Homework 3

Anthony-22421378

1. Introduction

The goal of this project is to implement scene recognition using the concept of bag of features. The idea has been derived from the concept of bag of textures which is a popular and accurate technique used in natural language processing for classification of text into different category. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. In document classification, a bag of words is a sparse vector of occurrence counts of words; that is, a sparse histogram over the vocabulary. In computer vision, a bag of visual words is a vector of occurrence counts of a vocabulary of local image features. We use three ways of representing our images using appropriate features.

2. Tiny images representation

2.1 Code implementation:

Implementing Tiny Image

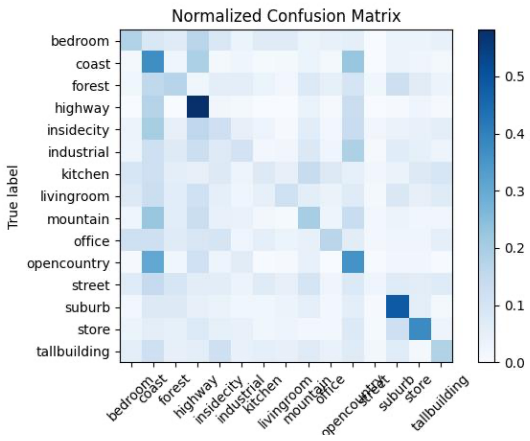
```
def tiny_image(image_path):
    # Getting the size of image in path
    n = len(image_path)
    # the size of pixel in image  for tiny image
    size = 16
    tiny_images = []
    for image in image_path:
        picture = Image.open(image)
        picture = picture.resize((size, size))
        picture = (picture - np.mean(picture)) / np.std(picture)
        picture = picture.flatten()
        tiny_images.append(picture)
    return np.asarray(tiny_images)
```

Implementing KNN algorithm:

```python
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
def knn_clasification(train_image, train_label, test_image, k=1):
    predictions = []
    for test_sample in test_image:
        distances = [
            euclidean_distance(test_sample, train_sample)
            for train_sample in train_image
        ]
        # Getting index of the nearest k neighbors
        sorted_indices = np.argsort(distances)[:k]
        # Collecting the labels of the k nearest neighbors
        nearest_labels = [train_label[i] for i in sorted_indices]
        if k == 1:
            predictions.append(nearest_labels[0])
        else:
            predictions.append(max(set(nearest_labels), key=nearest_labels.count))
    return predictions
```

Result of Experiment :

Class	Precision	Recall	F1-Score	Support
bedroom	0.17	0.18	0.17	116
coast	0.18	0.37	0.24	260
forest	0.21	0.17	0.19	228
highway	0.24	0.58	0.34	160
industrial	0.14	0.12	0.13	211
insidecity	0.17	0.11	0.13	208
kitchen	0.09	0.07	0.08	110
livingroom	0.25	0.12	0.16	189
mountain	0.25	0.20	0.22	274
office	0.21	0.17	0.19	115
opencountry	0.27	0.36	0.31	310
store	0.16	0.02	0.04	215
street	0.38	0.48	0.43	192
suburb	0.33	0.38	0.35	141
tallbuilding	0.34	0.18	0.23	256
accuracy		**0.24**		**2985**
macro avg	**0.23**	**0.23**	**0.21**	**2985**
weighted avg	**0.23**	**0.24**	**0.22**	**2985**



The result of this operation can be clearly seen in the tiny_image/visualization.md.

3. Bag of SIFT representation

3.1 Building the vocabulary of visual words

```
def vocab_build(image_paths, vocab_size, save_model_path="kmeans_model.pkl"):
    sift = cv2.SIFT_create()
    bag_of_features = []
    for image_path in image_paths:
        img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        keypoints, descriptors = sift.detectAndCompute(img, None)
        if descriptors is not None:
            bag_of_features.append(descriptors)
    bag_of_features = np.vstack(bag_of_features)
    try:
        kmeans = joblib.load(save_model_path)
```

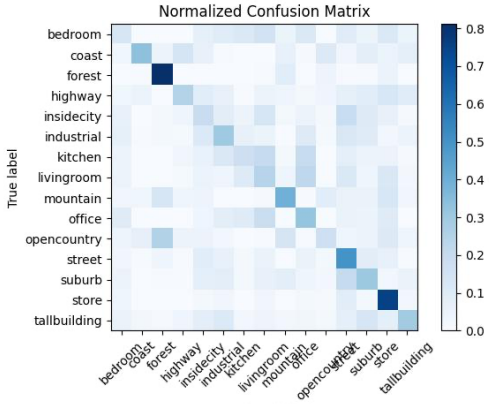
```
        print("Loaded existing k-means model")
    except FileNotFoundError:
        print("Making a new model kmean")
        kmeans = KMeans(n_clusters=vocab_size, random_state=0)
        kmeans.fit(bag_of_features)
        joblib.dump(kmeans, save_model_path)
        print(f"Saved k-means model to {save_model_path}")
    return kmeans
```

3.2 Building the vocabulary of visual words

```
def create_bag_of_sifts(image_paths, kmeans):
    f_hist = []
    for images in image_paths:
        #using SIFT algorihtm
        image = cv2.imread(images, cv2.IMREAD_GRAYSCALE)
        sift = cv2.SIFT_create()
        keypoints, descriptors = sift.detectAndCompute(image, None)
        if descriptors is not None:
            visual_words = kmeans.predict(descriptors)
            # Create histogram of visual word occurrences
            hist, _ = np.histogram(
                visual_words, bins=np.arange(kmeans.n_clusters + 1), density=True
            )
            # normalize histogram
            hist_norm = [float(i) / sum(hist) for i in hist]
            f_hist.append(hist_norm)
    image_feats = np.asarray(f_hist)
    return image_feats
```

3.3 Nearest neighbor classifier

Class	Precision	Recall	F1-Score	Support
bedroom	0.11	0.13	0.12	116
coast	0.66	0.33	0.45	260
forest	0.57	0.81	0.67	228
highway	0.32	0.25	0.28	160
industrial	0.20	0.18	0.19	211
insidecity	0.32	0.30	0.31	208
kitchen	0.19	0.17	0.18	110
livingroom	0.24	0.24	0.24	189
mountain	0.47	0.39	0.43	274
office	0.21	0.32	0.25	115
opencountry	0.38	0.17	0.24	310
store	0.33	0.50	0.39	215
street	0.25	0.31	0.27	192
suburb	0.33	0.75	0.46	141
tallbuilding	0.46	0.29	0.35	256
accuracy		**0.35**		**2985**
macro avg	**0.34**	**0.34**	**0.32**	**2985**
weighted avg	**0.37**	**0.35**	**0.34**	**2985**



The result of this operation can be clearly seen in the bag_word/visualization.md.

3.4 Comparison

Parameter (K)	Accuracy
10	22.75%
20	29.92%
30	32.16%
40	35.61%
50	34.27%
60	35.44%
70	34.74%
80	35.24%
90	34.94%
100	36.34%

Discussion

Considering that vocabulary size is one of the important factors affecting the performance in this bag-of-words scene recognition experiment, we can see that a reduced vocabulary captures fewer details and can be said to be underfitting, while an enlarged vocabulary captures more details with higher computational cost and risk of overfitting. Empirically, a proper trade-off can be established. I think it is more reasonable that with better feature recognition accuracy, better dimensionality reduction methods or better advanced classifiers such as SVM or deep learning can be optimized for this problem.