Question 2

a) Construct a model to classify images as Class A or B, and train it on the indicated data. You must code your own model from scratch. Specify your trained model. What does your model predict for each of the unlabeled images? Give the details of your model, its training, and the final result. Do the predictions make sense, to you?

In this question, I programmed in python3.7. First, in order to represent these pictures, I choose to use numpy and represent them in matrix. I store the two classes in files and load them into memory when I need them.

The model I use is K-nearest neighbor(KNN). And this model works like this: 1.Load the pictures with its label in the model. 2.For each picture in the Mystery class, calculate the average Euclidean distance between pictures in class A and class B. The number of each class picture is n, the height and the width of each picture is m. y represents the pixel in the labeled class and x represents the pixel in Mystery class. Thus, KNN do not a training process, this is also one of its advantage when dealing with small dataset. I slightly changed the process in the evaluation part, unlike the traditional KNN will hold a list of the distance between the test pictures and the training pictures. After sorting the list, it will compare the number of class A and class B in the smallest K elements of the distance list. If the number of class A in the K elements (the distance between class A with the test picture) is more than the number of class B, the model will label the picture with class A, otherwise, the model will assign the picture with class B.

Besides my own idea, I implement the traditional KNN as well to compare them.

The reason why I change the evaluation method is that:

1. The result can be more intuitive. Unlike the traditional way, I can show the average distance between the test data with the training data, which is related to the similarity between them.
2. Reducing the complexity. In order to store the list of the distances, the program will spend extra space to store the distance information. After that, it needs at least $O_{(nlog_n)}$ time complexity to sorting the list. Thus, the KNN model can be very time-consuming when the training data explodes.
3. Similar result with the traditional KNN. I compared the result with the traditional KNN. In this current given dataset, the prediction result is the same when K is less or equal than 3.
   When K is larger or equal than 5, the result will be(A, A, B, A, A), 40% of the results has changed. In this situation, I think the value of K maybe too large that it leads to some error in the prediction. Based on the reasons I talked about above, I

think in this situation, the average-distance evaluation is intuitive and reasonable.

$$average\ distance = \frac{\sum \sqrt{\sum_{i=0}^{m} \sum_{j=0}^{m} (y_{ij} - x_{ij})^2}}{n}$$

According to the average distance we just calculated, we can then get the average distance of picture.

Unlabeled picture	Class A	Class B	Result
1	2.9940613262802698	2.8515037428032297	B
2	2.778584445398434	3.5446725284604623	A
3	3.604482565660733	2.63194723121875	B
4	2.7661219270769117	3.3152507292744664	A
5	3.0909612800104647	3.0201115422233826	B

After got all those average distances, we can then label these pictures with class A and class B. In this label process, we can define different policies to affect the final result of the picture label. E.g. straightly comparing the distance: if the distance between A is smaller than the distance between B, we label this picture with A (The result column is based on this strategy). Otherwise, we assign it with B. Another method is to compare the distance with a threshold(predefine). If the distance between A and B is no larger than0.01, we can assign this picture with Same, which means it belongs to both A and B.

In the traditional KNN, the result is below:

K	result
1	B, A, B, A, B
3	B, A, B, A, B
5	A, A, B, A, A

As the table shows above, I can infer that when K is less than 5, the performance is similar between my own model and the traditional KNN. However, the traditional KNN contains more flexibility that you can change the K to avoid overfitting problem. Moreover, the robustness of the system can be insure.

The result of the prediction seems to be reasonable, but I still don't understand the inner relation between the labeled data and the test pictures. That's one of the disadvantages of KNN algorithm, it compares the distance between the input data with training data, but it won't analysis the inner feature between each other. As the consequence, the classification of the KNN can sometimes be unrational to human.

b) The data provided is quite small, and overfitting is a serious risk. What steps can you take to avoid it?

In traditional KNN model, we can choose different K value to avoid overfitting problem. In our own model, we have a different method to classify. One method to relief

overfitting is to predefine a threshold as hyperparameter, if the distance between A and B is smaller than this threshold, we can assign both A and B label to this picture. Another way to deal with this overfitting problem is to add some noisy data into the training set, that is to say, we can flip some points of the training data and add it to the training set. Thus, the unlabeled data will be compared with some data which is not in the original training data. To some extent, it will relief the overfitting and enhance the robustness of the classifier.

c) Construct and train a second type of model (again from scratch). Specify its details. How do its predictions compare to the first model? Are there any differences, and what about the two models caused the differences?

In the second model, I choose the Naïve Bayesian classifier. The representation method is the same with KNN, we use matrix to represent given pictures. After loading all the training data in the memory, we will compute the possibility of each pixel in the picture to be black, x represents the pixel in picture, (i, j) represent the row and column of the pixel:

$$P(x_{ij}) = \frac{\sum x_{ij}}{n}$$

We sum the pixel in position (i, j) of all the training data and divide by the number of pictures to get the possibility of x_{ij}. After assigning each pixel a possibility (training process), we will have a possibility picture. Then, we can input the Mystery data into the model. In this process, the algorithm will extract the black points' location of the input picture and comparing it with the possibility picture and times all the possibility together to get the possibility of this picture to be class A or class B.

Unlabeled picture	Class A	Class B	Result
1	0.0002286236854138088	0.0008573388203017832	B
2	0.00020322105370116337	4.7629934461210166e-06	A
3	2.3814967230605083e-06	0.0025720164609053494	B
4	0.0004572473708276176	4.762993446121017e-05	A
5	0.0005144032921810699	0.0020576131687242796	B

In the result set above, I did some modify to the possibility calculation formula

$$P(x_{ij}) = \frac{\sum x_{ij} + 1}{n + 1}$$

In the new possibility, I Laplace Smoothing into the calculation because when the classifier is deal with the sample which did not appeared in the training set, the possibility of the sample will be 0 and kill all the result, and we can't get the proper possibility.

You can see the last column of the two result-set, the KNN and the Naïve Bayesian classifier outputs the same result.

d) Why would a huge neural network be a bad idea here?

A huge neural network needs to be trained many epoch to achieve the converge. In the current situation, we only have 5 pictures per class, which means if we use a huge neural network, we need to go through these 10 pictures many times, and this process will lead to a severe overfitting problem. As b) described, overfitting is a serious risk, so we would like to choose other model to solve this problem.