COMS W4735y: Visual Interfaces To Computers – Assignment 2
Visual Information Retrieval

Michelle Tadmor – mdt2125@columbia.edu

March 17, 2015

In the following, I document an exploration of deciding the similarity among images.

Code is appended at the end of this file.

Software:

Matlab R2014b with Image Processing Toolbox

1. Gross Color Matching

Comparing two images strictly on their total color distributions. I load each `'.ppm'` image in Matlab using the command '`imread`' and save it into a cell array of images. I continue to extract the red (r), green (g), and blue (b) for the image 3 dimensional vector.

Background removal:

To account for images being of different sizes and to avoid two images having a high similarity based on similar concentration of background I remove the background heuristically by ignoring continuous regions of black pixels. First, create a binary image using a threshold for black at r,b,g<25. Then I select only continuous regions that are 15 pixels or more in size. I use Matlab method to '`bwboundaries`' to extract the continuous regions.

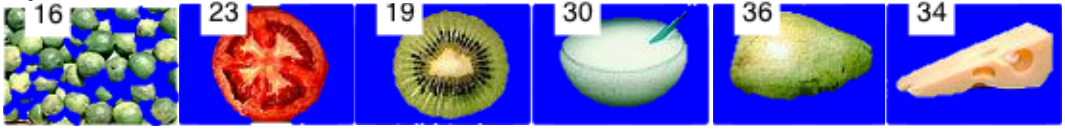
```
blackPixels = (double(r) < 25 & double(g) < 25 & double(b) < 25);
[~,regions] = bwboundaries(blackPixels,'noholes');

% sort by size
table = tabulate(regions(:));
background_regions = table(table(:,2)>15,1);

...   <handling to not remove continuous non black regions>

blackPixels = ismember(regions, background_regions);
r(blackPixels) = nan;
b(blackPixels) = nan;
g(blackPixels) = nan;
```

Here are some good examples of this threshold. Background is almost completely removed while the object remains:



Notice image 23 and 19 (tomato and kiwi) both contain black regions inside the object that remain unharmed. This is due to the selection of a continuous region.

Some images had a lighter black background that parts of were not selected for removal while other pictures had black in the object itself that was ignored. Here are some non-perfect examples on background cleaning with this method:

Compute Histogram:

I compute a color three-dimensional color histogram by binning each color's 1:255 possible values to bins of `bin_size`=40. I use a triple nested loop over the color bins and count the number of pixels in the image that fall into the combined r,g,b bin. The background pixels are ignored by setting the r,g,b values at those pixels to `'nan'`. The following is an except my function 'compute_color_hist.m'

```
for i=1:ceil(255/bin_size)
  for j=1:ceil(255/bin_size)
    for k=1:ceil(255/bin_size)
       hist(i,j,k) = sum(r>=bin_size*(i-1) & r<bin_size*i...
                         & b>=bin_size*(j-1) & b<bin_size*j...
                         & g>=bin_size*(k-1) & g<bin_size*k);
    end
  end
end
```

Compare Images:

Comparing images based on the previous step's color histogram. Each bin in the histogram contains the number of pixels that the image has of a specific color associated with that bin. To obtain the percentage that a certain color exists in an image I divide each bin by the sum of all bins for that image. Since we ignore background, two similar images of different scale with more or less background will have still have the same percentage, since the background pixels are not binned.

To compare the images I use L1 metric: sum over the absolute distance between each bin and then dive by 2 to obtain a metric in the range of 0 to 1. Note that without dividing by 2, the maximum difference (between images who share no common non-empty bins) is 2.

```
function score = l1_compare(imghist1, imghist2)
    vec1 = imghist1(:);
    vec2 = imghist2(:);
    score = sum(abs(vec1./sum(vec1) - vec2./sum(vec2))./2;
end
```

Next, A full pairwise similarity matrix, S, is generated by looping over all images and subtracting the L1 distance from 1. Now S contains 0 for dissimilar images, and 1 for identical images.

```
function A=pairwise_compare_hists(hists)

m_size = size(hists,2);
A = zeros(m_size, m_size);  % distance matrix
for i=1:m_size
    for j=1:m_size
        A(i,j) = 1-l1_compare(hists(:,i), hists(:,j));
    end
end
```

To find the most and least similar images for each image, S is sorted by rows. The first 4 and last 3 columns are printed below:

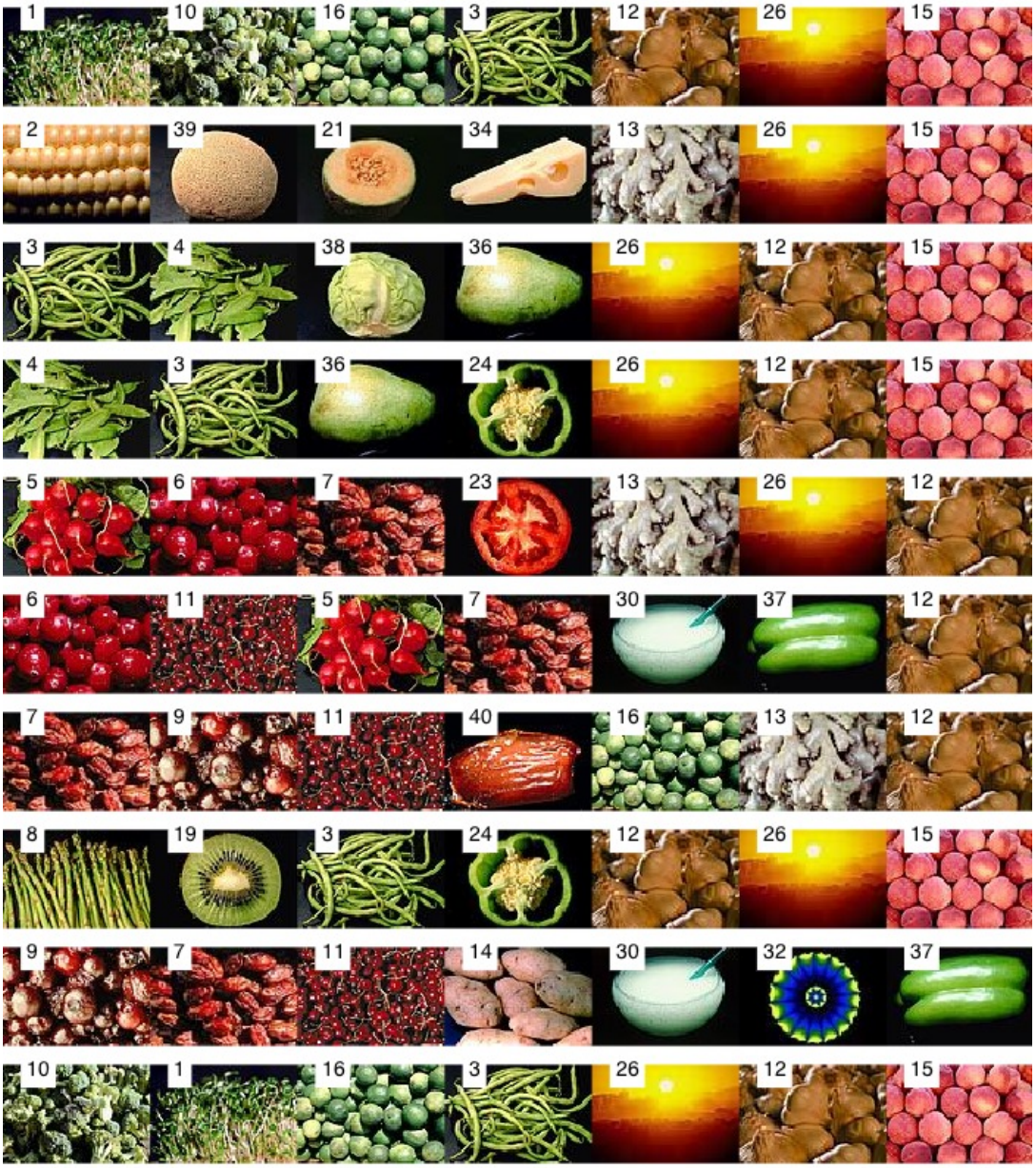


Figure 1 Img 1-10 Color Similarity

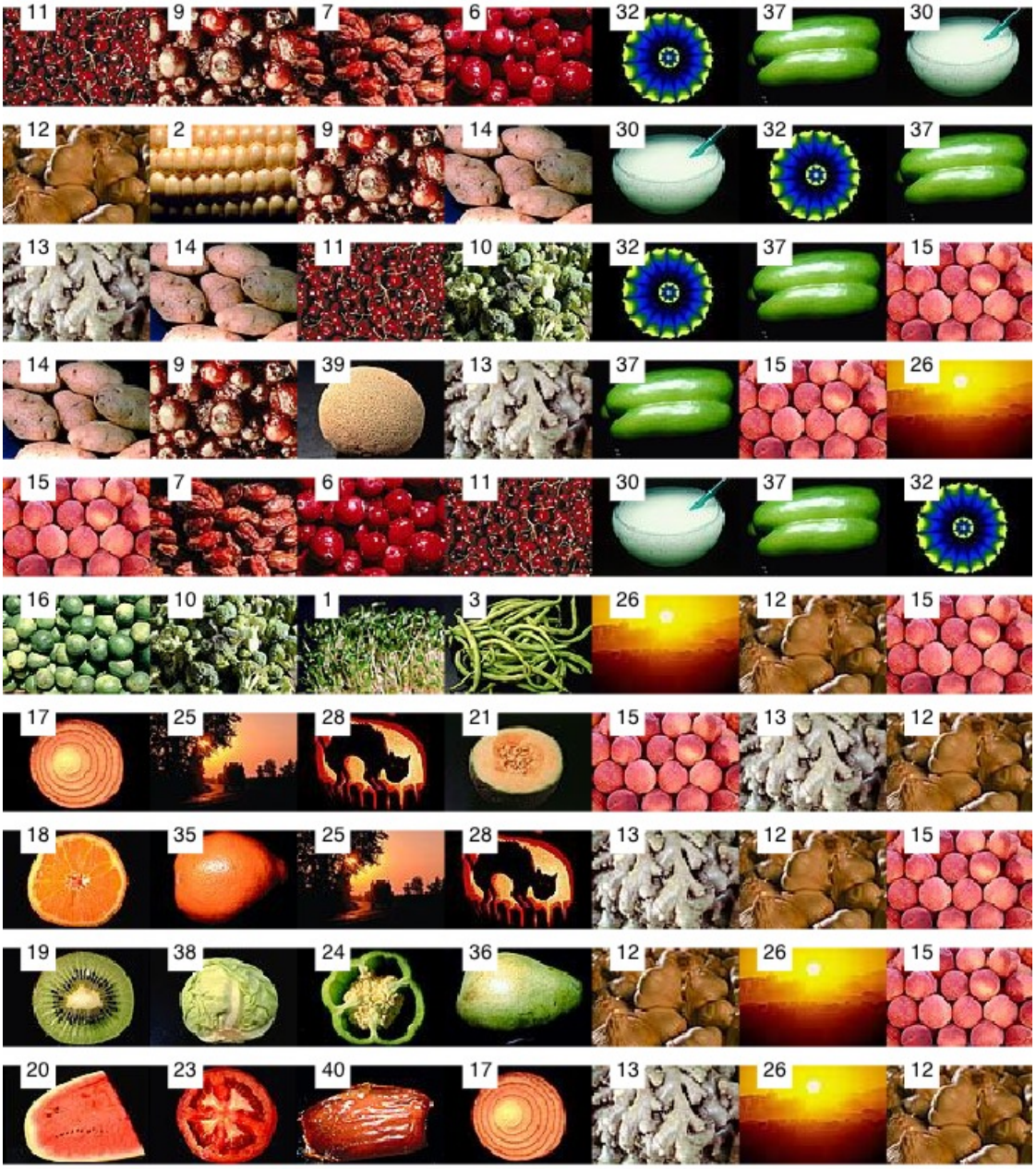


Figure 2 11-20 Color Similarity

Figure 3 21-30 Color Similarity

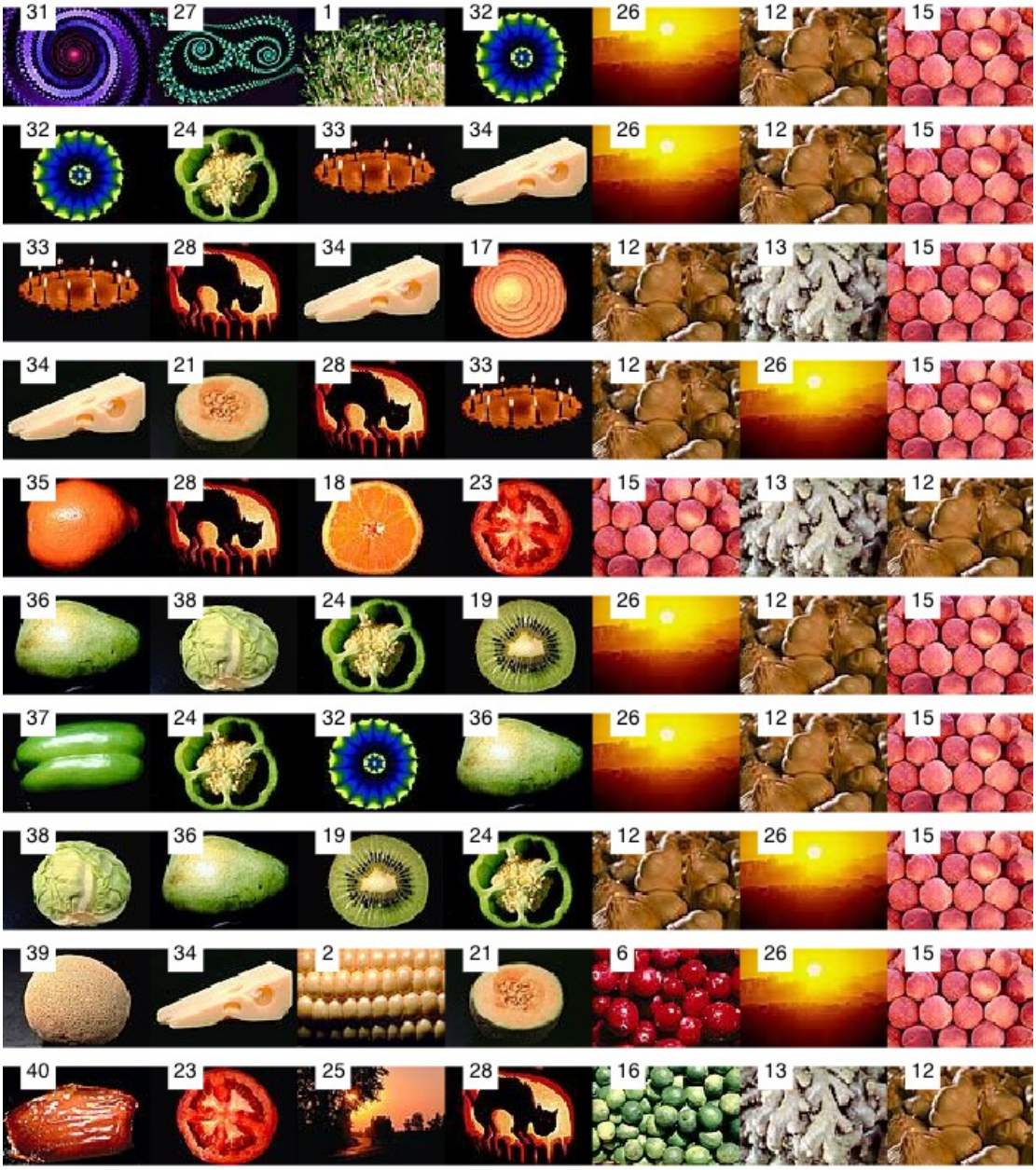


Figure 4 31-40 Color Similarity

Within the groups of four most similar images I find the groups that has the most similarity between the images and the one with the least. I sum the first four columns of the sorted similarity matrix S along its rows, and find the rows that have the maximum sum (those are the most similar) and the minimum sum (lease similar):

Figure 5 most similar group

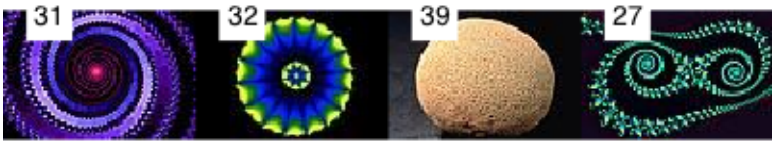


Figure 6 least similar group

2. Gross Texture Matching

Comparing two images strictly on their texture. As in step 1, I load each `'.ppm'` image in Matlab using the command '`imread`' and save it into a cell array of images. I use Matlab's '`rgb2gray`' to convert the image to a gray scale image and compute the laplacian of the image described in the assignment. In the code below, `'bw'` is the gray-scale image. I multiple the pixel's in sensitivity by 9 (instead of 8) because I count it an unnecessary time in the sum of surrounding pixels being deducted.

```
% compute laplacian
bw_laplacian = zeros(size(bw));
for r=2:(size(bw(:,:), 1)-1)
  for c=2:(size(bw(:,:), 2)-1)
    bw_laplacian(r,c) =...
        double(bw(r,c))*9-sum(sum(double(bw(r-1:r+1, c-1:c+1))));
  end
end
```

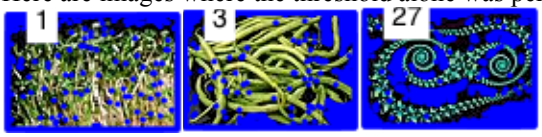
Background Removal:

Following the professor reply on piazza that we are not allowed to use color in this step, naturally I went to remove areas of low texture, along the lines of absolute change is less than some threshold

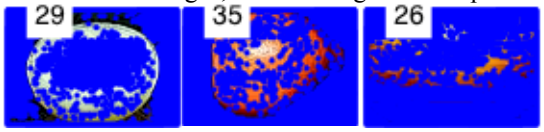
'`abs(bw_laplacian) < texture_threshold`'.

