Homework 5 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- **Please make this document anonymous.**

In the beginning...

Starting by coding the different algorithms. It was not that hard to extract the features from the image. In the first iteration I implemented the HOG, PCA, Bag of Words and SVM part. When this code was running properly, I added the SIFT and spatial pyramid algorithms.
Starting with the HOG feature extraction, using the predefined opencv function reduced the difficulty by far:

```
grid_size = 16
half_win_size = 16
results = []
for i in range(half_win_size, img.shape[0]-half_win_size, grid_size):
    for j in range(half_win_size, img.shape[1]-half_win_size,
    grid_size):
        result = hog.compute(img=img[i-half_win_size:i+half_win_size,j
    -half_win_size:j+half_win_size])
        results.append(result)
```

Looping through the image patches and calculating the features for each. One special thing about this is the grid we should use. This is realized by increasing the loop index by the grid size.

Luckily I know the PCA code from a previous class, so it was not that hard to implement this part:

```
vocab = np.load(f'vocab_{feature}.npy')
mean = np.mean(vocab, axis=0)
std = np.std(vocab, axis=0)
vocab = (vocab - mean) / std

cov = np.cov(vocab, rowvar=False)
eigvals, eigvecs = np.linalg.eig(cov)
basis = eigvecs[:, :feat_num]

reduced_vocab = vocab @ basis

return reduced_vocab
```

After normalizing the features, I performed a eigenvalue/vector decomposition. Since the eigenvalues are already descending ordered, just taking the first vectors fulfills the requirements. Representing the feature-points in the new coordinate-system gives the feature reduced representation.

For the bag of words, we used the previously written feature extracting function.

```
1  vocab = np.load(f'vocab_{feature}.npy')
2  vocab_size = vocab.shape[0]
3  hists = np.zeros((len(image_paths), vocab_size))
4  for i in range(len(image_paths)):
5      img = cv2.imread(image_paths[i])
6      descriptors = feature_extraction(img, feature)
7      distances = pdist(descriptors, vocab)
8      nearest_cluster_index = np.argsort(distances, axis=1)[:,0]
9      hist = np.zeros((vocab_size))
10     for j in range(vocab_size):
11         hist[j] = np.count_nonzero(nearest_cluster_index == j)
12     l2_norm = np.sqrt(np.sum(np.power(hist, 2)))
13     hists[i,:] = hist/l2_norm
14 return hists
```

As seen in line 7, the distance calculation was outsourced by a pre-implemented method, so I just had to take the cluster with the smallest distance and distribute the amount to the histogram. After normalization I obtained the features for the SVM.

Using the sklearn SVM classifier, it was not that hard to train a SVM model. First I struggled with the Multiclassifier, but after some time i realized that there is a parameter i could use to train on multi labels. Also i had some issues with the RBF parameter you provided. For me it has to be small, so i added a replacement into my code (see line 4 and 5)

```
1  categories = np.unique(train_labels)
2  C = 1.0
3  results = []
4  if(kernel_type == "RBF"):
5      kernel_type = "rbf"
6  svm_instance = svm.SVC(C=C, kernel=kernel_type,
       decision_function_shape="ovr")
7  svm_instance.fit(train_image_feats, train_labels)
8  for i in range(test_image_feats.shape[0]):
9      img_feature = test_image_feats[i,:]
10     img_feature = img_feature.reshape((1,img_feature.shape[0]))
11     decision = svm_instance.decision_function(img_feature)[0]
12     pred_category = categories[np.argmax(decision)]
13     results.append(pred_category)
14 return np.array(results)
```

Interesting Implementation Detail

In the following i provided the different results for different parameters. The diagrams show how the algorithm classifies the images. The (theoretically) best result is a dark blue image with a light yellow diagonal.

