CSCI-677 Home Work Assignment #5
Ye Joo Park (USC# 1128-6851-51)

1. **A brief description of the program**

Python 3 / OpenCV 3 + contrib modules are used.

The program starts by retrieving the list of image files and corresponding classes for training images.

```
# directories are separated by labels
def parseImageDirectory(dir_path):
	class_names = os.listdir(dir_path)

	# image paths
	image_paths = []

	# image classes corresponding to image_paths list
	image_classes = []

	# class ID begins at 0, increment by when navigating to a new directory
	class_id = 0

	# iterate over each folder
	for class_name in class_names:
		dir = os.path.join(dir_path, class_name)
		listed_image_paths = [os.path.join(dir, f) for f in os.listdir(dir)]
		image_paths += listed_image_paths

		# add class labels for each image inside the same directory
		image_classes += [class_id] * len(listed_image_paths)

		class_id += 1

	return image_paths, image_classes, class_names
```

Then, PCA-SIFT features are computed for each image. Only the top 20 dimensions are left.

```
def getPCASIFT(image_path):
	img = cv2.imread(image_path)

	# find keypoints and descriptors with SIFT
	kpts, des = fea_det.detectAndCompute(img, None)

	# perform PCA with SciKit module
	# note that the module is imported as sklearnPCA instead of PCA to avoid conflicts with
other modules
	sklearn_pca = sklearnPCA(n_components=20)
	pca_des = sklearn_pca.fit_transform(des)

	return pca_des
```

After vertically stacking the PCA-SIFT features, they are clustered using K-means clustering.

```
for image_path, descriptor in des_list[1:]:
	descriptors = np.vstack((descriptors, descriptor))  # Stacking the descriptors

# k-means clustering with 'num_clusters'
voc, variance = kmeans(descriptors, num_clusters, 1)
```

After being clustered, histograms are computed.

```
from scipy.cluster.vq import *

# histogram of features
im_features = np.zeros((len(image_paths), num_clusters), "float32")
for i in range(len(image_paths)):
	words, distance = vq(des_list[i][1], voc)
	for w in words:
		im_features[i][w] += 1
```

Now, we are done through steps a) through d) from the homework assignment. We move on to classification. Using a KNN classifier, we are able to get class predictions for test images.

```
clf = cv2.ml.KNearest_create()
clf.train(samples, cv2.ml.ROW_SAMPLE, responses)

# repeat PCA-SIFT calculations and histogram generations for test images here
# code abbreviated since they are almost identical to the training code

# classify (prediction)
retval, results, neigh_resp, dists = clf.findNearest(test_features, k=num_neighbors)
```

Match accuracy values and confusion matrices are derived.

```
import matplotlib.pyplot as plt
import itertools
from sklearn.metrics import confusion_matrix

correct_count = 0
for i in range(0, len(results)):
	if(results[i] == image_classes[i]):
		correct_count += 1

accuracy = round(correct_count / len(results), 2)

cnf_matrix = confusion_matrix(y_true=image_classes, y_pred=results)

# Plot normalized confusion matrix
helper.plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
title=str(num_clusters) + ' clusters, ' + str(num_neighbors) + ' neighbors, accuracy=' +
str(accuracy))

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix',
cmap=plt.cm.Blues):
	if normalize:
		cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
		print("Normalized confusion matrix")
	else:
		print('Confusion matrix, without normalization')
	print(cm)

	plt.imshow(cm, interpolation='nearest', cmap=cmap)
	plt.title(title)
	plt.colorbar()
	tick_marks = np.arange(len(classes))
	plt.xticks(tick_marks, classes, rotation=45)
	plt.yticks(tick_marks, classes)

	fmt = '.2f' if normalize else 'd'
	thresh = cm.max() / 2.
	for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
		plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white"
if cm[i, j] > thresh else "black")

	plt.tight_layout()
	plt.ylabel('True label')
	plt.xlabel('Predicted label')
	plt.show()
```

