Homework 5 Writeup

Code Implementation and Details

Feature Extraction

```python
def feature_extraction(img, feature):
    """
    This function computes defined feature (HoG, SIFT) descriptors of
    the target image.

    :param img: a height x width x channels matrix,
    :param feature: name of image feature representation.

    :return: a number of grid points x feature_size matrix.
    """

    if feature == 'HoG':
        # HoG parameters
        win_size = (32, 32)
        block_size = (32, 32)
        block_stride = (16, 16)
        cell_size = (16, 16)
        nbins = 9
        deriv_aperture = 1
        win_sigma = 4
        histogram_norm_type = 0
        l2_hys_threshold = 2.0000000000000001e-01
        gamma_correction = 0
        nlevels = 64

        hog = cv2.HOGDescriptor(win_size, block_size, block_stride,
    cell_size, nbins, deriv_aperture, win_sigma, histogram_norm_type,
    l2_hys_threshold, gamma_correction, nlevels)
        hog_features = hog.compute(img)

        return hog_features.reshape(-1, 36)
    elif feature == 'SIFT':
        sift = cv2.SIFT_create()
        step_size = 20
        keypoints = [cv2.KeyPoint(x, y, step_size) for y in range(0,
    img.shape[0], step_size) for x in range(0, img.shape[1], step_size
    )]
        keypoints, descriptors = sift.compute(img, keypoints)


        if descriptors is None:
            return np.zeros((0, 128), dtype=np.float32)
        return descriptors
```

- **HoG Descriptor** For HoG Descriptor, I simply used cv2.HoGDescriptor function to obtain HoG feature points. Then I reshaped the output into -1*36 size, which automatically makes the row number.

- **SIFT Descriptor** For SIFT Descriptor, I implemented the regular grid feature extraction with grid size 20. First I devided the image into 20*20 size grids, and for each grid made keypoints with cv2.Keypoint function. Then for each keypoints I computed the SIFT descriptor. If there are no descriptors found, the code gives zero array for output.

Principal Component Analysis

```
def get_features_from_pca(feat_num, feature):

    vocab = np.load(f'vocab_{feature}.npy')

    vocab_mean = np.mean(vocab, axis=0)
    vocab_centered = vocab - vocab_mean

    covariance_matrix = np.cov(vocab_centered, rowvar=False)

    eigen_values, eigen_vectors = np.linalg.eigh(covariance_matrix)

    idx = np.argsort(eigen_values)[::-1]
    principal_components = eigen_vectors[:, idx[:feat_num]]

    reduced_vocab = np.dot(vocab_centered, principal_components)

    return reduced_vocab
```

- First, from the loaded vocabulary I centered it by extracting the mean.
- Then, with the centered vocabulary I calculated the covariance matrix, and performed eigenvalue decompostition to obtain eigenvalues and vectors.
- I sorted the eigenvalues from largest one to the smallest one, and then chose the eigenvectors corresponding to largest feat_num indices.
- Finally, I obtained the reduced vocabulary by making a dot product with the centered vocabulary and selected eigenvectors.

Bag of Words

```
def get_bags_of_words(image_paths, feature):
    vocab = np.load(f'vocab_{feature}.npy')

    vocab_size = vocab.shape[0]

    bags_of_words = np.zeros((len(image_paths), vocab_size))

    for i, path in enumerate(image_paths):
        img = cv2.imread(path)
        features = feature_extraction(img, feature)

```

```
        distances = pdist(features, vocab)
        closest_vocab = np.argmin(distances, axis=1)

        for idx in closest_vocab:
            bags_of_words[i, idx] += 1

        bags_of_words[i, :] /= linalg.norm(bags_of_words[i, :])

    return bags_of_words
```

- First I made an empty array with size of the number of test images and vocabulary size, to use it as histogram.
- For each test images, I extracted the feature points with pre-implemented feature_extraction function.
- Calculated the distance between the feature vectors of the test image and the vocabulary by pdist function.
- Then I sorted the distances array to get the vocabulary that is closest to the given image's feature vectors.
- If a typical vocabulary is chosen as the closest one, then I incremented the count of the histogram for that index of vocabulary.
- Finally, I applied L2 normalization for the histogram, by deviding the feature vector with the size of the vector.

Spatial Pyramid Representation

```
def get_spatial_pyramid_feats(image_paths, max_level, feature):

    vocab = np.load(f'vocab_{feature}.npy')
    vocab_size = vocab.shape[0]

    d = 0
    for l in range(max_level + 1):
        d += (vocab_size * (4 ** l))
    pyramid_features = np.zeros((len(image_paths), d))

    for i, path in enumerate(image_paths):
        img = cv2.imread(path)
        h, w = img.shape[:2]
        current_feature = []
        for l in range(max_level + 1):
            sub_h, sub_w = h // (2 ** l), w // (2 ** l)
            for y in range(2 ** l):
                for x in range(2 ** l):
                    sub_img = img[y*sub_h:(y+1)*sub_h, x*sub_w:(x+1)*
    sub_w]
                    sub_features = feature_extraction(sub_img, feature
    )
```

```
21
22                      distances = pdist(sub_features, vocab)
23                      closest_vocab_indices = np.argmin(distances, axis
    =1)
24
25                      hist = np.zeros(vocab_size)
26                      for idx in closest_vocab_indices:
27                          hist[idx] += 1
28
29                      weight = 2 ** (-max_level+l-1) if l > 0 else -
    max_level
30                      current_feature.extend(weight * hist)
31
32          normalized_feature = np.array(current_feature) / linalg.norm(
    current_feature)
33
34          pyramid_features[i, :len(normalized_feature)] =
    normalized_feature
35
36      return pyramid_features
```