Unfortunately, even for very low threshold (`texture_threshold =10`) this method would remove the smooth area in smooth objects. It would be less effective to draw a similarity between an apple to a pepper based on its non-smooth areas but in fact an apple is more similar to the pepper than the cabbage exactly because it has "less texture". Here are some examples of what was found with this low threshold.

Here are images where the threshold alone was performing pretty well:



but for some images, that had large smooth parts the information was lost:



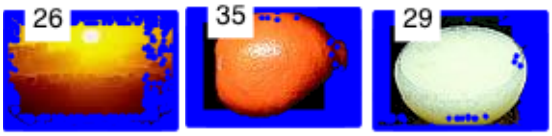
To preserve some object information I decided to make a dead zone in the center of the images that will not be ignored regardless of texture. The width of the border that was evaluated for texture threshold is a fraction of the width\height (respectively). I found that using 1/10 is a good approximation.

```
% remove background - if low texture and not in the center of image
edgewidth  = floor(size(bw_laplacian,2)/frac);
edgeheight = floor(size(bw_laplacian,1)/frac);

% create dead zone mask
mask = ones(1, size(bw_laplacian,2));
mask(edgewidth:edgewidth*(frac-1)) = 0;

% iterate over each image row
for r=1:size(bw_laplacian,1)
    if (r>edgeheight && r<edgeheight*(frac-1))
        bw_laplacian(r, mask & abs(bw_laplacian(r, :)) < 10 ) = nan;
    else
        bw_laplacian(r, abs(bw_laplacian(r, :)) < 10 ) = nan;
    end
end
```

Here are the same images with center preserved:



Compute Histogram:

Similarly to Step 1, I compute a texture histogram by binning. The texture space can take values from negative 0 to positive 255*8 over the absolute value of the laplacian. I bin the laplacian values to bins of `bin_size`=2. I noticed that gave me more consistent results. Below are two example of a similar group of four before adding the absolute value and after:

Figure 7 Group of similar four images by binning over the raw laplacian values

Figure 8 Group of similar four images by binning over the absolute of the laplacian values

For some images the removal of background created strange texture matching. I see the corn more similar to objects like the cherris, potatoe, brussel sprouts but for some reason it kept being matched to the orange. Perhaps my eyes are too distracted by the background to see the similarity.

Figure 9 before background removal

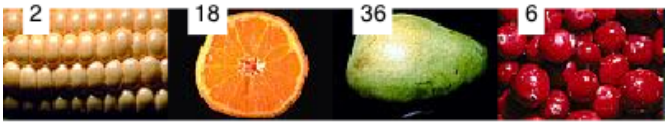


Figure 10 After background removal

Setting the laplacian values at those pixels to 'nan' ignores the background pixels.

```
% texture histogram - iteration over bins
for i=1:ceil(nbins)
  minbin=min_t+bin_size*(i-1);
  maxbin=min_t+bin_size*(i);

  % count the number of pizels that belong to current bin
  hist(i) = sum(bw_laplacian>=minbin & bw_laplacian<maxbin);
end
```

Once I have the histogram of the bins, I normalize to attain a percentage in each cell and continue to compare the histograms exactly as in step 1, using the same functions:
`pairwise_compare_hists` which uses `l1_compare` to obtain similarity matrix.

Here is the 40 by 7 matrix where the first column is the image, the next 3 are the most similar (in texture) to the image, and the last three are the least similar.

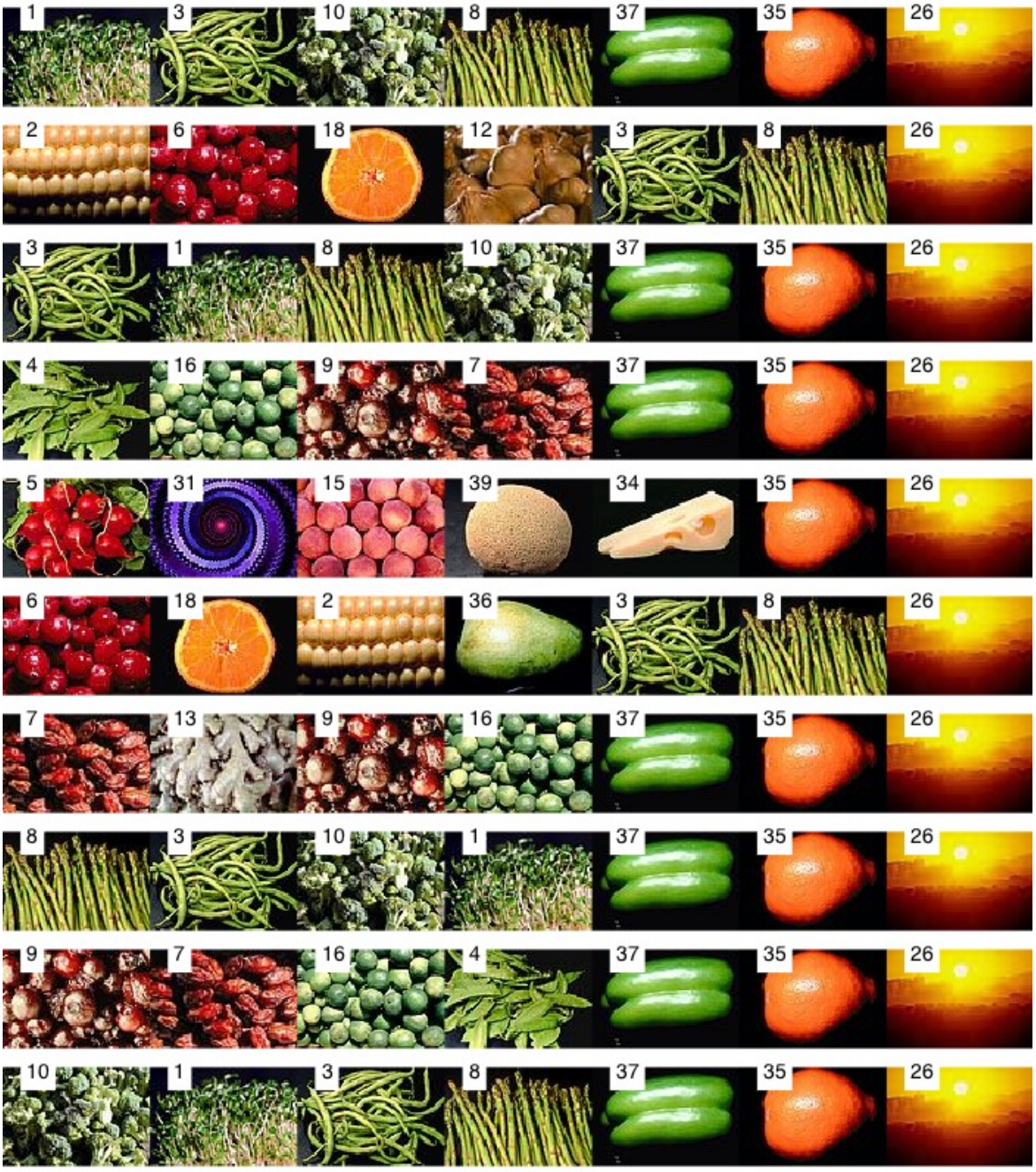


Figure 11 1-10 Texture Similarity

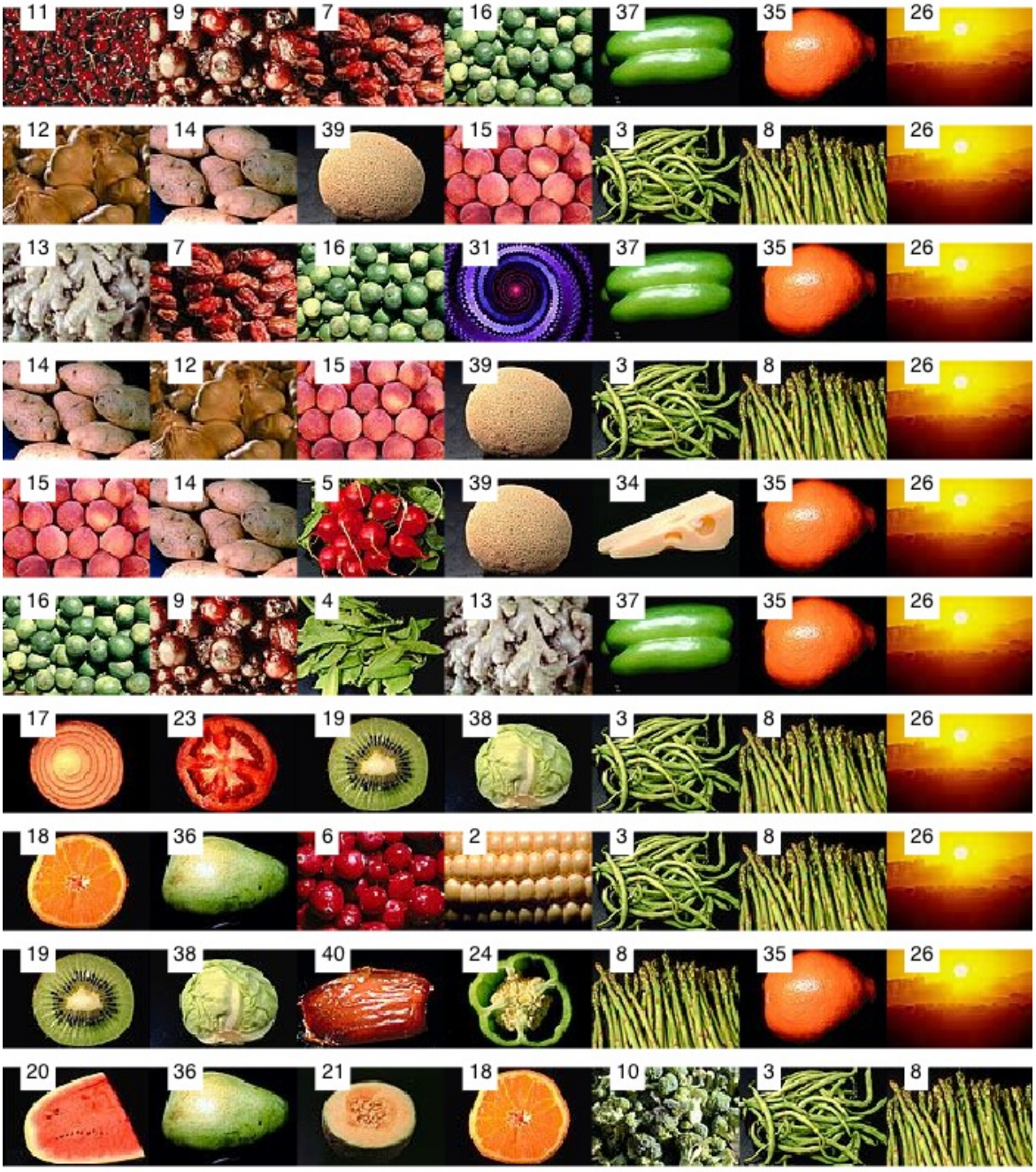


Figure 12 11-20 Texture Similarity

Figure 13 21-30 Texture Similarity

Figure 14 31-40 Texture Similarity

Identically to step one, here are the four most similar and four least similar by summing over the similarity matrix.

Figure 15 most similar group

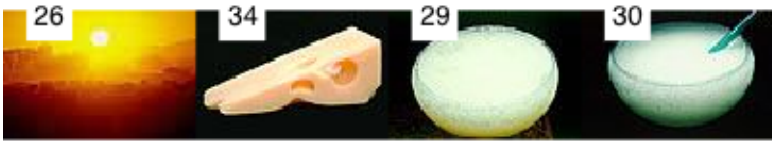
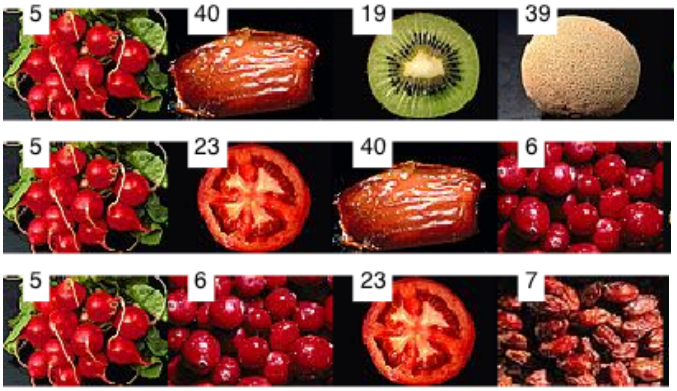
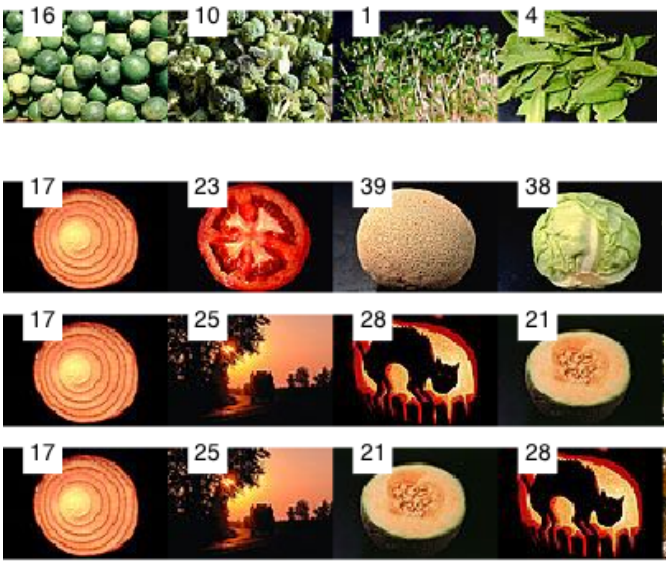


Figure 16 least similar group

3. Combine Similarities, and cluster
   I combine the texture similarity matrix (I will refer to it here as T) and the color similarity matrix (C) to a combined similarity matrix S in the same fashion as was recommended in the assignment

   S = r*T+(r-1)C
   I tested a few values for r to see what made sense for me as a human.
   Here is a few examples for r=.8 (more by texture), .5 (even contribution), and .2 (more by color) respectively, each example is shown in three rows.





I examined more r values as well and ultimately decided to use r=.6. Though initially I went with r=.2 because it was more visually appealing I kept getting a very large cluster in the next step. Increasing the influence of texture gave me well-sized clusters that were also more consistent with my friends clusters in step 4.

