CS484

Spring 2019

HW3 Report, Discussion, Code

Name: Unsal Ozturk

ID: 21601565

Section: 1

Part 1 – Superpixel Segmentation

Theoretical Discussion

For this part, I used the superpixel implementation provided by MATLAB, which is an implementation of the **SLIC0** algorithm. The reason why I did not use the algorithm provided in the project description is because I wanted to do everything in MATLAB as much as possible. The compactness measure for the compacting of the superpixels is not taken as a parameter in this implementation, and is calculated on the fly according to the pixel configuration. According to the MATLAB documentation, the compactness measure between two pixels are computed as follows:

$$d_{intensity} = \sqrt{(l_i - l_j)^2}$$

$$d_{spatial} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$D = \sqrt{\left(\frac{d_{intensity}}{m}\right)^2 + \left(\frac{d_{spatial}}{S}\right)^2}$$

This measure is then later normalized to the $[0.01, 1]$ interval for normalized compaction over the entirety of the image. The rest of this algorithm is the same as the SLIC version: It considers a grid step size of S, initializes cluster centers where the domain is R^5: The first three entries correspond to the entries in the LAB color space, and the last two entries are the coordinates of the pixel. The algorithm then shifts these cluster centers to the lowest position of the gradient in an nxn windows around the cluster center for robustness reasons. After this initialization, for every cluster center, in a $2Sx2S$ window around the cluster, the pixels are assigned to the cluster if the distance measure between the cluster center and the pixel is appropriate. After the assignments, clusters are reassigned according to the l_1 norm between previous cluster centers and recomputed cluster centers, and a residual error of the reassignment is calculated. The cluster assignment and error calculation steps are repeated until the residual error is below some desired value.

Parameters

The only parameter to be supplied to the algorithm is the expected number of superpixels i.e. the initial number of cluster centers. Note that a cluster may become empty during the execution of the algorithm due to the computed distance and compactness measures.

I chose $N = 2048$ for the number of superpixels ***for every image***, i.e. the algorithm will attempt to find 2048 superpixel clusters in each image. I chose this number because I think it is a good tradeoff between the computational complexity caused by high number of superpixels and the granularity of the results. The actual number of superpixels obtained from the algorithm for each image given below (retrieved from MATLAB workspace):

152_5219.JPG: 1939

152_5220.JPG: 1913

153_5341.JPG: 1937

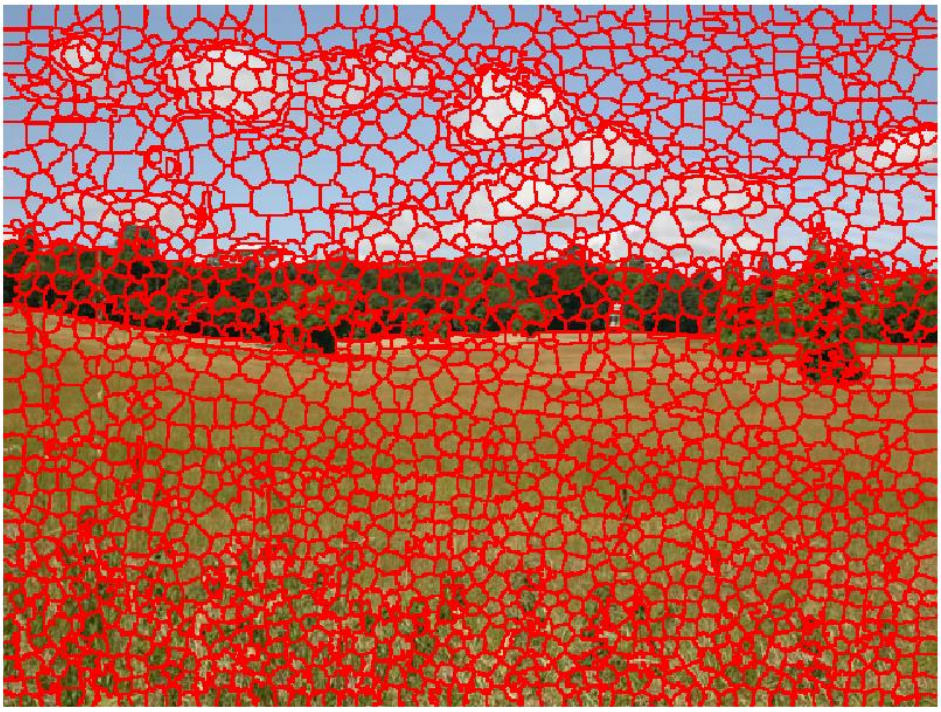
153_5396.JPG: 1937

157_5768.JPG: 1957

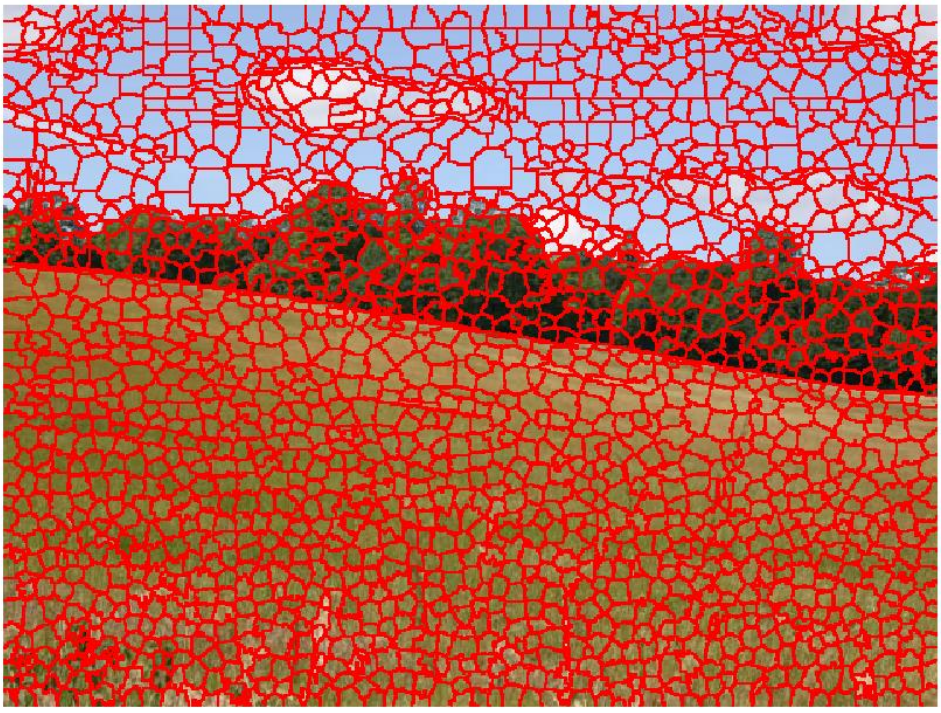
Results

The following images are the superpixel segmentations of the input images.

