Homework Writeup

Instructions

- This write-up is intended to be 'light'; its function is to help us grade your work and not to be an exhaustive report.
- Be brief and precise.
- Please describe any non-standard or interesting decisions you made in writing your algorithm.
- Show your results and discuss any interesting findings.
- List any extra credit implementation and its results.
- Feel free to include code snippets, images, and equations. Below are useful markup templates for these.
- **Please make this document anonymous.**

Assignment Overview

This assignment had four major parts. The first two parts were different ways to get features for later analysis. One was the tiny images implementation which required taking small chunks from larger images. Second was a bag of words implementation focusing on using HOGs and building a vocabulary. Finally we had two classifier functions to implement, nearest neighbors and multiclass SVM model.

Implementation Detail

1 Tiny Images

This part was very straight forward to me. In the end, it came down to resizing the images, adding them to a list and then converting the list to an array to return it. The one thing I forgot to do at the beginning was reshape from (16, 16) to (256,).

2 Get Bag of Words

For the first half of this part of the assignment in the build vocabulary algorithm the built in libraries were super helpful. Being able to use the feature.hog function, then MiniBatchKMeans and cluster centers, made it a lot more straight forward and efficient. I ended up using (4,4) size for both the pixels per cell and pixel per block. For get bag of words it was a similar set up to the previous part and then these lines

```
    distance = sci.spatial.distance.cdist(features, vocab, 'euclidean'
                                          )
    closest_word = np.argmin(distance, axis=1)
```

did a lot of the calculating in this function to end the end return the correct features.

3 SVM

This part was at the same time one of the easier ones for me but also the most interesting. I really struggled on this function for a while until one of my friends recommended using a dictionary to store stuff versus using arrays like I had been. This ended up being extremely valuable. This is how I ended up using the dictionary to help store the different classes

```
    SVM_dictionary = {}
    for sect in np.unique(train_labels):
        binary_labels = np.where(np.array(train_labels) == sect, 1, -1
                                                 )
        weights, bias = SVM().train(train_image_feats, binary_labels)
        SVM_dictionary[sect] = weights.reshape(-1), bias
```

Which ended up working really nicely

4 K nearest neighbors

For this algorithm, it came down to making sure the shapes were right and that the lists I was updating and resetting I was doing so in the right places. For instance, one issue I had for a while was I was not resetting the close labels list for every image so it was just becoming one giant list, which in turn made it think every single image was a highway. Also, I ended up using k=30 which seemed to work well.

5 Results

Bag of words with SVM was 61.4 percent. Bag of words with nearest neighbor was 54.2 percent. Tiny image with nearest neighbor was 18.8 percent.

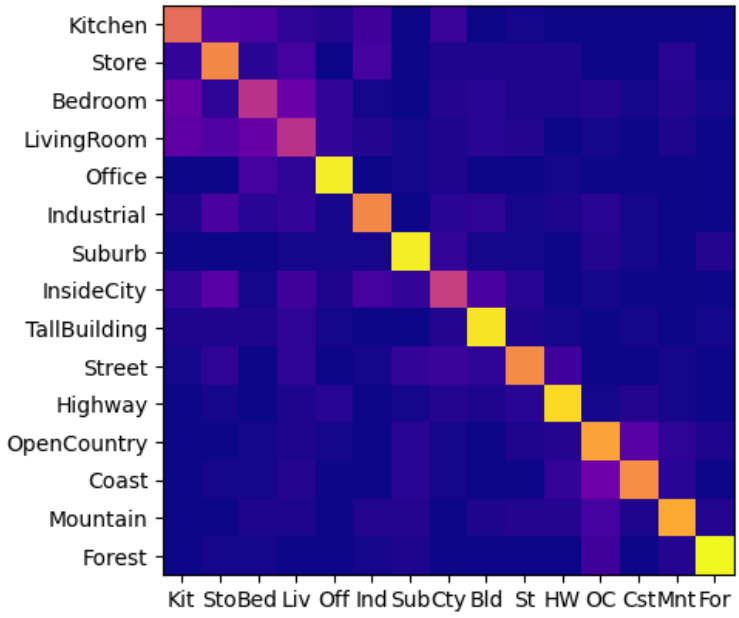

Figure 1: Confusion Matrix

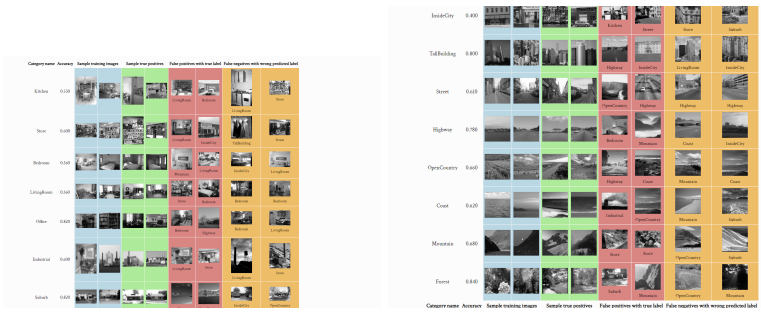

Figure 2: Table