I sorted the combined similarity matrix and printed out the 40 by 7 matrix

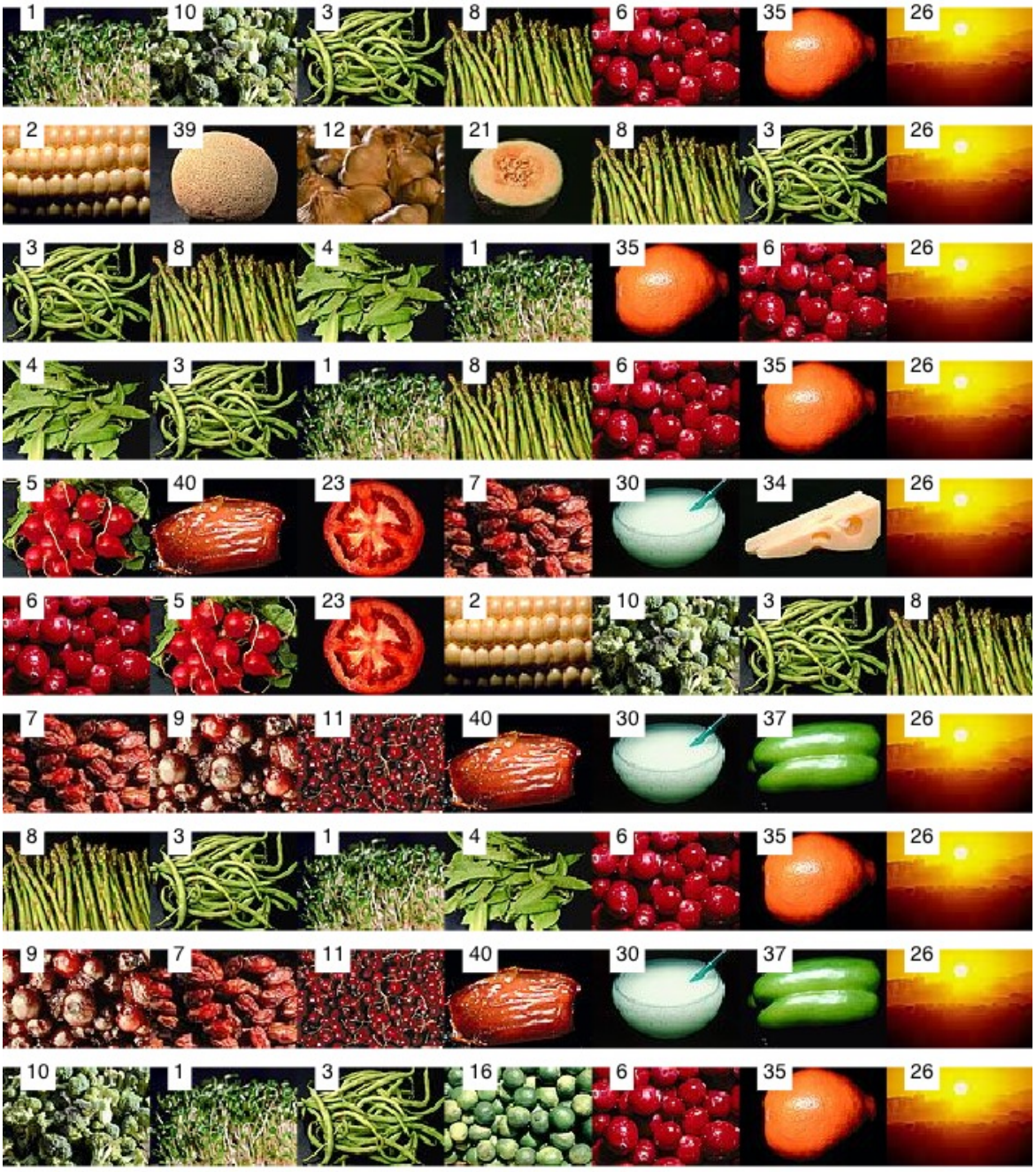


Figure 17 Imgs 1-10 combined similarity

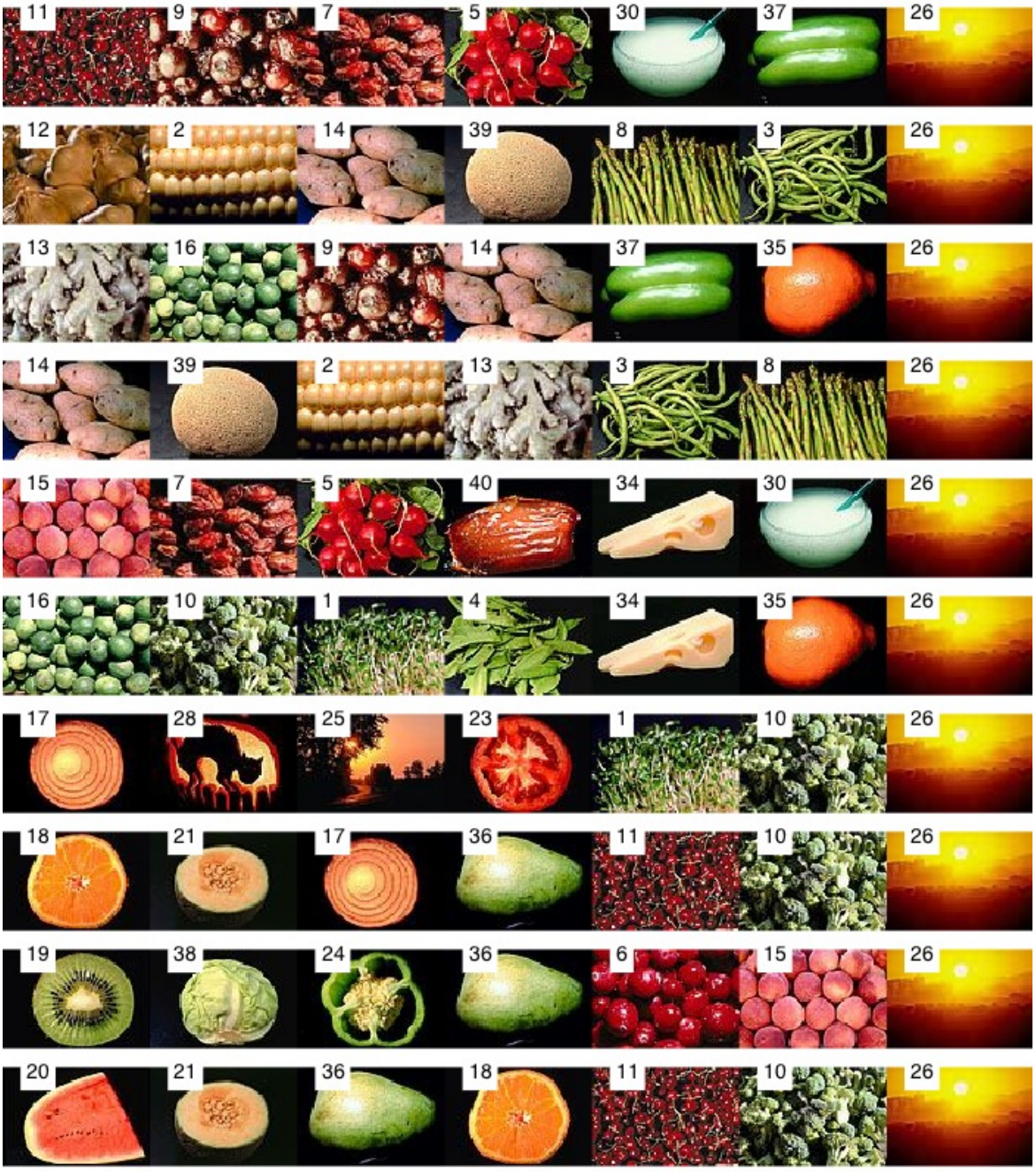


Figure 18 Imgs 11-20 combined similarity

Figure 19 Imgs 21-30 combined similarity

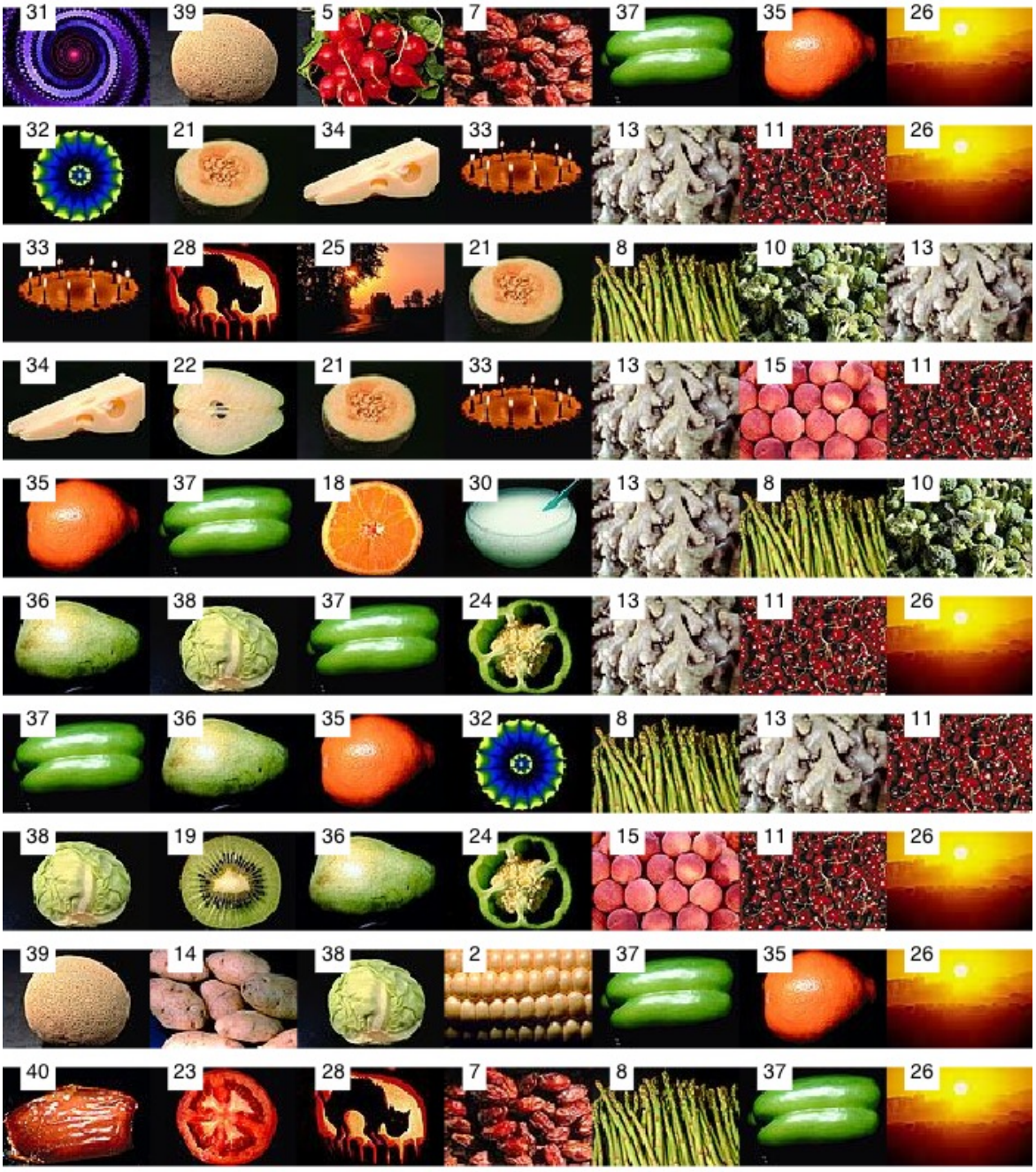


Figure 20 Imgs 31-40 combined similarity

In addition here are the over all most and least similar group of four pictures

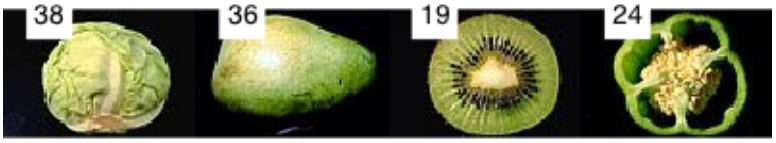


Figure 21 combined similarity - most similar group

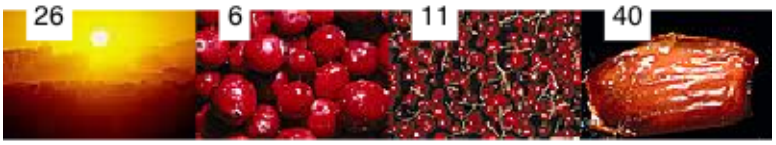


Figure 22 combined similarity least similar group

Clustering:

I implemented the clustering algorithm (Hierarchical Clustering?) that was described in the homework. I iterate a merge cluster action until I only have n [=7] clusters left. I compute a pairwise cluster similarity matrix for all clusters in current iteration at each iteration. I then select the two closest clusters and merge them. The similarity between clusters can be either complete link or single link. Here is in pseudo-code, you can find the full code in the bottom under the function '`hier_clusters`':

```
while number of clusters > n
      compute cluster similarity matrix (complete or single link)

      find the two closest clusters

      merge closest clusters
end

return clusters
```

Below are the resulting clusters for complete link. As mentioned above, I played with r value until I found a reasonably sized and correct looking clustering at r=.6.

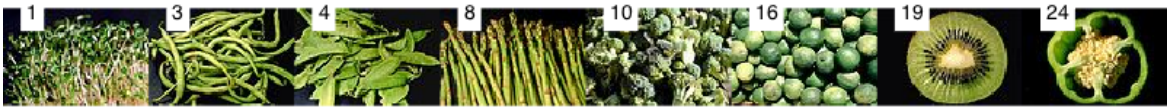
Cluster 1

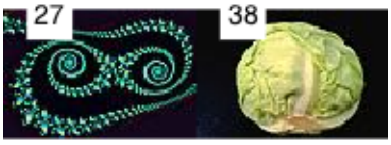
Cluster 2

Cluster 3

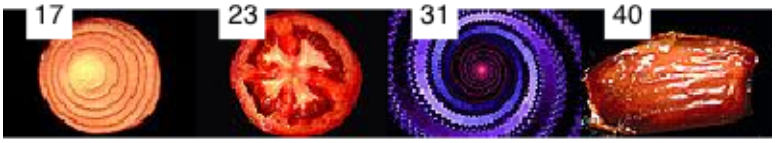
Cluster 4

Cluster 5

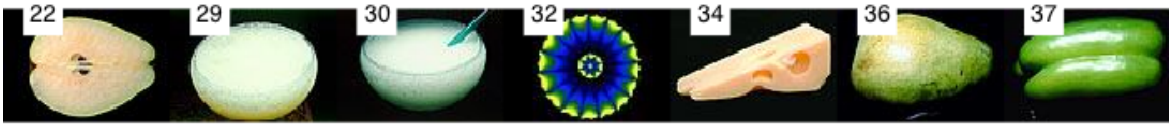




Cluster 6

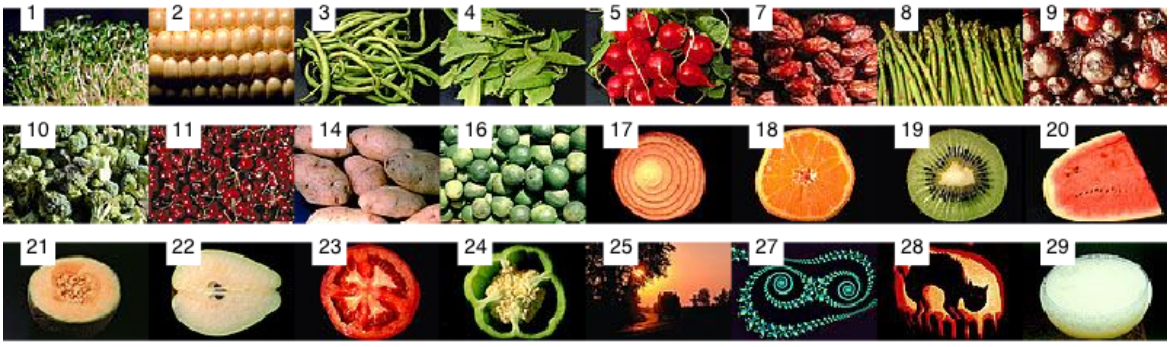


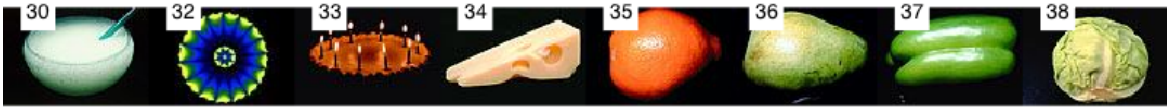
Cluster 7

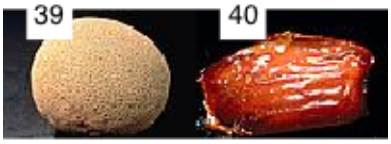


Here are the resulting clusters for single link, which as was explained in the exercise will result in more of a spanning tree behavior where one giant cluster eats up all the images around it.

