Brianne Trollo
CS558: Computer Vision
7 November 2019
Assignment 3

Problem 1: k-means Segmentation

For this implementation of k-means segmentation, k =10. First, the k-cluster centers are randomly chosen, while making sure that no two chosen points are the same or have the exact same RGB values. All of the pixels in the image are then assigned to the closest cluster. Next, the center of each cluster is recalculated. Then, if the new center does not match the original center, then the pixels are clustered again and the process repeats until the newly calculated center equals the original center. Finally, the resulting image is created by setting the RGB values of the pixels in each cluster to the average RGB values of the cluster.

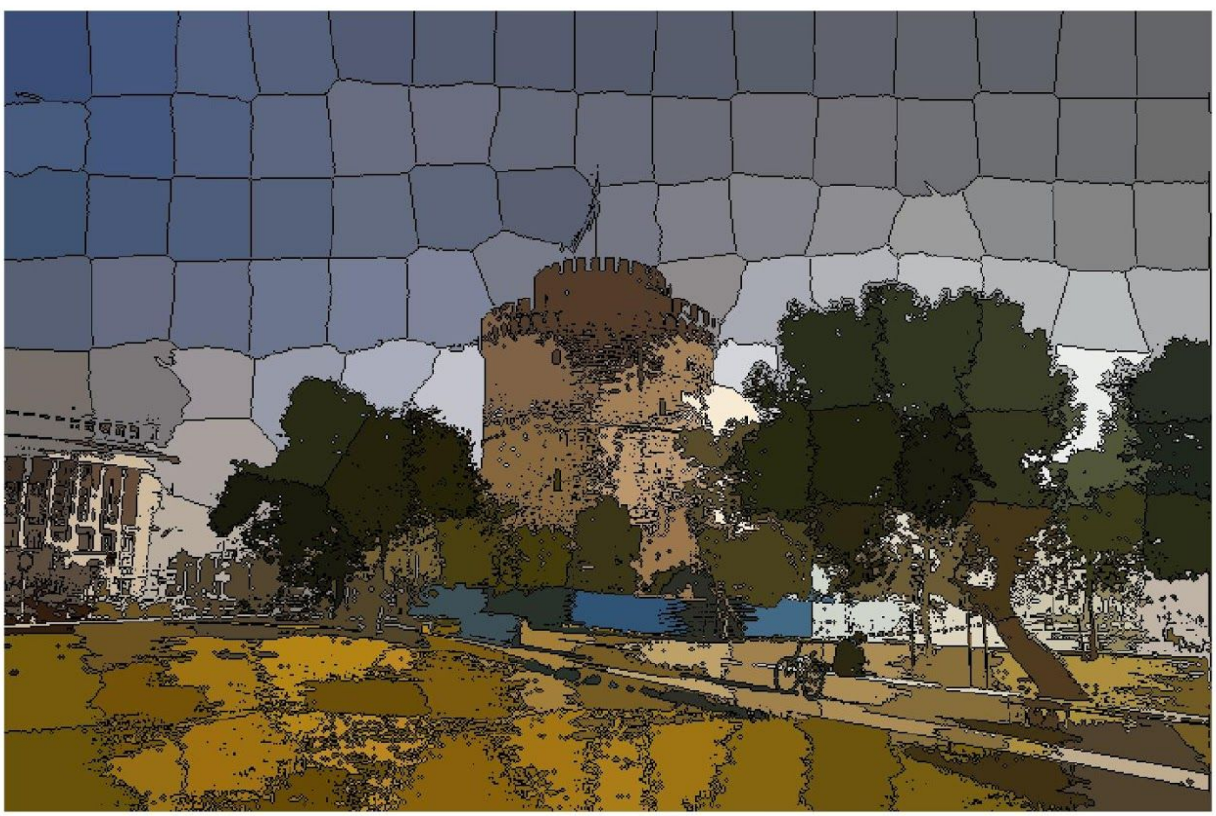
Problem 2: SLIC

For this implementation of SLIC, the initial segmentation of the image was done in blocks of 50x50. First the centroids are initialized as the center of each block. Then the gradients are computed in each of the RGB channels and the centroids are moved to the pixel with the smallest gradient that is a direct neighbor of the original centroid. All of the pixels in the image are then assigned to the cluster with the closest centroid using the 5D space of x, y, and RGB values. The problem stated that the x and y distance should be divided by 2, which can be seen in the first image below. Additionally, there are images included where the x and y distance were not modified and where the x and y distance were multiplied by 2. The optional step where pixels are only compared to centroids within a distance of 71 pixels is included when the second parameter in function call is 1, e.g. slic(img, 1). The centroids are then recomputed to be the average of the pixels in their clusters. If the centroids have changed, then the process repeats from computing the gradients. If the centroids have converged or the maximum number of iterations allowed has been reached, in this implementation it is 3, then the program computes the final image. The resulting image is created by taking the average of the RGB values of each cluster and setting all pixels to the average of their clusters, except for the pixels that are on the border of their cluster which are colored black. This black border around the clusters is 1 pixel wide by only coloring pixels on an edge that do not have a surrounding pixel in a different cluster is already colored black.