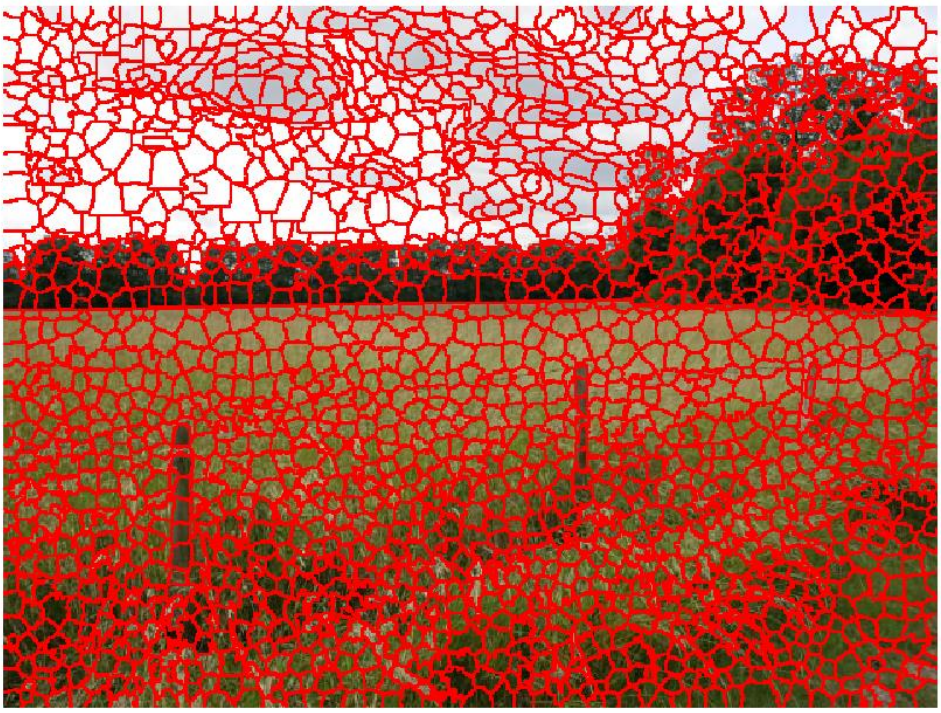
Superpixel Segmentation:152_5219.JPG



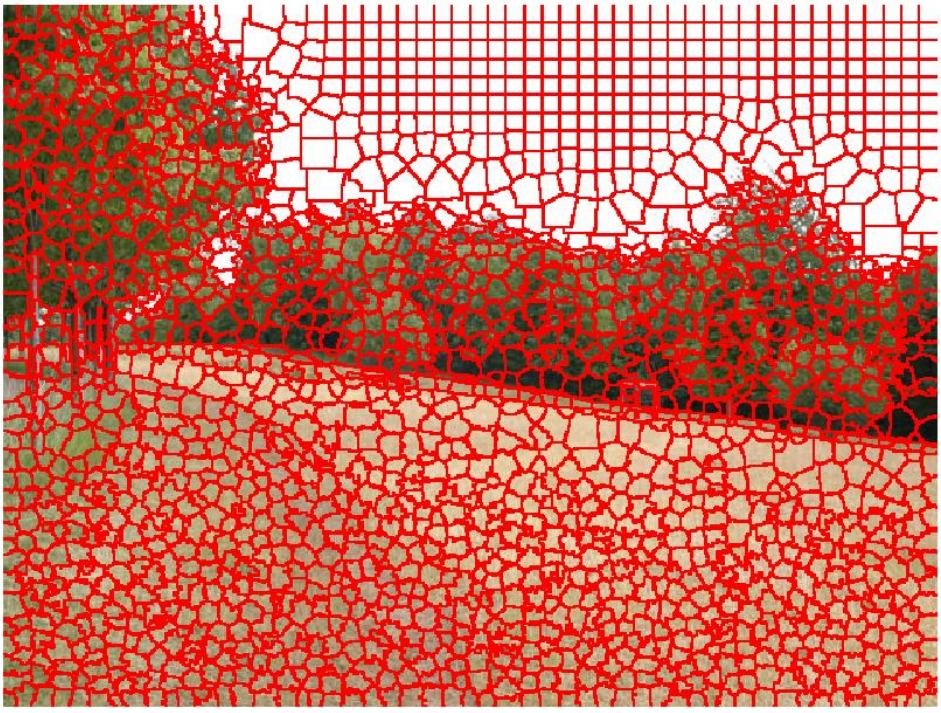
Superpixel Segmentation:152_5220.JPG



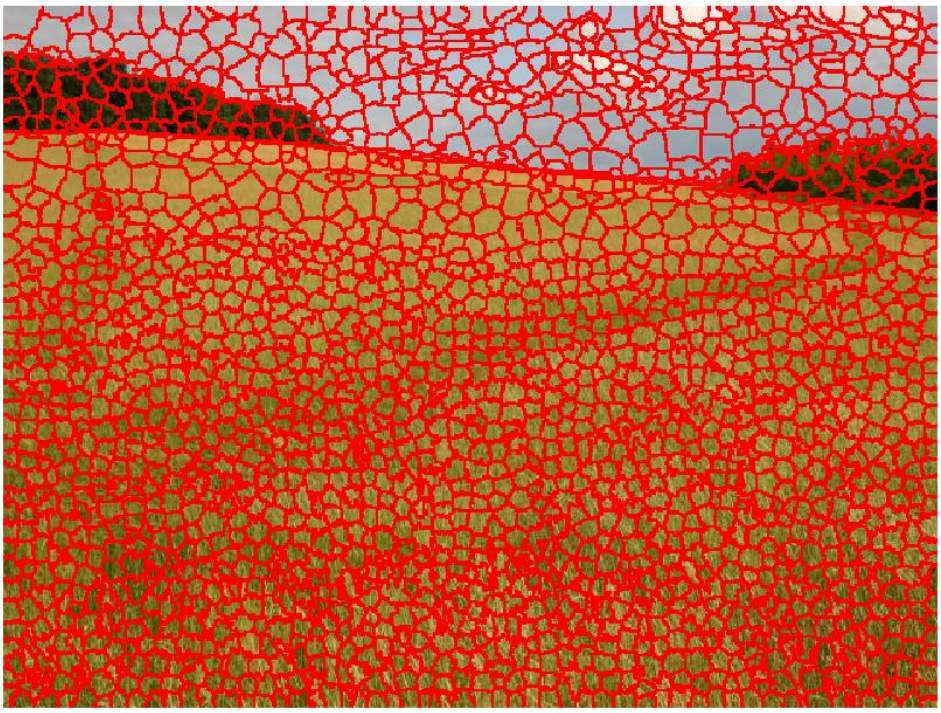
Superpixel Segmentation:153$_5$396.JPG



Superpixel Segmentation:153$_5$341.JPG



Superpixel Segmentation:157$_5$768.JPG



[This space was intentionally left blank]

Part 2 – Gabor Filter Design and Parameters

Theoretical Discussion

The following expression can be used to generate a real valued Gabor filter for a given orientation, scale, phase shift and "spatial scaling":

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = e^{-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}} \cos\left(\frac{2\pi x'}{\lambda} + \psi\right)$$

where

$$x' = xcos\theta + ysin\theta$$

$$y' = -xsin\theta + ycos\theta$$

i.e.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = R_\theta \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The parameters of this linear filter correspond to the following:

$$x, y: Spatial\ Location$$

$$\lambda: Wavelength$$

$$\theta: Orientation\ of\ the\ filter$$

$$\psi: Phase\ shift$$

$$\sigma = Standard\ deviation\ of\ the\ Gaussian\ (scale)$$

$$\gamma: Aspect\ Ratio$$

Filter Design and Parameters

In the design of the filters, I considered all of the parameters above. The effect of x, y are automatically achieved via the construction of the filter itself. I set λ to 10.5 for all filters, and I determined the value of this through trial and error, and I achieved the best result at that particular value. θ was chosen to be $0^\circ, 45^\circ, 90^\circ, 135^\circ$, and σ was chosen to be $1, 2, 3, 4$ respectively to obtain a Gabor bank of 16 filters, with these particular orientations and scales. γ was chosen to be 1, but could also have been ideally set to $\frac{480}{640}$ or its reciprocal. I chose 0 for the phase shift.

To sum up, I generated the following 16 filters

$$g\big(x, y; \lambda = 10.5, \theta_i, \psi = 0, \sigma_j, \gamma = 1\big)$$

where

$$\theta_i \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$$

$$\sigma_j \in \{1, 2, 3, 4\}$$

Implementation

