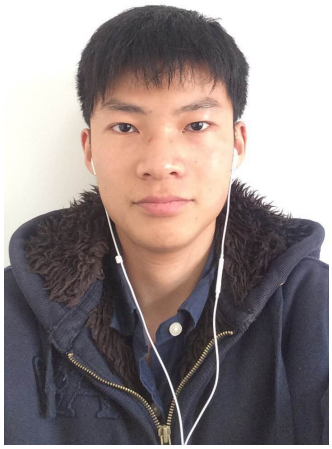
Assignment 6: Image Classification

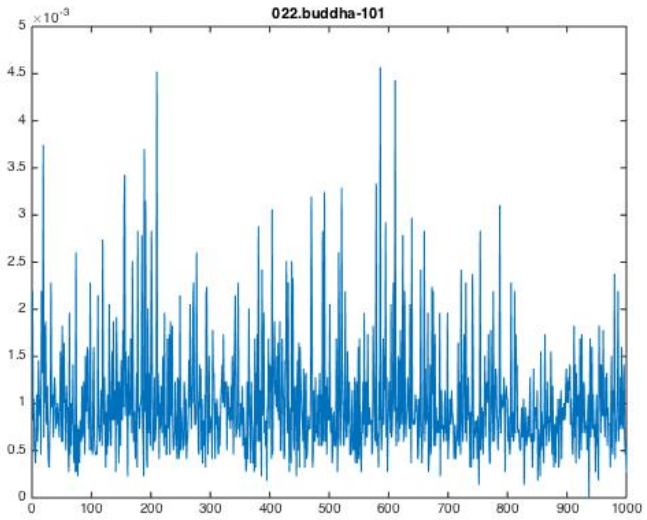
Name: Suozhi Qi
NetId: sq6

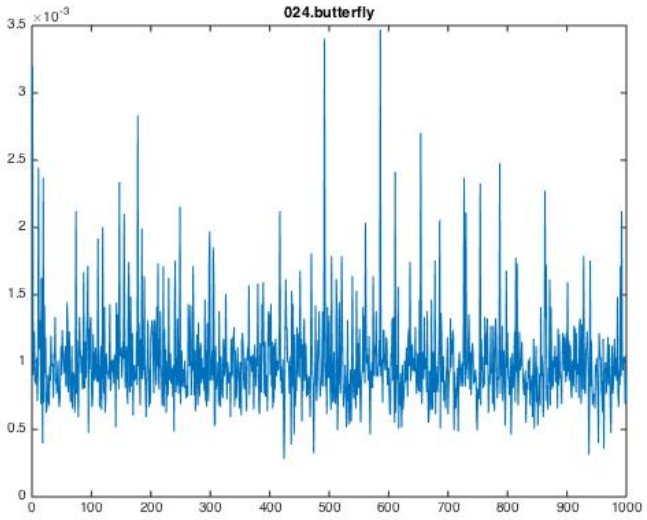


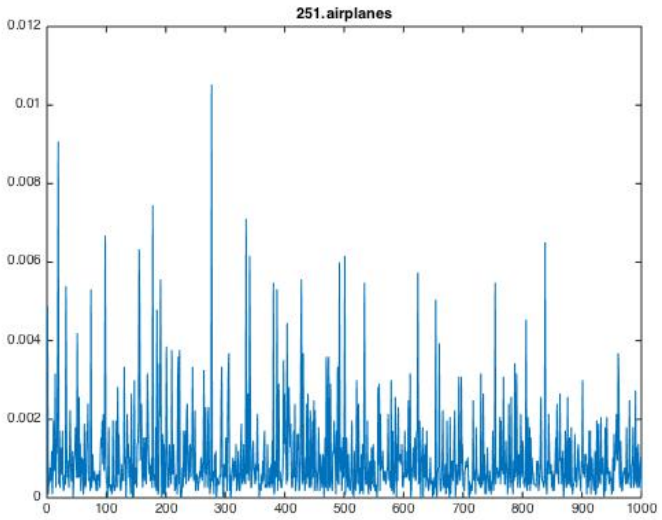
1 Bag of Features Classification

1.1 Algorithm implementation

1. I use vl_sift function to get all the SIFT features and store them in struct trainingRes.
2. I use vl_kmeans for clustering. I use distance of 'cityblock' during the clustering process. The clustering process took about 20mins. Very long. This step may be optimized by reducing the number of SIFT feature points obtained from step 1. In step 1, we use 128 SIFT descriptor. According to our instructions in class, this number may be reduced to about 80 descriptors.
3. Please refer to my code for generation of histograms. I use knnsearch to search for the nearest neighbors in the SIFT features of each individual classes in the training set. I set the threshold to 1.5 times the median of distance. Show below are the training histograms of the three classes in reduced data set. You can see from the histograms that everything works well here. The three histograms show different patterns of peaks and valleys.







4. Please refer to my code for getting the SIFT features, which is very similar to getting the SIFT features from the training data sets.

