# CS 558 – Computer Vision

# Homework 2

# (Note : Run the main.mlx)

## Problem 1: K-means segmentation

clear;

%Problem Number 1

tower = imread( 'white-tower', 'png' );

imshow( tower )



The original input image for the program.

%k-means segmentation

k = 10

clusters = random\_centers( tower, k );

k is set to the number of different clusters that will be used in this program. An array of those cluster is initialized to random points in the image by choosing x and y randomly.

%Change the image into an array of RGBPoints

W = size(tower, 2);

H = size(tower, 1);

points(H, W) = RGBPoint( tower(H,W,1), tower(H,W,2), tower(H,W,3) );

for y = 1:H

for x = 1:W

points(y, x) = RGBPoint( tower(y,x,1), tower(y,x,2), tower(y,x,3) );

end

end

%Clean up the environment

clear tower;

The image is then changed from matrix of pixels into a matrix of point objects.

%Begin the while loop

iterate = true;

while( iterate )

%Find out which cluster each point is in and add that point to the working

%average of that cluster

cluster\_amount = zeros(k, 1);

cluster\_average = zeros(k, 3);

for y = 1:H

for x = 1:W

points(y, x) = points(y, x).find\_cluster( clusters );

c = points(y, x).getcluster();

cluster\_average(c, 1) = (cluster\_average(c, 1) \* cluster\_amount(c) + points(y,x).getr()) / (cluster\_amount(c) + 1);

cluster\_average(c, 2) = (cluster\_average(c, 2) \* cluster\_amount(c) + points(y,x).getg()) / (cluster\_amount(c) + 1);

cluster\_average(c, 3) = (cluster\_average(c, 3) \* cluster\_amount(c) + points(y,x).getb()) / (cluster\_amount(c) + 1);

cluster\_amount( c ) = cluster\_amount(c) + 1;

end

end

The number of points belonging to each cluster and the average RGB values of each cluster respectively is held by the variables cluster\_amount and cluster\_average. The RGBPoint.find\_cluster function takes in an array of cluster points and assigns the closest cluster to the point. After it has been determined that which cluster is closest to the point, the average RGB value of that cluster gets updated, and the amount of points which are contained in that cluster is incremented. After this , all points have been filtered into their respective cluster groups for this iteration of K-means.

%Create usable points out of the averages of each cluster

new\_clusters(k) = RGBPoint();

for i = 1:k

new\_clusters(i) = RGBPoint( round( cluster\_average(i, 1)), round( cluster\_average(i, 2)), round( cluster\_average(i, 3)) );

end

The values of the cluster centers for the next iteration of K-means is stored in new\_cluster. The average RGB values of each cluster are rounded off to an integer value.RGBPoint is initialized to hold those values.

thresh = 150; %If RGB-Distance between two points are below this threshold, we consider them the same point.

[iterate, distance\_array] = same\_clusters( clusters, new\_clusters, thresh );

iterate = ~iterate;

disp( "Distances between SSD of old clusters averages and new ones" );

disp( distance\_array );

clusters = new\_clusters;

end %End of while loop

Distance between each of the cluster centers in cluster and in new\_clusters is calculated at the end of K-mean iteration. This process is equivalent to calculating how far each of the centers have moved in RGB space. If any of the centers have moved more than thresh distance, they are considered as different points, otherwise they are considered as congruent. If no single center moved more than thresh, than break out of the while loop.

(Note : Sqaure root is not used in distance calculation , so the true threshold value is closer to the root of the thresh variable.)

