Computer Vision Assignment 8 - Image Classification

Kai-Po Lin (kl72); Department of Computer Science

1. Bag of Features Classification with SIFT Descriptors

1.1 Algorithm Implementation

(1) Use SIFT to find features (and their descriptors) in all of the training set images. Do this carefully (see Matlab's struct datatype), so that you can easily identify which features belonged to which training class.

⇨ I defined a file called "SIFT.m" to find all features and their descriptors within all training images. The inputs are the training folder path and subfolder names (Note: since the training data starts with category "butterfly" and the test data category starts with category "hat", I switched the subfolder name "butterfly" and the "hat" of the training folder). After **changing all RGB training images to grayscale images with type single**, we then use **vl_sift() to fetch each images' features and feature descriptors** and stack them together as a variable.
I also have a variable called "class_descriptors_cnt". This variable will record the descriptors' range that belongs to each training class.

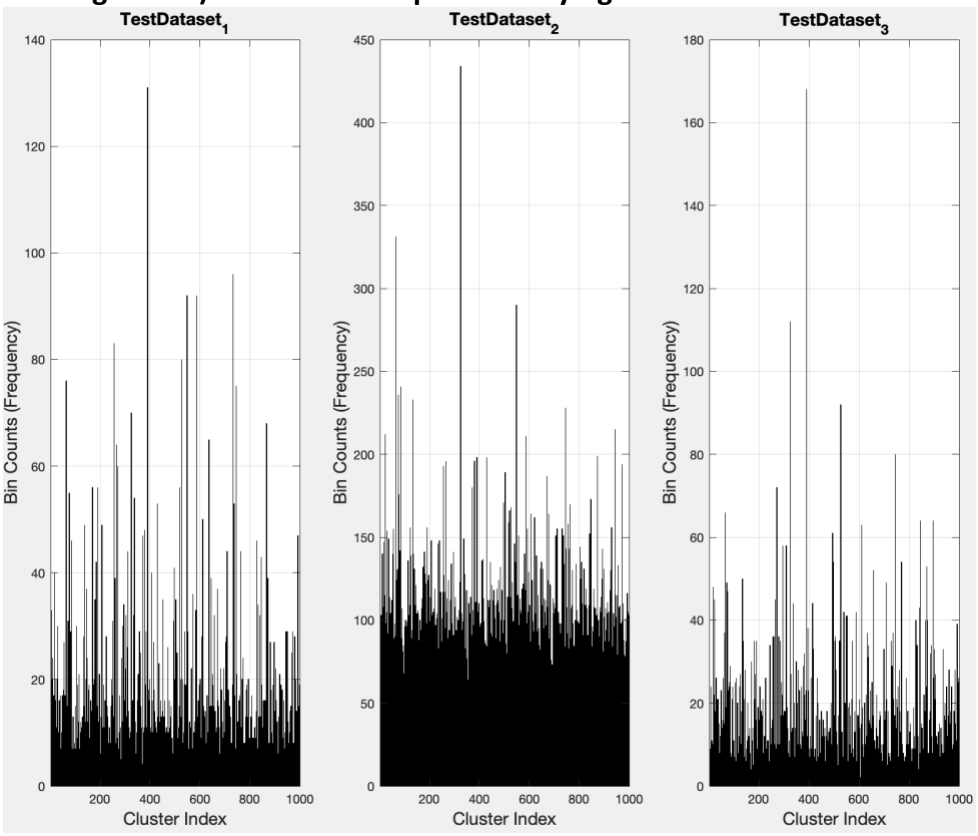
(2) Cluster all the SIFT feature descriptors you found using a distance metric of your choice (be sure to include your chosen metric in your report along with your rationale). We suggest using the k-means clustering algorithm with N=1000.

⇨ I used k-means (with N=1000) to cluster all feature descriptors from SIFT. The distance metric I chose is the **L1 distance (the "cityblock" distance metric)**. The reason why I chose this instead of all other distance metrics like cosine or Squared Euclidean distance is simply that it performs better (about 83.66% accuracy) after some experiments (L1 distance: about 78.33%, cosine: about 65%, correlation: about 59.58%). I consider the reason is that **L1 distance will reduce the number of coefficients to reduce loss, while squared of L2 will only try to minimize the value of the coefficients instead of eliminating them**. Since we have 1000 clusters, it is more important to learn which performs better and eliminate the useless ones. **The only drawback of L1 distance is that it requires more operating time than all other distance metrics** since calculating absolute value is more computationally expensive.

