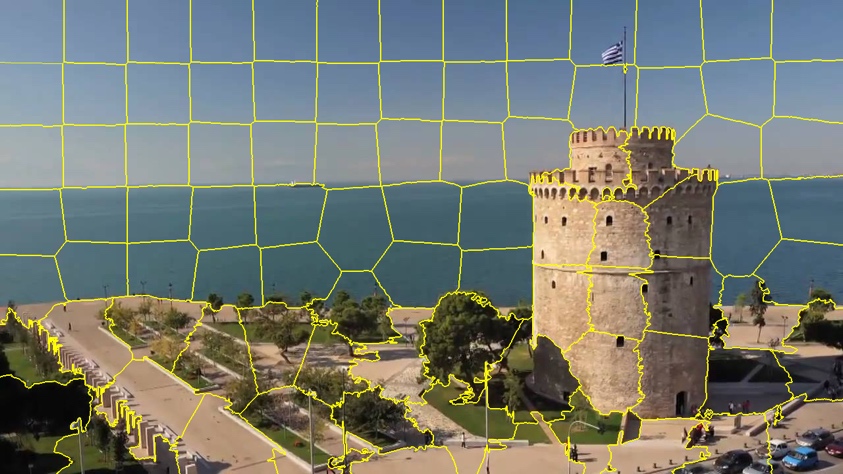
A close up of a fence

Description automatically generatedA picture containing map, text

Description automatically generatedThen is the SNIC’s output:

A picture containing grass, fence, tree

Description automatically generatedThose are the output of the SLIC in the package skimage:



For the result, SNIC use more time to process than the SLIC package. Both of them are better than my implement. And the SNIC has more unregular selection than the SLIC, such as the tree area in the image, it could handle more details than SLIC, but it is slower than SLIC, I think it is better than the SLIC.

1. Code

Project 1

clear all;

folder = 'project1/superset1/';

bin=4;

trainl=zeros(12,1);

trainh=zeros(12,bin\*3);

testl=zeros(12,1);

testh=zeros(12,bin\*3);

class={'building','sculpture','door'};

count=1;

for i=1:29

for j=1:length(class)

% training

train=imread([folder num2str(i) '.jpg']);

trainl(count) = j;

trainh(count,:) = [Histograms(train(:,:,1),bin), Histograms(train(:,:,2),bin), Histograms(train(:,:,3),bin)];

% testing

% test=imread([folder num2str(i) '.jpg']);

% testl(count) = j;

% testh(count,:) = [Histograms(test(:,:,1),bin), Histograms(test(:,:,2),bin), Histograms(test(:,:,3),bin)];

% count=count+1;

% end

end

% index2=MeanShift(trainh,testh,'k',1,'Distance','euclidean');

count=0;

for i=1:size(testh,1)

if testl(i)==trainl(index2(i))

count=count+1;

end

% disp(['Test image ' num2str(i) ' of class ' num2str(testl(i)) ' has been assigned to class ' num2str(testl(index2(i))) '.']);

end

Project 2

clear all;

folder = 'castle\_dense/';

traini=double(imread([folder '0008.jpg']));

maski=double(imread([folder 'mask2.jpg']));

trainimage = imresize(traini, 0.125);

maskimage = imresize(maski, 0.125);

car=[];

noncar=[];

carcolor = [];

carcolor(1,1,1) = 255;

carcolor(1,1,2) = 255;

carcolor(1,1,3) = 255;

paintcolor = [];

paintcolor(1,1,1) = 255;

paintcolor(1,1,2) = 255;

paintcolor(1,1,3) = 255;

skyindex=1;

nonskyindex=1;

for i=1:size(trainimage,1)

for j=1:size(trainimage,2)

if maskimage(i,j,:)== carcolor

car(skyindex,:)=trainimage(i,j,:);

skyindex=skyindex+1;

else

noncar(nonskyindex,:)=trainimage(i,j,:);

nonskyindex=nonskyindex+1;

end

end

end

% k-means

k=8;