HOG vs SIFT

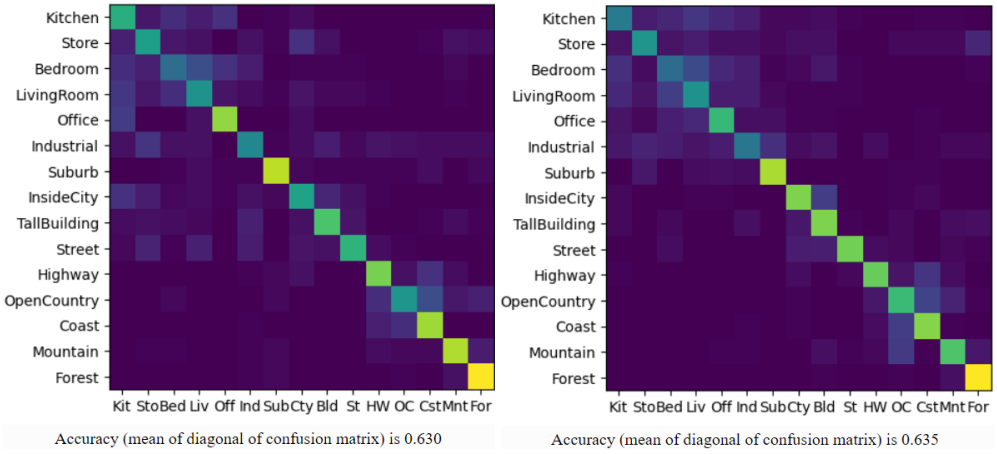

Figure 1: *Left:* HOG (63%) *Right:* SIFT (63,5%)

Both methods have a score of around 63% but the SIFT method performs slightly better than the HOG. It is interesting that some image types getting much better recognised than other one. For instance: Forest and Suburban images. One possible explanation could be, that these type of images provide some very unique properties compared to the other types.
After viewing the images, shows that forest and Suburban have very unique shapes. These shapes lead to a easier classification rule. Other images like Kitchen and Store look nearly the same (especially if you consider them as gray scale image).

PCA of the vocabulary

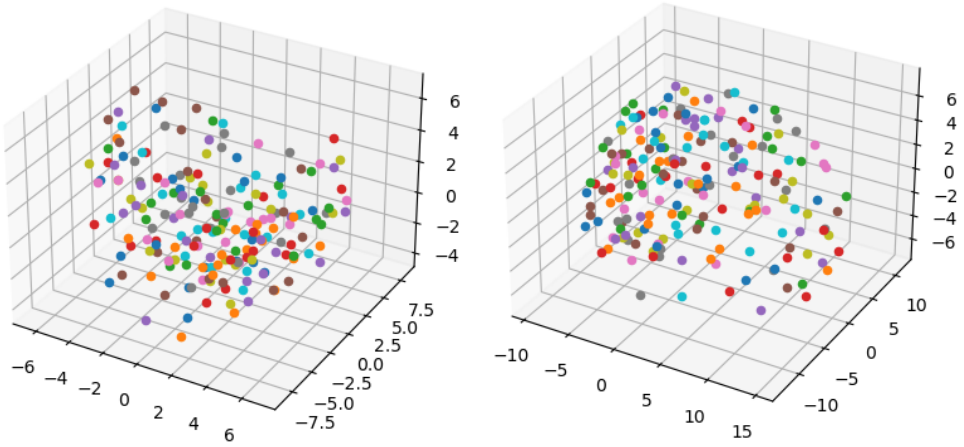

Figure 2: *Left:* HOG *Right:* SIFT

Here you can see the different PCA components of the HOG and SIFT features. This methods reduced the feature space while ensuring the maximal variance in the data. As you can see the points of the HOG seem to be a in a bowl shape (2D quadratic function) and the SIFT seems to be a bowl shape upside down.

Linear vs RBF

Using a kernel function puts the feature points into an other feature dimension. So, the SVM can train a different type of function, which usually leads to a better classification. One drawback of kernels is the danger of over-fitting. The learned function is too much adjusted on the training data points, so it can't score good on the test data set.

