4.

(1)

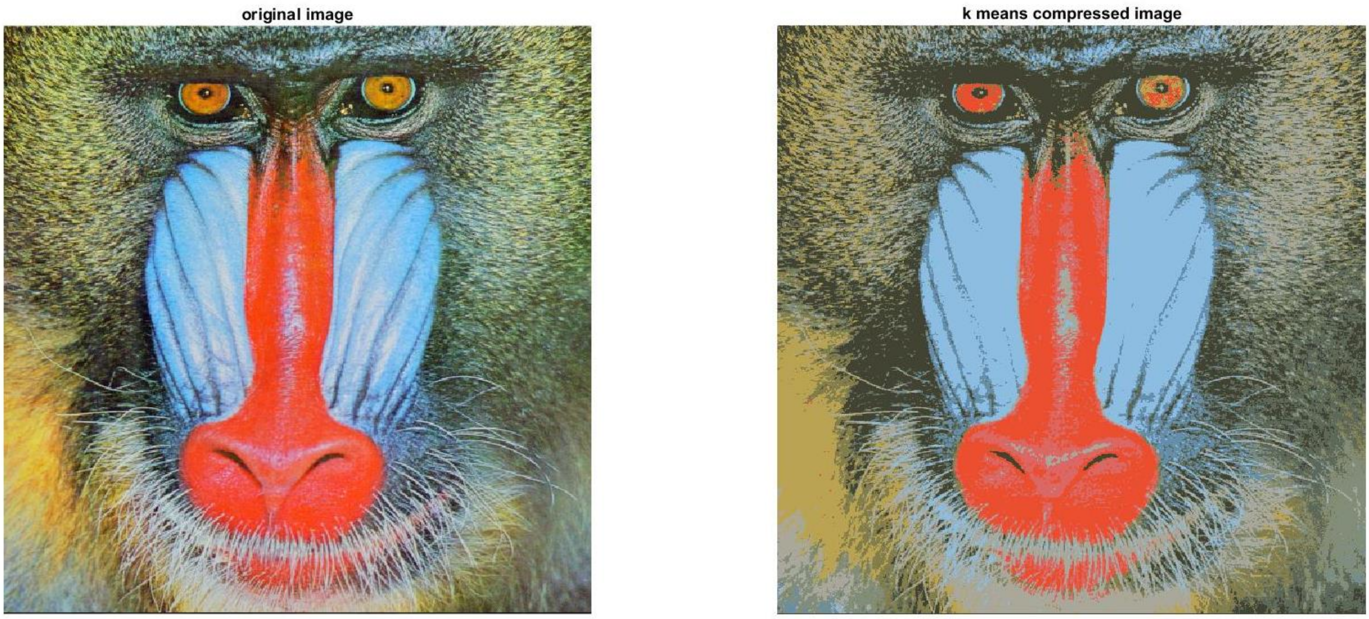
```
large = 'mandrill-large.tiff';
filename = large;
A = double(imread(filename));
imwrite(uint8(round(A)), 'p4_1.tiff');
```

(2)

```
k = 16; nIters = 40;
filename = small;
[m, kgroup] = myKmeans(filename, k, nIters);
function [ m, kgroup] = myKmeans(filename, k, nIters)
    %%
    color_scale = 256;
    im = double(imread(filename));
    [rows, cols, dim] = size(im);
    m = floor(color_scale * rand(dim, k));
    kgroup = zeros(rows, cols);
    for h = 1:nIters
        kgroup = findClosestCenterOf(kgroup, im, m);
        m = findMeans(kgroup, im, m, k);
        format long
        disp(['percentage: ', num2str(round(h / nIters * 100)) , '%']);
    end
    %%
    function m = findMeans(kgroup, im, m, k)
```

```
        for l = 1:k
            [row, col] = find(kgroup == l);
            if ~isempty(row)
                count = 0;
                for i = 1:length(row)
                    count = count + 1;
                    m(:, l) = m(:, l) + squeeze(im(row(i), col(i),:));
                end
                m(:, l) = m(:, l)/ count;
            end
        end
    end
end

function kgroup = findClosestCenterOf(kgroup, im, m)
    k = size(m, 2);
    for i = 1:size(im, 1)
        for j = 1:size(im, 2)
            tmp = zeros(1, k);
            for l = 1:k
                tmp(l) = norm(squeeze(im(i, j, :)) - m(:, l), 2);
            end
            [~,I] = min(tmp);
            kgroup(i, j) = I;
        end
    end
end
```

