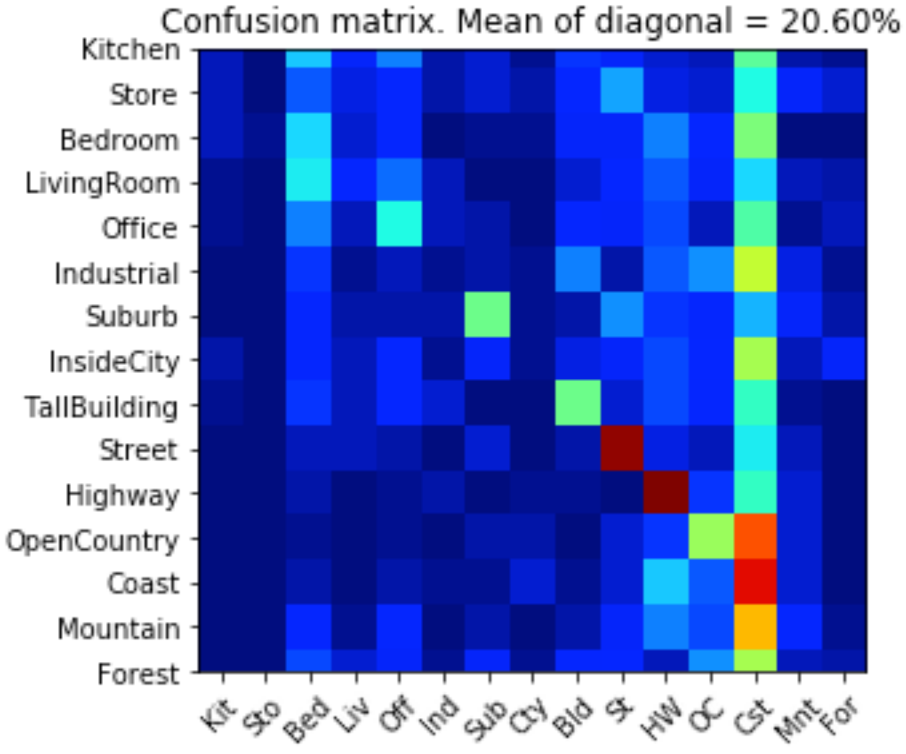
CS 4476 Project 5

Ashwin Rathie
Ashwin.rathie@gmail.com
903281887

Part 1: Your confusion matrix, together with the accuracy for Part 1 with the standard param set (image_size = 16, k = 3)



Part 1: Experiments: change image size and k individually using the following values, and report the accuracy (when tuning one param, keep the other as the standard (16 x 16, 3)):

ie. when you’re tuning image size, keep k at 3, when changing k, keep image size as 16x16

image size:

8 x 8: 19.00%

16 x 16: 20.60%

32 x 32: 19.67%

k:

1: 22.13%

3: 20.60%

5: 21.07%

10: 21.73%

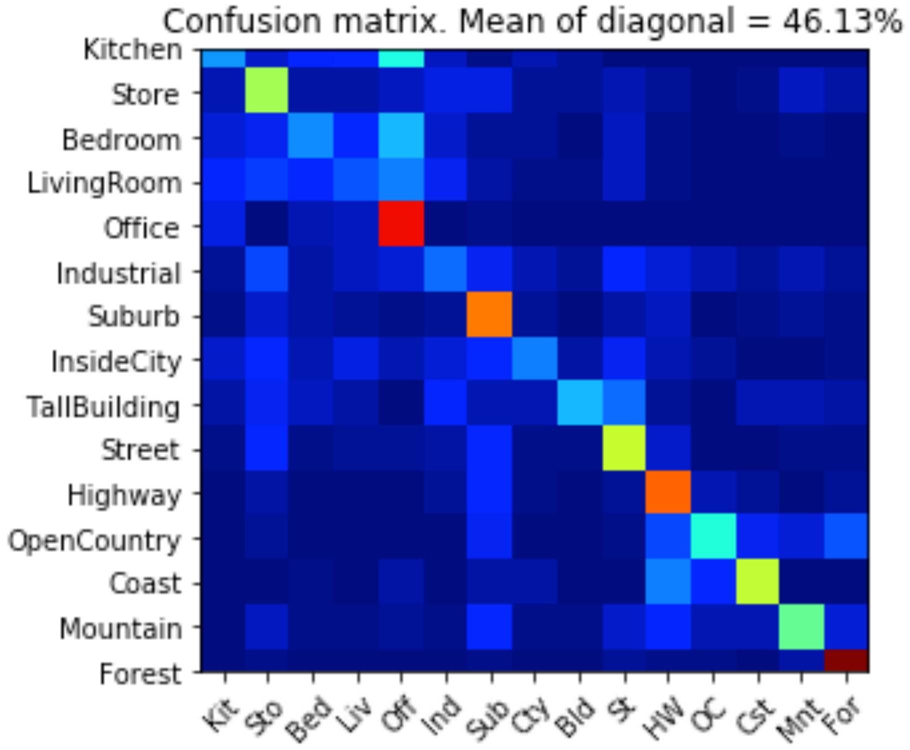
15: 21.67%

Part 1: Reflection: when tuning the parameters, what have you observed about the *processing time and accuracy*? What do you think might lead to this observation?

Processing time: When tuning k, I found that increasing the k value also (slightly) increased processing time. This makes intuitive sense to me as a higher k means that for every queried point, we must look at a greater number of nearby points to evaluate the test point. When tuning image size, I also found that increasing image size increased the processing time. This also makes sense because image size dictates the number of features, and a higher number of features to compare means more computation is required.

Accuracy: A k valued of 1 resulted in the highest accuracy, but I believe this to be an unreliable conclusion and the result of coincidence. K values of 5-15 were pretty stable and were reasonably high. I think this is because it is a range that is high enough to be resilient to noise but not so high that every query it generalizes to the mean (or in this case, the mode). 16x16 was most accurate image size for me, with a default interpolation setting (interpolation=INTER_LINEAR) for cv2.resize(). 16x16 appears to be enough pixels to retain distinguishable details but not too high to make reach feature too specific.

Part 2: Your best confusion matrix, together with the accuracy for Part 2. Also report your param settings to get this result.



Param settings:

vocab_size: 80

stride (build_vocab): 20

step_size(get_bags_of_sifts): 10

max_iter (k-means): 50

k (kNN): 16

Part 2: Reflection: when experimenting with the value k in kNN, what have you observed? Compare this performance difference with the k value experiment in Part 1, what can you tell from this?

In part 1, k values from 5-15 were generally very stable and similar. In part 2, my optimal k value was 16, and a range of 16-19 gave pretty stable and similar results. While the 2 optimal ranges in part 1 and part 2 almost overlap, they are distinct. There is a clear and steady drop in accuracy in part 2 when decreasing the k value from 16.

From these observations I would conclude that generally, the methodology we used in part 2 favors higher k values than that of part 1. This makes sense because in part 1, the “features” were simply the pixel values of the resized image, the tiny image. In part 2, the features of derived from SIFT, which is a much more descriptive metric that takes into account gradient.

Reflection on Tiny Image Representation vs. Bag of Words with SIFT features

Why do you think that the tiny image representation gives a much worse accuracy than bag of words? As such, why is Bag of Words better in this case?

In the Tiny Image representation, the “features” were the pixel values of the the tiny image, i.e. the 256 (16x16) pixels of the tiny image. In Bag of Words, the features of derived from SIFT and put into histograms, that were then represented on a graph where centroids (visual vocab points) were calculated. This is a much more descriptive metric that takes into account gradient, meaning that comparing SIFT features to identify classification is going to be much more effective than comparing pixel values.

Conclusion: briefly discuss what you have learned from this project.

This project really broadened how I thought of the kMeans algorithm. Specifically, it applied the processes in ways I hadn’t considered before. Previously, I never considered the use of kMeans for image processing/classification because I primarily associated kMeans with more traditional forms of datasets, numerical datasets rather than image datasets. But in this project, we performed kMeans on a representation of SIFT features, which are in itself a representation of images. I suppose I had never really “put 2 and 2 together” and connected what we learned earlier in this class (representing the an image using its features) with what I had already known about kMeans (processing numerical data). The idea of a visual vocab in the form of the centroids was also very clever.

Code and Misc. (DO NOT modify this page)

Part 1

Part 2

Late hours

Violations

Extra Credit

<Discuss what extra credit you did and analyze it. Include images of results as well >