Cluster 1







Cluster 2

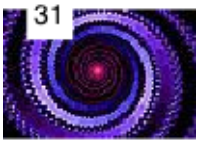
Cluster 3

Cluster 4

Cluster 5

Cluster 6

Cluster 7



4. Creativity Step

   I called on three of my friends to recreate 2 40 by 7 matrices from step 1&2. I printed out the images and had each of them write down the answers (in exchange for pizza) and finally cluster the images to 7 groups. Here is an image from my living room table that night

   Here are the similarity by color and then texture of my friends:

Figure 23 1-10 Color Similarity (friend1)

Figure 24 11-20 Color Similarity (friend1)

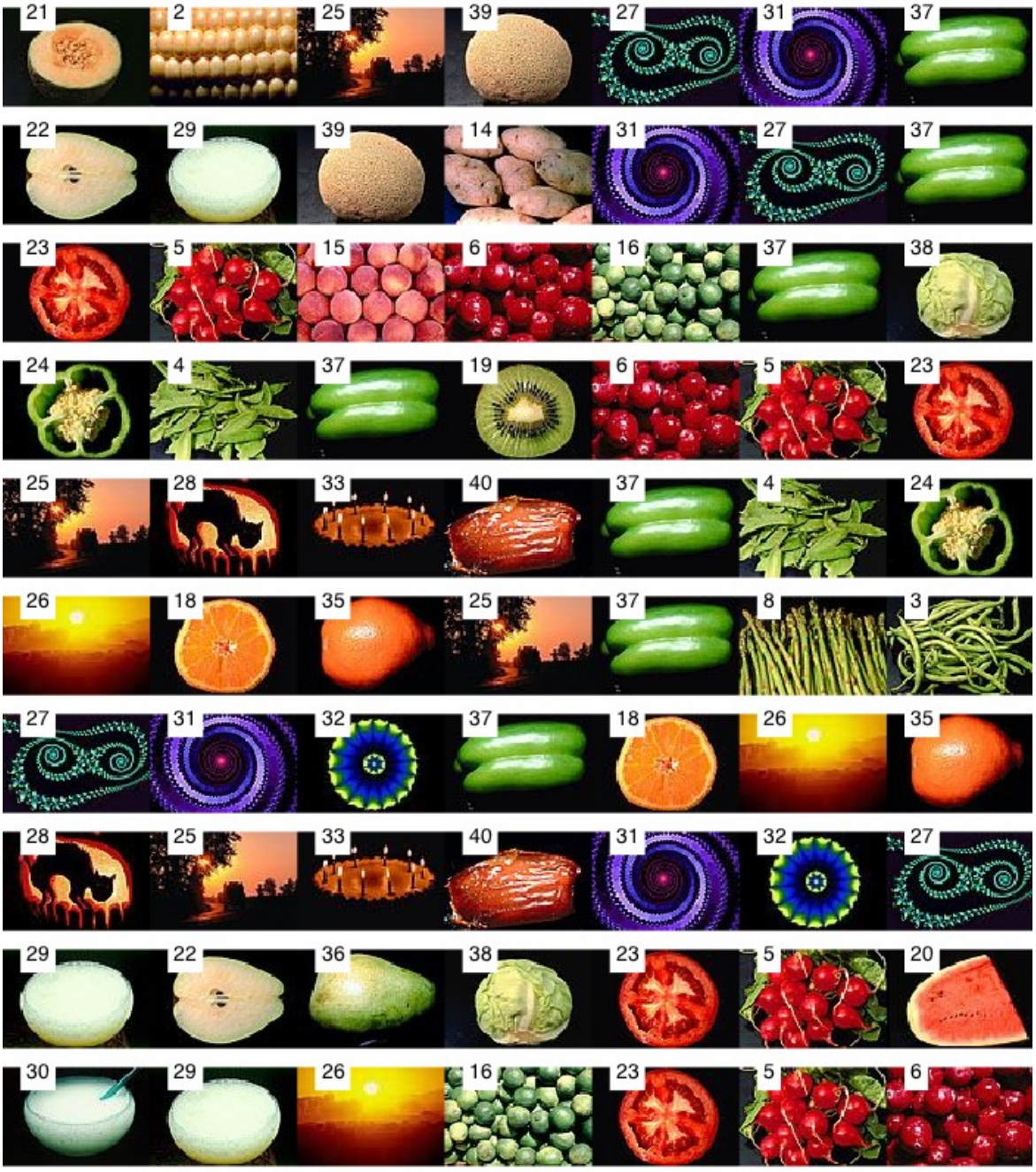


Figure 25 21-30 Color Similarity (friend1)

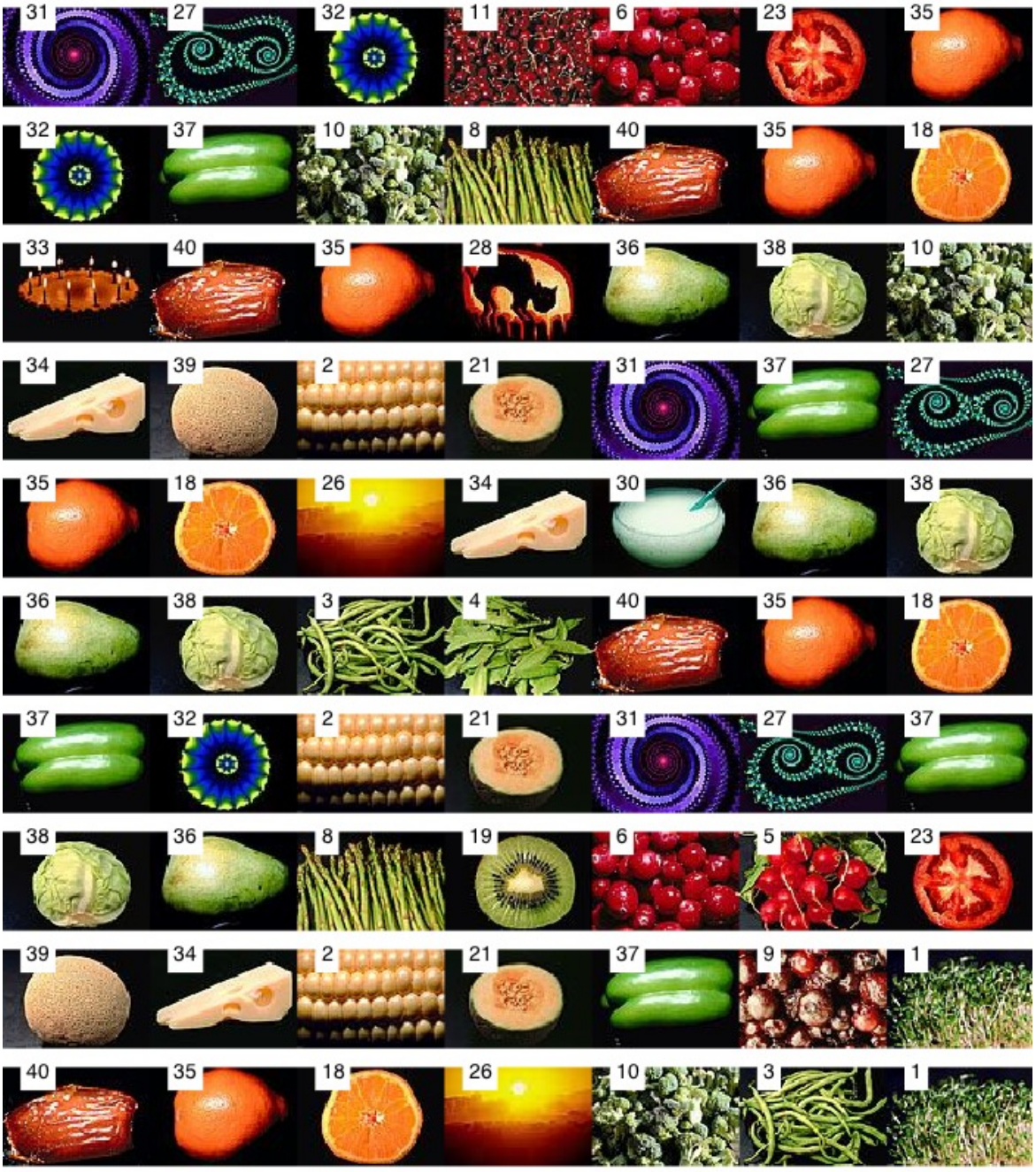


Figure 26 31- 40 Color Similarity (friend1)

Here by texture:

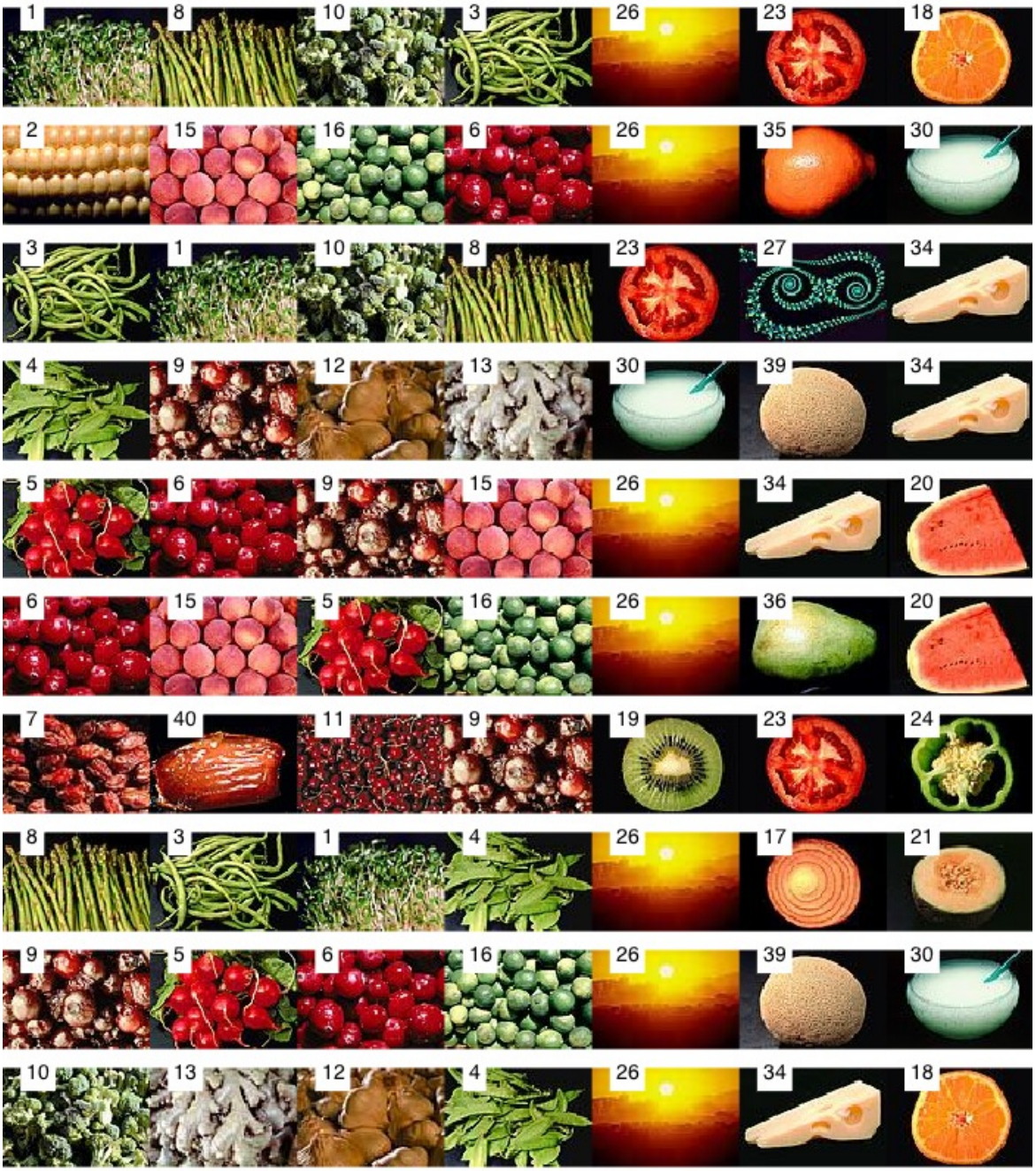


Figure 27 1-10 Texture Similarity (friend1)

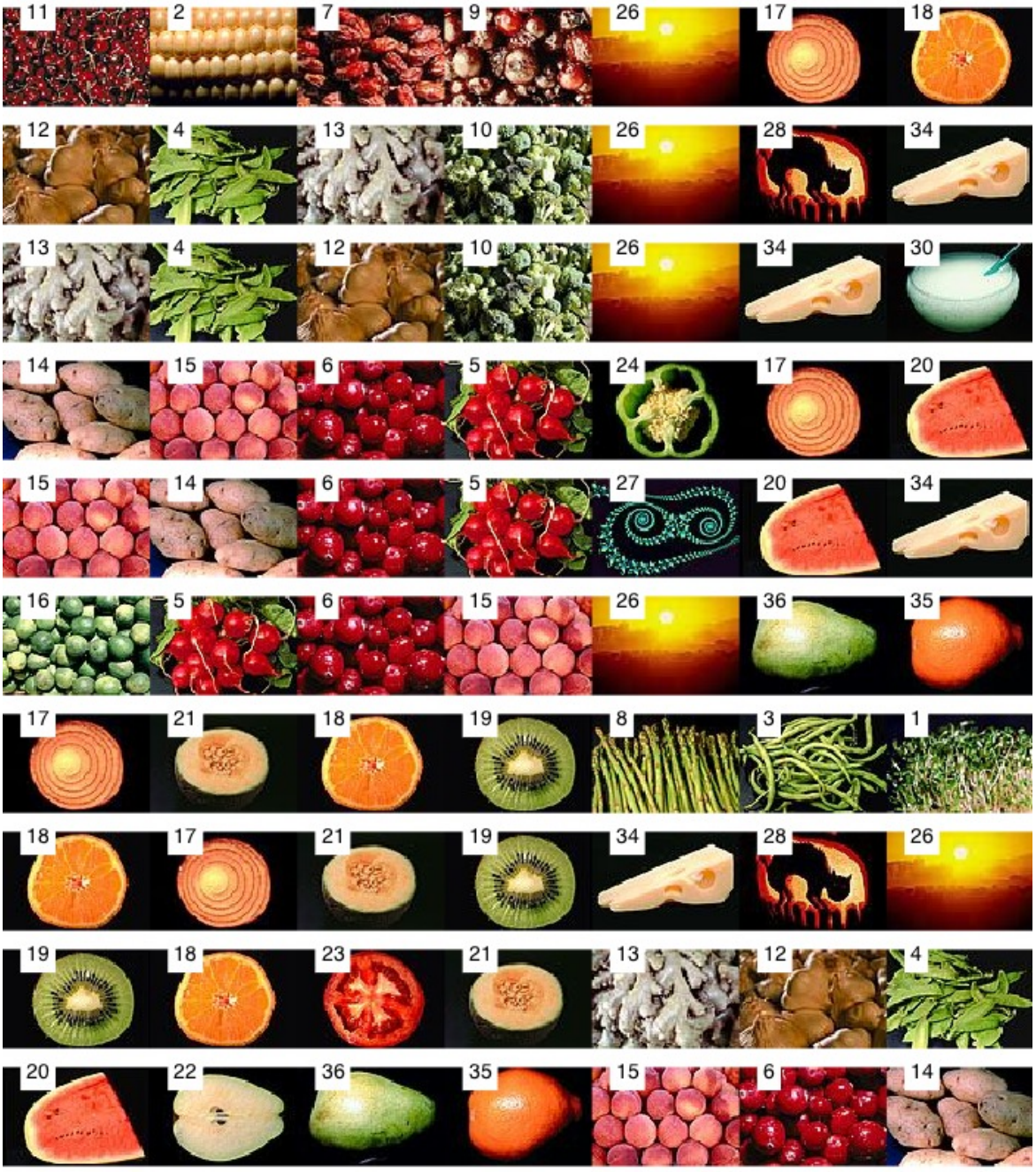


Figure 28 11-20 Texture Similarity (friend1)

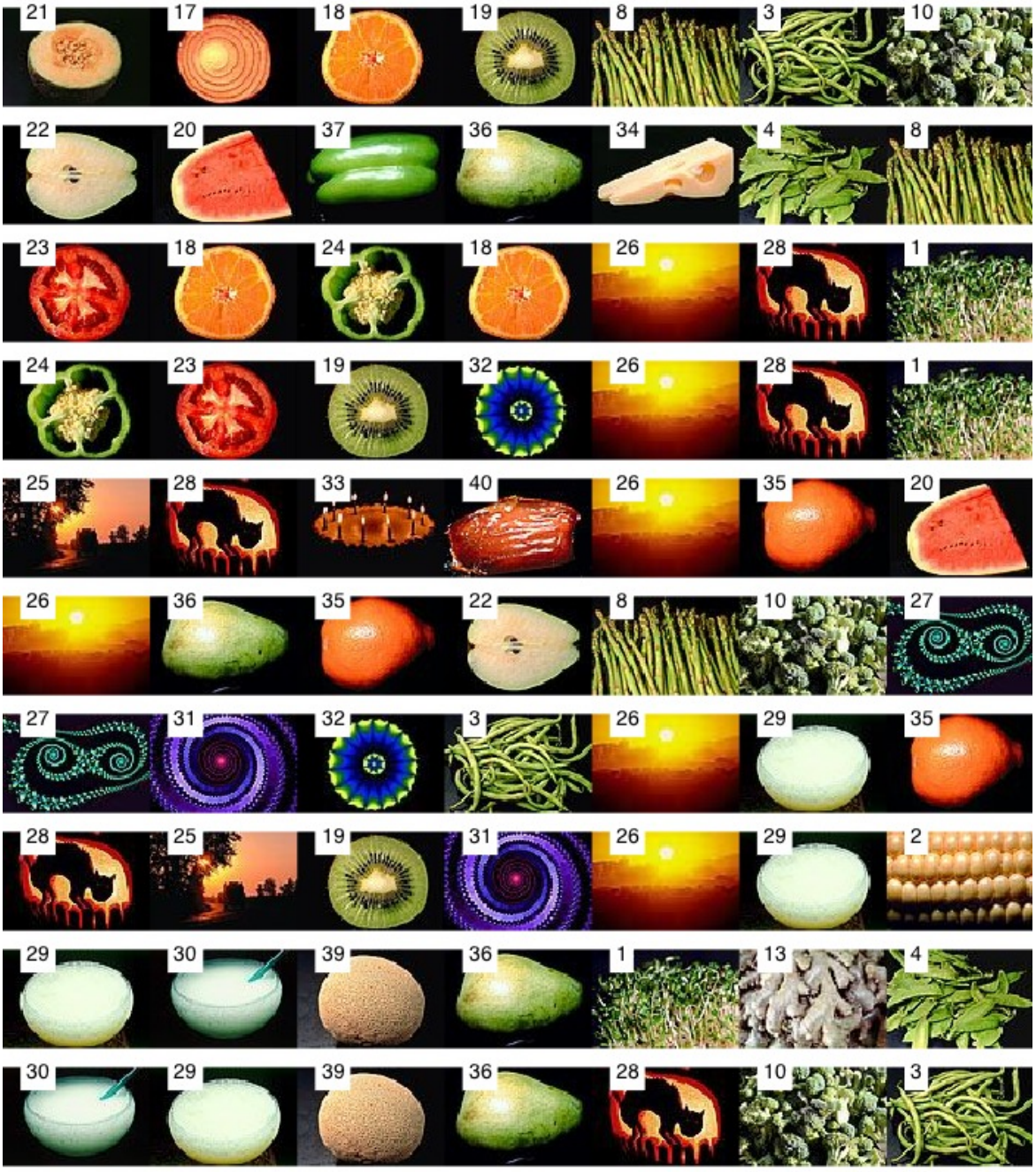


Figure 29 21-30 Texture Similarity (friend1)

