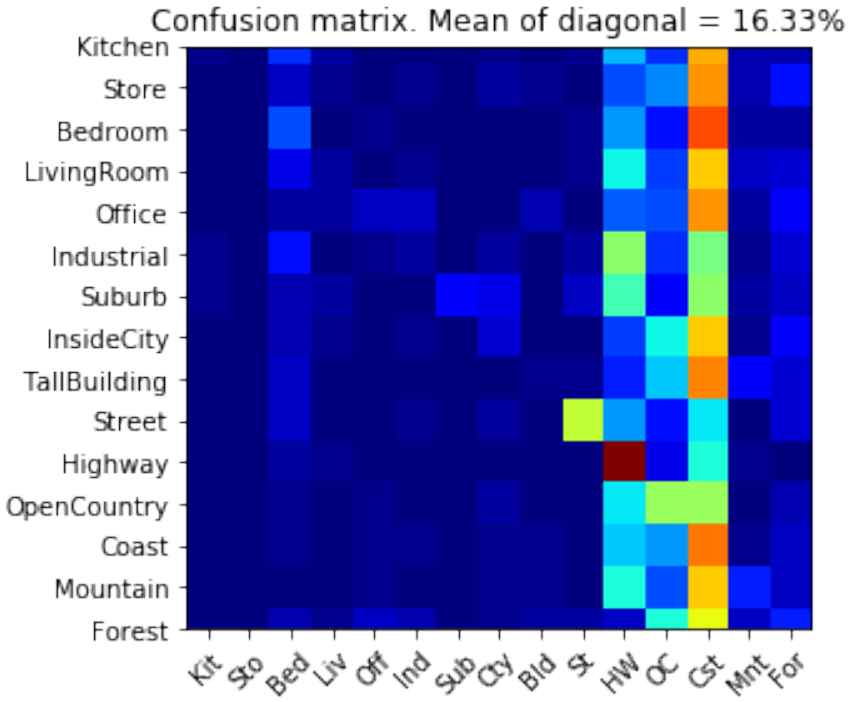
CS 4476 Project 5

Jie Lyu
jlyu31
903329676

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: 16.87%

16 x 16: 16.33%

32 x 32: 15.93%

k:

1: 19.40%

3: 16.33%

5: 17.07%

10: 15.53%

15: 15.60%

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?

When image size is bigger, processing time takes longer. This is because bigger the tiny image size, more dimension a feature will have, so the distance computation for clustering takes more computation. Also, when image size is increased from 8x8 to 16x16 to 32x32, the accuracy drops from 16.87% to 16.33% to 15.93% (with k = 3). This is because when we use tiny image algorithm, we can trying to capture the ”big picture” of an image. If the resolution is too high, details will be added, so the distance between similar features will be bigger because of the unnecessary details

When changing the k size, processing time did not change much. Theoretically the processing time will be longer for a little bit when k is bigger, since the voting algorithm is querying from more labels. However, here the change is small because the k are (k = 1, 3, 5, 10, 15). Among the five options (with image size = 16x16), k = 1 gives the best result with accuracy of 19.40%. In general, as k increases beyond the optimal value, the accuracy drops as we can see with k = 10 and k = 15, the accuracy drops to about 15%. This is because when k is too big, labels from other clusters are added. The voting algorithm will take votes from them, and the result is more likely to be incorrect.

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?