MATLAB's **imgaborfilt** implementation does not support the manipulation of the above parameters when it comes to obtaining filter banks. The implementation by Dr. Peter Kovesi also does not allow the individual setting of the parameters of a given bank, but it rather generates a general set of filters by multiplying the scale and rotating the filter for the Cartesian product of the sets of orientation and scale.

Therefore, I used the implementation provided in the following link, which literally implements the Gabor filter as described in the theoretical discussion section.

https://uk.mathworks.com/matlabcentral/fileexchange/28751-gabor-filtering-on-an-image

Using the function provided in this implementation, I obtained a 16-bank filter and the filtered versions of the images via the following MATLAB script.

```
function out = gabor_bank(image, sigma, theta)
out = zeros([size(image), length(sigma)*length(theta)]);
    for i=1:length(sigma)
        for j = 1:length(theta)
            filter = gabor_fn(sigma(i),theta(j),10.5,0,1);
            out(:,:,(i-1)*length(theta)+j) = conv2(image,double(filter),'same');
        end
    end
end
```

Example Filtered Images

The following images are the Gabor filter responses of the images to Gabor filters with different orientations and scales. Note that higher scale values capture finer features, and different orientations capture higher responses from the image if the said features are oriented the same way.

This report contains 32 images from the entire set of pictures. The first 16 correspond to the responses of the first image to the filters, and the rest the second image. The rest of the responses are under the directory out/gabor in the root file of the project.

Gabor Response for Image152$_5$219.JPG, Scale1 Orientation0

Gabor Response for Image152$_5$219.JPG, Scale1 Orientation45

Gabor Response for Image152$_5$219.JPG, Scale1 Orientation90

Gabor Response for Image152$_5$219.JPG, Scale1 Orientation135

Gabor Response for Image152$_5$219.JPG, Scale2 Orientation0

Gabor Response for Image152$_5$219.JPG, Scale2 Orientation45

Gabor Response for Image152$_5$219.JPG, Scale2 Orientation90

Gabor Response for Image152$_5$219.JPG, Scale2 Orientation135

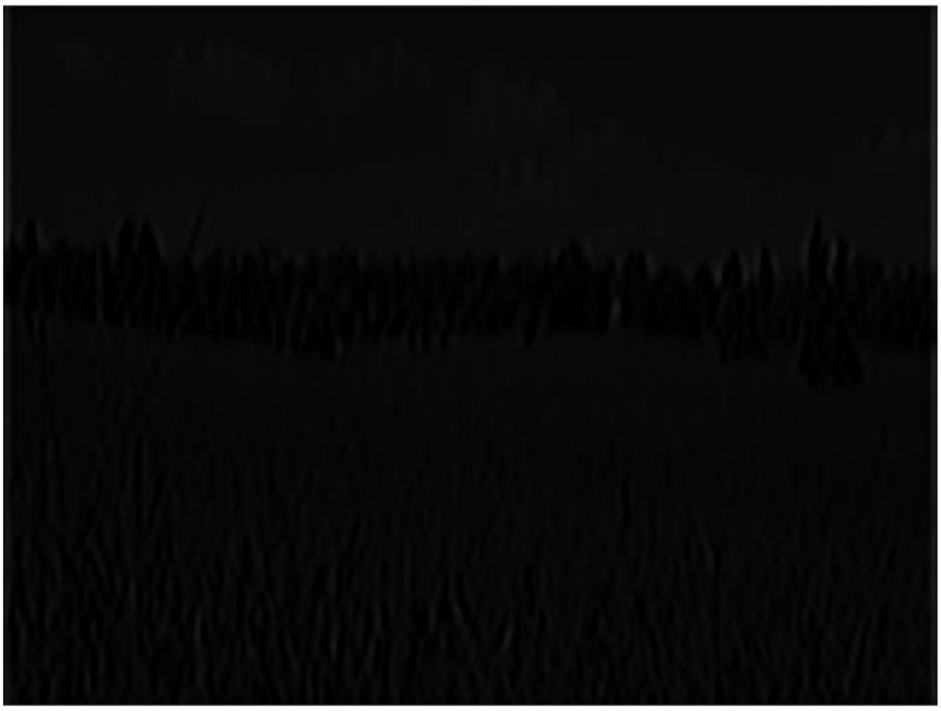
Gabor Response for Image152$_5$219.JPG, Scale3 Orientation0