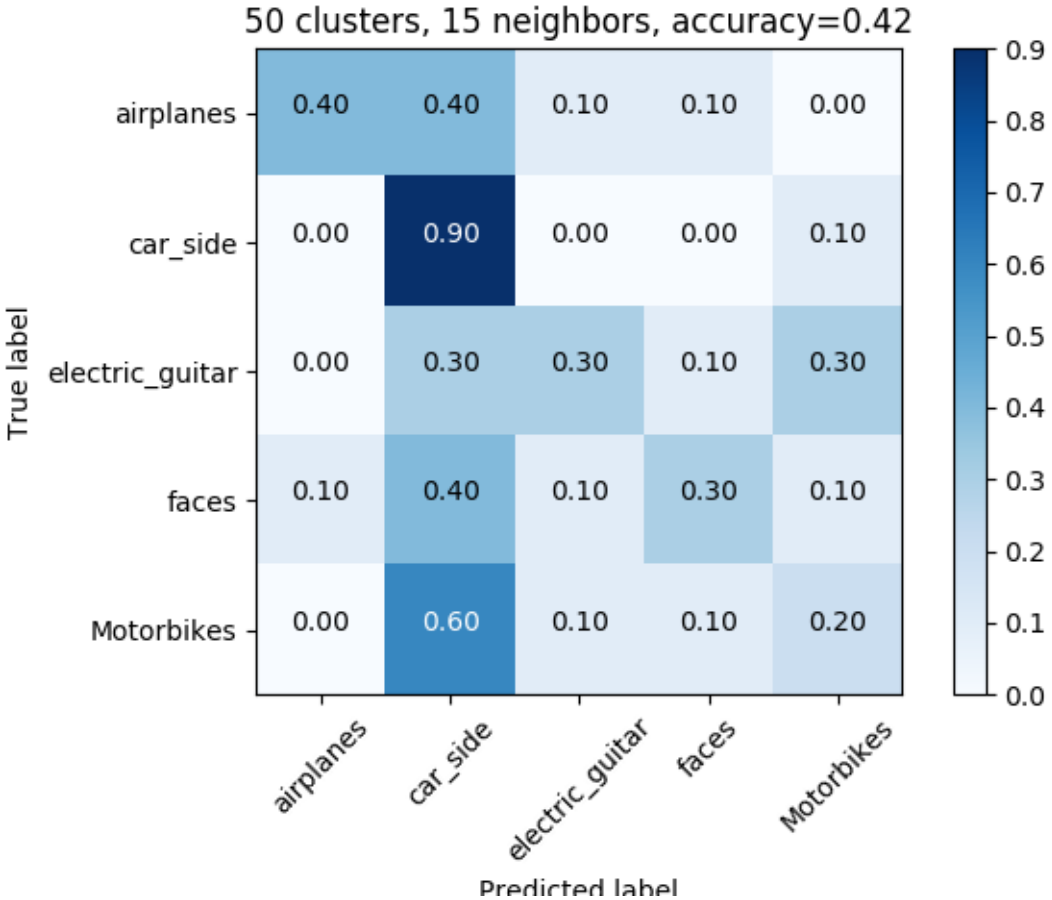
The plotted images for the confusion matrices are shown below (at part 2).

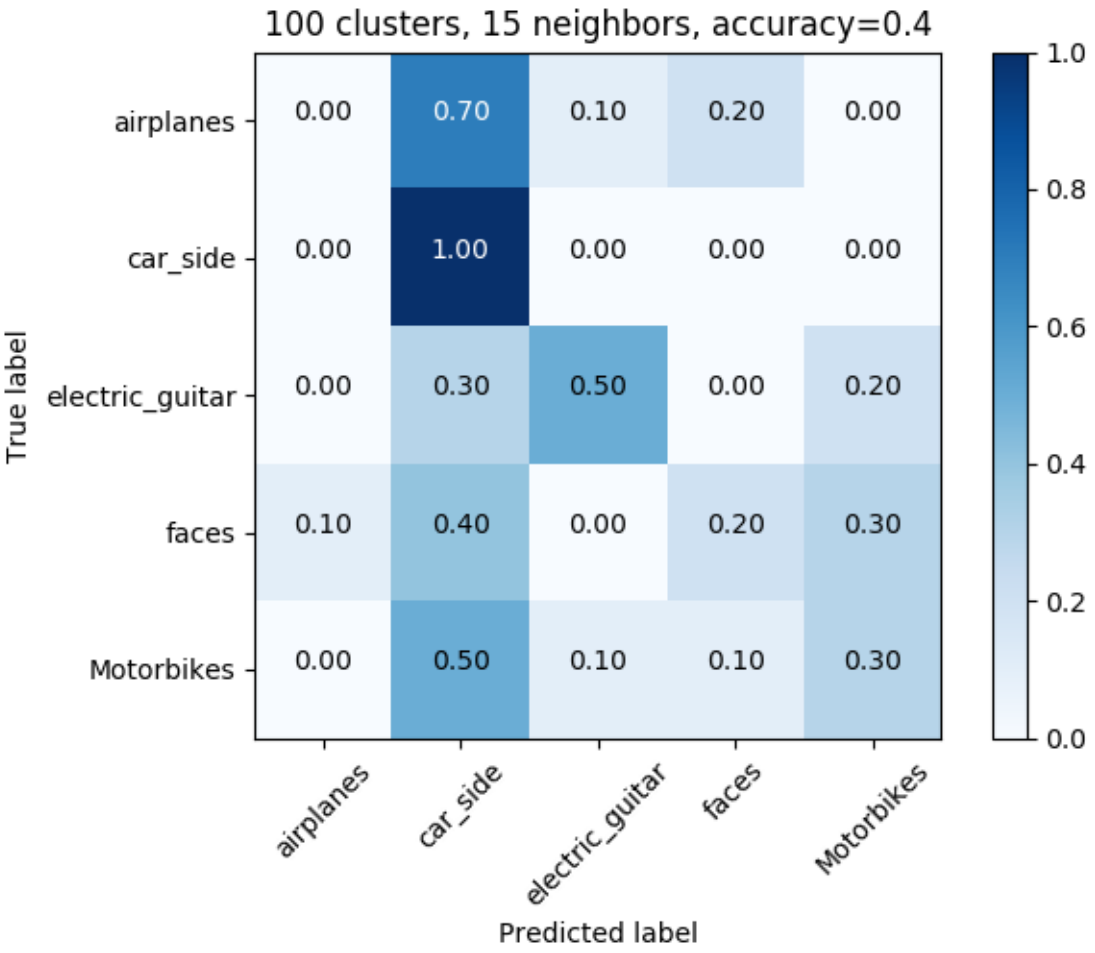
2. **Step by step results with different parameters for number of parameters and neighbors**