clear distance\_array cluster\_amount cluster\_average i iterate new\_clusters thresh

disp( "These are the resulting clusters of the k-means algorithm." )

for i = 1:k

clusters(i).show()

end

image = zeros( H, W, 3, 'uint8');

for y = 1:H

for x = 1:W

c = points(y, x).getcluster();

image(y, x, 1) = uint8(clusters(c).getr());

image(y, x, 2) = uint8(clusters(c).getg());

image(y, x, 3) = uint8(clusters(c).getb());

end

end

imshow( image );

clear k clusters points c clusters H i image k points W x y

|  |  |
| --- | --- |
| The result of displaying all of the separate clusters that resulted from the K-means algorithm is shown to the right.  The resulting image is created. The RGB value of the cluster that the point belongs to is used instead of the RGB value of the point.  The resulting image from the algorithm is shown below. | These are the resulting clusters of the k-means algorithm.  R:114, G:137, B:163, C:0  R:77, G:69, B:49, C:0  R:91, G:117, B:141, C:0  R:82, G:90, B:87, C:0  R:28, G:25, B:18, C:0  R:155, G:141, B:129, C:0  R:100, G:126, B:158, C:0  R:137, G:155, B:170, C:0  R:200, G:171, B:144, C:0  R:81, G:103, B:111, C:0 |



## Problem 2 : SLIC

%%This is problem 2

clear;

SLIC = imread( 'wt\_slic.png', 'png' );

imshow( SLIC );

The input image.



S = 50; %The distance between the beginning superpixel groups

W = size(SLIC, 2);

H = size(SLIC, 1);

%Change the image into the point form we used before

points(W, H) = RGBPoint( SLIC(H,W,1), SLIC(H,W,2), SLIC(H,W,3) );

for y = 1:H

for x = 1:W

points(x, y) = RGBPoint( SLIC(y,x,1), SLIC(y,x,2), SLIC(y,x,3) );

end

end

clear SLIC

Instead of dealing with a matrix of 8-bit integers, Image is read into an array of RGBPoint objects.

%Initialize the first centers of the superpixels to be every 50 pixels

num\_super = ceil((H-S/2)/S) \* ceil((W-S/2)/S)

super\_pixels( num\_super ) = Point();

new\_super\_pixels( num\_super ) = FullPoint();

index = 1;

for y = S/2:S:H

for x = S/2:S:W

super\_pixels(index) = Point(x, y);

index = index + 1;

end

end

clear index

num\_super = 150

num\_supervalue is the result of a mathematical calculation of finding out the amount of superpixels in the image. This is calculated to allocate space for the necessary arrays beforehand for efficiency purpose. The superpixel centers are initialized into super pixel array as Point objects that hold x and y values.

%Check the gradient around each superpixel, and move to the minimum in a

%3x3 area

gradient = zeros(3, 3);

for i = 1:size(super\_pixels, 2)

%gradient 2,2 is where our pixel will be

px = super\_pixels(i).x;

py = super\_pixels(i).y;

for x = -1:+1

for y = -1:+1

if px-x > 1 && px-x < W && py-y > 1 && py-y < H

gradient(x+2, y+2) = ...

points(px-x-1, py).RGB\_distance( points(px-x+1, py) ) + ...

points(px, py-y-1).RGB\_distance( points(px, py-y+1));

end

end

end

%Gradient has been calculated for each point around the superpixel in a

%3x3 window

minx = 1;

miny = 1;

min\_distance = gradient(1, 1);

for x = 1:3

for y = 1:3

if gradient(x, y) < min\_distance

min\_distance = gradient(x, y);

minx = x;

miny = y;

end

end

end

%minx and miny point to the least value in gradient

new\_super\_pixels(i) = points( px+minx-2, py+miny-2).to\_full\_point(px+minx-2, py+miny-2).set\_cluster(i);

end

clear gradient min\_distance minx miny

super\_pixels = new\_super\_pixels;

Each superpixel has been iterated over. The gradient of each point in a 3 by 3 window around that superpixel is found, then the superpixel is moved to the position of the pixel with the minimum gradient value. The form of the superpixel center is changed to FullPoint objects. RGBxy values are present in FullPoint objects, and it has methods for calculating distance in five dimensions.

%All the superpixels are now set

%Run k-means in a 100 by 100 neighboorhood around each superpixel

iterate = true;

while( iterate )

distance = Inf( W, H ); %accessed by a y,x pair. Stores distance to nearest superpixel.