Distance modified: x and y divided by 2

Distance not modified

Distance modified: x and y multiplied by 2



Distance modified: x and y divided by 2 with the optional step

Code:

```
function main ()
%%%%%k-means
    k = 10;
    img = imread("white-tower.png");
    figure(1);
    imshow(img);
    result = k_means(k, img);
    figure(2);
    imshow(uint8(result));
    title("k-means");

%%%%%SLIC
% works with other image
% 1 for optional step (does not work for other image)

    img2 = imread("wt_slic.png");
    figure(3);
    imshow(img2);
    result = slic(img2, 0);
    figure(4);
    imshow(uint8(result));
    title("SLIC");

    result = slic(img2, 1);
    figure(5);
    imshow(uint8(result));
    title("SLIC with Optional Step");
end

function result = point_check(lst, rgb)
    % Parameters
        % lst -> matrix of rgb values
        % rgb -> rgb values to check
    % Return
        % result ->
            % 1 -> if x,y is NOT in lst
            % 0 -> if x,y is in lst

    l = size(lst);
    result = 1;
    r = rgb(1);
    g = rgb(2);
```

```
    b = rgb(3);
    for i = 1:l
        if lst(i, 1) == r && lst(i, 2) == g && lst(i, 3) == b
            result = 0;
            return;
        end
    end
    return;
end

function result = closest_cluster(img, px, py, c)
    % Parameters
        % img -> original image
        % px,py -> point
        % cx,cy -> cluster center
    % Return
        % result -> i of closest cluster ci
    closest_dist = inf;
    closest_c = 0;
    k = length(c);
    % Point r, g, b values
    prgb = img(px, py, :);
    pr = prgb(1);
    pg = prgb(2);
    pb = prgb(3);
    for i = 1:k
        % Cluster RGB values
        crgb = c(i, :);
        cr = crgb(1);
        cg = crgb(2);
        cb = crgb(3);

        % Calculate color distance
        r = (cr - pr)^2;
        g = (cg - pg)^2;
        b = (cb - pb)^2;
        dis = sqrt(r + g + b);
        if dis < closest_dist
            closest_dist = dis;
            closest_c = i;
        end
    end
    result = closest_c;
```

```
    return;
end

function result = check_ci(og, newc)
    % Parameters
        % og -> original cluster centers
        % newc -> new cluster centers
    % Return
        % result ->
            % 1 -> original and new cluster centers are the same
            % 0 -> original and new cluster centers are NOT the same
    og = sortrows(og);
    newc = sortrows(newc);
    result = isequal(og, newc);
    return;
end

function result = k_means(k, img)
    img = double(img);

    %Get size of image
    [X, Y, col] = size(img);

%%%%% STEP 1: Randomly initialize the cluster centers, c1, ..., ck

    % Initialize cluster centers
    cluster_centers = zeros(k, 3);

    % Randomly pick k clusters and get rgb values
    for c = 1:k
        px = randi([1, X], 1, 1);
        py = randi([1, Y], 1, 1);

        crgb = img(px, py, :);


        % Check that the cluster center has not already been chosen
        while point_check(cluster_centers, crgb) == 0
            px = randi([1, X], 1, 1);
            py = randi([1, Y], 1, 1);
            crgb = img(px, py, :);
        end
        cluster_centers(c, 1) = crgb(1);
```

```
        cluster_centers(c, 2) = crgb(2);
        cluster_centers(c, 3) = crgb(3);
    end

    flag = 0;
    while flag == 0
%%%%%%%% STEP 2: Given cluster centers, determine points in each cluster
%%%%%%%%         For each point p, find the closest ci. Put p into cluster i