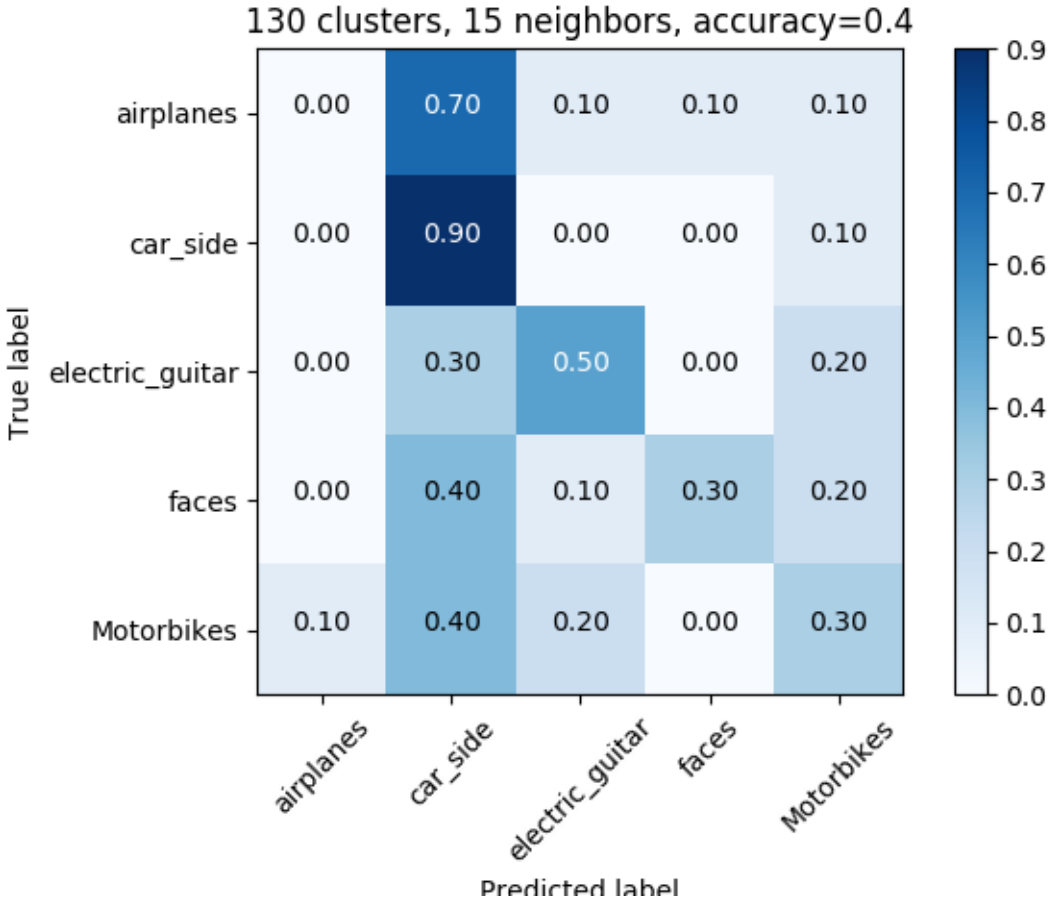
To find the optimal parameter (number of clusters for K-means and number of neighbors to count for KNN), I have computed normalized confusion matrices along with accuracy values (between 0 and 1).