(3)



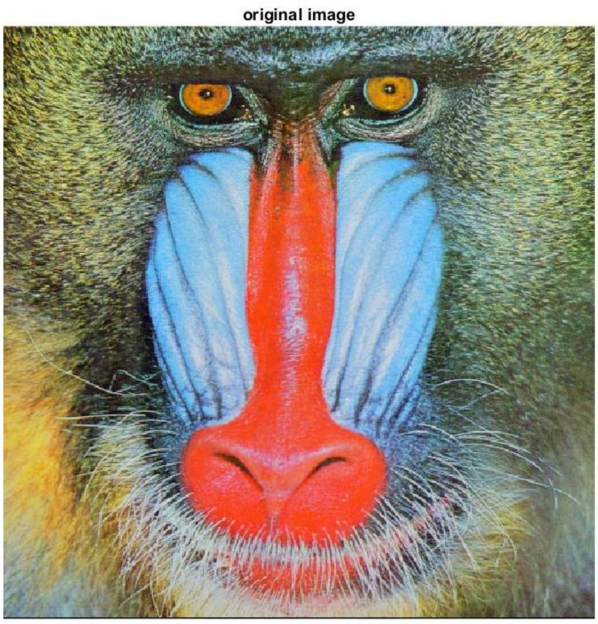

```
filename = large;
A = double(imread(filename));
image = compress(m, A);
figure,
subplot(1,2,1),
imshow(uint8(A));
title('original image');
subplot(1,2,2),
imshow(image);
title('k means compressed image');


function image = compress(m, im)
```

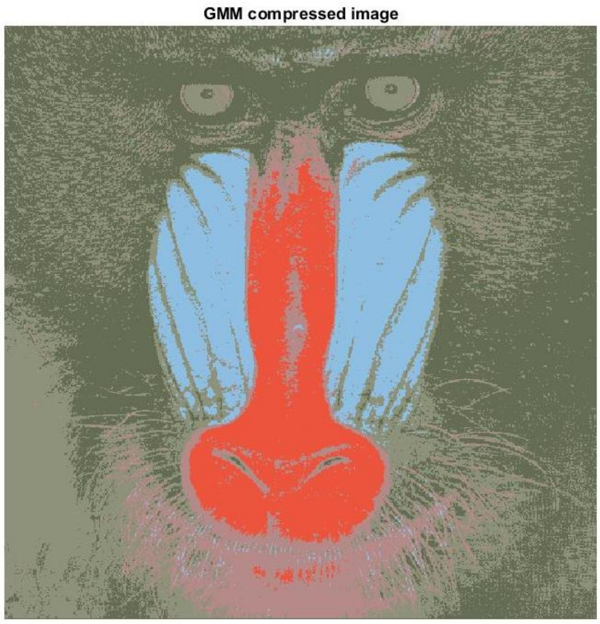
```
    [rows, cols, dim] = size(im);
    image = zeros(rows, cols, dim);
    kgroup = zeros(rows, cols);
    kgroup = findClosestCenterOf(kgroup, im, m);
    for i = 1:rows
        for j = 1:cols
            idx = kgroup(i, j);
            image(i, j, :) = reshape(m(:, idx), [1, 1, dim]);
        end
    end
    image = uint8(image);
end
```

(4)

Each pixel in the original image has the value from 0 to 256 on RGB. Therefore, it cost 24 bits per pixel. On the other hand, the kmeans only need to save the value of cluster from 0 to 16, which only costs 4 bits. As a result, if we assume that the means are the same for all the images and ignore them on calculating compressed factor. We use only $\frac{4}{24} = 16.6\%$ of the original image after compression.