        % initialize matrix for holding points per cluster
        % X*Y points, [x, y, ci]
        clustered = zeros(X*Y, 3);
        cnt = 1;
        for x=1:X
            for y=1:Y
                % Find closest cluster center
                ci = closest_cluster(img, x, y, cluster_centers);
                clustered(cnt, 3) = ci;
                clustered(cnt, 1) = x;
                clustered(cnt, 2) = y;
                cnt = cnt + 1;
            end
        end
%%%%%%%% STEP 3: Given points in each cluster, solve for ci
%%%%%%%%         Set ci to be the mean of points in cluster i
        new_cluster_centers = zeros(k, 3);
        for ci=1:k
            n = 0;
            sumr = 0;
            sumg = 0;
            sumb = 0;
            for i=1:(X*Y)
                if clustered(i, 3) == ci
                    n = n + 1;
                    % pixel coordinates from cluster ci
                    px = clustered(i,1);
                    py = clustered(i,2);
                    % RGB value from original image
                    rgb = img(px, py, :);
                    sumr = sumr + rgb(1);
                    sumg = sumg + rgb(2);
                    sumb = sumb + rgb(3);
```

```
            end
        end
        new_cluster_centers(ci, 1) = floor(sumr/n);
        new_cluster_centers(ci, 2) = floor(sumg/n);
        new_cluster_centers(ci, 3) = floor(sumb/n);
    end

%%%%%%%% STEP 4: If ci have changed, repeat STEP 2
    flag = check_ci(cluster_centers, new_cluster_centers);
    if flag == 0
        % Check for NaN values and
        % keep original center is new one is NaN

%           new_cluster_centers = remove_NaN(cluster_centers, new_cluster_centers, k);
        cluster_centers = new_cluster_centers;
    end
  end

  % Create final image - Represent each cluster with the average RGB
  % value of its members

  % Initialize resulting image
  c_img = zeros(X, Y, 3);
  % Calculate average RGB value of cluster
  c_avg_rgb = zeros(k, 3);
  for ci=1:k
    n = 0;
    sumr = 0;
    sumg = 0;
    sumb = 0;
    for i=1:(X*Y)
        if clustered(i, 3) == ci
            n = n + 1;
            % pixel coordinates from cluster ci
            px = clustered(i,1);
            py = clustered(i,2);
            % RGB value from original image
            rgb = img(px, py, :);
            sumr = sumr + rgb(1);
            sumg = sumg + rgb(2);
            sumb = sumb + rgb(3);
        end
        c_avg_rgb(ci, 1) = floor(sumr/n);
```

```
            c_avg_rgb(ci, 2) = floor(sumg/n);
            c_avg_rgb(ci, 3) = floor(sumb/n);
        end
    end

    % Set new image coordinates to RGB average of each cluster
    for ci=1:k
        for i=1:(X*Y)
            if clustered(i, 3) == ci
                x = clustered(i, 1);
                y = clustered(i, 2);
                r = c_avg_rgb(ci, 1);
                g = c_avg_rgb(ci, 2);
                b = c_avg_rgb(ci, 3);
                c_img(x, y, 1) = r;
                c_img(x, y, 2) = g;
                c_img(x, y, 3) = b;
            end
        end
    end

    result = c_img;
end

function mag = gradient_mag(a, b)
    % Parameters
        % a - single rgb channel value
        % b - single rgb channel value
    % Results
        % mag - magnitude of gradient

    mag = sqrt(a^2 + b^2);
    return;
end

function result = rbg_gradient(c, low, right)
    % Parameters
