CV Assignment#2 Report

Image Segmentation by Gaussian Mixture Model

Setup

Components

There are **3 folders and 3 files** in this project directory. The detail information is as follows.

- **GMM_Examples:** This folder has all the example codes. There are **5 ".m"** files in this folder.
  - **gaussian1D.m:** calculate Gaussian for 1-dimensional data
  - **gaussianND.m:** calculate multivariate Gaussian for N-dimensional data
  - **GMMExample_1D.m:** Gaussian Mixture Model for 1-dimensional data
  - **GMMExample_2D.m:** Gaussian Mixture Model for N-dimensional data
  - **weightedAverage.m:** calculate the weighted average the data and the current corresponding μ
- **result:** This folder has all the results for problem 2. There are 2 folders in this folder.
  - **shiba1:** There are **6** images after segmentation of **"Shiba_1.jpg"** in this folder. Which are 1D-K_3.png, 1D-K_5.png, 1D-K_10.png, 3D-K_3.png, 3D-K_5.png, 3D-K_10.png
  - **shiba2:** There are 6 images after segmentation of **"Shiba2.jpg"** in this folder. Which are 1D-K_3.png, 1D-K_5.png, 1D-K_10.png, 3D-K_3.png, 3D-K_5.png, 3D-K_10.png
- **src:** There are **2 images** in this folder which are **Shiba_1.jpg** and **Shiba2.jpg**. They are the source images for problem 2.
- **hw2_2_live.mlx:** The live script for problem 2.
- **hw2_2.m:** The source code for problem 2.
- **CV-hw2_report.pdf:** The report for this assignment.

Prerequisite

1. MATLAB: recommended version R2019b

Execution

1. Open **MATLAB**
2. Select **"406410035_hw2_v1"** this folder to open in your MATLAB
3. Open **"GMM_Examples/GMMExample_ND.m"**, **"GMM_Examples/gaussianND.m"** and **"GMM_Examples/weightedAverage.m"** to see the detail comment
4. Open **hw2_2_live.mlx** or **hw2_2.m**
5. Run the program
6. Go to **"406410035_hw2_v1/result"** to see all the results

Method Description

I split the assignment into **3 parts**. **Part1** is for **problem 1** of this assignment. **Part2 and Part3** are for **problem 2** of this assignment. The detail is as follow:

Part1: Trace Code

From the **GMMExample_2D.m** I can generalize GMM algorithm to some simple steps.

1. Generate data from 2D Gaussian distribution (if you have mean and variance you have the Gaussian distribution)
2. Plot the data points and their pdfs (probability density function)
3. Choose initial values for the parameters (**K-means clustering**)
4. Run expectation maximization (**EM**: find blob parameters θ that maximize the likelihood function)
    a. Loop until convergence
    b. **E step (Expectation)**: evaluate the Gaussian for all data points for cluster j (given a guess of blobs, compute ownership of each point)
    c. **M step (Maximization)**: calculate the probability for each distribution (given ownership probabilities, update blobs to maximize likelihood function)
5. Plot the data points and their estimated PDFs (probability density function)

Part2: Use 1D to do GMM

Before doing anything first import functions from **GMM_Examples**. We will use them in our program later.

1. Read in and process the image
    a. Read in the image
    b. Turn the image to grayscale by **rgb2gray()**
    c. Get the size of the original image (we need it for reshaping in the last step)
    d. Turn the value of every pixel to double data type in the convenience of doing the calculation in the future by **im2double()**
    e. Stretch the image into a vector (1D)
    f. Get the length of the vector as the number of data points by **length()**
2. Pass into GMM model-> EM step (1D)
   This step is basically same as **Step4** in the sample code **GMMExample_1D.m**
    a. Randomly select k data points as the initial cluster centers by **randperm()**
    b. Use the overall **variance** of the dataset as the initial variance for each cluster
    c. Initialize equal probabilities to each cluster
    d. E step:
        i. Evaluate the Gaussian for all data points for cluster j
        ii. Multiply each pdf value by the prior probability for cluster
        iii. Divide the weighted probabilities by the sum of weighted probabilities for each cluster
    e. M step:
        i. Store the previous mean
        ii. Calculate the probability for each cluster

iii. Calculate the new mean
iv. Calculate the new variance
v. Check for convergence (if cluster centers remain the same)

3. Output the image after segmentation
    a. Iterate all the pixels
    b. Assign the cluster to the pixel according to the probability of that pixel belong to
    c. Give that pixel the color of its cluster center
    d. Reshape the segmentation image back to its original size by **reshape()**
    e. Transform the pixel value to grayscale by **mat2gray()**
    f. Save the segmentation image by **imwrite()**