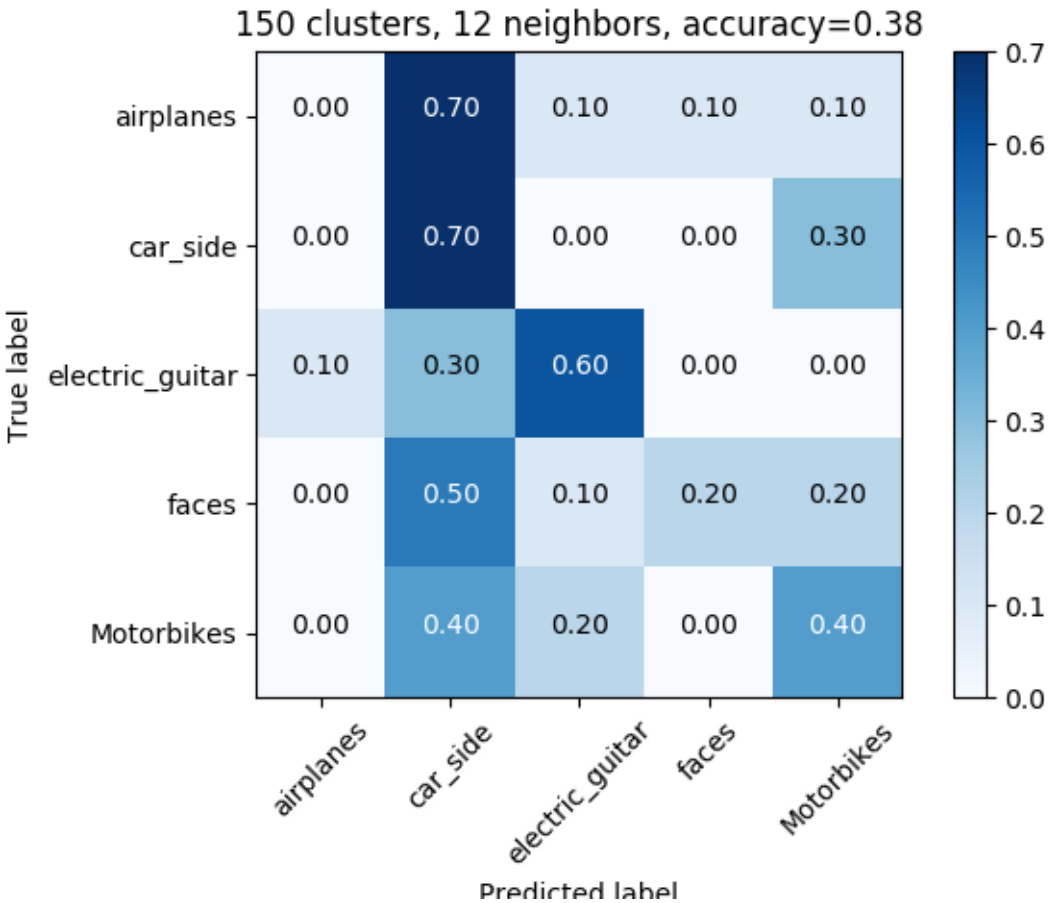
```
Park@Park-PC MINGW64 ~/Downloads/hw5-yejoopark
$ python classify.py
Normalized confusion matrix
[[ 0.   0.7  0.1  0.1  0.1]
 [ 0.   0.7  0.   0.   0.3]
 [ 0.1  0.3  0.6  0.   0. ]
 [ 0.   0.5  0.1  0.2  0.2]
 [ 0.   0.4  0.2  0.   0.4]]
```