(4)





We use 3 bits to store the MAP estimate for each pixel. In GMM, We use $\frac{3}{24} = 12.5\%$ of the original image after compression.

	μ		
K	R	G	B
1	181.577695440685	137.930878010427	135.850026990273
2	141.791300841145	146.007393447149	122.277218588609
3	237.362424634810	84.1546764501801	61.9142384005461
4	141.583117117354	190.533438387129	226.278000572344
5	100.809940463799	110.026302800986	84.8502275241879

Σ_1		
110.94	948.9	32.6
948.9	1156.5	403.1
32.5	403.2	594.4

Σ_2

815.91	644.31	251.73
644.31	658.33	446.36
251.73	446.36	891.47

Σ_3		
195.65	-154.88	321.65
-154.88	334.52	584.75
321.65	584.75	1245.6

Σ_4		
412.39	456.29	278.17
456.29	604.71	417.85
278.17	417.85	391.82

Σ_5		
920.57	13.93	-463.68
13.93	801.55	1047.13
-463.68	1047.13	1693.85

```
k = 5;
nIters = 100;
filename = small;
[m, sigma, prior] = gmm(filename, k, nIters);
filename = large;
A = double(imread(filename));
[rows, cols, dim] = size(A);

W = expectation(reshape(A, [rows * cols, dim]), m, k, sigma, prior);

image = zeros(rows, cols, dim);
for i = 1:rows
    for j = 1:cols
        idx = (i - 1) * cols + j;
        [~, I] = max(W(idx, :));
        image(j, i, :) = reshape(m(I, :), [1, 1, dim]);
    end
end
image = uint8(image);
figure,
subplot(1,2,1),
imshow(uint8(A));
title('original image');
subplot(1,2,2),
imshow(image);
title('GMM compressed image');

function [m, sigma, prior] = gmm( filename, k, nIters )
%GMM Summary of this function goes here
%   Detailed explanation goes here

    im = double(imread(filename));
    [rows, cols, dim] = size(im);
    X = reshape(im, [rows * cols, dim]);

    %% initialization
    indeces = randperm(rows * cols);
    m = X(indeces(1:k), :);
    sigma = zeros(dim, dim, k);
    % Use the overal covariance of the dataset as the initial variance for each
cluster.
    for i = 1:k
        sigma(:, :, i) = cov(X);
```

```
    end
    % Assign equal prior probabilities to each cluster.
    prior = ones(1, k) * (1 / k);
    %% EM
    for h = 1:nIters
        prevM = m;
        W = expectation( X, m, k, sigma, prior);
        [prior, m, sigma ] = maximization(m, W, X, k, prior, sigma);
        if m == prevM
            break
        end
    end
end
function W = expectation( X, m, k, sigma, prior)
    [row, n] = size(X);
    pdf = zeros(row, k);
    for i = 1 : k
        Sigma = sigma(:, :, i);
        meanDiff = bsxfun(@minus, X, m(i, :));
        pdf(:, i) = 1 / sqrt((2*pi)^n * det(Sigma)) * exp(-1/2 * sum((meanDiff /
Sigma .* meanDiff), 2));
    end
    pdf_w = bsxfun(@times, pdf, prior);
    W = bsxfun(@rdivide, pdf_w, sum(pdf_w, 2));
end

function [prior, m, sigma ] = maximization(m, W, X, k, prior, sigma)
    [row, n] = size(X);
    for i = 1 : k
        prior(i) = mean(W(:, i), 1);

        % Divide by the sum of the weights.
        m(i, :) = (W(:, i)' * X) ./ sum(W(:, i), 1);
        sigma_k = zeros(n, n);
        meanDiff = bsxfun(@minus, X, m(i, :));
        for j = 1 : row
            sigma_k = sigma_k + (W(j, i) .* (meanDiff(j, :)' * meanDiff(j, :)));
        end
        sigma(:, :, i) = sigma_k ./ sum(W(:, i));
    end
end
```