Gabor Response for Image152$_5$219.JPG, Scale3 Orientation45

Gabor Response for Image152$_5$219.JPG, Scale3 Orientation90

Gabor Response for Image152$_5$219.JPG, Scale3 Orientation135

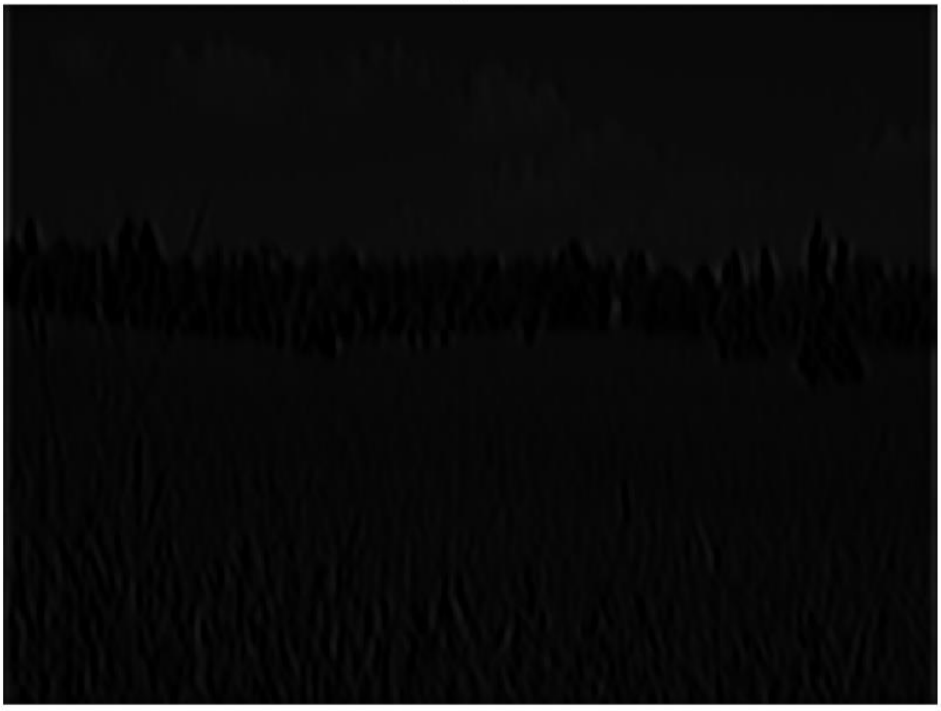
Gabor Response for Image152$_5$219.JPG, Scale4 Orientation0



Gabor Response for Image152$_5$219.JPG, Scale4 Orientation45

Gabor Response for Image152$_5$219.JPG, Scale4 Orientation90

Gabor Response for Image152$_5$219.JPG, Scale4 Orientation135



Gabor Response for Image152$_5$220.JPG, Scale1 Orientation0

Gabor Response for Image152$_5$220.JPG, Scale1 Orientation45

Gabor Response for Image152$_5$220.JPG, Scale1 Orientation90

Gabor Response for Image152$_5$220.JPG, Scale1 Orientation135

Gabor Response for Image152$_5$220.JPG, Scale2 Orientation0

Gabor Response for Image152$_5$220.JPG, Scale2 Orientation45

Gabor Response for Image152$_5$220.JPG, Scale2 Orientation90

Gabor Response for Image152$_5$220.JPG, Scale2 Orientation135

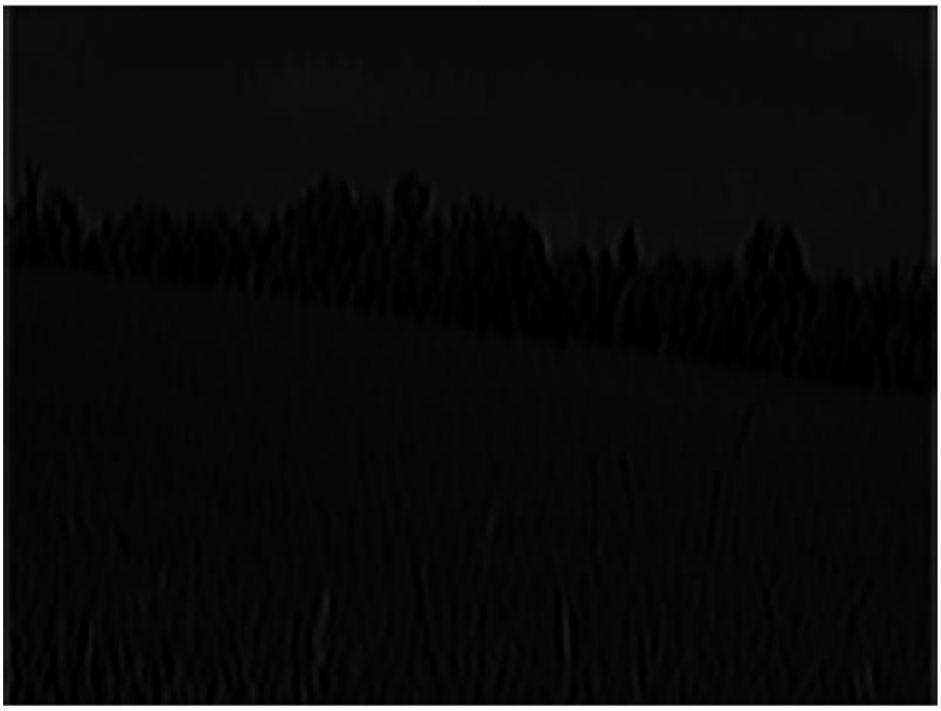
Gabor Response for Image152$_5$220.JPG, Scale3 Orientation0