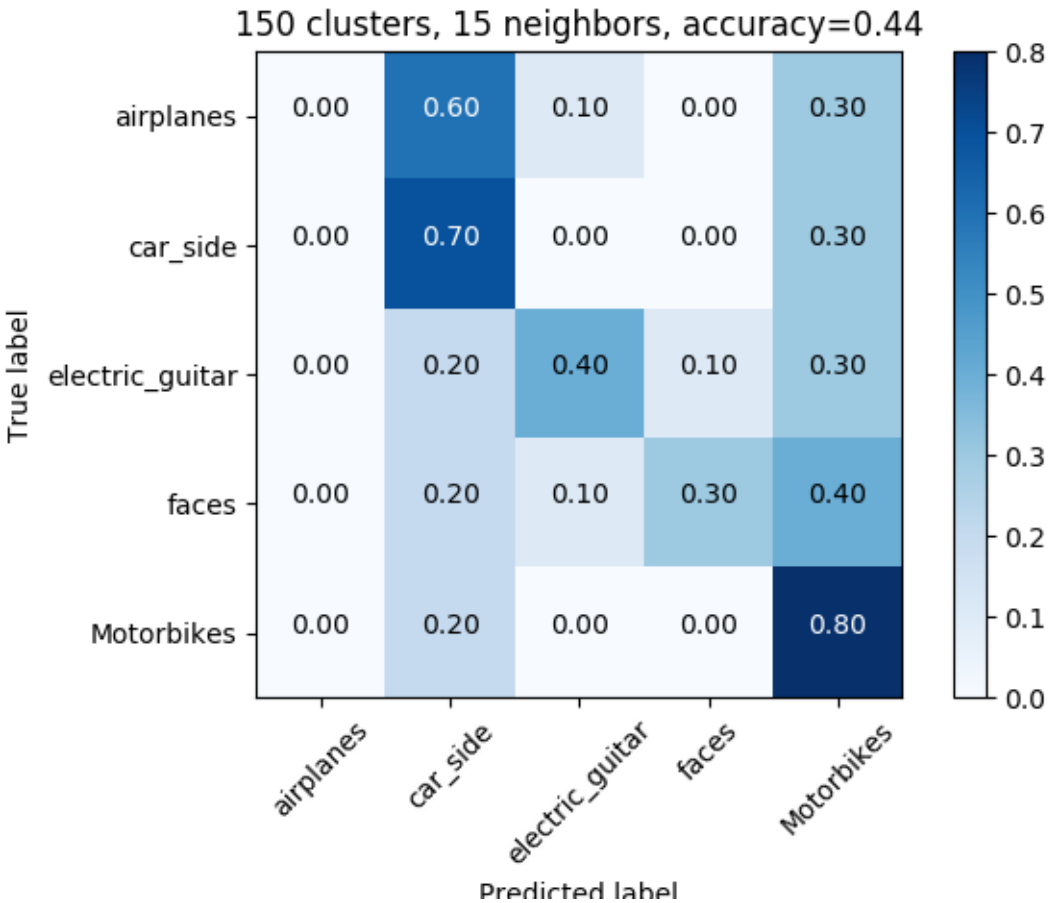
Please refer to the plots below for multiple runs with different parameters.