5. Please refer to my code for clustering. I use vl_kmeans for clustering, and this is similar to the clustering process above.
6. Please refer to my code for getting the histogram for each image. Basically I just use knnsearch to find which cluster each SIFT feature point in the test belongs to. Then use the same procedure as above to generate the histogram.
7. Please refer to my code for this part. I use knnsearch to find the predicted category by locating the points for each histogram generated from each image to the three histogram images got from training. These results will be included in the variable reducedRes. Then I just get the percent of each prediction by dividing each value in the reducedRes by the total sum of predictions. Below is the result.

Actual \ Predicted	Buddha	Butterfly	Airplane
Buddha	0.71428571	0.21428571	0.07142857
Butterfly	0.2	0.8	0
Airplane	0.125	0	0.875

The diagonal three spaces are the input data for the accuracy of prediction. Basically over 70% is correctly predicted as class1, 80% is correctly predicted as class2, and near 90% is correctly predicted as class3. These results showed that this method works well.

1.2 Technical write up

Possible improvement:

This algorithm runs very slow, about 30mins. Most of the time was taken to do clustering. As I mentioned above, we can reduce the number of SIFT descriptors to about 80, which will make the whole process faster. To improve the accuracy of the prediction, first we can further modify the parameter of threshold. Here I am using 1.5 * median(D). We may try other numbers like 1.6, 1.7, 1.4, etc. Since the whole process lasts for a long time, I did not put too much time in trying this out.

1.3 Competition

Accuracy of the classifier: (0.714 + 0.8 + 0.875) / 3 = 0.796, which is about 80%, a very good result.

2 Can you fix it? Yes you can!

2.1 Algorithm implementation revisited

2.2 Technical Write up

I chose the SURF technique as mentioned in reducing the dimensionality of the feature descriptors, and I use a “a bag of features” method which I found on Mathworks.com (ref: https://www.mathworks.com/examples/matlab-computer-vision/mw/vision_product-ImageCategoryClassificationExample-image-category-classification-using-bag-of-features). This implementation toolbox was introduced in 2016, and it works very well and fast for image category classification. Basically, this method put all of the training images in a imagedatastore, and then construct a bag of features for the training data. Using this “bag of features” and statistics and machine learning toolbox, matlab will help generate a category classifier. After that, we just do similar things to our test data, i.e., load every image to the imagedatastore, then use the trained image classifier as above for predicting the results. This method is much faster than the one I used above. Before doing everything, I first manually divide the

expanded data into two subfolders: training and testing. For the testing subfolder, I manually divide everything into a subfolder named with its category number.

It took me less than 30min for the all expanded data set.

Shown below is the classification result (confusion matrix).

		Predicted Classes																										
	Classes	006	007	011	012	022	024	025	026	028	031	037	045	051	054	063	072	093	102	129	145	159	171	178	180	251
Actual Classes	006	0.25	0	0	0	0	0	0	0.08333333	0	0.08333333	0.08333333	0	0	0	0	0	0	0.08333333	0	0.08333333	0	0.33333333	0	0	0
	007	0	0.14285714	0	0	0	0.07142857	0.07142857	0.14285714	0.07142857	0.07142857	0.14285714	0	0	0	0	0.07142857	0.07142857	0	0.14285714	0	0	0	0	0	0
	011	0	0	0.6	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0.1	0	0	0	0	0	0.2	0
	012	0	0.1	0.1	0.4	0	0	0	0	0	0	0.1	0	0.1	0	0	0	0	0	0	0.1	0	0	0	0	0	0.1
	022	0	0	0	0	0.42857143	0	0.07142857	0.07142857	0	0	0	0	0	0	0	0.14285714	0	0	0.07142857	0	0	0.07142857	0.07142857	0.07142857	0	0
	024	0.1	0.1	0	0	0	0.6	0.1	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0
	025	0	0.1	0	0	0	0.1	0.5	0	0	0.1	0	0	0	0	0	0	0	0.1	0	0.1	0	0	0	0	0	0
	026	0	0	0.1	0	0.2	0	0	0.2	0	0	0	0	0	0	0	0.1	0	0	0.1	0	0	0.1	0.2	0	0	0
	028	0	0	0.1	0	0	0	0	0	0.4	0.1	0.1	0	0	0	0	0	0	0	0.1	0.1	0	0	0	0	0.1	0
	031	0	0	0	0.2	0.06666667	0	0	0	0	0.33333333	0.06666667	0	0	0	0	0	0.06666667	0.2	0	0	0	0	0	0	0.06666667	0
	037	0	0.07142857	0	0	0	0.14285714	0	0	0	0	0.28571429	0	0	0	0	0.07142857	0.07142857	0	0	0.07142857	0.07142857	0	0.21428571	0	0	0
	045	0	0	0.07142857	0.14285714	0	0	0	0	0.07142857	0.07142857	0	0.14285714	0.07142857	0.07142857	0.07142857	0	0.07142857	0.14285714	0	0	0	0	0	0	0.07142857	0
	051	0	0.1	0.2	0	0	0	0	0	0.2	0.1	0.1	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0.1
	054	0	0	0	0	0	0.1	0	0	0	0.1	0	0	0.1	0.5	0	0	0	0	0	0	0	0	0	0.1	0.1	0
	063	0.1	0	0	0	0.1	0	0	0	0	0.1	0	0	0	0	0.4	0	0	0.1	0.2	0	0	0	0	0	0	0
	072	0.1	0	0.1	0	0	0	0	0.1	0	0	0	0	0	0	0	0.4	0	0.2	0	0	0	0	0.1	0	0	0
	093	0	0.2	0	0	0	0	0.1	0	0	0.1	0.1	0	0.1	0.2	0	0	0	0	0	0	0.1	0	0	0	0	0.1
	102	0	0	0.1	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0.1
	129	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	145	0	0	0	0	0.1	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0.2
	159	0	0.1	0.1	0	0	0.1	0.2	0.2	0	0	0.1	0	0	0	0	0	0	0	0	0	0.1	0.1	0	0	0	0
	171	0	0	0.125	0.0625	0	0	0	0	0	0	0	0	0	0	0	0.125	0	0	0	0	0	0.5	0.0625	0	0.125
	178	0	0	0	0.1	0	0	0.1	0	0.1	0	0	0.1	0	0	0	0.1	0	0.1	0	0	0	0.1	0.2	0	0.1
	180	0	0	0	0.07142857	0.07142857	0	0	0	0	0	0	0.07142857	0	0	0.07142857	0	0.07142857	0	0	0.07142857	0	0.14285714	0	0.35714286	0.07142857
	251	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0625	0	0	0	0	0	0	0	0.9375