%Initialized to inf because initialliy, it's closest to no superpizel.

for i = 1:num\_super

%We do this for each superpixel

px = super\_pixels(i).x;

py = super\_pixels(i).y;

%We search in a 2S by 2S square for associated pixels

for x = px-S:px+S

for y = py-S:py+S

%Make sure the pixel we're testing is within image

%boundaries

if x > 0 && x <= W && y > 0 && y <= H

dis = super\_pixels(i).full\_distance( points(x, y), x, y );

if dis < distance(x, y)

%Set the cluster and distance to the new center

distance(x, y) = dis;

points(x,y) = points(x,y).set\_cluster(i);

end

end

end

end

end

For each superpixel, the neighborhood of the center for pixels that may belong to it is searched. The distance matrix contains the distance from each RGBPoint in points to the nearest superpixel cluster. This has been done because, in SLIC algorithm, iteration is done by superpixel to superpixel instead of by point to point, hence a single point cannot be compared to each superpixel at once. Since the superpixel each point belongs to can change in later iterations of the for loop, the superpixel averages in this run over the image cannot be calculated.

sup\_amount = zeros(num\_super, 1); %Holds the amount of pixels contained within each superpixel.

sup\_average = zeros(num\_super, 5); %Holds the average of R, G, B, x, y for each superpixel

for y = 1:H

for x = 1:W

%Find the averages of each superpixel so we can move them

c = points(x, y).getcluster();

sup\_average(c, 1) = (sup\_average(c, 1) \* sup\_amount(c) + points(x,y).getr()) / (sup\_amount(c) + 1);

sup\_average(c, 2) = (sup\_average(c, 2) \* sup\_amount(c) + points(x,y).getg()) / (sup\_amount(c) + 1);

sup\_average(c, 3) = (sup\_average(c, 3) \* sup\_amount(c) + points(x,y).getb()) / (sup\_amount(c) + 1);

sup\_average(c, 4) = (sup\_average(c, 4) \* sup\_amount(c) + x) / (sup\_amount(c) + 1);

sup\_average(c, 5) = (sup\_average(c, 5) \* sup\_amount(c) + y) / (sup\_amount(c) + 1);

sup\_amount( c ) = sup\_amount(c) + 1;

end

end

Iteration is done over each point in the image. Those points RGBxy values are averaged into the corresponding superpixel averages in the sup\_average. Sup\_amount has been used to store the amount of points in each superpixel .

%Create usable data out of the averages of each superpixel

new\_super\_pixels( num\_super ) = FullPoint();

distance\_arr = zeros( num\_super, 1);

for i = 1:num\_super

new\_super\_pixels(i) = FullPoint(...

round( sup\_average(i, 1)),...

round( sup\_average(i, 2)),...

round( sup\_average(i, 3)),...

round( sup\_average(i, 4)),...

round( sup\_average(i, 5)) );

distance\_arr( i ) = new\_super\_pixels(i).distance( super\_pixels(i) );

end

%Determine whether or not to iterate

thresh = 200;

if max( distance\_arr ) < thresh

iterate = false;

end

The new\_super\_pixels array is populated. This array contains, as FullPoint objects, what the new center of each superpixel will be in the next iteration. Each object in new\_super\_pixels contains RGBxy values. In each iteration a corresponding value is also added to the distance\_arr array. This array contains the distance each superpixel center moved in five-dimensional RGBxy space. If any superpixel has there center move more than thresh distance, than the while loop is used again with the super\_pixels values replaced with the new\_super\_pixels values.

(Note: The square root in distance calculation is not taken, so the true threshold value is closer to the root of the thresh variable).

%Set superpixels for the next iteration

super\_pixels = new\_super\_pixels;

end %End of k-means iteration

clear distance distance\_arr min\_distance new\_super\_pixels sup\_amount sup\_average

%super\_pixels contains the final versions of super\_pixels

%Each pixel in the points array has its cluster set accordingly

%Building the resulting image again

image = zeros( H, W, 3, 'uint8');

for y = 1:H

for x = 1:W

c = points(x, y).getcluster();

image(y, x, 1) = uint8(super\_pixels(c).getr());

image(y, x, 2) = uint8(super\_pixels(c).getg());

image(y, x, 3) = uint8(super\_pixels(c).getb());

end

end

%add the black lines between each cluster boundary

for y = 3:H

for x = 3:W

if points(x,y).getcluster() ~= points(x-1,y).getcluster() ||...