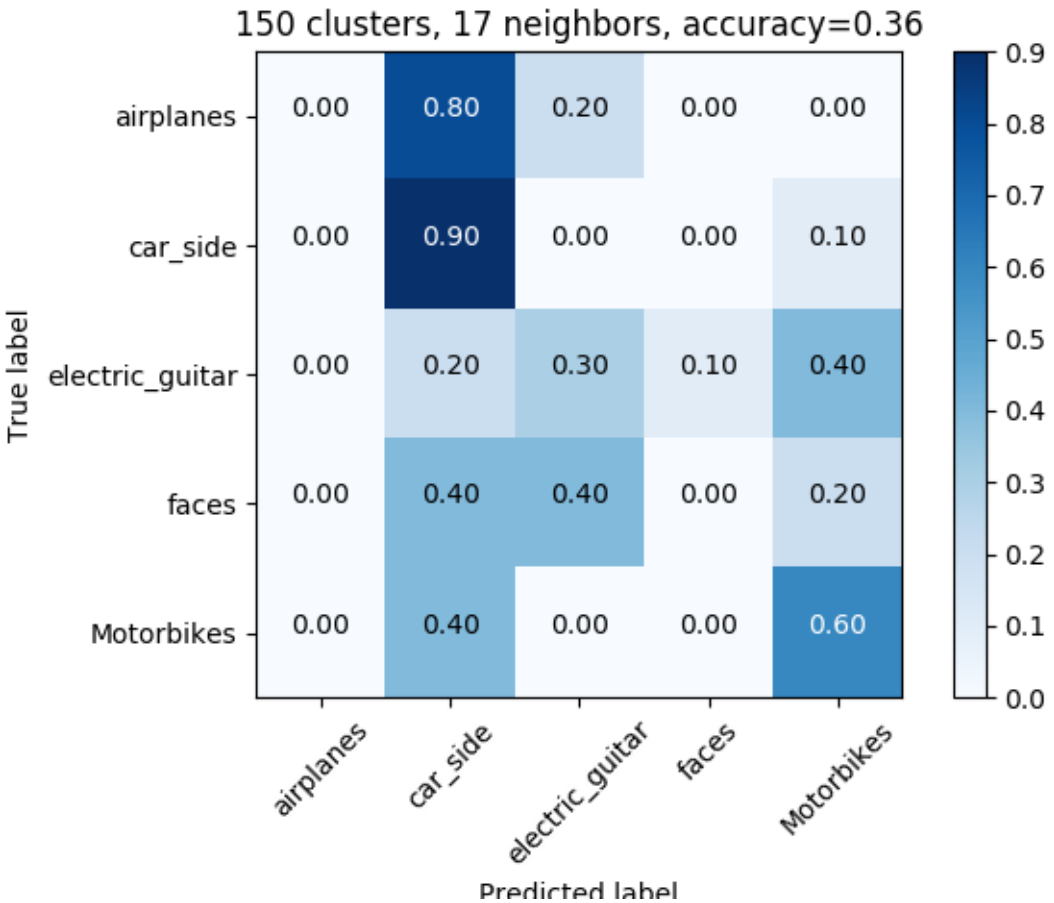


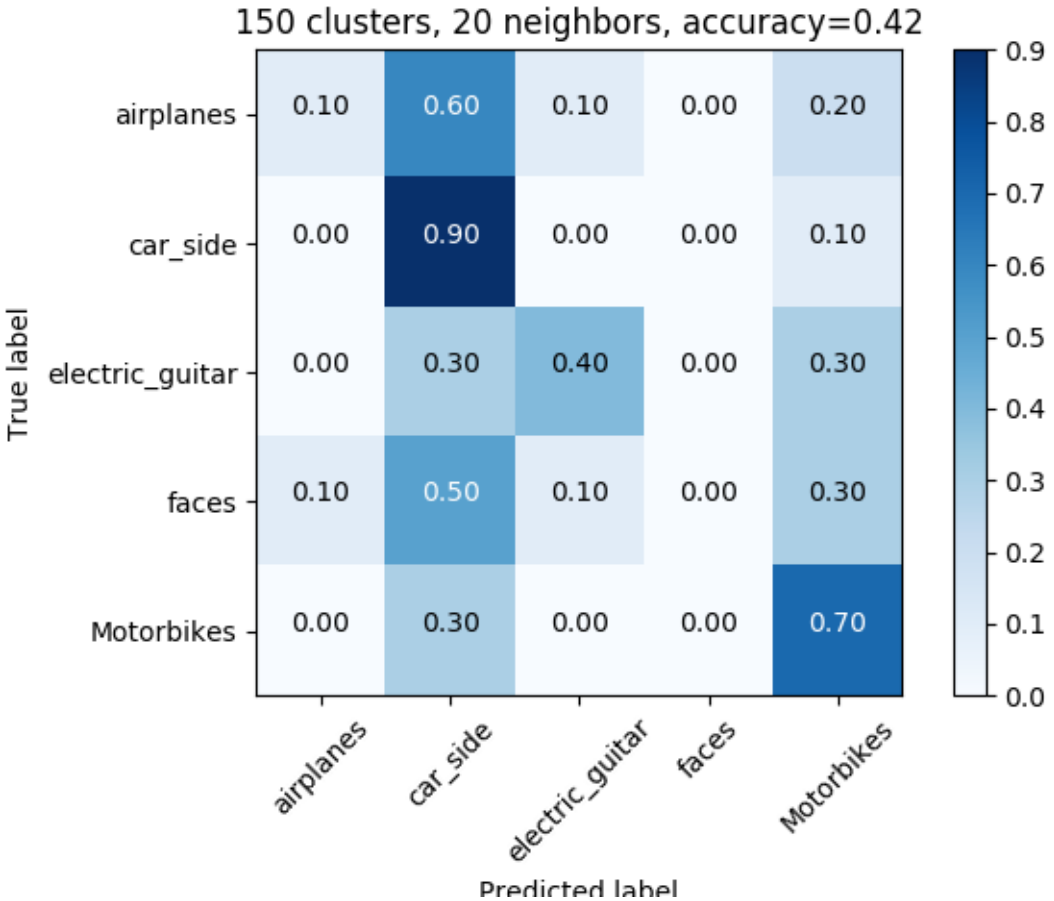

100 clusters, 15 neighbors, accuracy=0.4
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.00 0.70 0.10 0.20 0.00
0.00 1.00 0.00 0.00 0.00
0.00 0.30 0.50 0.00 0.20
0.10 0.40 0.00 0.20 0.30
0.00 0.50 0.10 0.10 0.30
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
1.0
0.8
0.6
0.4
0.2
0.0