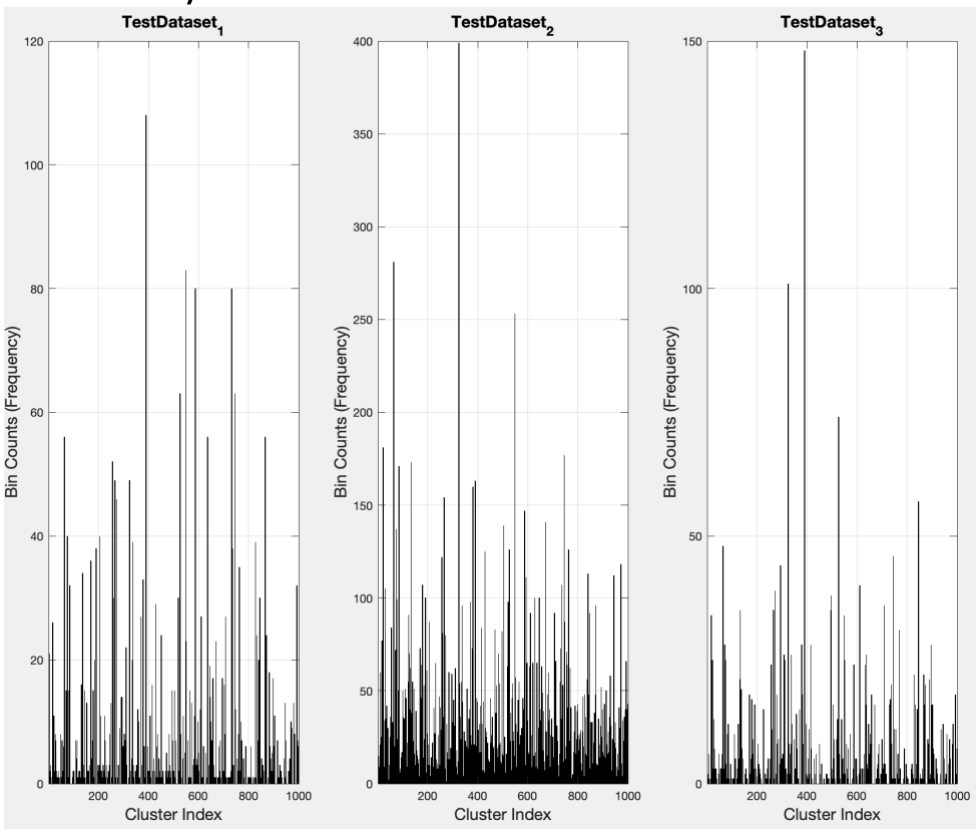
(3) (a) For each training class, form a histogram of N bins, where each bin corresponds to a cluster found above. Find how many SIFT features from that class's training images belong to each of the SIFT descriptor clusters. Keep in mind that there will be features present which does not help in identifying classes (such as useless junk in the image's background for instance). These "bad" features should be removed with a cluster distance threshold.

⇨ My threshold algorithm that helps remove junk feature descriptors that do not help identify classes => My threshold algorithm is **to pick the median value (I've also tried using the max value or the mean value) of the distance from the identified cluster and multiplied it by the threshold. Any distance value that is bigger than the calculated result above will be treated as a useless feature and discarded.**

1. Histogram "before removing features" (threshold=-1 => which means don't utilize my threshold algorithm) that do not help in identifying classes.