Part3: Output the image after segmentation

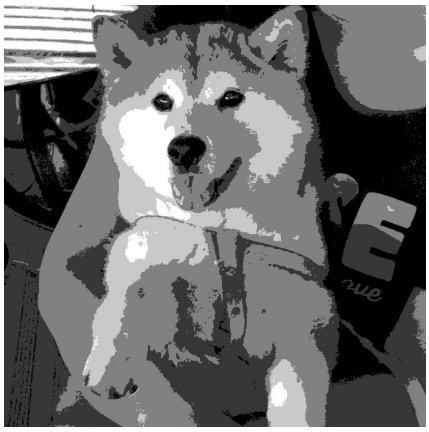
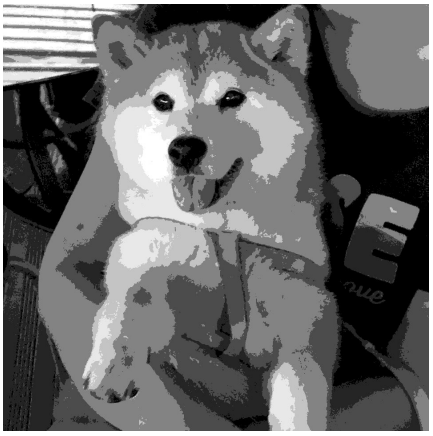
1. Process the image to fit in GMM functions
    a. Get the row, col and channels of the original image using **size()**
    b. Change the original pixels' datatype to double by **im2double()**
    c. Reshape the image into a channel a vector so there will be 3 vectors (columns) and each vector size is $row \times col$ (rows)
    d. Get the length the reshape image as the number of data points by **length()**
2. Pass into GMM model-> EM step (3D)
    a. The parameters initialization is basically the same as **Part2** but the initial variance for each cluster is the overall **covariance** of the dataset using **cov()**
    b. E step is the same as **Part2**
    c. M step is mostly the same as **Part2**. The only difference is in the check for convergence. For saving time I didn't check if the cluster centers are absolutely the same instead I check if the **Squared Euclidean distance** of the current mean and the previous mean is **less then 0.000005**
3. Output the image after segmentation: This step is basically the same as **Step3 in Part2**. The only difference is that we have **3 channels** to cope with rather than 1 channel

Results

All the resulting images after segmentation are in **"406410035_hw2_v1/result/"** folder

1D Gaussian Mixture Model

1. **Shiba 1:** **3 grayscale images in "shiba1/" folder** corresponding to **k=3,5,10**. They are also shown in Figure (1.1), Figure (1.2) and Figure (1.3) below.

Figure(1.1) shiba1/1D-K_3.png *Figure (1.2) shiba1/1D-K_5.png* *Figure (1.3) shiba1/1D-K_10.png*

2. **<u>Shiba2:</u> 3 grayscale images in "shiba2/"** folder corresponding to **k=3,5,10**. They are also shown in Figure (2.1), Figure (2.2) and Figure (2.3) below.

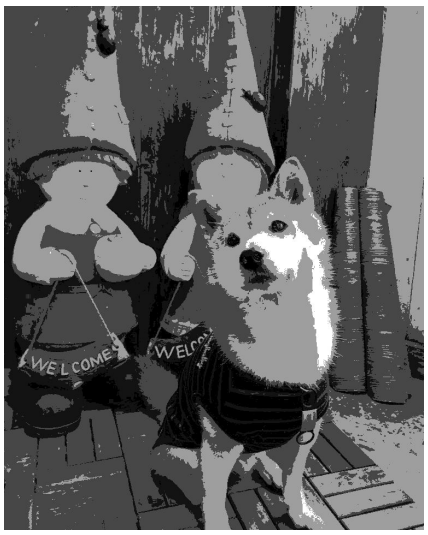




Figure (2.1) shiba2/1D-K_3.png *Figure (2.2) shiba2/1D-K_5.png* *Figure (2.3) shiba2/1D-K_10.png*

<u>3D Gaussian Mixture Model</u>

1. **<u>Shiba1:</u> 3 color images in "shiba1/"** folder corresponding to **k=3,5,10**. They are also shown in Figure (3.1), Figure (3.2) and Figure (3.3) below.

Figure(3.1) shiba1/3D-K_3.png

Figure (3.2) shiba1/3D-K_5.png

Figure (3.3) shiba1/3D-K_10.png

2. **<u>Shiba2:</u> 3 color images in "shiba2/"** folder corresponding to **k=3,5,10**. They are also shown in Figure (4.1), Figure (4.2) and Figure (4.3) below.

Figure (4.1) shiba2/3D-K_3.png

Figure (4.2) shiba2/3D-K_5.png

Figure (4.3) shiba2/3D-K_10.png

Discussion

From the results above I found that if k is larger than the image after segmentation looks better, more accurately to say is that the image looks more detailed. I think if you want to get better segmentation result the key will be k. If k is larger you can get the better result a more detailed image. But the tradeoff will be computational time because if k is larger that means you have more clusters, and each pixel will calculate the probability it is in the specific cluster numerous time, from here you can tell that the computational time is longer. If k=3 takes 3 minutes then k=10 may take more then 10 minutes to compute. Also when k is larger it will take longer to converge, this is what I found in my case when k=10 it takes more iterations in the EM step. As for the effect of the image in my point of view, I like when k=3 and k=10. I like when k=3 is because although there are only 3 clusters the resulting images look quite smooth, not weird at all, this surprised me. And when k=10 I think the image is detailed enough to represent the original image.

Difficulties and Problems

The first problem I encountered is that when tracing code. At the beginning of the example code when I see X1 and X2 I can't relate to the data that will be passed into the GMM. And the generation of data isn't that straightforward so I spend quite a long time to realize that is the data points. After that, the tracing code process becomes smoother. The second problem is after the EM step. When segmenting image I don't know how should I present the image? What color should I use? Random select color to each cluster or other methods. At last, I choose to use the cluster centre's color to represent all the pixels in the same cluster. And I think the resulting image looks quite great.