Gabor Response for Image152$_5$220.JPG, Scale3 Orientation45

Gabor Response for Image152$_5$220.JPG, Scale3 Orientation90

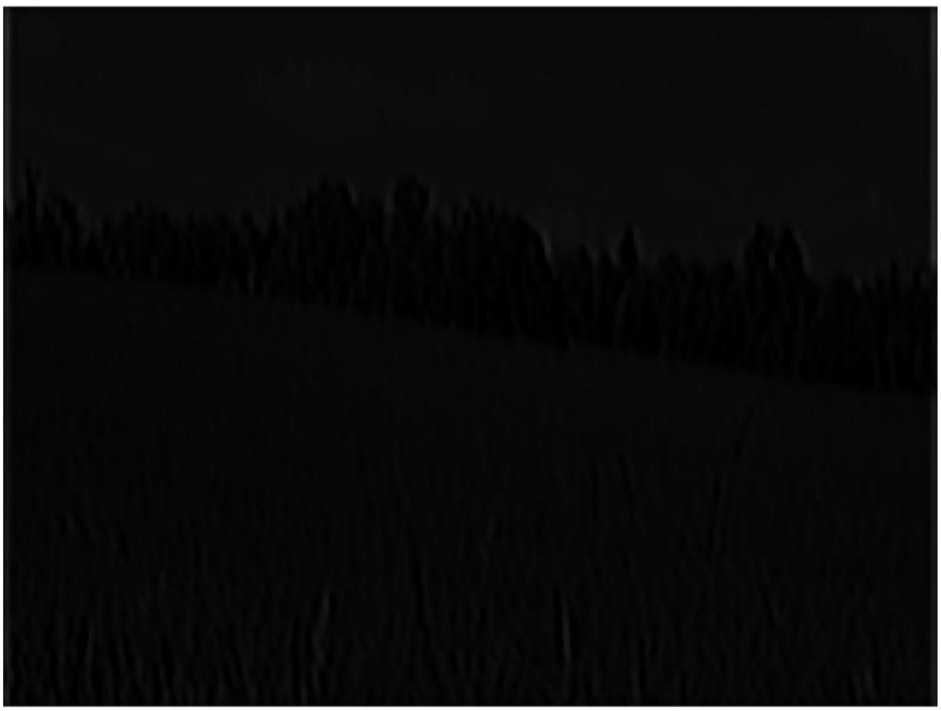
Gabor Response for Image152$_5$220.JPG, Scale3 Orientation135

Gabor Response for Image152$_5$220.JPG, Scale4 Orientation0



Gabor Response for Image152$_5$220.JPG, Scale4 Orientation45

Gabor Response for Image152$_5$220.JPG, Scale4 Orientation90



Part 3 – Clustering, Final Segmentation, False Color Images

Theoretical Discussion

After extracting the average Gabor response for every superpixel, one ends up with 9683 feature vectors for this set of images, with this particular configuration. Each vector has 16 entries.

To cluster the texture patches extracted from the dataset, I used the K-Means algorithm provided by MATLAB. To the algorithm, I supplied a $9683x16$ matrix and the number of clusters, M as four. This choice for the number of clusters results from trial and error. Decreasing the number of clusters decreases the quality of the segmentation and increasing the number of clusters above 4 makes the resulting clustering noisier.

The k-means algorithm provided by MATLAB which I used in my implementation works as follows. The initial step of the algorithm is the classic k-means algorithm: k cluster centers are initialized randomly and points in the dataset are assigned to one of these clusters (the cluster which the point is closest to according to the l_2 norm). Then, the mean of the cluster is calculated and the cluster center is moved to the mean. This is repeated for either a certain number of iterations or until the cluster centers move less than a threshold for a given iteration. After this step is completed, the implementation saves this clustering and then repeats the algorithm, starting from the beginning, but with differently assigned cluster centers. This is repeated five times until five clusterings are achieved. The implementation then chooses the clustering with the lowest SSE.

Results

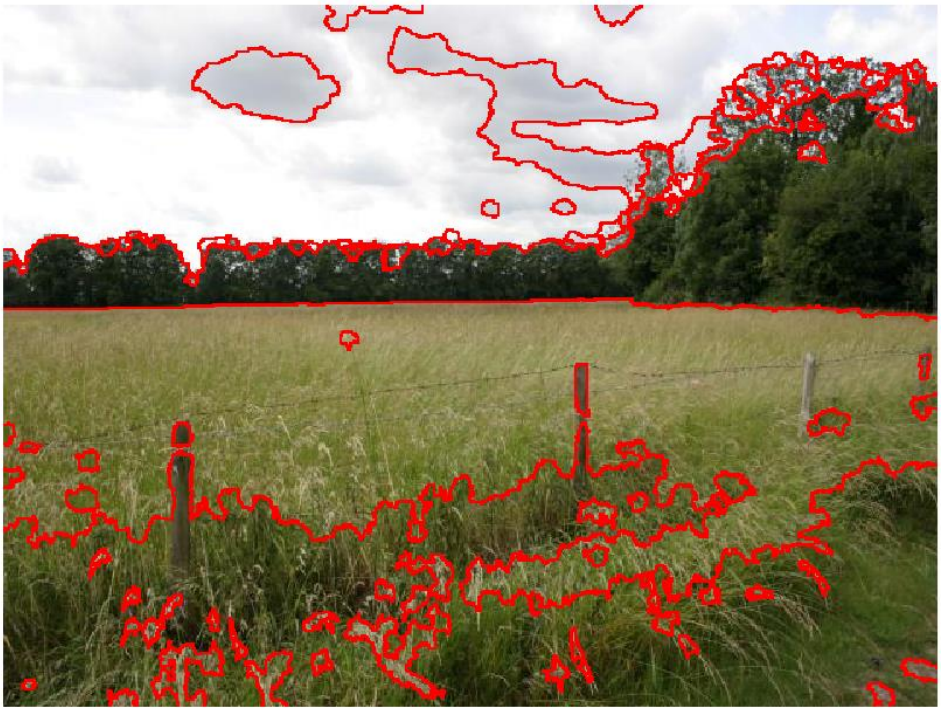
The following images are the results of k-means clustering over the superpixel gabor feature space.