[~,sw]=kmeans(car,k,'EmptyAction','singleton');

[~,nsw]=kmeans(noncar,k,'EmptyAction','singleton');

words=[ones(k,1) sw;zeros(k,1) nsw];

% testing

for n=1:9

test1=imread([folder '000' num2str(n) '.jpg']);

[s1,s2,s3] = size(test1);

test2=double(reshape(test1,s1\*s2,s3,1));

index3=knnsearch(words(:,2:end),test2,'k',1,'Distance','euclidean');

test3=words(index3,1);

[x,y]=ind2sub([s1 s2],1:s1\*s2);

% painting sky

for i=1:s1\*s2

if test3(i)==1

test1(x(i),y(i),:)= paintcolor;

end

end

figure,imshow(test1);

imwrite(test1, [folder 'nocar' num2str(n) '.jpg']);

end

for n=10:18

test1=imread([folder '00' num2str(n) '.jpg']);

[s1,s2,s3] = size(test1);

test2=double(reshape(test1,s1\*s2,s3,1));

index3=knnsearch(words(:,2:end),test2,'k',1,'Distance','euclidean');

test3=words(index3,1);

[x,y]=ind2sub([s1 s2],1:s1\*s2);

% painting sky

for i=1:s1\*s2

if test3(i)==1

test1(x(i),y(i),:)= paintcolor;

end

end

figure,imshow(test1);

imwrite(test1, [folder 'nocar' num2str(n) '.jpg']);

end

k-means

function image2=kmeans(image1,center,k)

h=size(image1,1);%height

w=size(image1,2);%width

m=size(image1,3);

% get RGB

center1=zeros(k,m);

center2=zeros(k,m);

for i=1:k

center1(i,1:m)=image1(center(i,1),center(i,2),:);

end

% initialize distance

distance=inf(h,w,k);

%calculate the distance

flag=0;

while ~flag

for i=1:k

for d=1:m

temp(:,:,d)=repmat(center1(i,d),h,w);

end

distance(:,:,i)=sqrt(sum(power(image1-temp,2),3));

end

% find out which closest center

[~,d] = min(distance,[],3);

% find the centroid of these points in each cluster

for i=1:k

index=d==i;

rgb=reshape(image1(repmat(index,1,1,m)),sum(index(:)),m);

center2(i,:)=round(mean(rgb));

end

if abs(norm(center1-center2))<0.1

flag=1;

end

center1=center2;

end

image2=zeros(h,w,m);

for i=1:h

for j=1:w

image2(i,j,:)=center1(d(i,j),:);

end

end

Filter

function image\_filter = f(image,filter)

%image info

iw = size(image,2);%image width

ih = size (image,1);%image height

%filter info

fw = size(filter,2);%filter width

fh = size(filter,1);%filter height

fhw = (fw - 1)/2;%filter half width

fhh = (fh - 1)/2;%filter half height

image1 = zeros(iw+fhw\*2, ih+fhh\*2);

for i = 1:iw

for j = 1:ih

image1(j+fhh, i+fhw) = image(j, i);

end

end

%replicate boundary

for i = 1:iw + fhw \* 2

for j = 1:ih + fhh \*2

if i<=fhw && j>=fhh && j<=ih+fhh

image1(j,i)=image1(j,fhw+1);

elseif i>=iw+fhw+1 && j>=fhh+1 && j<=ih+fhh

image1(j,i)=image1(j,fhw+iw);

elseif j<=fhh && i>=fhw+1 && i<=iw+fhw

image1(j,i)=image1(fhh,i);

elseif j>=ih+fhh+1 && i>=fhw+1 && i<=iw+fhw

image1(j,i)=image1(ih+fhh,i);

elseif i<=fhw && j<=fhh

image1(j,i)=image1(1+fhh,1+fhw);

elseif i<=fhw && j>=ih+fhh+1

image1(j,i)=image1(ih+fhh,1+fhw);

elseif j<=fhh && i>=iw+fhw+1

image1(j,i)=image1(1+fhh,iw+fhw);

elseif i>=iw+fhw+1 && j>=ih+fhh+1

image1(j,i)=image1(iw+fhh,iw+fhw);

end

end

end

%filter

image2 = image1;

for i = 1+fhw:iw+fhw

for j = 1+fhh:ih+fhh

image2(j,i)=sum(sum(filter.\*image1(j-fhh:j+fhh, i-fhw:i+fhw)));

end

end

image\_filter=zeros(ih,iw);

for i=1:iw

for j=1:ih

image\_filter(j,i)=image2(j+fhh,i+fhw);

end

end

Project 4

clear all;