        % c - center x,y coordinates
        % low - x,y coordinates below c
        % right - x,y coordinates to right of c
    % Results
        % result - rgb gradient
```

```
    r = gradient_mag(c(1)-low(1), c(1)-right(1));
    g = gradient_mag(c(2)-low(2), c(2)-right(2));
    b = gradient_mag(c(3)-low(3), c(3)-right(3));
    result = sqrt(r^2 + g^2 + b^2);
    return;
end

function ci = fived_distance(n, centroids, x, y, rgb)
    % Parameters
        % n - number of centroids
        % centroids - list of centroids -> [ x y r g b ]
        % x - x-coordinate of image pixel
        % y - y-coordinate of image pixel
        % rgb - rgb values of image pixel -> [ r g b ]
    % Results
        % ci - index of closest centroid
    closest_dist = inf;
    closest_c = 0;

    % Point rgb values
    pr = rgb(1);
    pg = rgb(2);
    pb = rgb(3);
    for ci=1:n
        % Cluster RGB values
        cxyrgb = centroids(ci, :);
        cx = cxyrgb(1);
        cy = cxyrgb(2);
        cr = cxyrgb(3);
        cg = cxyrgb(4);
        cb = cxyrgb(5);

        % Calculate color distance
        r = (cr - pr)^2;
        g = (cg - pg)^2;
        b = (cb - pb)^2;
        % Calculate x,y distance divided by 2 (as stated in assignment)
        % (Also try with multiplied by 2 and no modifications)
        xd = (((cx - x)/2))^2;
        yd = (((cy - y)/2))^2;
        dis = sqrt(xd + yd + r + g + b);
        if dis < closest_dist
            closest_dist = dis;
```

```
            closest_c = ci;
        end
    end
    ci = closest_c;
end

function ci = fived_distance_op(n, centroids, x, y, rgb)
    % Parameters
        % n - number of centroids
        % centroids - list of centroids -> [ x y r g b ]
        % x - x-coordinate of image pixel
        % y - y-coordinate of image pixel
        % rgb - rgb values of image pixel -> [ r g b ]
    % Results
        % ci - index of closest centroid
    closest_dist = inf;
    closest_c = 0;

    % Point rgb values
    pr = rgb(1);
    pg = rgb(2);
    pb = rgb(3);
    for ci=1:n
        % Cluster RGB values
        cxyrgb = centroids(ci, :);
        cx = cxyrgb(1);
        cy = cxyrgb(2);
        cr = cxyrgb(3);
        cg = cxyrgb(4);
        cb = cxyrgb(5);

        % Calculate color distance
        r = (cr - pr)^2;
        g = (cg - pg)^2;
        b = (cb - pb)^2;
        % Calculate x,y distance divided by 2 (as stated in assignment)
        % (Also try with multiplied by 2 and no modifications)
        xd = (((cx - x)/2))^2;
        yd = (((cy - y)/2))^2;
        dis = sqrt(xd + yd + r + g + b);
        xydis = sqrt(xd + yd);
        if dis < closest_dist && xydis <= 71
            closest_dist = dis;
```

```
            closest_c = ci;
        end
    end
    ci = closest_c;
end

function result = cluster_edge(c_img, clusters, i, img)
    % Parameters
        % c_img - pixel ci marked at x,y pixel coordinate
        % clusters - [ ci x y r g b ]
        % i - clusters(i) = current pixel
        % img - resulting image
    % Results
        % result -
            % 0 - is not between two clusters
            % 1 - is the edge between two clusters
    [X,Y] = size(c_img);
    ci = clusters(i, 1);
    cx = clusters(i, 2);
    cy = clusters(i, 3);
    if cx-1 > 0 && c_img(cx-1, cy) ~= ci
        result = 1;
        rgb = img(cx-1, cy, :);
        if rgb(1) == 0 && rgb(2) == 0 && rgb(3) == 0
            result = 0;
            return;