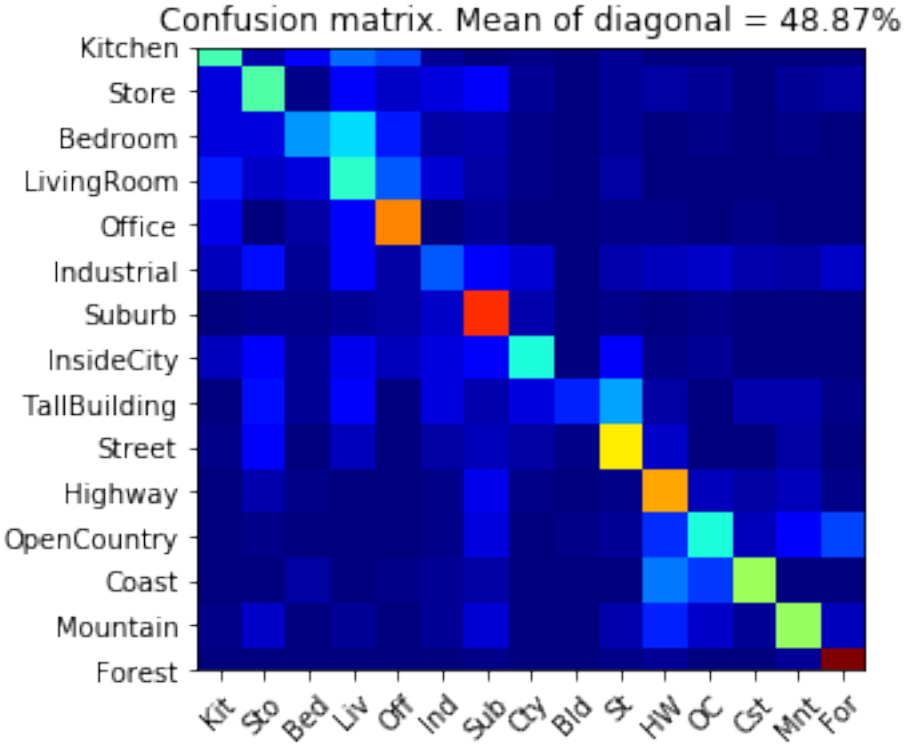
The optimum k (with other hypermeters shows below the table) is 20 with an accuracy of 45.33%. As we decrease k or increase k from 20, the accuracy drops.

Compared to the k experiment in part 1, here is accuracy is significantly higher. In part 1, k = 1 gives 19.40% and k = 10 gives 15.53%. Here k = 1 gives 40.87% and k = 10 gives 44.87%. It shows the bag of word with SIFT is much better than tiny image approach.

k size	accuracy
1	40.87%
5	42.93%
10	44.87%
20	45.33%
30	44.40%

vocab size = 50
stride = 20
step size = 10
max_iter = 10

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: 100

stride (build_vocab): 20

step_size(get_bags_of_sifts): 10

max_iter (k-means): 10

k (kNN): 15

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 tiny image approach, we are treating the tiny image as a feature. It is bad because we size the images down, details get lost e.g. high frequency contents. Also, it is bad when there is brightness and spatial shift, which can be common. One solution to the shift problem is to use alignment. Since we are not using any alignment method in this project, the result is bad.

In the bad of words approach, we use SIFT as a method to extract features. SIFT is robust against the shifting problem described above, and by using SIFT on the original image, the high frequency content and details can be captured. Also, here we are able to adjust hypermeters like vocab size, so we can tune them to get a better vocab list from training set, and we can adjust k and step size to get a better KNN query result.

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

Overall, this project is fun. Before I always used classification as a black box. Now I just wrote one and that feels awesome. What is nice is that I used SIFT that I built in previous projects here, so I can see how things builds up, from pixels level to high level bag of words. I learnt a few new useful numpy method e.g. np.unique. Also it is nice to see the simple bag of word algorithm in practice. I did not think such a simple algorithm will work this well (>45% accuracy). Representing features using vectors reminds me of word2vec in NLP, and refreshed my understanding of vectors being just an array of numbers. I can see how many thing can be done by vectors and how useful it is.

The trade off between time and space complexity was an issue. My first version of pairwise_distances() is fully vectorized, but it ended up taking too much RAM and froze my laptop and jupyter notebook kernel kept dying. I revised it with a nested for loop, and problem solved. So I learnt that the best algorithm is the one that suits your hardware best. Theoretical best is not equal to the best in practice.

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 >