% SLIC

folder = 'project4/';

image=imread([folder 'white-tower.png']);

h2=size(image,1);

w2=size(image,2);

image=double(image);

% parameter

s=50;

% slic

image1=SLIC(image,s);

image1=uint8(image1(:,:,1:3));

figure,imshow(image1);

imwrite(image1,[folder 'tower\_SLIC1.jpg']);

% color the pixel that touch two different clusters black

simage=image1;

for i=2:h2-1

for j=2:w2-1

flag=0;

for t=1:3

if image1(i,j,t)~=image1(i+1,j,t)

flag=1;

end

if image1(i,j,t)~=image1(i,j+1,t)

flag=1;

end

if image1(i,j,t)~=image1(i-1,j,t)

flag=1;

end

if image1(i,j,t)~=image1(i,j-1,t)

flag=1;

end

if image1(i,j,t)~=image1(i+1,j+1,t)

flag=1;

end

if image1(i,j,t)~=image1(i-1,j-1,t)

flag=1;

end

if image1(i,j,t)~=image1(i+1,j-1,t)

flag=1;

end

if image1(i,j,t)~=image1(i-1,j+1,t)

flag=1;

end

end

if flag==1

simage(i,j,:)=[0 0 0];

end

end

end

figure,imshow(simage);

imwrite(simage,[folder 'tower\_SLIC2.jpg']);

SLIC.m

function image2=SLIC(image1,s)

h=size(image1,1); % image height

w=size(image1,2); % image width

% gradient magnitude

sobel\_x=[-1,0,1;-2,0,2;-1,0,1];

sobel\_y=[1,2,1;0,0,0;-1,-2,-1];

gradient(:,:,1)=f(image1(:,:,1),sobel\_x);

gradient(:,:,2)=f(image1(:,:,2),sobel\_x);

gradient(:,:,3)=f(image1(:,:,3),sobel\_x);

gradient(:,:,4)=f(image1(:,:,1),sobel\_y);

gradient(:,:,5)=f(image1(:,:,2),sobel\_y);

gradient(:,:,6)=f(image1(:,:,3),sobel\_y);

gradient=sqrt(sum(power(gradient,2),3));

% initialize the centroids and move it to the position

% with the smallest gradient magnitude

count=1;

centroids=zeros(h,w);

for i=round((s+1)/2):s:h

for j=round((s+1)/2):s:w

window=gradient(i-1:i+1,j-1:j+1);

[~,small]=min(window(:));

[i2,j2]=ind2sub(size(window),small);

centroids(i+i2-2,j+j2-2)=1;

center(count,:)=[i+i2-2,j+j2-2];

count=count+1;

end

end

% divide x and y by 2

[x,y]=meshgrid(1:h,1:w);

x=x';

y=y';

image3(:,:,1:3)=image1;

image3(:,:,4)= x./2;

image3(:,:,5)= y./2;

% image3(:,:,4)= x.\*2;

% image3(:,:,5)= y.\*2;