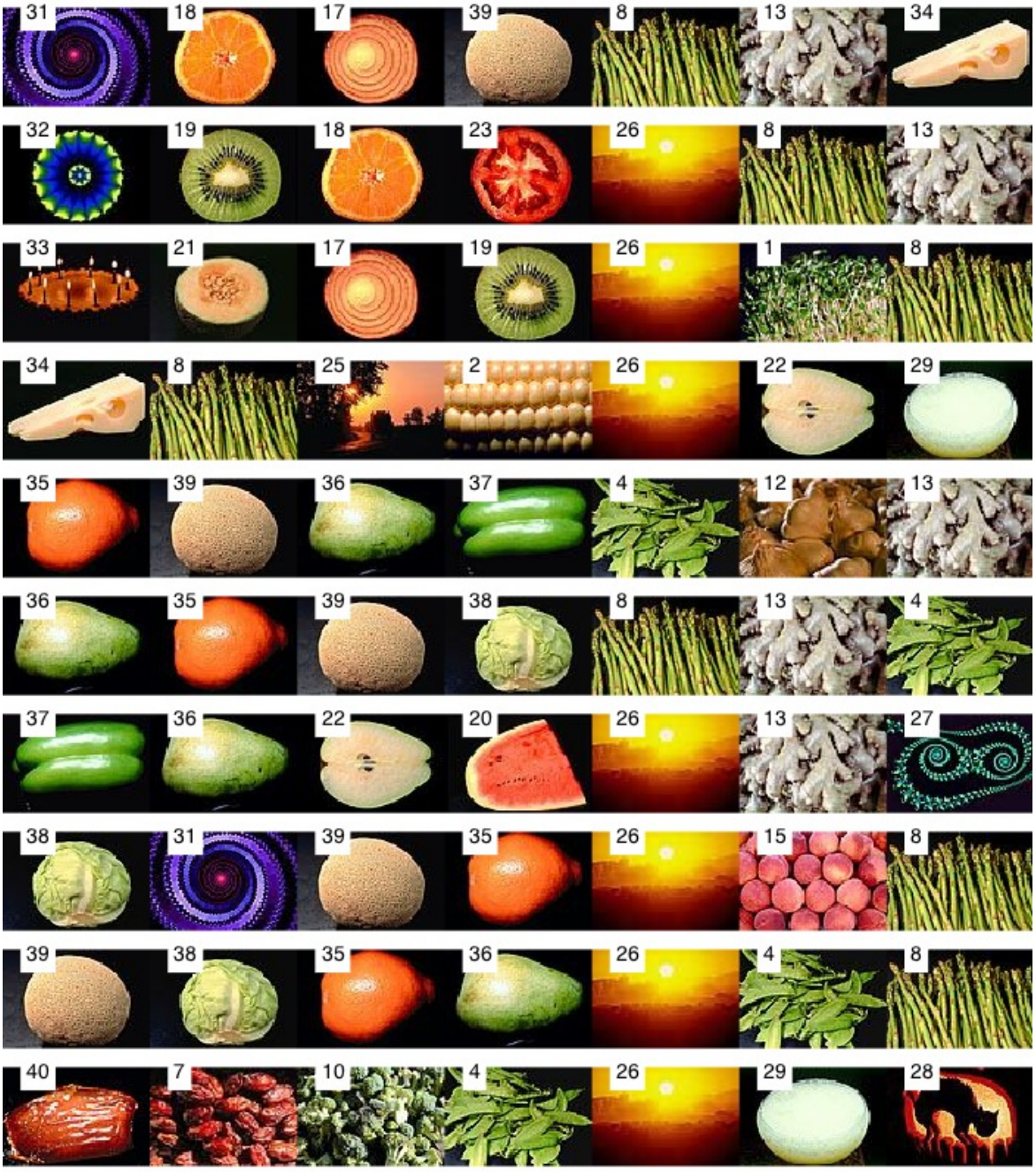


Figure 30 31-40 Texture Similarity (friend1)