- First, I initialized the feature map with the size of number of test images, and the total number of subimages, which is $4^l * vocab\_size$ for each level of pyramid.
- Iterate through the test images and the level of spatial pyramid. For each level, the images are divided into 4^l subimages.
- For each subimages, feature vectors are extracted, and the closest vocabulary is calculated, simillar to the bag of words.
- After computing the histogram, weight for spational imformation is multplied to the histogram.
- Finally, the histogram is L2 normalized, and applied to the total pyramid feature array.

Multi-Class SVM

```
1  def svm_classify(train_image_feats, train_labels, test_image_feats,
       kernel_type):
2
3      categories = np.unique(train_labels)
4
5      n_categories = len(categories)
6
7      svms = {category: svm.SVC(kernel=kernel_type.lower(), C=10) for
       category in categories}
8
9      for category in categories:
10         labels = (train_labels == category).astype(int)
11         svms[category].fit(train_image_feats, labels)
12
```

```
13      predictions = np.zeros((len(test_image_feats), n_categories))
14      for i, category in enumerate(categories):
15          predictions[:, i] = svms[category].decision_function(
        test_image_feats)
16
17      predicted_categories = categories[np.argmax(predictions, axis=1)]
18
19      return predicted_categories
```

- Number of categories to be classified is determined by extracting the unique categories from train labels.
- Using svm.SVC function, I made the svm object with the given kernel and lambda value for each categories. Since lambda value is the hyperparamter, I tired some different values, which will be explained later.
- For each of categories, I made binary label which gives positive label for the labels in the target category, and negative label for the labels not in it.
- Then, I trained the SVM model for each of categories by fitting the train image features to the train lables. This gives 15 trained SVM model for each of 15 categories.
- Test image features are used as input in the model, and the prediction probability for each classes are calcuated by SVM decision function for each class.

Analysis of the result

HoG Descriptor VS SIFT Descriptor

Descriptor	Accuracy (percent)
HoG	59.3
SIFT	60.9

Table 1: Accuracy Comparison between HoG and SIFT Descriptor

Accuracy was compared between HoG and SIFT Descriptor. Bag of Words representation was used, and fixed linear kernel for SVM with lambda=10. SIFT Descriptor showed higher accuracy for about 1.6 percent.

PCA Visualization of the Vocabulary

This is the result of PCA for vocabulary. SIFT descriptor was used to extract features. Left figure is reduction to dimension 2, and right figure is reduction to dimension 3.

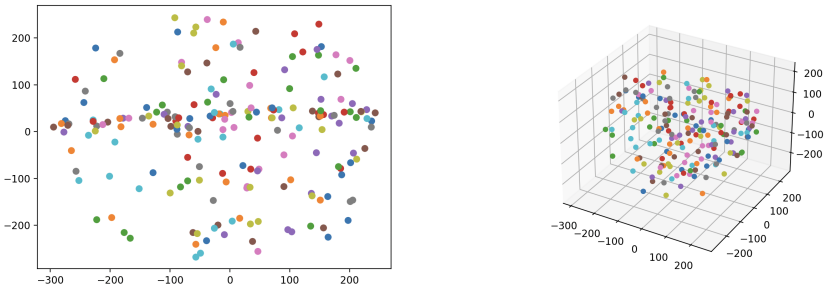

Figure 1: *Left:* Reduction to dimension 2 *Right:* Reduction to dimension 3

Linear Kernel VS RBF Kernel

Kernel Type	Accuracy (percent)
Linear	60.9
RBF	63.5

Table 2: Accuracy Comparison between Linear and RBF kernel

The train features were extracted with SIFT, and represented with bag of word without spatial pyramid. Bag of words representation was used without spatial pyramid. Usage of RBF kernel showed about 2.6 percent improvement in accuracy.

Effect of Spatial Pyramid Representation

Representation	Accuracy (percent)
Bag of Words	63.5
Bag of Words + Spatial Pyramid	64.0

Table 3: The effect of spatial pyramid representation on accuracy

Adding spatial pyramid representation on bag of words improved accuracy about 0.5 percent. SIFT descriptor was used for feature extraction, and RBF kernel was used.

Choosing Hyperparamter

Hyperparamter in this project is the lambda value of multi-class SVM. Lambda, which is written as “C” in svm function, decides the intensity of normalization in model. When C is large, the model tries to fit more on data, and when C is small, the model tries to find more generalized solution. Decision of C is important since too large or too small C could bring overfitting of underfitting.

I varied the value of C from 0.1 to 100 with other parameters fixed, since C could be found by log scaling. I chosed the C value which showed the highest accuracy.

C	Accuracy (percent)
0.1	57.3
1	63.5
10	64.0
100	62.9

Table 4: The effect of spatial pyramid representation on accuracy

I used SIFT descriptor for feature extraction, and RBF kernel. Bag of words representation was used without spatial pyramid. Since the accuracy was highest when C=10, I choosed C for 10.