        end
    elseif cx+1 <= X && c_img(cx+1, cy) ~= ci
        result = 1;
        rgb = img(cx+1, cy, :);
        if rgb(1) == 0 && rgb(2) == 0 && rgb(3) == 0
            result = 0;
            return;
        end
    elseif cy-1 > 0 && c_img(cx, cy-1) ~= ci
        result = 1;
        rgb = img(cx, cy-1, :);
        if rgb(1) == 0 && rgb(2) == 0 && rgb(3) == 0
            result = 0;
            return;
        end
    elseif cy+1 <= Y && c_img(cx, cy+1) ~= ci
        result = 1;
```

```
        rgb = img(cx, cy+1, :);
        if rgb(1) == 0 && rgb(2) == 0 && rgb(3) == 0
            result = 0;
            return;
        end
    else
        result = 0;
        return;
    end
end

function result = slic(img, option)
%     get size of image
    [X, Y, colors] = size(img);

%     Get img values as double
    img = double(img);

%     Set maximum iteration value
    max_iter = 3;

%%%%% STEP 1: Initialization: Divide the image in blocks of
%%%%% 50x50 pixels
    blocksize = 50;
    nblocks = (floor(X/blocksize))*(floor(Y/blocksize));

%%%%% initialize a centroid at the center of each block
    % centroids holds the (x,y) coordinates and rgb values
    % of the centroids
    % centroids - > [ x y r g b ]
    centroids = zeros(nblocks, 5);
    n = 1;
    for i=0:blocksize:X-1
        for j=0:blocksize:Y-1
            x= i+(blocksize/2);
            y=j+(blocksize/2);
            if x <= X && y <= Y
                ci_rgb=img(x,y,:);
                centroids(n, 1) = x;
                centroids(n, 2) = y;
                centroids(n, 3) = ci_rgb(1);
                centroids(n, 4) = ci_rgb(2);
                centroids(n, 5) = ci_rgb(3);
```

```
            end
            n = n + 1;
        end
    end


%%%%% STEP 2: Local Shift: Compute the magnitude of the
%%%%% gradient in each of the RGB channels
%   iteration count
    iter = 1;
%   flag for stopping loop
    flag = 0;
    while flag == 0
        magnitudes = zeros(X, Y);
        for i=1:X-1
            for j=1:Y-1
%%%%% use the square root of the sum of squares of the three
%%%%% magnitudes as the combined gradient magnitude
                magnitudes(i, j) = rbg_gradient(img(i, j, :), img(i+1, j, :), img(i, j+1, :));
            end
        end

%%%%% Move the centroids to the position with the smallest
%%%%% gradient magnitude in 3x3 windows centered on the
%%%%% initial centroids.
        for n=1:nblocks
            x = centroids(n,1);
            y = centroids(n,2);
            window = zeros(3,3);
            for wi=-1:1
                for wj=-1:1
                    if x+wi > X || y+wj > Y
                            window(wi+2, wj+2) = inf;
                    else
                        window(wi+2, wj+2) = magnitudes(x+wi, y+wj);
                    end

                end
            end
            % Find minimum
            win_min = min(min(window));
            [mx, my] = find(window == win_min);
            mx = mx(1) + x;
```

```
    my = my(1) + y;
    r = img(mx, my, 1);
    g = img(mx, my, 2);
    b = img(mx, my, 3);
    centroids(n, 1) = mx;
    centroids(n, 2) = my;
    centroids(n, 3) = r;
    centroids(n, 4) = g;
    centroids(n, 5) = b;
  end


%%%%% STEP 3: Centroid Update: Assign each pixel to its
%%%%% nearest centroid in the 5D space of x, y, R, G, B
%%%%% and recompute centroids. Use the Euclidean distance
%%%%% in this space, but divide x and y by 2.