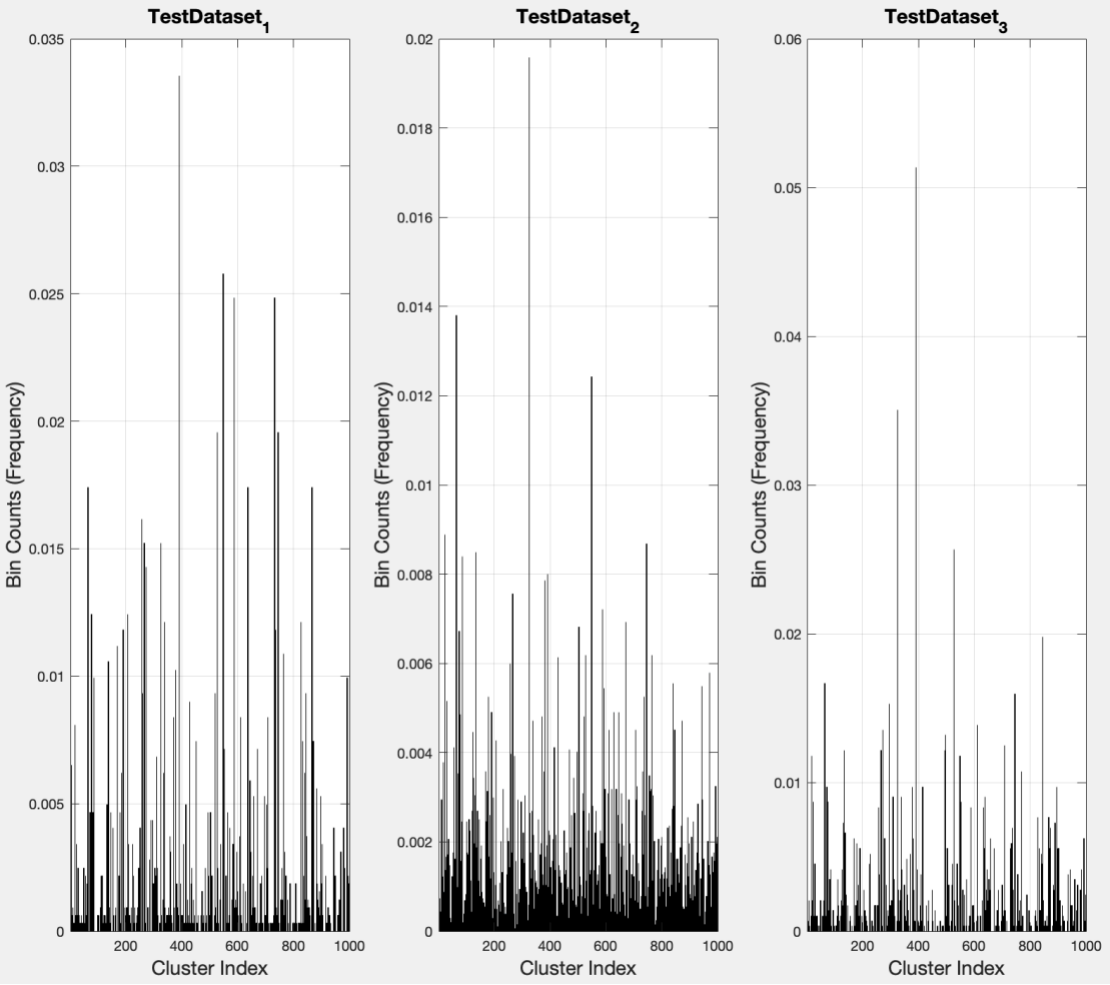


2. Histogram after "removing features" that do not help in identifying classes (max strategy with threshold=0.7).



(b) Normalize the bin counts by dividing by the total number of SIFT features binned for that particular class. This normalized histogram now forms a descriptor for that particular training class.

⇨ **Normalize the bin counts (max strategy with threshold=0.7).**



(4) Find the SIFT feature descriptors within each test image.

⇨ The procedure for this question is exactly the same as question (1) except we now read the directories of the test images instead. We will use the variable “descriptors” and “class_image_cnt” (this is a list recording the accumulated number of images for each class) from the return values of the self-defined function SIFT().

(5) Assign these features to clusters as you did when creating the class descriptors (removing “bad” features with a cluster distance threshold as before).

(6) Use these assignments to calculate a normalized cluster histogram for each test image.

⇨ I have a self-defined function test_knn() for these questions. The operations are similar to questions (2) and (3), except now we don’t have to do the k-means again. Instead, I use knnsearch with the default 'Euclidean' distance metric to calculate the distance of each image with the centers we obtain for the training process (k-means).

(7) Assign each test image to one of the possible classes by comparing its cluster histogram to the cluster histograms of the various classes you trained with previously. One conceptual way to do this comparison is to think of each normalized cluster histogram as a vector. Finding the class is akin to

finding the class histogram vector closest (with respect to your chosen distance metric) to the one associated with the image being tested. This can be implemented using MATLAB's knnsearch.

⇨ I use knnsearch() (using the default 'Euclidean' distance metric) again in test_knn() to find out which class has the most similar histogram to the current image's histogram.

1.2 Technical Write-up: Results and Discussion

• Clearly and cogently document your methods and results. From your PDF report, it should be clear what you did, how/why you did it, and how well it worked, without needing to run code or sift through 300 figures.

⇨ Most of my methods are stated in question 1.1 Algorithm Implementation (including why choosing L1 distance, and how my threshold algorithm was implemented) by following the general procedure. In this question, I would state other special implementations within the code.

1. **The threshold value setting**

⇨ I used a for-loop with a 0.01 increment from 0.3 to 1.2 (like the following figure) to find out which threshold setting for the training and testing set obtains the best accuracy. After doing lots of experiments, I found out that setting a threshold between 0.9~1 will provide the best results regarding my threshold algorithm (my threshold algorithm method are mean, median, and max).

```
Current Threshold: 0.94, Current Sum: 231.25, Best Threshold: 0.91
Current Threshold: 0.95, Current Sum: 237.50, Best Threshold: 0.91
Current Threshold: 0.96, Current Sum: 237.50, Best Threshold: 0.91
Current Threshold: 0.97, Current Sum: 247.50, Best Threshold: 0.97
Current Threshold: 0.98, Current Sum: 247.50, Best Threshold: 0.97
Current Threshold: 0.99, Current Sum: 241.25, Best Threshold: 0.97
fx Best Sum: 247.50, Best Threshold: 0.97
```