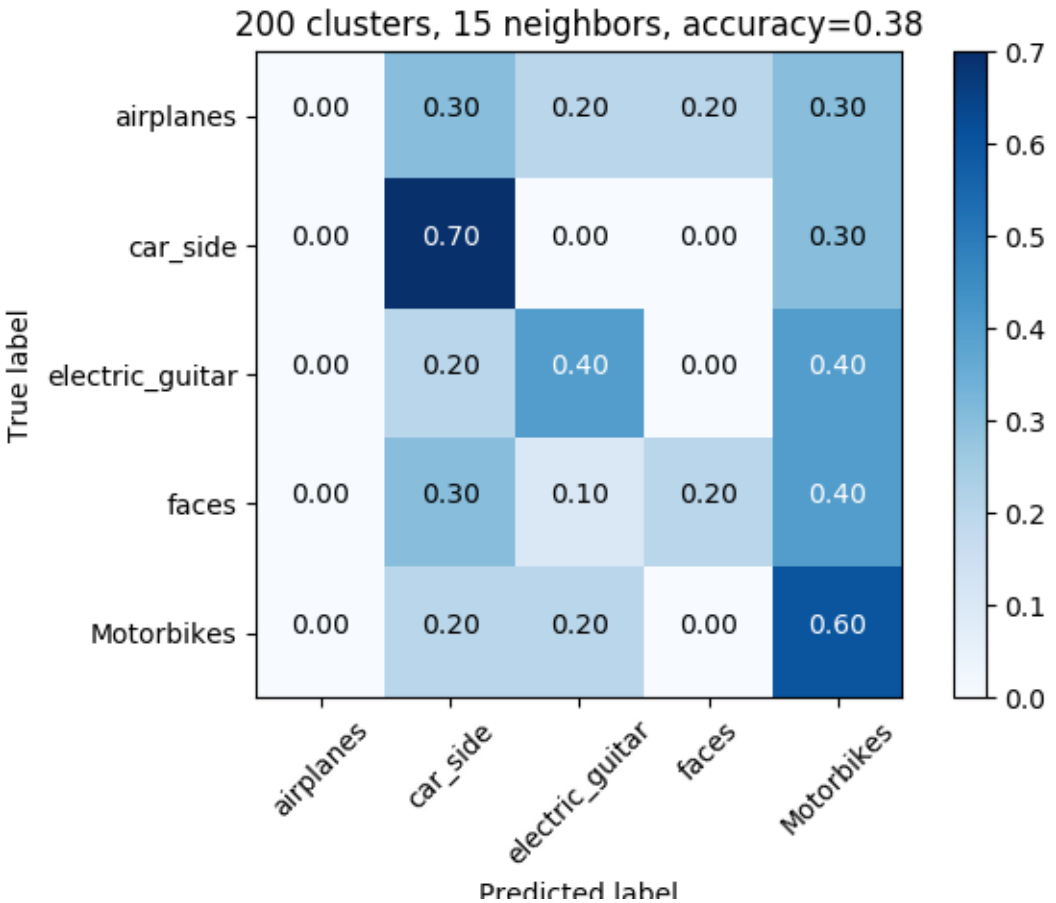

130 clusters, 15 neighbors, accuracy=0.4
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.00 0.70 0.10 0.10 0.10
0.00 0.90 0.00 0.00 0.10
0.00 0.30 0.50 0.00 0.20
0.00 0.40 0.10 0.30 0.20
0.10 0.40 0.20 0.00 0.30
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


150 clusters, 12 neighbors, accuracy=0.38
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.00 0.70 0.10 0.10 0.10
0.00 0.70 0.00 0.00 0.30
0.10 0.30 0.60 0.00 0.00
0.00 0.50 0.10 0.20 0.20
0.00 0.40 0.20 0.00 0.40
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


150 clusters, 15 neighbors, accuracy=0.44
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.00 0.60 0.10 0.00 0.30
0.00 0.70 0.00 0.00 0.30
0.00 0.20 0.40 0.10 0.30
0.00 0.20 0.10 0.30 0.40
0.00 0.20 0.00 0.00 0.80
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


150 clusters, 17 neighbors, accuracy=0.36
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.00 0.80 0.20 0.00 0.00
0.00 0.90 0.00 0.00 0.10
0.00 0.20 0.30 0.10 0.40
0.00 0.40 0.40 0.00 0.20
0.00 0.40 0.00 0.00 0.60
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


150 clusters, 20 neighbors, accuracy=0.42
airplanes
car_side
electric_guitar
faces
Motorbikes
True label
0.10 0.60 0.10 0.00 0.20
0.00 0.90 0.00 0.00 0.10
0.00 0.30 0.40 0.00 0.30
0.10 0.50 0.10 0.00 0.30
0.00 0.30 0.00 0.00 0.70
airplanes
car_side
electric_guitar
faces
Motorbikes
Predicted label
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0



It seems that the best overall prediction rate is around 0.44 with 150 clusters and 15 neighbors. For analysis, please see the next section.

3. Summary and discussion

As observed in the previous section, the accuracy was highest at 150 clusters with 15 neighbors.