points(x,y).getcluster() ~= points(x,y-1).getcluster()

%If a cluster has changed, make this pixel black

image(y,x, 1) = uint8(0);

image(y,x, 2) = uint8(0);

image(y,x, 3) = uint8(0);

end

end

end

imshow( image );

disp( "SLIC Algorithm has finished" );

Once outside the while loop, the final image is prepared. The point array is read back into an 8-bit integer matrix. Then, each pixel in the image is iterated over, if it is in a different superpixel than its immediate neighbors, it is painted black to show superpixel boundaries. After showing the final image, it has been indicated that program has been finished.



SLIC Algorithm has finished

## Additional notes about Object/Method:

|  |  |
| --- | --- |
| Object/Method | Description |
| FullPoint.  distance | This function does not just calculate the 5-dimensional distance between two FullPoints in RGBxy space. It divides the xy values by 2 in order to make the RGB values hold more weight in the SLIC algorithm. |
| RGBPoint | This object stores each of its RGB and cluster value as 8-bit unsigned integers. Therefore, when k-means is run with more than 255 clusters, or a large enough image in SLIC that needs over 255 superpixels, than the program will crash. |
| RGBPoint.  find\_cluster() | This method, when given an array of RGBPoints, compares the distance, in RGB space, from itself to each of the RGBPoints. It then sets the cluster value of itself to the index of the closest point in the array. This method is used to find out in which cluster a point belongs to in k-means algorithm. |
| FullPoint.  distance()  And  RGBPoint.  RGB\_distance() | These two methods each, when given a point, computes the distance between the object point, and the parameter point, in either RGBxy or RGB space depending on class. Neither of these functions calculate true distance since they never use a square root function. Since distance values are only ever compared to other distance values, the same exact logical conclusion is still reached. The distance values in program are so large because the are essentially the algebraic square of the true distance values. |

## Problem 3 : Pixel Classification

%%Problem Number 3

clear;

train\_img = imread(“sky/sky\_train.jpg" );

mask\_img = imread(“sky/sky\_mask.jpg");

## Training image is selected.

mask = mask\_img == 255;

num\_sky\_mask = max( uint8(mask), [], 3);

sky\_mask = cat(3, num\_sky\_mask, num\_sky\_mask, num\_sky\_mask);

sky = bsxfun( @times, train\_img, uint8(sky\_mask));

points = reshape(sky, [], 3);

points(points(:, 3)== 0, :)= [];

[~, sky\_words] = imsegkmeans( reshape(points,1,[],3), 10 );

## Sky words now contains the array of points that describe the sky.

num\_ground\_mask = 1 - num\_sky\_mask;

ground\_mask = cat(3, num\_ground\_mask, num\_ground\_mask, num\_ground\_mask);

ground = bsxfun( @times, train\_img, uint8(ground\_mask));

points = reshape(ground, [], 3);

points(points(:, 3)== 0, :)= [];

[~, ground\_words] = imsegkmeans( reshape(points,1,[],3), 10 );

## Ground words now contains the array of points that describe the ground.

clear ground ground\_mask mask mask\_img num\_ground\_mask num\_sky\_mask points sky sky\_mask train\_img

## Ground\_words and sky\_words are now fully populated. Images are tested.

test\_paths = {'sky/sky\_test1.jpg', 'sky/sky\_test2.jpg', 'sky/sky\_test3.jpg', 'sky/sky\_test4.jpg'};

limit = length(test\_paths);

for i = 1:limit

disp( test\_paths{i}(5:end) );

image = sky\_mask(sky\_words, ground\_words, test\_paths{i} );

imshow( image );

figure;

drawnow;

end

## Output of the program :

## A picture containing circle Description automatically generated

## sky\_test1

## Map Description automatically generated

## sky\_test2

## Chart Description automatically generated with medium confidence

## Sky\_test3

## A picture containing map Description automatically generated

## sky\_test4