  % Initialize clustered to hold pixel locations and rgb values
  % clustered -> [ ci x y r g b ]
  clustered = zeros(X*Y, 6);
  cluster_label_img = zeros(X,Y);
  n = 1;
  for i=1:X
    for j=1:Y
      %img x,y coordinates
      clustered(n, 2) = i;
      clustered(n, 3) = j;
      %pixel rgb values
      rgb = img(i, j, :);
      clustered(n, 4) = rgb(1);
      clustered(n, 5) = rgb(2);
      clustered(n, 6) = rgb(3);

      % compute 5D distance
      if option == 0
        ci = fived_distance(nblocks, centroids, i, j, rgb);
      else
%%%%% Optionally: only compare pixels to centroids
%%%%% within a distance of 71 pixels (~sqrt(2)*50 block size)
%%%%% during the updates.
        ci = fived_distance_op(nblocks, centroids, i, j, rgb);
      end
      clustered(n,1) = ci;
```

```
            cluster_label_img(i, j) = ci;

            n = n + 1;
        end
    end

    % recompute centroids (x,y) average is center
    % centroids - > [ x y r g b ]
    new_centroids = zeros(nblocks, 5);
    for ci=1:nblocks
        sumx = 0;
        sumy = 0;
        cnt = 0;
        for i=1:(X*Y)
            c = clustered(i, 1);
            if c == ci
                cnt = cnt + 1;
                sumx = sumx + clustered(i, 2);
                sumy = sumy + clustered(i, 3);
            end
        end
        new_x = floor(sumx/cnt);
        new_y = floor(sumy/cnt);
        if isnan(new_x)
            disp(ci);
        end
        new_centroids(ci, 1) = new_x;
        new_centroids(ci, 2) = new_y;
        new_rgb = img(new_x, new_y, :);
        new_centroids(ci, 3) = new_rgb(1);
        new_centroids(ci, 4) = new_rgb(2);
        new_centroids(ci, 5) = new_rgb(3);
    end

%%%%% STEP 4: If (not converged) and (iterations < max_iter)
%%%%% THEN go to STEP 2. max_iter = 3
    flag = check_ci(centroids, new_centroids);
    if flag == 0 && iter < max_iter
        centroids = new_centroids;
        iter = iter + 1;
    else
        flag = 1;
    end
```

```
    end

%%%%% STEP 5: Display the output image as in the SLIC slide:
%%%%% colorpixels that touch two different clusters black
%%%%% and the remaining pixels by the average RGB value of
%%%%% their cluster.

    % Initialize resulting image
    c_img = zeros(X, Y, 3);
    % Calculate average RGB value of cluster
    c_avg_rgb = zeros(nblocks, 3);
    for ci=1:nblocks
        n = 0;
        sumr = 0;
        sumg = 0;
        sumb = 0;
        for i=1:(X*Y)
            % clustered -> [ ci x y r g b ]
            if clustered(i, 1) == ci
                n = n + 1;
                % pixel coordinates from cluster ci
                px = clustered(i,2);
                py = clustered(i,3);
                % RGB value from original image
                rgb = img(px, py, :);
                sumr = sumr + rgb(1);
                sumg = sumg + rgb(2);
                sumb = sumb + rgb(3);
            end
            c_avg_rgb(ci, 1) = floor(sumr/n);
            c_avg_rgb(ci, 2) = floor(sumg/n);
            c_avg_rgb(ci, 3) = floor(sumb/n);
        end
    end

    % Set new image coordinates to RGB average of each cluster
    for ci=1:nblocks
        for i=1:(X*Y)
            if clustered(i, 1) == ci
                x = clustered(i, 2);
                y = clustered(i, 3);
                if cluster_edge(cluster_label_img, clustered, i, c_img) == 1
```

```
                c_img(x, y, 1) = 0;
                c_img(x, y, 2) = 0;
                c_img(x, y, 3) = 0;
            else

                r = c_avg_rgb(ci, 1);
                g = c_avg_rgb(ci, 2);
                b = c_avg_rgb(ci, 3);
                c_img(x, y, 1) = r;
                c_img(x, y, 2) = g;
                c_img(x, y, 3) = b;
            end
        end
    end
end

result = c_img;
end
```