The prediction rate was very sensitive to the number of neighbors. At 150 clusters, the accuracy rate goes from 0.38 with 12 neighbors for KNN to 0.44 with 15 neighbors. The accuracy rate goes back down to 0.36 with 17 neighbors, but goes up again to 0.42 with 20 neighbors.

Number of clusters	Number of neighbors for KNN	Accuracy
150	12	0.38
150	15	0.44
150	17	0.36
150	20	0.42

What does this abnormal fluctuation indicate? Intuitively, this classification is not working too well. The accuracy rate is under 50% (still better than a random guess of ~20%, but much lower than the average accuracy stated in the referenced paper.

TABLE I THE COMPARISON OF THE EXPERIMENTAL RESULTS

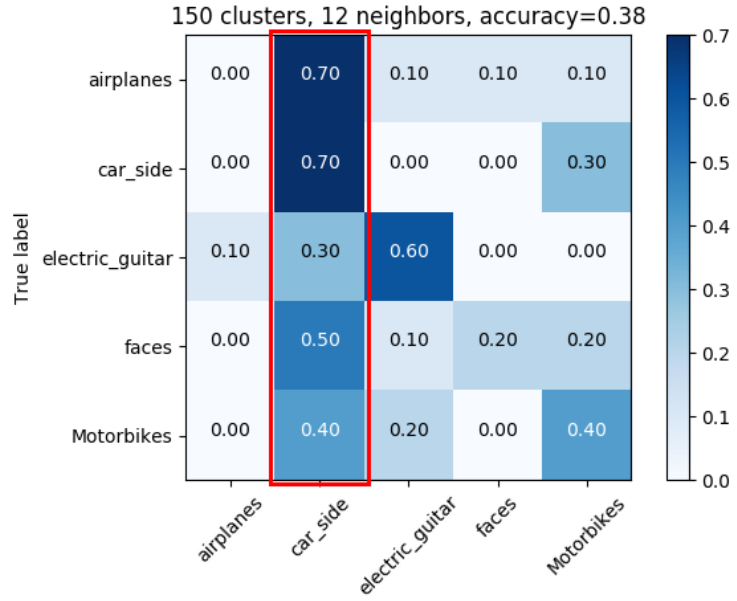
Algorithm	Correctly classified images	Average accuracy
Traditional SIFT	88/120	73.33%
SIFT based on patches	95/120	79.17%
SIFT based on patches employing PCA theory	104/120	86.67%

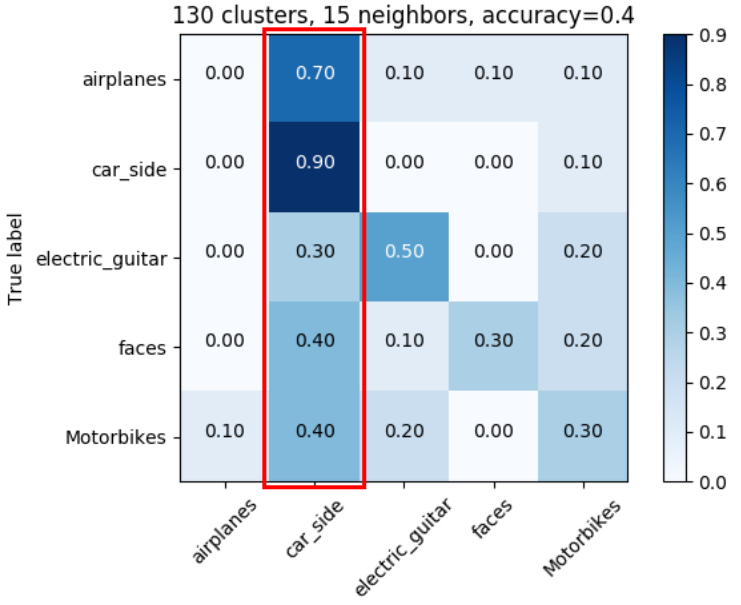
Results from “Image Classification with Bag-of-Words Model Based on Improved SIFT Algorithm”

Using KNN to classify the test images in this assignment has two shortcomings.

First, the number of images to train with is too small. In fact, the size is so small that it won’t show any meaningful improvement even if we use SVM. If we used a larger image set (Caltech-101 as an example), the program will be likely to show better results.

Second, PCA-SIFT features are not the best features to use to classify the given image sets. Many images were classified as “car_side” class even if they were other types. Images of airplanes and faces were rarely correctly classified. These trends are shown below.





Over half of class images that are not “car_side” images are classified as “car_side”.

What can be done to improve the accuracy? We can try increasing the size of training data set, using different features (other than PCA-SIFT that are more agnostic to different classes compared to SIFT descriptors), or using a different classifier (such as SVM).