% k-means

image2=kmeans(image3,center,count-1);

end

SNIC.py

from math import sqrt  
  
"""  
 Computes superpixels from a given image.  
 > reference = {Achanta, Radhakrishna, and Sabine Süsstrunk.   
 "Superpixels and polygons using simple non-iterative clustering."   
 Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. Ieee, 2017.}  
 {https://github.com/MoritzWillig/pysnic}  
 """  
  
class QueueElement(object):  
 \_\_sub\_index = 0  
  
 def \_\_init\_\_(self, dist, value):  
 self.\_key = dist  
 self.value = value  
  
 def \_\_lt\_\_(self, other):  
 *"""* ***:type*** *other: QueueElement  
 """* return self.\_key < other.\_key  
  
  
class Queue(object):  
 # define a priority queue  
 def \_\_init\_\_(self, \_buffer\_size=0):  
 self.heap = []  
  
 def add(self, priority, value):  
 heapq.heappush(self.heap, QueueElement(priority, value))  
  
 def is\_empty(self):  
 return len(self.heap) == 0  
  
 def pop\_value(self):  
 item = heapq.heappop(self.heap)  
 return item.value  
  
 def pop(self):  
 return heapq.heappop(self.heap)  
  
 def length(self):  
 return len(self.heap)  
  
  
def compute\_centroids(image, grid\_of\_x, grid\_of\_y):  
 # initialized coordinate and lab value of centroids  
 image\_size = np.array([len(image), len(image[0])])  
  
 step\_x = image\_size[0] // grid\_of\_x  
 step\_y = image\_size[1] // grid\_of\_y  
  
 centroids\_pos = np.array([[[  
 int(step\_x / 2 + x \* step\_x),  
 int(step\_y / 2 + y \* step\_y)  
 ] for x in range(grid\_of\_x)] for y in range(grid\_of\_y)])  
  
 centroids = []  
  
 for i in range(len(centroids\_pos)):  
 for j in range(len(centroids\_pos[0])):  
 pos = centroids\_pos[i][j]  
 centroid = [centroids\_pos[i][j], image[pos[0]][pos[1]], 0]  
 centroids.append(centroid)  
  
 return centroids  
  
  
def get\_neighbourhood\_pos(pos, image\_size):  
 # outputs candidates 1 pixel away from the image border.  
 n = 0  
 neighbourhood = [None, None, None, None]  
  
 x = pos[0]  
 y = pos[1]  
  
 if x - 1 >= 0:  
 neighbourhood[0] = [x - 1, y]  
 n += 1  
  
 if y - 1 >= 0:  
 neighbourhood[n] = [x, y - 1]  
 n += 1  
  
 if x + 1 < image\_size[0]:  
 neighbourhood[n] = [x + 1, y]  
 n += 1  
  
 if y + 1 < image\_size[1]:  
 neighbourhood[n] = [x, y + 1]  
 n += 1  
  
 return neighbourhood, n  
  
  
def update(centroid, candidate, num\_pixel):  
 # online update centroids value with candidates  
 return (centroid \* (num\_pixel - 1) + candidate) / num\_pixel  
  
  
def snic\_distance\_mod(pos\_i, pos\_j, col\_i, col\_j, s, m):  
 # Computes the SNIC pixel distance between i and j  
 # param s: normalization factor = 1 / np.sqrt(num\_pixels\_in\_image/num\_super\_pixels)  
 # param m: user provided, higher m leads to more compact superpixels and poorer boundary adherence  
  
 pos\_d = ((pos\_i[0] - pos\_j[0]) \*\* 2 + (pos\_i[1] - pos\_j[1]) \*\* 2) / s  
 col\_d = ((col\_i[0] - col\_j[0]) \*\* 2 + (col\_i[1] - col\_j[1]) \*\* 2 + (col\_i[1] - col\_j[1]) \*\* 2) / m  
 distance = pos\_d + col\_d  
 return distance  
  
  
def snic(  
 image,  
 grid\_of\_x,  
 grid\_of\_y,  
 compactness):  
  
 # initializa basic parameters  
 image\_size = np.array([len(image), len(image[0])])  
 label\_map = np.ones(image\_size) \* -1  
 s = sqrt(image\_size[0] \* image\_size[1] / (grid\_of\_x \* grid\_of\_y)) # normalization factor  
 m = compactness  
 centroids = compute\_centroids(image, grid\_of\_x, grid\_of\_y) # [position, color at position]  
  