Gabor Feature Segmentation:152$_5$219.JPG

Gabor Feature Segmentation:$152_5$220.JPG

Gabor Feature Segmentation:$153_5$341.JPG

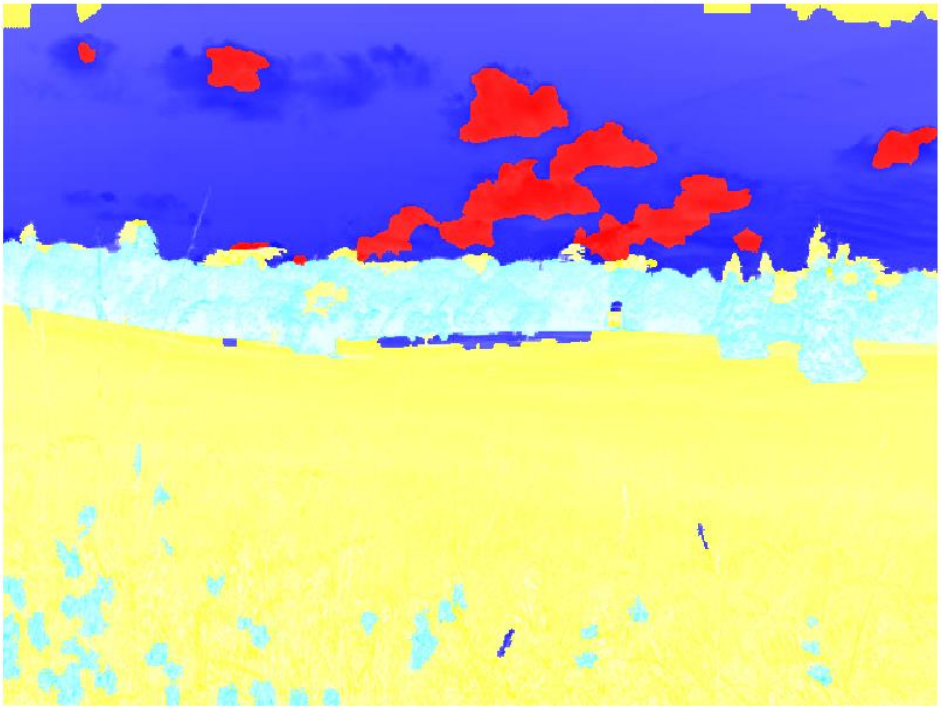
Gabor Feature Segmentation:153$_5$396.JPG



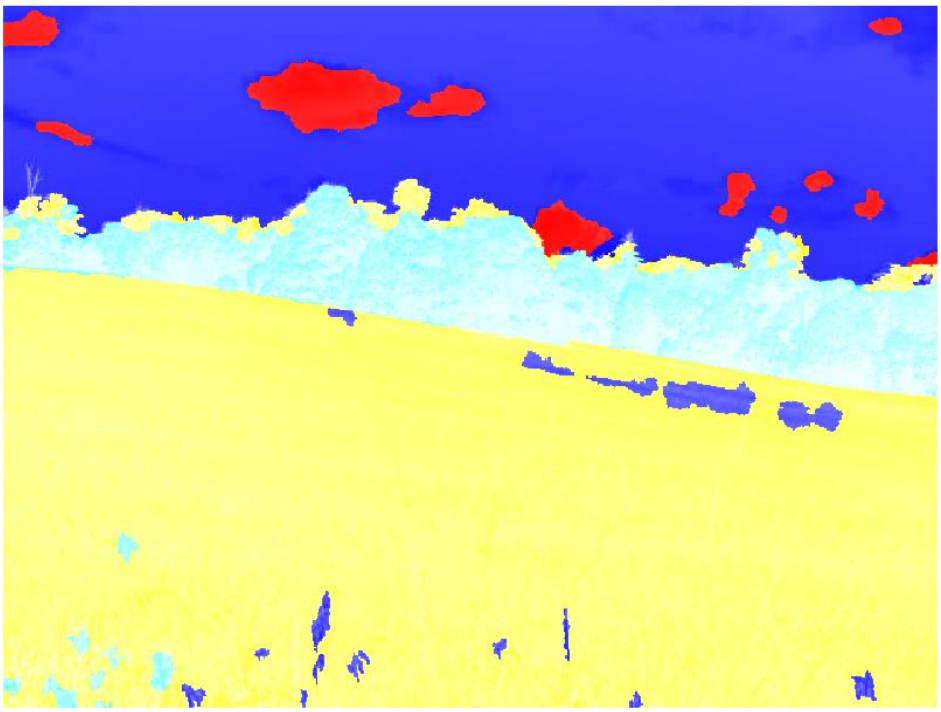
Gabor Feature Segmentation:157$_5$768.JPG

The following images are the false color overlays of the clusters over the image.

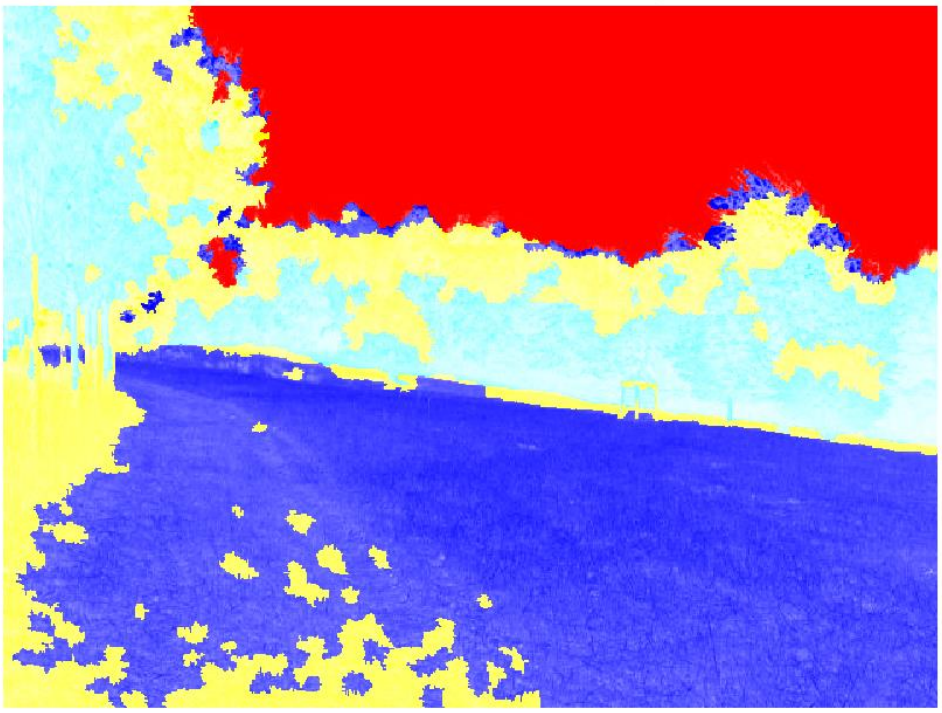
Segmentation False Color Overlay:$152_5$219.JPG