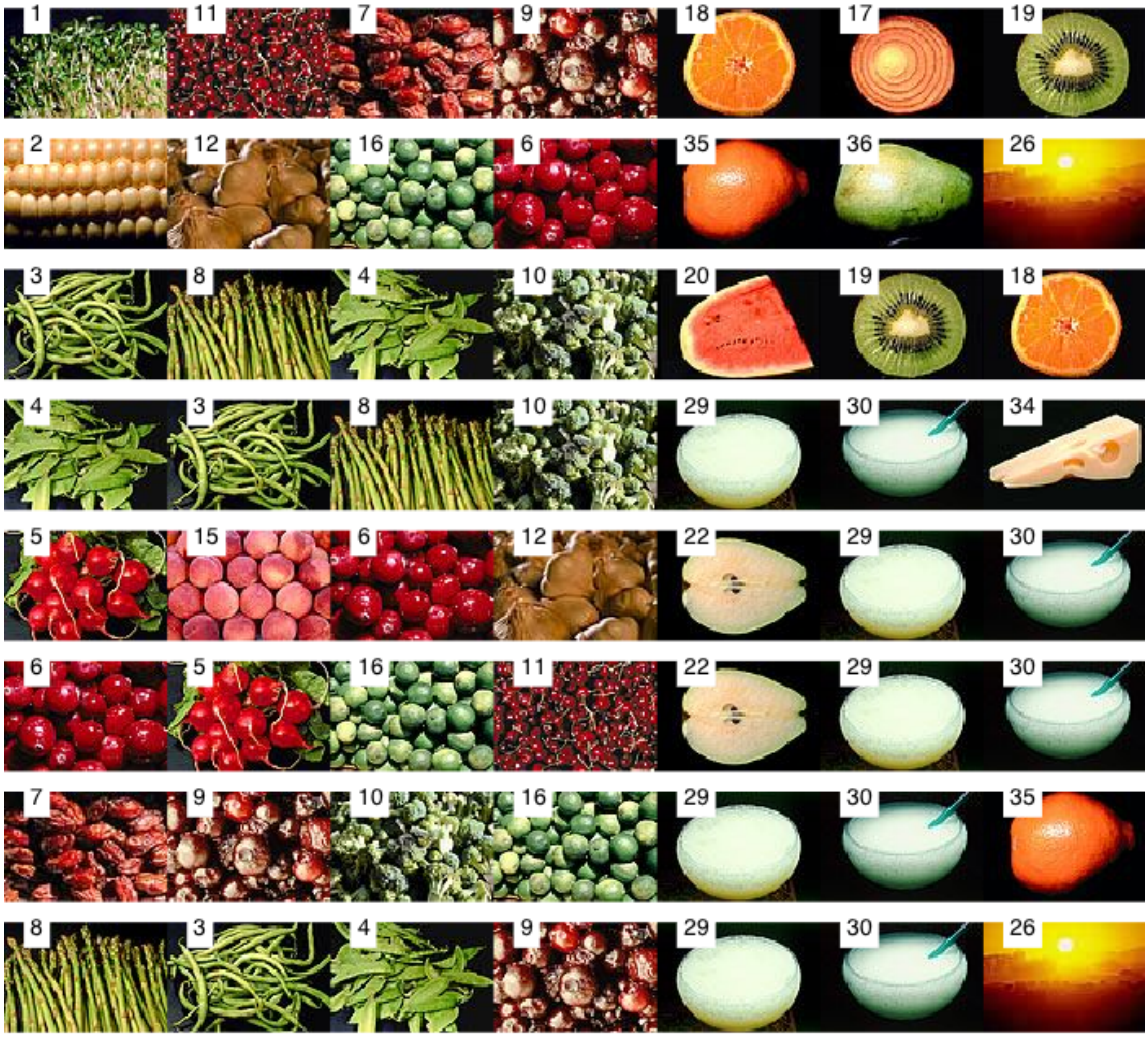
Friend 2 Color Similarity 1

Friend 2 Color Similarity 2

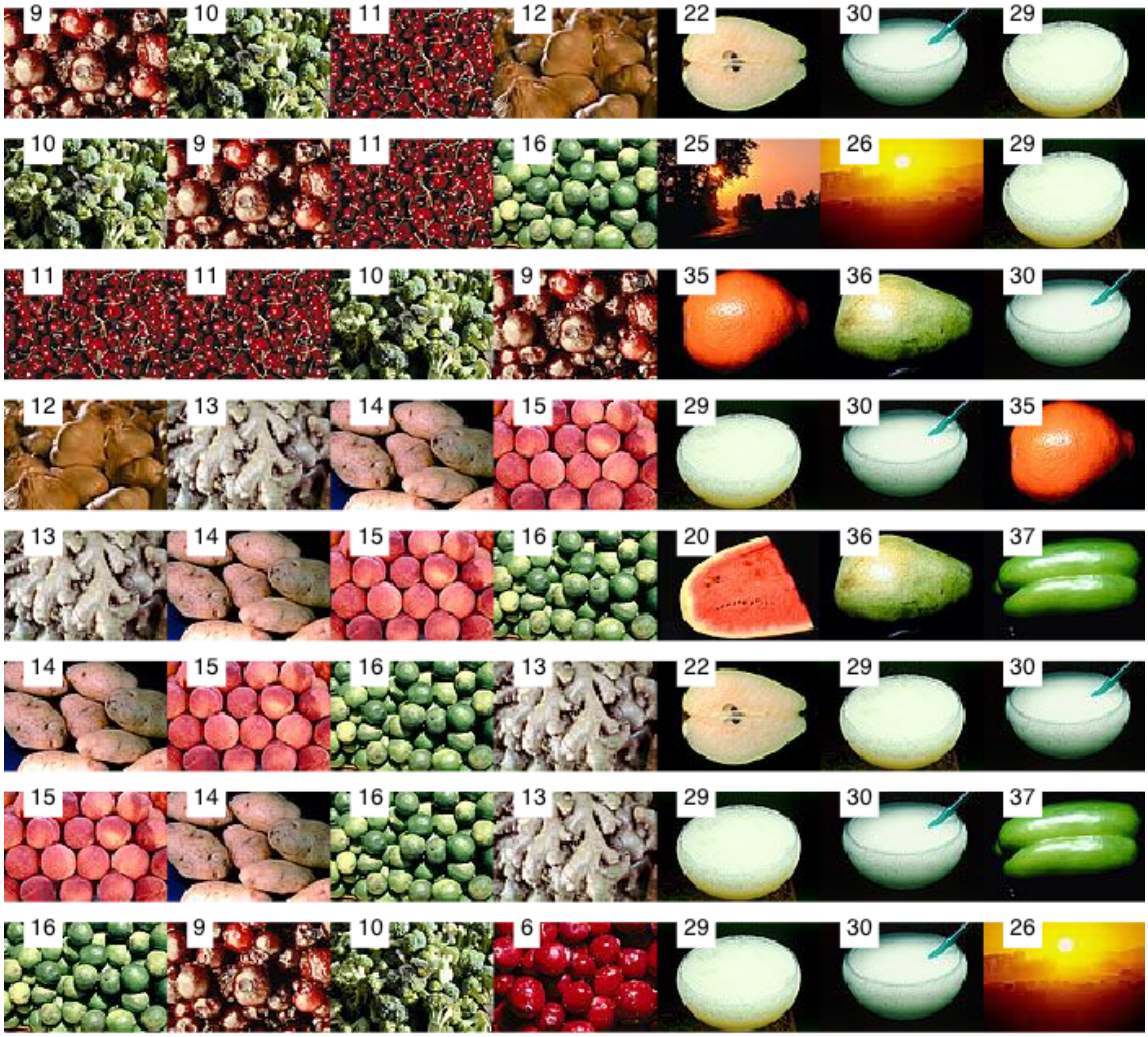
Friend 2 Color Similarity 3

Friend 2 Color Similarity 4

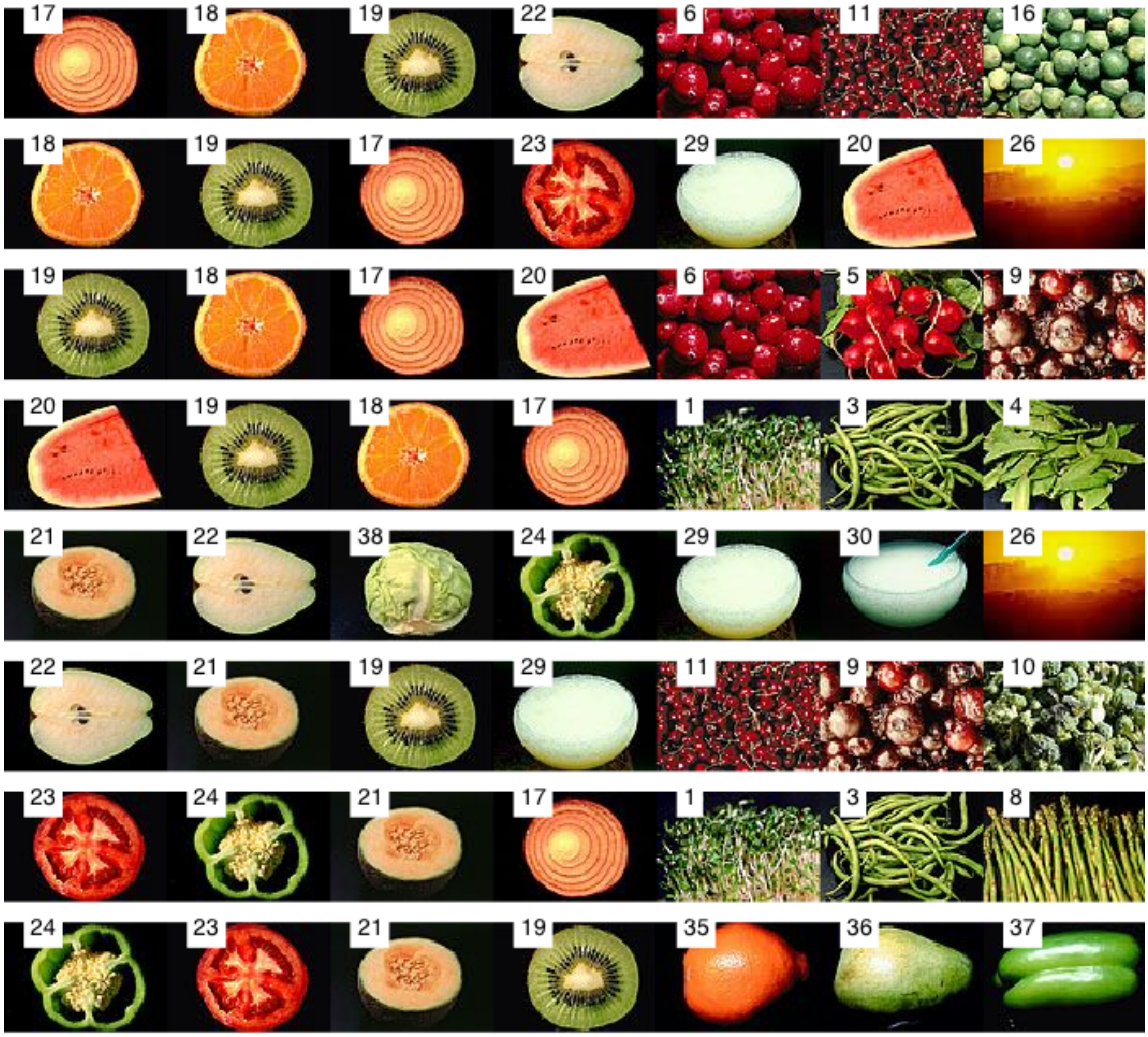
Friend 2 Color Similarity 5



Friend 2 Texture Similarity 1



Friend 2 Texture Similarity 2



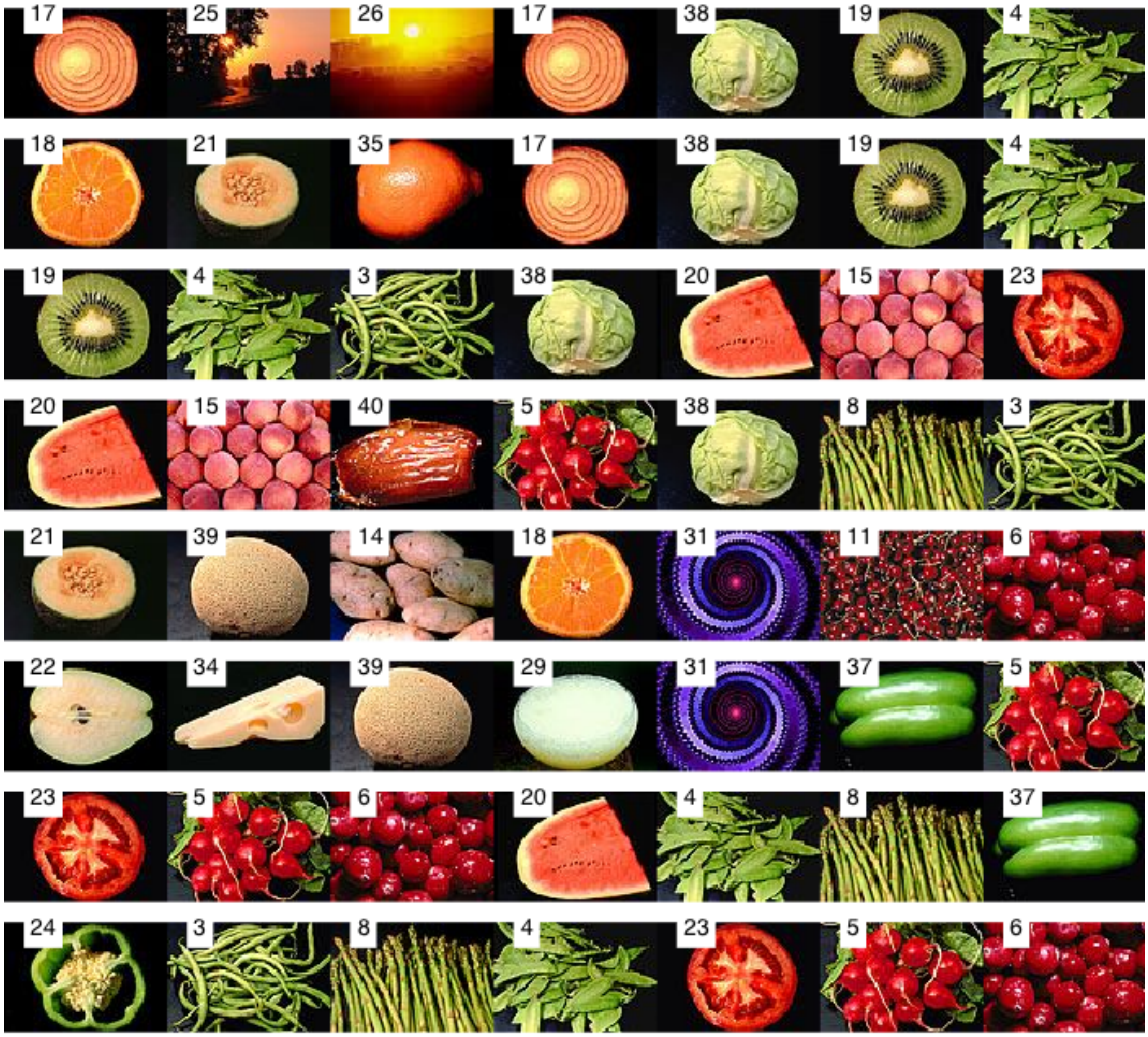
Friend 2 Texture Similarity 3

Friend 2 Texture Similarity 4

Friend 2 Texture Similarity 5

Friends 3 Color Similarity 1

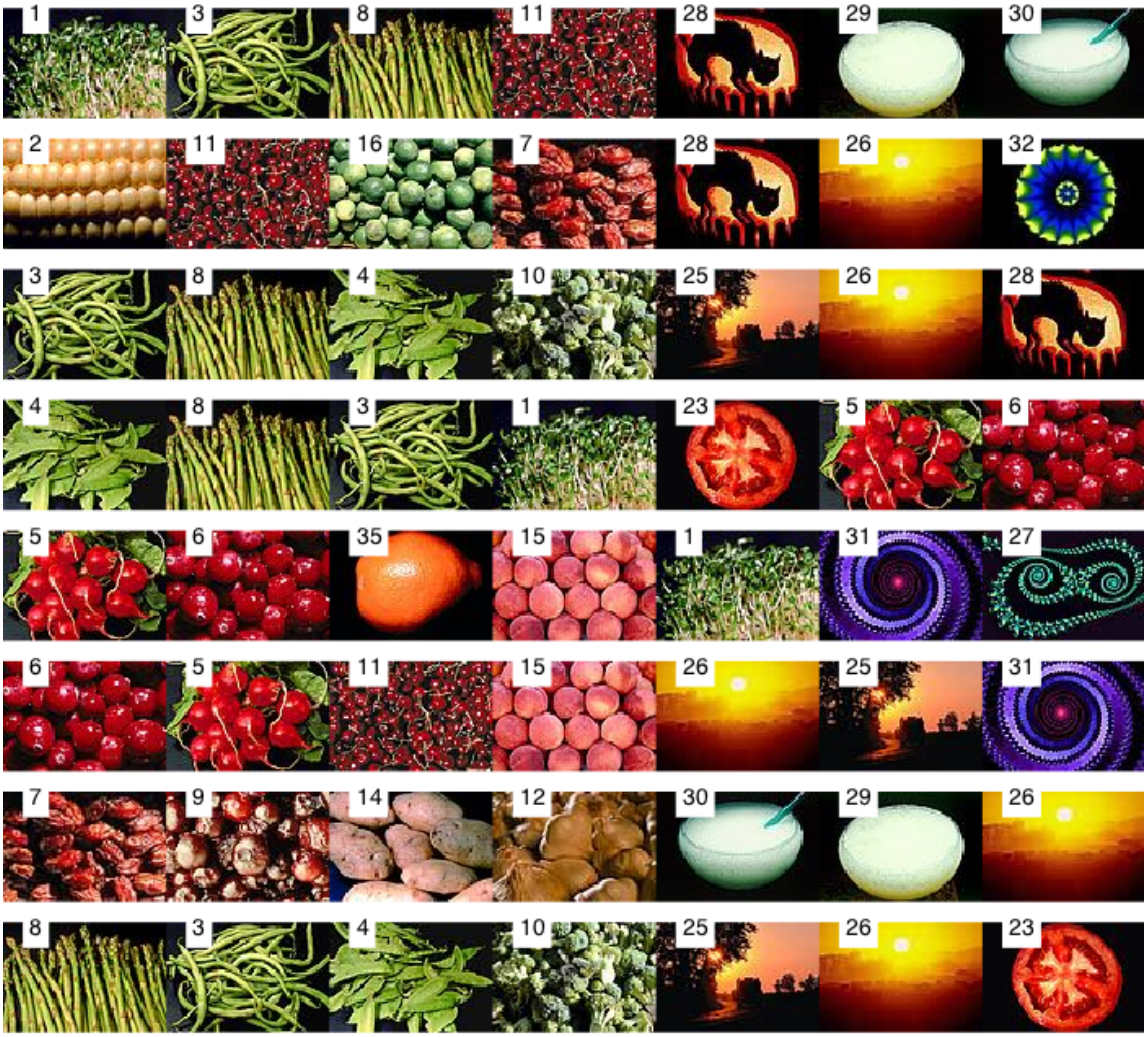
Friends 3 Color Similarity 2



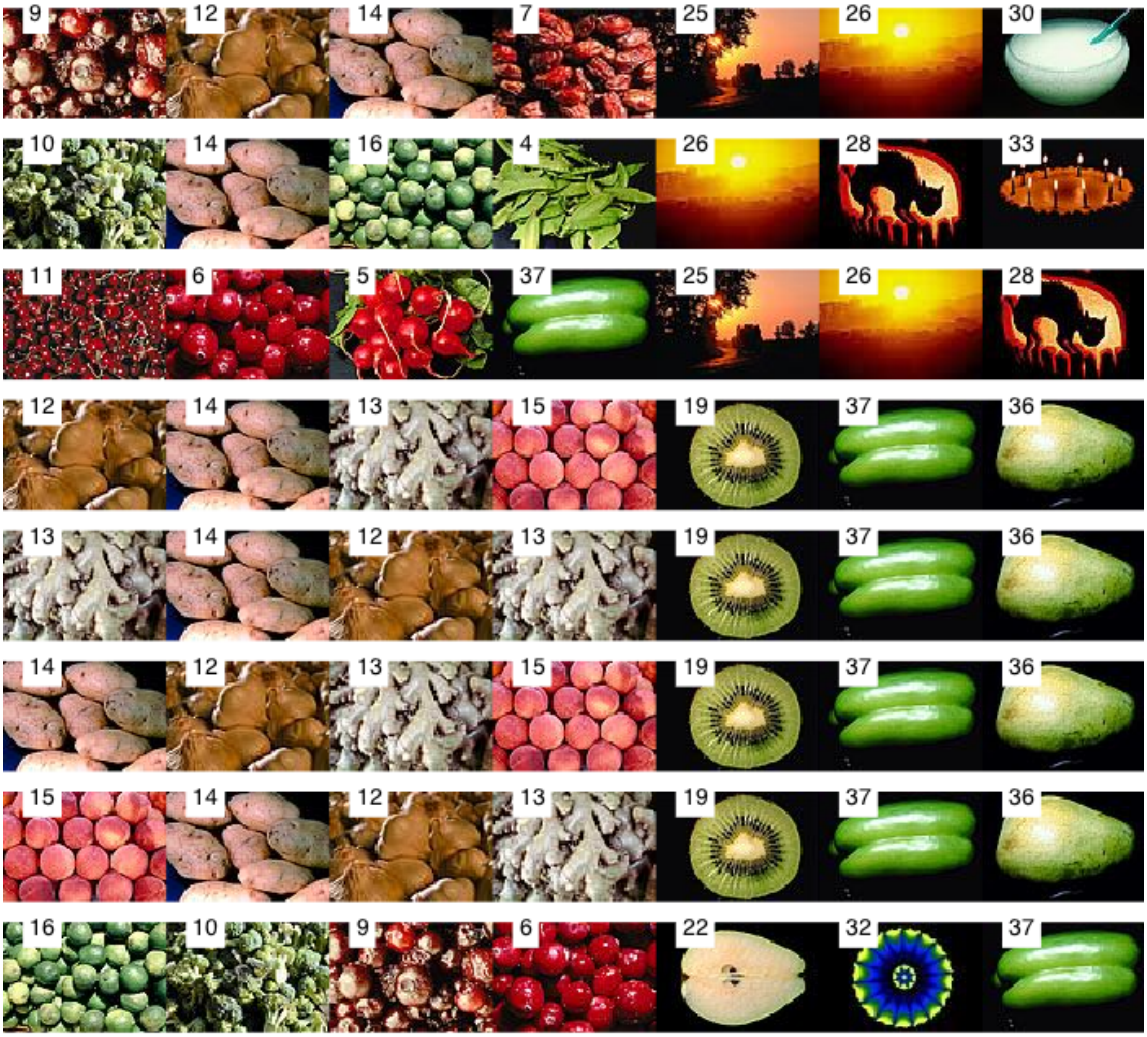
Friends 3 Color Similarity 3

Friends 3 Color Similarity 4

Friends 3 Color Similarity 5



Friend 3 Texture Similarity 1



Friend 3 Texture Similarity 2

Friend 3 Texture Similarity 3

Friend 3 Texture Similarity 4

Friend 3 Texture Similarity 5

To compare my friends C, and T matrices to mine I counted a point towards accuracy each time a friend selected a photo that also appeared in the same row in my matrices. The code for that can be found in '`compare_matches`' Here is an excerpt:

```
for i=1:size(S,1)
    % count if they share at least one similarity or dissimlarity
    if sum(ismember(S(i, 2:end), S1(i, 2:end)))>0
        accuracy = accuracy+1;
    end
end
accuracy = accuracy/size(S,1);
```

Below are the results:
C vs. C friend1: 0.825
C vs. C friend2: 0.950
C vs. C friend3: 0.850
Color Similarity mean: 0.875

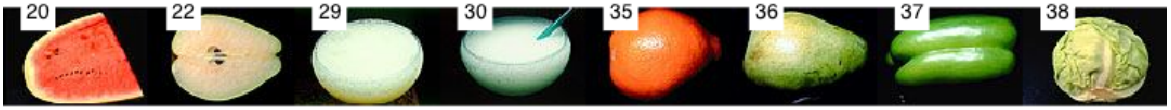
T vs. T friend1: 0.975

T vs. T friend2: 0.850
T vs. T friend3: 0.850
Texture Similarity mean: 0.892

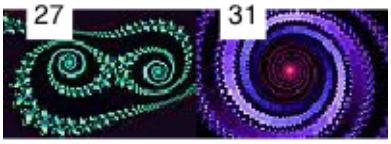
Human Clustering results:

Friend 1 - Cluster 1

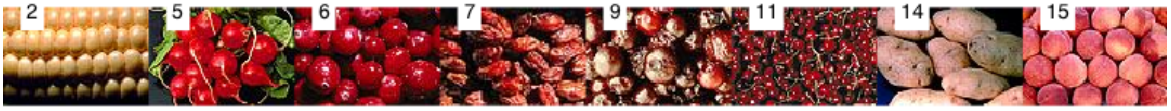
Friend 1 - Cluster 2

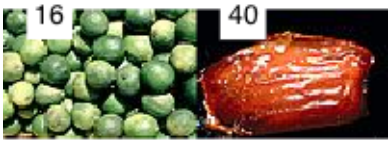


Friend 1 - Cluster 3



Friend 1 - Cluster 4



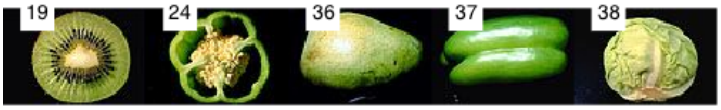


Friend 1 - Cluster 5

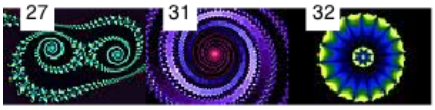
Friend 1 - Cluster 6

Friend 1 - Cluster 7

Friend 2 - Cluster 1



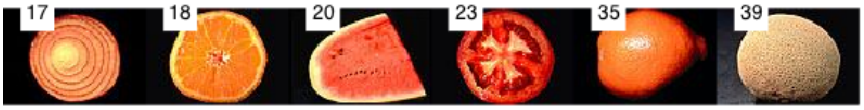
Friend 2 - Cluster 2



Friend 2 - Cluster 3

Friend 2 - Cluster 4

Friend 2 - Cluster 5



Friend 2 - Cluster 6

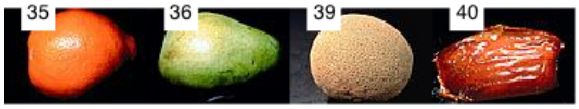
Friend 2 - Cluster 7

Friend 3 Cluster 1

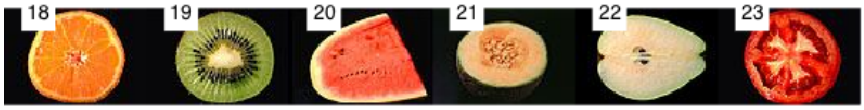
Friend 3 Cluster 2

Friend 3 Cluster 3

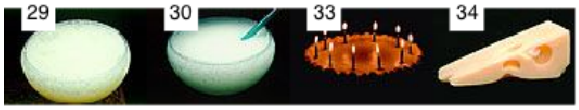
Friend 3 Cluster 4



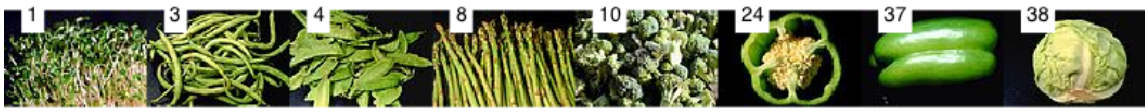
Friend 3 Cluster 5



Friend 3 Cluster 6



Friend 3 Cluster 7



To compare this result with my automated clustering result I used Rand Index. I used code that was freely available on the mathworks forum at this address:
http://www.mathworks.com/matlabcentral/fileexchange/13916--simple--tool-for-estimating-the-number-of-clusters/content/valid_RandIndex.m

The function accepts two cluster results and provides an error ranging from 0 (worst) to 1 (best). Rand Index is computed using all pairs from the data set and counting whether they were clustered together in both results or clustered apart in both results. If they are clustered together in one result and then apart in the other it is not counted. The total is divided by the number of possible pairs (n choose 2).
In pseudo code:

```
RI=0
for points i and j
    if i,j in c1 in X and i,j in c2 in Y
        RI++
    elseif i,j not in c1 in X and i,j not c2 in Y
        RI++
    end
end
RI = RI/(n choose 2)
```

I compared my complete link and single link clustering result to my 3 friends

R and Index - Complete link vs. friend1: **0.773**
R and Index - Complete link vs. friend2: **0.835**
R and Index - Complete link vs. friend3: **0.815**
R and Index - Complete link mean: **0.808**

R and Index - Single link vs. friend1: **0.365**
R and Index - Single link vs. friend2: **0.327**
R and Index - Single link vs. friend3: **0.341**
R and Index - Single link mean: **0.344**