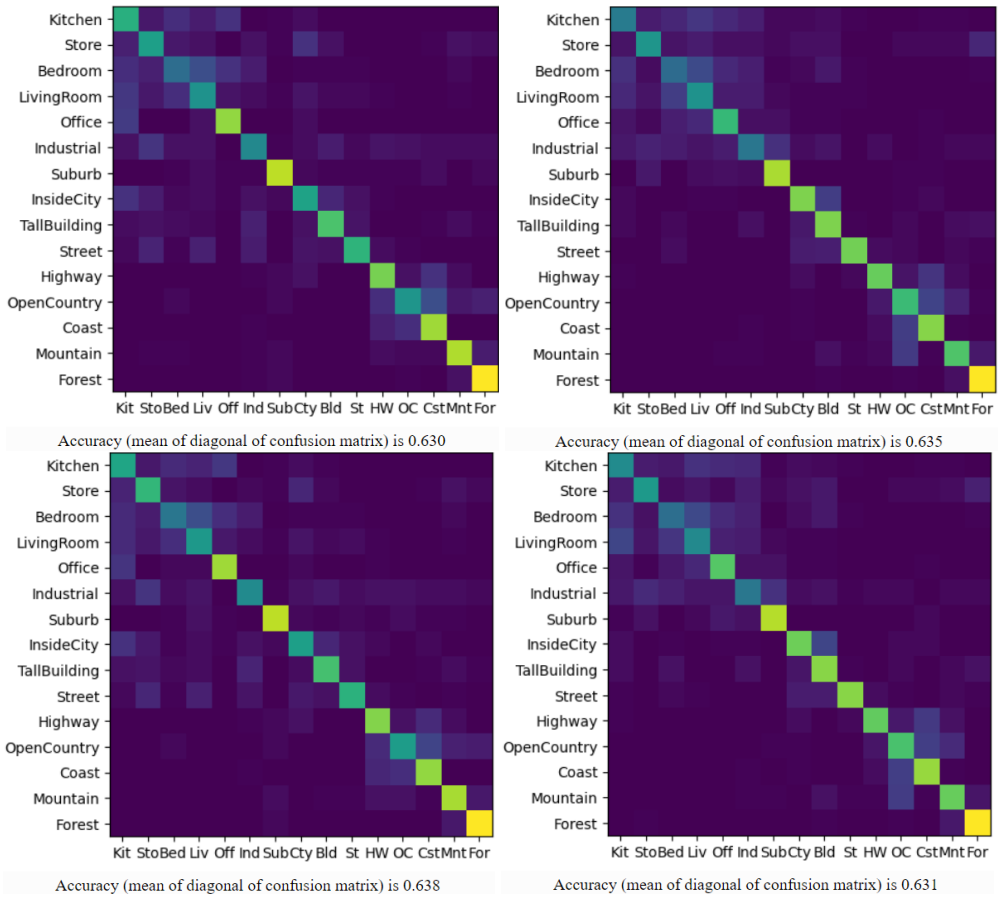

Figure 3: *Left-Top:* HOG-linear (63,0%) *Right-Top:* SIFT-linear (63,5%)
Left-Bottom: HOG-RBF (63,8%) *Right-Bottom:* SIFT-RBF (63,1%)

I would assume that using the RBF kernel would always result in a better performance, since we have a big training set. But actually the RBF just performs better for the HOG case. It improves by 0.8% , while SIFT lost 0.4% of accuracy. This bad behaviour could result due to over-fitting.

bag of words vs spatial pyramid

Increasing the computational power to gain more feature points out of the images. This might result into better results. In general you have always to consider between computational costs and performance gain.

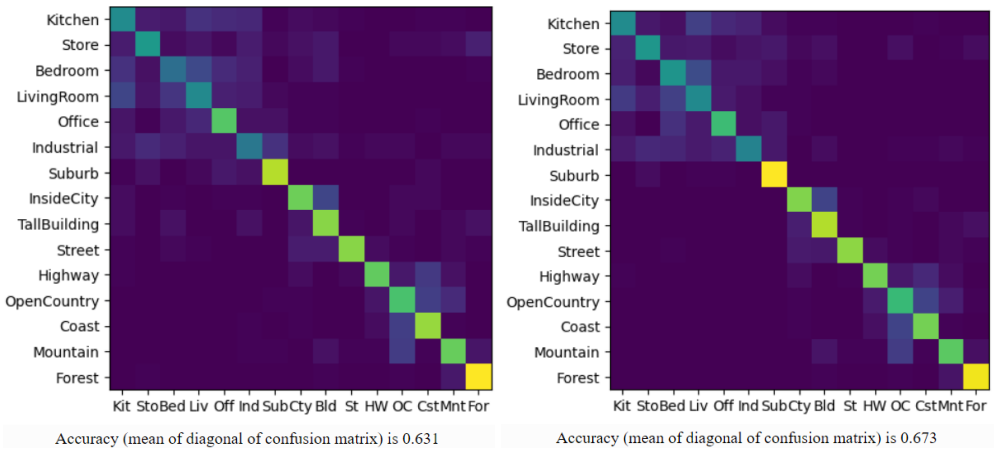

Figure 4: *Left:* Bag of Words (63%) *Right:* Spatial Pyramid (67,3%)

Comparing the SIFT version of Bag of Words and Spatial pyramid, there is a significant performance increase by using the Spatial Pyramid. The accuracy increases by 4.3%. Still there are the same types of images which perform much better than other types of images. This should have the same reason as before.

A Result

Here you can see some image categories and right/wrong classified images. Images that get classified wrong, usually show some similarities with images from the classified category. One good example is the Kitchen and Store category. Since images from both categories appear very similar, they have a low right classification rate.

Figure 5: Sample images that perform good/bad of the best trained model