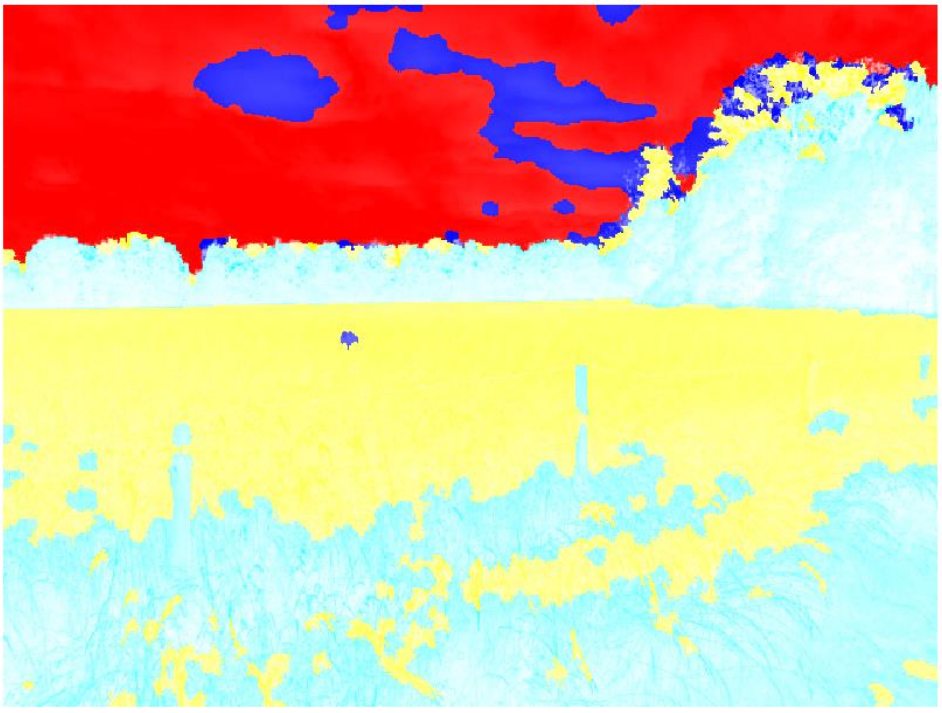
Segmentation False Color Overlay:$152_5$220.JPG

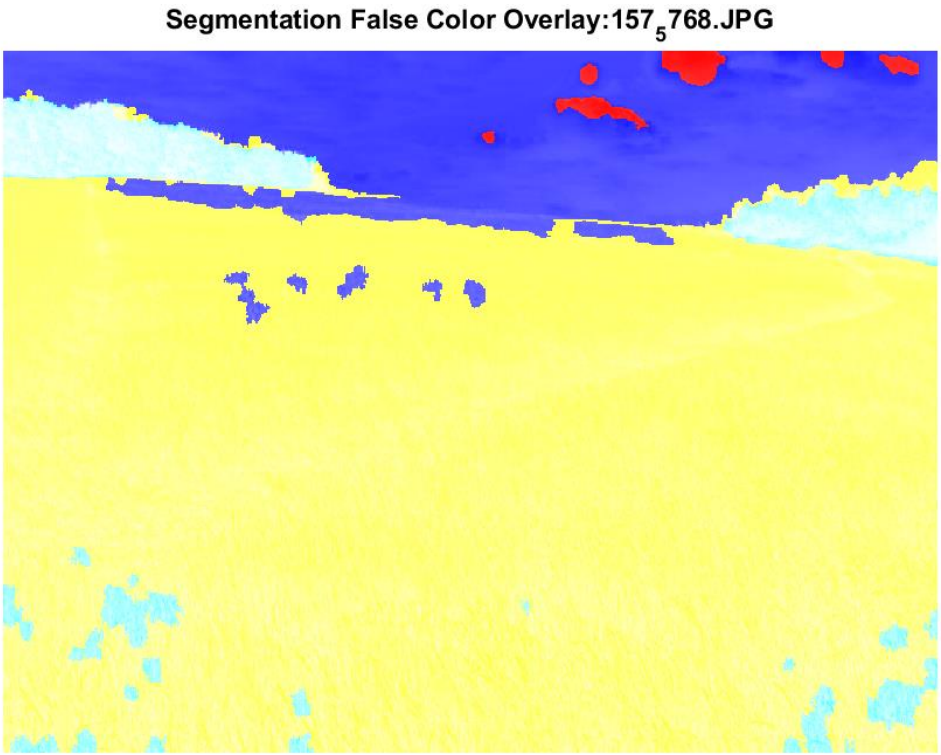


Segmentation False Color Overlay:$153_5$341.JPG



Segmentation False Color Overlay:$153_5$396.JPG





Conclusion and Discussion

I was able to obtain segmentations for certain regions quite accurately for all the images in the data set. I was able to segment the skies, the clouds, the grass, and the trees in most images based on their texture properties, however there is some noise and misclassification in the segmentation. This is caused by the fact that the texture for the misclassified/noisy regions resemble the other regions (e.g. grass closer to the camera resembles trees wherever they are greener than the average patch of grass in the dataset). The clouds classified as sky are caused by the clouds having weak Gabor responses, similar to that of the skies. The lack of color information in Gabor filtering, and the lack of context causes the segmentation to be noisy, and at some places plain wrong. However, considering the big picture, these segmentations can offer a very good approximation for the regions, and can be easily parallelized. Also, one can consider certain algorithms that eliminate the noise in the labels so that segmented regions are contiguous.

Another factor that introduces noise in the dataset is the presence of regions that are only present in one image. For instance, one of the images feature a wooden fence and another image features a dirt road. The algorithm tries to cluster these in the same category as the trees, skies, clouds, and grassland. This is obviously not the case, and these outliers have an effect on the robustness of the method, because during clustering their presence affects the cluster centers as well. This also introduces some form of error in the segmentation.