CODE for assignment:

```
% runs steps 1-4 of Assignment 2 in visual interfaces
function assignment2
```

```
close all;;
clc;

files = dir([curr_path filesep 'Images/*.ppm']);

imgObjs = cell(1,40);
bws = cell(1,40);
bws_laplacian = cell(1,40);
chists=[];
m_size = length(files);
color_bin_size = 40;
texture_bin_size = 20;

% load and process all images by texture and color
for i=1:m_size
    imgObj = imread([curr_path filesep 'Images' filesep
files(i).name]);
    imgObjs{i} = imgObj;

    % == ignore background and create color histograms
    r = imgObj(:,:,1);
    b = imgObj(:,:,2);
    g = imgObj(:,:,3);

    blackPixels = (double(r) < 25 & double(g) < 25 & double(b) < 25);
    [~,regions] = bwboundaries(blackPixels,'noholes');

    % sort by size
    table = tabulate(regions(:));
    background_regions = table(table(:,2)>15,1);

    % remove continous regions of color
    rems = [];
    for regi=1:numel(background_regions)
        if any(~blackPixels(regions==background_regions(regi)))
            rems(end+1) = regi;
            break;
        end
    end
    background_regions(rems) = [];

    blackPixels = ismember(regions, background_regions);

    print_image_and_bin(imgObj, getInds(blackPixels), i,
sprintf('%s%scolorbackgrounds25%s%g.png',curr_path, filesep, filesep,
i));

    % set relevant pixels to nan
    r(blackPixels) = nan;
    b(blackPixels) = nan;
    g(blackPixels) = nan;
    imgObj(:,:,1) = r;
    imgObj(:,:,2) = b;
    imgObj(:,:,3) = g;

    % get color histogram
```

```
    threedimhist = get_color_hist(imgObj, color_bin_size);
    chists(:,i) = threedimhist(:)./ sum(threedimhist(:));

    % == ignore background and create texture histograms
    bw = rgb2gray(imgObj);
    bws{i} = bw;

    % compute laplacian
    bw_laplacian = zeros(size(bw));
    for r=2:(size(bw(:,:), 1)-1)
        for c=2:(size(bw(:,:), 2)-1)
            bw_laplacian(r,c) =...
                double(bw(r,c))*9-sum(sum(double(bw(r-1:r+1, c-
1:c+1))));
        end
    end

    frac = 10;

    % remove background - if low texture and not in the center of image
    edgewidth  = floor(size(bw_laplacian,2)/frac);
    edgeheight = floor(size(bw_laplacian,1)/frac);

    % create dead zone mask
    mask = ones(1, size(bw_laplacian,2));
    mask(edgewidth:edgewidth*(frac-1)) = 0;

    % iterate over each image row
    for r=1:size(bw_laplacian,1)
        if (r>edgeheight && r<edgeheight*(frac-1))
            bw_laplacian(r, mask & abs(bw_laplacian(r, :)) < 3 ) = nan;
        else
            bw_laplacian(r, abs(bw_laplacian(r, :)) < 3 ) = nan;
        end
    end

    bws_laplacian{i} = bw_laplacian;

    print_image_and_bin(imgObj, getInds(isnan(bw_laplacian)), i,
sprintf('%s%stexturebackgrounds25%s%g.png',curr_path, filesep, filesep,
i));

    % compute texture histogram
    thists(:,i) =
get_texture_hist(abs(bws_laplacian{i}),texture_bin_size);
    thists(:,i) = thists(:,i) ./ sum(sum(thists(:,i)));
end

% === Step 1: Gross color matching ==
C=compute_print_similarities(imgObjs, chists, 'color');

% === Step 2: Gross texture matching. ==
T=compute_print_similarities(imgObjs, thists, 'texture');


% === Step 3: Combine similarities, and cluster.
r=.6;
```

```
S = r*T + (1-r)*C;
print_similarities(imgObjs, S, sprintf('combination%gprctext',100*r));

% = Cluster to 7 clusters
complete_cluster = hier_clusters(S, 7, 'complete');
print_cluster_result(imgObjs,complete_cluster,'complete');


single_cluster   = hier_clusters(S, 7, 'single');
print_cluster_result(imgObjs,single_cluster,'single');

% === Part 4 benchmark against live human beings

[C1, T1, clusts1] = loadAndrey;
[C2, T2, clusts2] = loadRon;
[C3, T3, clusts3] = loadMichal;

print_img_similarity_mat(imgObjs, C, 'Friend1Color');
print_img_similarity_mat(imgObjs, T, 'Friend1Texture');
print_cluster_result(imgObjs,clusts,'Friend1');

print_img_similarity_idx(imgObjs, C2, 'Friend2Color');
print_img_similarity_idx(imgObjs, T2, 'Friend2Texture');
print_cluster_result(imgObjs,clusts2,'Friend2');

print_img_similarity_idx(imgObjs, C3, 'Friend3Color');
print_img_similarity_idx(imgObjs, T3, 'Friend3Texture');
print_cluster_result(imgObjs,clusts3,'Friend3');

% compare color and texture result to three friends
[~, idx] = sort(C, 2, 'descend');
CIDX = idx(:, [1:4 40-(2:-1:0)]);

ec_adj1 = compare_matches(CIDX, C1);
ec_adj2 = compare_matches(CIDX, C2);
ec_adj3 = compare_matches(CIDX, C3);

[~, idx] = sort(T, 2, 'descend');
TIDX = idx(:, [1:4 40-(2:-1:0)]);

et_adj1 = compare_matches(TIDX, T1);
et_adj2 = compare_matches(TIDX, T2);
et_adj3 = compare_matches(TIDX, T3);

fprintf('\nC vs. C friend1: %1.3f', ec_adj1);
fprintf('\nC vs. C friend2: %1.3f', ec_adj2);
fprintf('\nC vs. C friend3: %1.3f', ec_adj3);
fprintf('\nColor Similarity mean: %1.3f\n', mean([ec_adj1 ec_adj2
ec_adj3]));

fprintf('\nT vs. T friend1: %1.3f', et_adj1);
fprintf('\nT vs. T friend2: %1.3f', et_adj2);
fprintf('\nT vs. T friend3: %1.3f', et_adj3);
fprintf('\nTexture Similarity mean: %1.3f\n', mean([et_adj1 et_adj2
et_adj3]));
```

```
% compare cluster result to three friends
c_v1 = convert_cluster_cell_to_vec(clusts1);
c_v2 = convert_cluster_cell_to_vec(clusts2);
c_v3 = convert_cluster_cell_to_vec(clusts3);

c_complete = convert_cluster_cell_to_vec(complete_cluster);
ec1 = clusteringError(c_v1, c_complete);
ec2 = clusteringError(c_v2, c_complete);
ec3 = clusteringError(c_v3, c_complete);

c_single = convert_cluster_cell_to_vec(single_cluster);
es1 = clusteringError(c_v1, c_single);
es2 = clusteringError(c_v2, c_single);
es3 = clusteringError(c_v3, c_single);

fprintf('\nR and Index - Complete link vs friend1: %1.3f', ec1);
fprintf('\nR and Index - Complete link vs friend2: %1.3f', ec2);
fprintf('\nR and Index - Complete link vs friend3: %1.3f', ec3);
fprintf('\nR and Index - Complete link mean: %1.3f\n', mean([ec1 ec2
ec3]));

fprintf('\nR and Index - Single link vs friend1: %1.3f', es1);
fprintf('\nR and Index - Single link vs friend2: %1.3f', es2);
fprintf('\nR and Index - Single link vs friend3: %1.3f', es3);
fprintf('\nR and Index - Single link mean: %1.3f\n', mean([es1 es2
es3]));
end

function print_cluster_result(imgObjs,clusters, folderprefix)

k=8; %number of pics to print in a row
for c=1:numel(clusters);
    cluster = clusters{c};

    print_all_imgs(imgObjs(cluster), cluster,...
      sprintf('%s-clusters/cluster%g.png',folderprefix,c),...
      k);
end

end

function print_img_row(imgObjs, inds,imgname, opt_h)

% optionally create a new figure
if ~exist('opt_h', 'var')
    m_size = numel(imgObjs);
    opt_h = figure('Position', [100, 100, 100*m_size, 80]);
end

% size of picture row
m_size = numel(imgObjs);

% print a row of images with a white label
for row=1:numel(imgObjs)
    subplot('position', [(row-1)/m_size, 0, 1/m_size, 1]);
    imshow(imgObjs{row});
    text(15,5,num2str(inds(row)), 'background', 'w');
```

```
end
fprintf('\n...Printing image to file %s', imgname);

% save to picture to file
screen2png(opt_h,imgname);

end

function print_all_imgs(imgObjs,inds,imgname, columns,opt_h)

rows = max(numel(imgObjs)/columns, 1);

% optionally create a new figure
if ~exist('opt_h', 'var')
    opt_h = figure('Position', [100, 100, 100*columns, 80*rows]);
end

% print a row of images with a white label
for row=1:rows
    for column=1:columns
        if ((row-1)*columns+column > numel(imgObjs))
            break;
        end
        subplot('position', [(column-1)/columns, 1-((row)/rows),
1/columns, 1/rows]);
        imshow(imgObjs{(row-1)*columns+column});
        text(15,5,num2str(inds((row-1)*columns+column)), 'background',
'w');
    end
end
fprintf('\n...Printing image to file %s', imgname);

% save to picture to file
screen2png(opt_h,imgname);
end

function print_img_similarity_mat(imgObjs, S, foldername)
[~, idx] = sort(S, 2, 'descend');
SIDX = idx(:, [1:4 40-(2:-1:0)]);

print_img_similarity_idx(imgObjs, SIDX, foldername);
end

function print_img_similarity_idx(imgObjs, idx, foldername)

m_size = numel(imgObjs);
h = figure('Position', [100, 100, 700, 640]);
for row=1:8:m_size
    inds = idx(row:(row+8-1), :)';
    imgObj = imgObjs(inds(:));

    print_all_imgs(imgObj,...
                   inds(:),...
                   sprintf('%s%sImg%g-%g_similarity.png',
foldername,filesep,row, row+8-1),...
                   7,...
                   h);
```

```
end

end

function print_image_and_bin(img,binimg,label, filename)
    h= figure(1);
    imshow(img);
    hold on;
    scatter(binimg(:,2), binimg(:,1), 40, '.b');
    hold off;
    text(15,5,num2str(label), 'background', 'w');


    % save to picture to file
    screen2png(h,filename);
end

function S=compute_print_similarities(imgObjs, hists, prefix)
S = pairwise_compare_hists(hists);

% print_similarities(imgObjs, S, prefix)
end

function print_similarities(imgObjs, S, prefix)
[S_sorted, idx(:,:)] = sort(S, 2, 'descend');

[~, l_sim] = min(sum(S_sorted(:,1:4),2));
[~, m_sim] = max(sum(S_sorted(:,1:4),2));

fprintf('\nMost similar group: %g', m_sim);

% most and least similar
imgname = sprintf('%s%smostleastsimilar/%smost.png',curr_path, filesep
, prefix);
print_img_row(imgObjs(idx(m_sim,1:4)), idx(m_sim,1:4), imgname);
imgname = sprintf('%s%smostleastsimilar/%sleast.png',curr_path, filesep
, prefix);
print_img_row(imgObjs(idx(l_sim,1:4)), idx(l_sim,1:4), imgname);

% print all color based similarities
output_folder = sprintf('%s%ssimilarities-%s',curr_path,
filesep,prefix);
print_img_similarity_idx(imgObjs, idx(:, [1:4 40-(2:-1:0)]),
output_folder);
end
% returns a two-column matrix with indices corresponding to the indices
% where the given matrix is positive
function inds=getInds(mat)
    [i,j,~] = find(mat);
    inds = [i,j];
end

function p = curr_path

    persistent f_p;

    if isempty(f_p)
```

```
        filename = mfilename('fullpath');
        [f_p, ~] = fileparts(filename);
    end
    p = f_p;
end

% compares two  matrices of that contain 3 similar and 3 unsimilar
images
% per row. First column is ignored (self similarity)
function accuracy = compare_matches(S, S1)

accuracy=0;
for i=1:size(S,1)

    % count a point if they share at least one similarity or
dissimlarity
    if sum(ismember(S(i, 2:end), S1(i, 2:end)))>0
        accuracy = accuracy+1;
    end
end
accuracy = accuracy/size(S,1);
end

function [RI,MI,HI]=clusteringError(c1,c2)
%RANDINDEX - calculates Rand Indices to compare two partitions
% ARI=RANDINDEX(c1,c2), where c1,c2 are vectors listing the
% class membership, returns the "Hubert & Arabie adjusted Rand index".
% [AR,RI,MI,HI]=RANDINDEX(c1,c2) returns the adjusted Rand index,
% the unadjusted Rand index, "Mirkin's" index and "Hubert's" index.
%
% See L. Hubert and P. Arabie (1985) "Comparing Partitions" Journal of
% Classification 2:193-218

%(C) David Corney (2000)        D.Corney@cs.ucl.ac.uk

if nargin < 2 | min(size(c1)) > 1 | min(size(c2)) > 1
   error('RandIndex: Requires two vector arguments')
   return
end

C=Contingency(c1,c2);   %form contingency matrix

n=sum(sum(C));
nis=sum(sum(C,2).^2);       %sum of squares of sums of rows
njs=sum(sum(C,1).^2);       %sum of squares of sums of columns

t1=nchoosek(n,2);       %total number of pairs of entities
t2=sum(sum(C.^2));  %sum over rows & columnns of nij^2
t3=.5*(nis+njs);

%Expected index (for adjustment)
nc=(n*(n^2+1)-(n+1)*nis-(n+1)*njs+2*(nis*njs)/n)/(2*(n-1));

A=t1+t2-t3;     %no. agreements
D=  -t2+t3;     %no. disagreements

if t1==nc
```

```
   AR=0;            %avoid division by zero; if k=1, define Rand = 0
else
   AR=(A-nc)/(t1-nc);       %adjusted Rand - Hubert & Arabie 1985
end

RI=A/t1;            %Rand 1971      %Probability of agreement
MI=D/t1;            %Mirkin 1970    %p(disagreement)
HI=(A-D)/t1;    %Hubert 1977    %p(agree)-p(disagree)

function Cont=Contingency(Mem1,Mem2)

if nargin < 2 | min(size(Mem1)) > 1 | min(size(Mem2)) > 1
   error('Contingency: Requires two vector arguments')
   return
end

Cont=zeros(max(Mem1),max(Mem2));

for i = 1:length(Mem1);
   Cont(Mem1(i),Mem2(i))=Cont(Mem1(i),Mem2(i))+1;
end

% compare clusters given adjecsency matrix
function s=compare_clusters(A, indsi, indsj, type)

if strcmp(type,'complete')
    s = max(max(A(indsi, indsj)));
else
    s = min(min(A(indsi, indsj)));
end

end

% converts a cell array of clusters to a grouping vector
function c_v= convert_cluster_cell_to_vec(clusts)
    c_v = zeros(1, 40);
    for clu=1:numel(clusts)
        c_v(clusts{clu}) = clu;
    end
end

% computes the histogram for a laplacian of a grayscale image.
function hist = get_texture_hist(bw_laplacian, bin_size)

    % init variables
    hist=[];
    min_t = 0;
    max_t = 255*8;

    nbins = ((max_t-min_t)/bin_size);