 # create priority queue  
 queue = Queue(image\_size[0] \* image\_size[1] \* 4) # [position, color, centroid\_idx]  
 q\_add = queue.add  
 q\_pop = queue.pop  
  
 # create a priority queue first filled with the centroids itself.  
 for k in range(len(centroids)):  
 init\_centroid = centroids[k]  
 q\_add(-k, [init\_centroid[0], init\_centroid[1], k])  
  
 try:  
 # process until the queue is empty  
 while True:  
  
 # pop current processing pixels  
 item = q\_pop()  
 candidate = item.value  
 candidate\_pos = candidate[0]  
 candidate\_color = candidate[1]  
  
 # test if pixel is not already labeled  
 # since the key of first element in queue is 0, we set key of unlabeled pixel to be -1  
 if label\_map[candidate\_pos[0]][candidate\_pos[1]] == -1:  
 centroid\_idx = candidate[2]  
  
 # label new pixel  
 label\_map[candidate\_pos[0]][candidate\_pos[1]] = centroid\_idx  
  
 # online update of centroid  
 centroid = centroids[centroid\_idx]  
 num\_pixels = centroid[2] + 1  
  
 # update centroid position and color and the number of pixels corresponding to this superpixel  
 centroid[0] = update(centroid[0], candidate\_pos, num\_pixels)  
 centroid[1] = update(centroid[1], candidate\_color, num\_pixels)  
 centroid[2] = num\_pixels  
 centroids[centroid\_idx] = [centroid[0], centroid[1], centroid[2]]  
  
 # get four neighbour candidates to current processing pixel to queue  
 neighbours\_pos, neighbour\_num = get\_neighbourhood\_pos(candidate\_pos, image\_size)  
  
 # process neighbour pixels and add unlabeled ones into priority queue  
 for i in range(neighbour\_num):  
 neighbour\_pos = neighbours\_pos[i]  
 neighbour\_color = image[neighbour\_pos[0]][neighbour\_pos[1]]  
 if label\_map[neighbour\_pos[0]][neighbour\_pos[1]] == -1:  
 distance = snic\_distance\_mod(neighbour\_pos, centroid[0], neighbour\_color, centroid[1], s, m)  
 q\_add(distance, [neighbour\_pos, neighbour\_color, centroid\_idx])  
  
 except IndexError:  
 pass  
  
 return label\_map, centroids

Project4.py

from PIL import Image  
import numpy as np  
import time  
import skimage.color  
from skimage.segmentation import mark\_boundaries, find\_boundaries  
from skimage.segmentation import slic  
import imageio  
from SNIC import snic  
  
# load image  
image = np.array(Image.open('project4/white-tower.png'))  
image1 = np.array(Image.open('project4/wt\_slic.png'))  
lab\_image = skimage.color.rgb2lab(image)  
  
# SNIC parameters  
grid\_of\_x = 10  
grid\_of\_y = 10  
compactness = 40  
iteration = 10  
  
t1 = time.time()  
[label\_map\_snic, centroids\_snic] = snic(lab\_image, grid\_of\_x, grid\_of\_y, compactness,)  
t2 = time.time()  
label\_map\_snic = label\_map\_snic.astype(int)  
image\_seg\_snic = mark\_boundaries(image, np.array(label\_map\_snic))  
bd\_snic = find\_boundaries(np.array(label\_map\_snic)) \*1  
# img\_uint8 = image\_seg\_snic.astype(np.uint8)  
imageio.imwrite('tower\_snic.png', image\_seg\_snic)  
print('SNIC cost :', t2-t1)  
  
#SLIC used skimage  
segments = slic(image, n\_segments = 100, compactness = 40)  
t3 = time.time()  
slic\_skimage = mark\_boundaries(image, segments)  
bd\_slic\_skimage = find\_boundaries(np.array(segments)) \*1  
# img1\_uint8 = slic\_skimage.astype(np.uint8)  
t4 = time.time()  
imageio.imwrite('tower\_originalslic.png', slic\_skimage)  
print('Original SLIC cost:', t4-t3)