2.3 Competition
Accuracy for 006-045: 0.357
Total Accuracy: 0.41

We can see that the total accuracy is not very bad, but still have a lot of room for further improvement. But this method really improves the speed a lot. I tested the reduced data set using this method and it only took about 2 – 3mins, which is about 10% of the time using the method in the first part. That is to say, if I use the method in part I for expanded data set, this will at least take 30min * 10 = 300min, which is about 5hrs.

3 Grad Credits

3.1 Reading

The authors in this paper demonstrated how they use Support Vector Machines (SVM) to solve problems in object recognition and image classification. They found that SVM can be trained even if there are way more dimensionality than number of examples in the input space. They also found that a $\chi 2$-based kernel provided the best results and could be extended to other problems.

SVMs will find the hyperplane given a set of points which belong to two classes, and this hyperplane will separate these two classes, which leave the largest possible fraction of points of the same class on the same side and maximizing the distance of either class from the hyperplane. Though SVM is a binary classifier, we can use it to do multi-class classification by combining several binary classifiers.

In this paper, the author use one method is to combination of binary classifiers called “one against the others”, in which n hyperplanes are constructed to separate one class from the other class and n is the number of classes. In order to use SVM in image classification, the authors first preprocess the images, then separate the images to training and test sets, followed by choice of representation and way of training, and finally training and testing.

To handle noise, this paper use transition histogram to bypass the difficulty of handling images with noise, which is to count jumps in the signal in the histogram of the derivative. This works very well for them. To handle non-linearities, this paper use penalty term C for misclassification. To handle dimensionality, they use PCA to reduce the dimensionality, though the dimension of the input space is large compared with the size of the training data set, which makes the data linearly separable.

If I am to conduct these experiments, I may use SURF to get feature points instead of using PCA since the number of dimensionality here does not matter much. Also, I may divide the portion of training data set versus test data set to a different portion. In this paper, they use 2 training set with 1 test data set. I think the test set is too small compared with the training set. Instead, I may use 1 training set with 2 test data set.

3.2 Train SVM on 3 class dataset

For this part, I first load all of the images from the training set of the reduced dataset. I put all of them in the image datastore and get the bag of features using function “bagOfFeatures”. After that, I use two arrays, trainingPred, which stands for training predictor, and trainingLabel, which is the label of these predictors, and put the bag of features of each individual image in the predictors. The predictor and the label are then used for constructing the SVM training model. After I get the model, I then use the same process to get the data from the test set, and use it to predict the labels from the test predictors and the trained model. I then compare the predicted label results with the test labels, which are the real labels, and finally got the confusion matrix below.

Actual \ Predicted	Buddha	Butterfly	Airplane
Buddha	0.85714286	0.14285714	0
Butterfly	0.3	0.7	0
Airplane	0	0	1

The result is very good, especially for the airplane prediction, which reaches 100% of accuracy. It is not as good as the previous result in part 2 for butterfly prediction.

3.3 Competition

Accuracy of the classifier: (0.857 + 0.7 + 1) / 3 = 0.796, which is about 85%, a very good result, better than the previous result.