    bw_laplacian = bw_laplacian(:);

    % iteration over the bins
    for i=1:ceil(nbins)
        minbin=min_t+bin_size*(i-1);
        maxbin=min_t+bin_size*(i);
```

```
        % count the number of pizels that belong to current bin
        hist(i) = sum(bw_laplacian>=minbin & bw_laplacian<maxbin);
    end
end

% Performs a complete link or single link hierarchichal clustering down
to
% n requested clusters. Returns C the final clustering and H the entire
% hierarchy.
function [C, H] = hier_clusters(A, n, type)

H = {};
D = 1-A; % convert from adjencency to distance
clusters = num2cell(1:size(A, 1)); % init clusters
while numel(clusters)>n
    nclusters = numel(clusters);
    I = zeros(nclusters, nclusters);

    % create cluster similarity matrix
    for i=1:nclusters
        for j=1:nclusters
            I(i,j) = compare_clusters(D, clusters{i}, clusters{j},
type);
        end
    end

    % find the two closest clusters
    [I_sorted, iidx] = sort(I,2, 'ascend');
    [~, clostest_cluster] = min(sum(I_sorted(:,1:2), 2));
    if numel(clostest_cluster) > 1
        clostest_cluster = randsample(clostest_cluster,1);
    end

    % merge closest clusters
    merge_i = clostest_cluster;
    merge_j = iidx(merge_i, 2);
    clusters{merge_i} = union(clusters{merge_i},clusters{merge_j});
    clusters(merge_j) = []; % delete one cluster after merge
    H(end+1) = {clusters};
end

C = H{end};

end

% sum over the absolut difference at each vairable
function score = l1_compare(imghist1, imghist2)
    vec1 = imghist1(:);
    vec2 = imghist2(:);
    score = sum(abs(vec1./sum(vec1) - vec2./sum(vec2))./2;
end

function A=pairwise_compare_hists(hists)

m_size = size(hists,2);
A = zeros(m_size, m_size);  % distance matrix
for i=1:m_size
```

```
    for j=1:m_size
        A(i,j) = 1-l1_compare(hists(:,i), hists(:,j));
    end
end
end

% SCREEN2JPEG Generate a JPEG file of the current figure with
%   dimensions consistent with the figure's screen dimensions.
function screen2png(h,filename)

% create directory if needed
[pathstr, ~, ~] = fileparts(filename);
if ~exist(pathstr, 'dir')
   mkdir(pathstr);
end

set(gcf,'Units','pixels');

pos = get(h,'Position');
newpos = pos/100;

% sets the position\size of the current graphical object before
printing
set(h,'PaperUnits','inches',...
     'PaperPosition',newpos)

% print
print('-dpng', filename, '-r100');
drawnow;
end

function [C, T, clusts]=loadAndrey
C = zeros(40,6);
T = zeros(40,6);
clust_v=[];

C(1, :) = [10 8 4 40 6 5];

C(2, :) = [21 34 29 32 27 31];

C(3, :) = [1 10 8 23 6 5];

C(4, :) = [3 8 1 6 5 15];

C(5,:) = [9 12 13 30 39 34];

C(6,:) = [5 23 15 1 3 10];

C(7,:) = [11 9 6 37 16 24];

C(8,:) = [1 3 38 31 40 7];

C(9,:) = [11 7 6 37 4 3];

C(10,:) = [3 1 8 28 6 11];
```

```
C(11,:) = [7 28 23 19 4 24];

C(12,:) = [14 39 2 37 27 28];

C(13,:) = [14 23 22 6 5 11];

C(14,:) = [22 39 2 31 27 32];

C(15,:) = [23 5 20 37 36 16];

C(16,:) = [37 36 19 6 11 5];

C(17,:) = [18 35 26 31 32 37];

C(18,:) = [35 17 21 31 37 16];

C(19,:) = [24 4 16 6 5 15];

C(20,:) = [15 23 35 37 1 3];

C(21,:) = [2 25 39 27 31 37];

C(22,:) = [29 39 14 31 27 37];

C(23,:) = [5 15 6 16 37 38];

C(24,:) = [4 37 19 6 5 23];

C(25,:) = [28 33 40 37 4 24];

C(26,:) = [18 35 25 37 8 3];

C(27,:) = [31 32 37 18 26 35];

C(28,:) = [25 33 40 31 32 27];

C(29,:) = [22 36 38 23 5 20];

C(30,:) = [29 26 16 23 5 6];

C(31,:) = [27 32 11 6 23 35];

C(32,:) = [37 10 8 40 35 18];

C(33,:) = [40 35 28 36 38 10];

C(34,:) = [39 2 21 31 37 27];

C(35,:) = [18 26 34 30 36 38];

C(36,:) = [38 3 4 40 35 18];

C(37,:) = [32 2 21 31 27 37];
```

```
C(38,:) = [36 8 19 6 5 23];

C(39,:) = [34 2 21 37 9 1];

C(40,:) = [35 18 26 10 3 1];

C = [(1:40)' C];

T(1,:) = [8 10 3 26 23 18];

T(2,:) = [15 16 6 26 35 30];

T(3,:) = [1 10 8 23 27 34];

T(4,:) = [9 12 13 30 39 34];

T(5,:) = [6 9 15 26 34 20];

T(6,:) = [15 5 16 26 36 20];

T(7,:) = [40 11 9 19 23 24];

T(8,:) = [3 1 4 26 17 21];

T(9,:) = [5 6 16 26 39 30];

T(10,:) = [13 12 4 26 34 18];

T(11,:) = [2 7 9 26 17 18];

T(12,:) = [4 13 10 26 28 34];

T(13,:) = [4 12 10 26 34 30];

T(14,:) = [15 6 5 24 17 20];

T(15,:) = [14 6 5 27 20 34];

T(16,:) = [5 6 15 26 36 35];

T(17,:) = [21 18 19 8 3 1];

T(18,:) = [17 21 19 34 28 26];

T(19,:) = [18 23 21 13 12 4];

T(20,:) = [22 36 35 15 6 14];

T(21,:) = [17 18 19 8 3 10];

T(22,:) = [20 37 36 34 4 8];

T(23,:) = [18 24 18 26 28 1];
```

```
T(24,:) = [23 19 32 26 28 1];

T(25,:) = [28 33 40 26 35 20];

T(26,:) = [36 35 22 8 10 27];

T(27,:) = [31 32 3 26 29 35];

T(28,:) = [25 19 31 26 29 2];

T(29,:) = [30 39 36 1 13 4];

T(30,:) = [29 39 36 28 10 3];

T(31,:) = [18 17 39 8 13 34];

T(32,:) = [19 18 23 26 8 13];

T(33,:) = [21 17 19 26 1 8];

T(34,:) = [8 25 2 26 22 29];

T(35,:) = [39 36 37 4 12 13];

T(36,:) = [35 39 38 8 13 4];

T(37,:) = [36 22 20 26 13 27];

T(38,:) = [31 39 35 26 15 8];

T(39,:) = [38 35 36 26 4 8];

T(40,:) = [7 10 4 26 29 28];

T = [(1:40)' T];

clust_v(1) = 1;

clust_v(2) = 4;

clust_v(3) = 1;

clust_v(4) = 1;

clust_v(5) = 4;

clust_v(6) = 4;

clust_v(7) = 4;

clust_v(8) = 1;

clust_v(9) = 4;
```

```
clust_v(10) = 1;

clust_v(11) = 4;

clust_v(12) = 1;

clust_v(13) = 1;

clust_v(14) = 4;

clust_v(15) = 4;

clust_v(16) = 4;

clust_v(17) = 5;

clust_v(18) = 5;

clust_v(19) = 5;

clust_v(20) = 2;

clust_v(21) = 5;

clust_v(22) = 2;

clust_v(23) = 5;

clust_v(24) = 5;

clust_v(25) = 6;

clust_v(26) = 6;

clust_v(27) = 3;

clust_v(28) = 7;

clust_v(29) = 2;

clust_v(30) = 2;

clust_v(31) = 3;

clust_v(32) = 5;

clust_v(33) = 6;

clust_v(34) = 6;

clust_v(35) = 2;

clust_v(36) = 2;
```

```
clust_v(37) = 2;

clust_v(38) = 2;

clust_v(39) = 2;

clust_v(40) = 4;

clusts = {};
clusts{1} = find(clust_v==1);
clusts{2} = find(clust_v==2);
clusts{3} = find(clust_v==3);
clusts{4} = find(clust_v==4);
clusts{5} = find(clust_v==5);
clusts{6} = find(clust_v==6);
clusts{7} = find(clust_v==7);
end

function [C, T, clusts]=loadMichal
C = zeros(40,6);
T = zeros(40,6);

clust_v=[];

C(1,:) = [3 8 11 25 26 29];
C(2,:) = [21 34 22 29 20 37];
C(3,:) = [8 1 4 34 29 30];
C(4,:) = [3 8 1 23 18 20];
C(5,:) = [6 9 16 4 29 20];
C(6,:) = [5 11 23 4 8 3];
C(7,:) = [9 33 14 30 29 13];
C(8,:) = [4 3 1 23 5 6];
C(9,:) = [10 16 12 30 29 37];
C(10,:) = [16 8 3 23 5 6];
C(11,:) = [23 5 7 37 4 3];
C(12,:) = [24 40 34 32 27 5];
C(13,:) = [29 14 12 28 25 27];
C(14,:) = [12 39 40 19 4 8];
C(15,:) = [20 5 6 37 36 4];
C(16,:) = [10 1 3 4 6 5];
C(17,:) = [25 26 17 38 19 4];
C(18,:) = [21 35 17 38 19 4];
C(19,:) = [4 3 38 20 15 23];
C(20,:) = [15 40 5 38 8 3];
C(21,:) = [39 14 18 31 11 6];
C(22,:) = [34 39 29 31 37 5];
C(23,:) = [5 6 20 4 8 37];
C(24,:) = [3 8 4 23 5 6];
C(25,:) = [39 21 17 31 13 30];
C(26,:) = [40 35 18 31 27 32];
C(27,:) = [16 36 37 23 5 36];
C(28,:) = [25 23 40 30 13 29];
C(29,:) = [30 22 13 28 25 31];
C(30,:) = [22 29 34 28 25 31];
C(31,:) = [7 40 6 29 30 37];
C(32,:) = [27 37 36 13 29 30];
C(33,:) = [40 7 11 32 27 1];
C(34,:) = [22 21 29 31 27 32];
```

```
C(35,:) = [18 26 40 31 32 27];
C(36,:) = [38 36 16 20 5 23];
C(37,:) = [24 4 3 23 5 6];
C(38,:) = [3 16 36 23 5 6];
C(39,:) = [21 2 17 31 27 32];
C(40,:) = [33 7 35 30 13 29];

C = [(1:40)' C];

T(1,:) = [3 8 11 28 29 30];
T(2,:) = [11 16 7 28 26 32];
T(3,:) = [8 4 10 25 26 28];
T(4,:) = [8 3 1 23 5 6];
T(5,:) = [6 35 15 1 31 27];
T(6,:) = [5 11 15 26 25 31];
T(7,:) = [9 14 12 30 29 26];
T(8,:) = [3 4 10 25 26 23];
T(9,:) = [12 14 7 25 26 30];
T(10,:) = [14 16 4 26 28 33];
T(11,:) = [6 5 37 25 26 28];
T(12,:) = [14 13 15 19 37 36];
T(13,:) = [14 12 13 19 37 36];
T(14,:) = [12 13 15 19 37 36];
T(15,:) = [14 12 13 19 37 36];
T(16,:) = [10 9 6 22 32 37];
T(17,:) = [18 21 39 1 10 6];
T(18,:) = [38 21 17 1 6 15];
T(19,:) = [18 23 17 1 10 15];
T(20,:) = [19 21 22 6 7 8];
T(21,:) = [22 29 30 10 1 15];
T(22,:) = [21 29 30 6 7 1];
T(23,:) = [24 19 32 37 26 25];
T(24,:) = [23 19 32 25 26 37];
T(25,:) = [26 33 30 11 6 7];
T(26,:) = [25 33 30 11 6 7];
T(27,:) = [32 31 24 29 30 26];
T(28,:) = [25 26 18 13 14 15];
T(29,:) = [30 20 22 15 16 6];
T(30,:) = [29 26 37 15 16 6];
T(31,:) = [32 27 3 25 26 33];
T(32,:) = [31 27 24 25 26 37];
T(33,:) = [19 18 25 37 36 29];
T(34,:) = [22 21 29 16 15 14];
T(35,:) = [36 37 40 13 12 9];
T(36,:) = [37 35 40 12 11 10];
T(37,:) = [36 35 40 6 7 3];
T(38,:) = [39 4 17 16 15 2];
T(39,:) = [38 17 40 11 2 9];
T(40,:) = [37 7 33 30 29 1];

T = [(1:40)' T];

clust_v(1) = 7;
clust_v(2) = 1;
clust_v(3) = 7;
clust_v(4) = 7;
clust_v(5) = 1;
clust_v(6) = 1;
clust_v(7) = 1;
```

```
clust_v(8) = 7;
clust_v(9) = 3;
clust_v(10) = 7;
clust_v(11) = 1;
clust_v(12) = 3;
clust_v(13) = 3;
clust_v(14) = 3;
clust_v(15) = 1;
clust_v(16) = 1;
clust_v(17) = 3;
clust_v(18) = 5;
clust_v(19) = 5;
clust_v(20) = 5;
clust_v(21) = 5;
clust_v(22) = 5;
clust_v(23) = 5;
clust_v(24) = 7;
clust_v(25) = 2;
clust_v(26) = 2;
clust_v(27) = 2;
clust_v(28) = 2;
clust_v(29) = 6;
clust_v(30) = 6;
clust_v(31) = 2;
clust_v(32) = 2;
clust_v(33) = 6;
clust_v(34) = 6;
clust_v(35) = 4;
clust_v(36) = 4;
clust_v(37) = 7;
clust_v(38) = 7;
clust_v(39) = 4;
clust_v(40) = 4;

clusts = {};
clusts{1} = find(clust_v==1);
clusts{2} = find(clust_v==2);
clusts{3} = find(clust_v==3);
clusts{4} = find(clust_v==4);
clusts{5} = find(clust_v==5);
clusts{6} = find(clust_v==6);
clusts{7} = find(clust_v==7);
end


 function [C, T, clusts]=loadRon

C = zeros(40,6);
T = zeros(40,6);

clust_v=[];


C(1,:) = [8 16 36 23 5 6];
C(2,:) = [21 34 22 31 27 11];
C(3,:) = [8 1 4 23 5 6];
C(4,:) = [8 3 16 6 15 31];
C(5,:) = [6 23 11 4 8 3];
```

```
C(6,:) = [5 11 23 4 8 3];
C(7,:) = [9 33 6 30 29 13];
C(8,:) = [3 4 24 23 6 15];
C(9,:) = [7 6 15 36 38 32];
C(10,:) = [16 1 8 23 32 35];
C(11,:) = [6 7 9 37 36 32];
C(12,:) = [2 39 18 6 5 23];
C(13,:) = [2 29 14 11 8 23];
C(14,:) = [15 13 12 6 11 27];
C(15,:) = [6 5 11 8 4 3];
C(16,:) = [4 8 3 6 5 15];
C(17,:) = [18 35 39 32 31 37];
C(18,:) = [35 1 39 38 37 6];
C(19,:) = [24 38 16 31 25 6];
C(20,:) = [5 23 11 32 31 29];
C(21,:) = [34 39 12 37 6 5];
C(22,:) = [29 30 34 21 23 29];
C(23,:) = [6 15 20 36 38 37];
C(24,:) = [36 38 19 23 20 25];
C(25,:) = [28 40 26 1 10 16];
C(26,:) = [25 28 33 8 37 36];
C(27,:) = [32 31 3 23 35 18];
C(28,:) = [25 26 33 28 19 37];
C(29,:) = [30 22 34 35 20 25];
C(30,:) = [29 22 24 35 20 25];
C(31,:) = [32 27 11 2 38 36];
C(32,:) = [31 27 19 23 6 4];
C(33,:) = [26 28 25 8 4 3];
C(34,:) = [29 30 22 23 20 28];
C(35,:) = [18 17 39 37 19 36];
C(36,:) = [38 19 24 23 20 6];
C(37,:) = [4 36 38 6 15 25];
C(38,:) = [36 19 17 28 25 6];
C(39,:) = [17 18 21 6 22 15];
C(40,:) = [9 7 12 36 37 19];

C = [(1:40)' C];

T(1,:) = [11 7 9 18 17 19];
T(2,:) = [12 16 6 35 36 26];
T(3,:) = [8 4 10 20 19 18];
T(4,:) = [3 8 10 29 30 34];
T(5,:) = [15 6 12 22 29 30];
T(6,:) = [5 16 11 22 29 30];
T(7,:) = [9 10 16 29 30 35];
T(8,:) = [3 4 9 29 30 26];
T(9,:) = [10 11 12 22 30 29];
T(10,:) = [9 11 16 25 26 29];
T(11,:) = [11 10 9 35 36 30];
T(12,:) = [13 14 15 29 30 35];
T(13,:) = [14 15 16 20 36 37];
T(14,:) = [15 16 13 22 29 30];
T(15,:) = [14 16 13 29 30 37];
T(16,:) = [9 10 6 29 30 26];
T(17,:) = [18 19 22 6 11 16];
T(18,:) = [19 17 23 29 20 26];
T(19,:) = [18 17 20 6 5 9];
T(20,:) = [19 18 17 1 3 4];
T(21,:) = [22 38 24 29 30 26];
```

```
T(22,:) = [21 19 29 11 9 10];
T(23,:) = [24 21 17 1 3 8];
T(24,:) = [23 21 19 35 36 37];
T(25,:) = [26 29 30 1 3 8];
T(26,:) = [25 28 33 33 10 9];
T(27,:) = [31 32 3 35 36 37];
T(28,:) = [26 25 18 22 37 25];
T(29,:) = [26 25 18 11 9 10];
T(30,:) = [29 37 35 27 21 32];
T(31,:) = [32 37 3 37 35 20];
T(32,:) = [31 37 3 18 35 36];
T(33,:) = [26 11 25 2 1 29];
T(34,:) = [22 23 29 27 31 32];
T(35,:) = [36 37 40 20 9 11];
T(36,:) = [35 37 40 1 3 8];
T(37,:) = [36 35 40 22 29 30];
T(38,:) = [40 17 23 6 11 9];
T(39,:) = [38 22 29 20 26 25];
T(40,:) = [35 37 6 11 1 8];

T = [(1:40)' T];

clust_v(1) = 6;
clust_v(2) = 7;
clust_v(3) = 6;
clust_v(4) = 6;
clust_v(5) = 7;
clust_v(6) = 7;
clust_v(7) = 7;
clust_v(8) = 6;
clust_v(9) = 7;
clust_v(10) = 6;
clust_v(11) = 7;
clust_v(12) = 7;
clust_v(13) = 7;
clust_v(14) = 7;
clust_v(15) = 7;
clust_v(16) = 6;
clust_v(17) = 5;
clust_v(18) = 5;
clust_v(19) = 1;
clust_v(20) = 5;
clust_v(21) = 4;
clust_v(22) = 4;
clust_v(23) = 5;
clust_v(24) = 1;
clust_v(25) = 3;
clust_v(26) = 3;
clust_v(27) = 2;
clust_v(28) = 3;
clust_v(29) = 4;
clust_v(30) = 4;
clust_v(31) = 2;
clust_v(32) = 2;
clust_v(33) = 3;
clust_v(34) = 4;
clust_v(35) = 5;
clust_v(36) = 1;
clust_v(37) = 1;
```

```
clust_v(38) = 1;
clust_v(39) = 5;
clust_v(40) = 3;

clusts = {};
clusts{1} = find(clust_v==1);
clusts{2} = find(clust_v==2);
clusts{3} = find(clust_v==3);
clusts{4} = find(clust_v==4);
clusts{5} = find(clust_v==5);
clusts{6} = find(clust_v==6);
clusts{7} = find(clust_v==7);
end
```