Code Written (Internal Resources)

main.m

```
clear
clc

N_init = 200;
img_dir = dir('data/*.JPG');

for i = 1:length(img_dir)
    images{i} = imread(img_dir(i).name);
end

[segmented, labels, gabor_images] = segment_images(images, 2048, 4);

for i = 1:length(img_dir)
    a = figure;
    mask1 = boundarymask(labels{i});
    imshow(imoverlay(images{i},mask1,'red'));
    title(strcat('Superpixel Segmentation: ', img_dir(i).name));

    b = figure;
    mask2 = boundarymask(segmented{i});
    imshow(imoverlay(images{i},mask2,'red'));
    title(strcat('Gabor Feature Segmentation: ', img_dir(i).name));

    c = figure;
    alpha = images{i};
    h = imshow(label2rgb(segmented{i}));
    set(h, 'AlphaData', rgb2gray(alpha));
    title(strcat('Segmentation False Color Overlay: ', img_dir(i).name));

    saveas(a, strcat('out/superpixel/superpixel_image', int2str(i), '.png'));
    saveas(b, strcat('out/segmentation/seg_image', int2str(i), '.png'));
    saveas(c, strcat('out/segmentation/false_color_seg_image', int2str(i), '.png'));
end


for n = 1:length(img_dir)
    gabor_sv = gabor_images{n};
    for i = 1:4
        for j = 1:4
            a = figure('visible', 'off');
            imshow(gabor_sv(:,:,(i-1)*4+j));
            title(strcat('Gabor Response for Image', img_dir(n).name , ', Scale ', int2str(i),
' Orientation ', int2str(45 * (j - 1))));
            saveas(a,strcat('out/gabor/gabor_', int2str(n),'_', int2str(i), int2str(j),
'.png'));
        end
    end
end
```

gabor_bank.m

```
function out = gabor_bank(image, sigma, theta)
out = zeros([size(image), length(sigma)*length(theta)]);
    for i=1:length(sigma)
        for j = 1:length(theta)
            filter = gabor_fn(sigma(i),theta(j),10.5,0,1);
            out(:,:,(i-1)*length(theta)+j) = conv2(image,double(filter),'same');
        end
    end
end
```

gabor_features.m

```
function [gabor_sup, gabor_resp] = gabor_features(image, labels, N)
image_gray = im2double(rgb2gray(image));
eo = gabor_bank(image_gray, [1 2 3 4], [0 45 90 135]);
gabor_resp = eo;
gabor_mag = eo;
```

```
idx = label2idx(labels);
sz = size(image_gray);
gabor_sup = zeros(size(gabor_mag,3),N);
for i = 1:N
    idx_lin = idx{i};
    [x,y] = ind2sub(sz, idx_lin);
    rws = [x,y];
    for j = 1:size(rws,1)
        gabor_response = gabor_mag(rws(j,1),rws(j,2),:);
        gabor_sup(:,i) = gabor_sup(:,i) + gabor_response(:);
    end
    gabor_sup(:,i) = gabor_sup(:,i) / size(rws,1);
end

gabor_sup = gabor_sup';
```

segment_images.m

```
function [segmented_image_labels, labels, gabor_images] = segment_images(images, N_init, M)
gabor_pixels_superset = [];
for i = 1:length(images)
    [labels{i}, N{i}] = superpixels(images{i}, N_init, 'Method', 'slic0');
    [gabor_pixels{i}, gabor_images{i}] = gabor_features(images{i}, labels{i}, N{i});
    gabor_pixels_superset = [gabor_pixels_superset; gabor_pixels{i}];
end

km_idx = kmeans(gabor_pixels_superset,M, 'Replicate', 5);

start_idx = 1;
end_idx = 0;
for i = 1:length(images)
    end_idx = end_idx + N{i};
    cluster_idx{i} = km_idx(start_idx:end_idx);
    start_idx = end_idx + 1;
end

for i = 1:length(images)
    segmented_img = zeros(size(images{i}, 1), size(images{i},2));
    label_idx = label2idx(labels{i});
    kmeans_idx = cluster_idx{i};
    for j = 1:N{i}
        idx_lin = label_idx{j};
        segmented_img(idx_lin) = kmeans_idx(j);
    end
    segmented_image_labels{i} = segmented_img;
end
end
```

Code Citations (External Resources)

gabor.fn

Taken from:

https://uk.mathworks.com/matlabcentral/fileexchange/28751-gabor-filtering-on-an-image

```
%https://uk.mathworks.com/matlabcentral/fileexchange/28751-gabor-filtering-on-an-image
function gb=gabor_fn(sigma,theta,lambda,psi,gamma)

sigma_x = sigma;
sigma_y = sigma/gamma;

% Bounding box
nstds = 3;
xmax = max(abs(nstds*sigma_x*cos(theta)),abs(nstds*sigma_y*sin(theta)));
xmax = ceil(max(1,xmax));
ymax = max(abs(nstds*sigma_x*sin(theta)),abs(nstds*sigma_y*cos(theta)));
ymax = ceil(max(1,ymax));
xmin = -xmax; ymin = -ymax;
[x,y] = meshgrid(xmin:xmax,ymin:ymax);
```

```
% Rotation
x_theta=x*cos(theta)+y*sin(theta);
y_theta=-x*sin(theta)+y*cos(theta);

gb= 1/(2*pi*sigma_x *sigma_y) * exp(-
.5*(x_theta.^2/sigma_x^2+y_theta.^2/sigma_y^2)).*cos(2*pi/lambda*x_theta+psi);
```

Instructions on How to Run the Script

The script can be run via main.m. Make sure that the following directories exist in the root directory of the script before running the script, and also make sure that everything is added to the PATH of MATLAB.

/data: This directory should have all the dataset images. Their names do not matter.

/out: This directory will be used to output the images that are on the report.

/out/gabor: This directory will be used to output the Gabor filter responses of all images.

/out/segmentation: This directory will be used to output the segmentation results for superpixel segmentation

/out/segmentation: This directory will be used to output the k-means segmentation based on the average Gabor features of superpixels, both as masks and overlay images.

Create these directories by hand. The script does not create them automatically, for the sake of the cleanliness of the filesystem.