2. **Beautify the confusion matrix and result**

⇨ I have a self-defined function called display_confusion_matrix.m to output the confusion matrix like the following figure. The x-axis is the prediction of each class, and the y-axis is the actual classes.

```
Best Sum: 283.75, Best Threshold: 0.99
---------- Result ----------
| Classes |    Hat    | Butterfly |  Airplane  |
----------------------------------------------------------
|    Hat    |  90.00%  |  0.00%  |  10.00%  |
----------------------------------------------------------
| Butterfly |  0.00%   |  100.00%  |  0.00%  |
----------------------------------------------------------
| Airplane |  6.25%   |  0.00%  |  93.75%  |
----------------------------------------------------------
```

• Indicate points of possible improvement and provide conceptual solutions to the extent you are able.

⇨ I think I could attempt more threshold strategies to potentially improve the results. Moreover, I consider if there are more training images included, the result will probably be better since there are only about 10 or 16 images within the training dataset of each class.

• Include and interpret a (3 x 3) confusion matrix (for your results on the reduced dataset). See below for an example:

⇨ I've tested three different kinds of threshold algorithms: max, mean, and median by running the whole program five times for each method and recording the best one. Eventually, I found out that the median method performs the best with 94.58% accuracy.

Best Result with the Max Threshold Method: 85.83% (x-axis: Predicted Class; y-axis: Actual Class)

```
Best Sum: 257.50, Best Threshold: 0.90
---------- Result ----------
| Classes |    Hat    | Butterfly |  Airplane  |
----------------------------------------------------------
|    Hat    |  80.00%  |  10.00%  |  10.00%  |
----------------------------------------------------------
| Butterfly |  10.00%  |  90.00%  |  0.00%  |
----------------------------------------------------------
| Airplane |  12.50%  |  0.00%  |  87.50%  |
----------------------------------------------------------
```

Best Result with the Mean Threshold Method: 83.75% (x-axis: Predicted Class; y-axis: Actual Class)

```
Best Sum: 251.25, Best Threshold: 0.99
---------- Result ----------
| Classes |    Hat    | Butterfly |  Airplane  |
----------------------------------------------------
|    Hat    |  70.00%  |  20.00%  |  10.00%  |
----------------------------------------------------
| Butterfly |  0.00%   |  100.00%  |  0.00%  |
----------------------------------------------------
| Airplane |  18.75%   |  0.00%  |  81.25%  |
----------------------------------------------------
```

Best Result with the Median Threshold Method: 94.58% (x-axis: Predicted Class; y-axis: Actual Class)

```
Best Sum: 283.75, Best Threshold: 0.99
---------- Result ----------
| Classes |    Hat    | Butterfly |  Airplane  |
----------------------------------------------------
|    Hat    |  90.00%  |  0.00%  |  10.00%  |
----------------------------------------------------
| Butterfly |  0.00%   |  100.00%  |  0.00%  |
----------------------------------------------------
| Airplane |  6.25%   |  0.00%  |  93.75%  |
----------------------------------------------------
```

1.3 Competition

		Predicted class		
	Classes	Hat	Butterfly	Airplane
Actual class	Hat	55%	31%	14%
	Butterfly	20%	70%	10%
	Airplane	12%	5%	83%

The average value of the diagonal elements in the confusion matrix above is the accuracy of the classifier. For the above case, the accuracy is 69.33%. Report the accuracy of your classifier. This part of the grade is based on your classifier's performance compared to the classifiers trained by your peers in the class.
Note: Failure to report this number will automatically award you zero points and reporting wrong numbers is against the honor code.

⇨ Method: mean strategy with threshold = 0.99

⇨ Accuracy: 283.75 / 3 = 94.58%

```
Best Sum: 283.75, Best Threshold: 0.99
------------------------- Result -------------------------
| Classes |    Hat    | Butterfly |  Airplane  |
----------------------------------------------------------
|    Hat    | 90.00%  |  0.00%  |  10.00%  |
----------------------------------------------------------
| Butterfly |  0.00%  | 100.00% |  0.00%   |
----------------------------------------------------------
| Airplane |  6.25%   |  0.00%  |  93.75%  |
----------------------------------------------------------
```

(x-axis: Predicted Class; y-axis: Actual Class)

2. Conclusion

(1) **Issue**

The biggest issue I encountered was to ensure the indexes are correct when reading the training and the testing data. I've tried to find out the error for two days by tracing all of my code comprehensively.

(2) **Surprise**

I think this assignment helps me truly implement and clarify the concepts of clustering. Moreover, I learned a lot of MATLAB syntax when I try to load the data. Although I was struggle finding the bug within my code, I eventually gain sense of achievement with a very decent result.