电 子 科 技 大 学

实　验　报　告

学生姓名：刘文晨　　学 号：2018080901006　　指导教师：沈复民，徐行

一、实验项目名称：

基于词袋模型的场景检测（Scene Recognition with Bag of Words）

二、实验原理：

1. 图像分类

图像分类是机器视觉中一个重要的问题，其基本概念是，使用计算机自动把图像划分到特定的概念类别中。图像分类问题可以描述为：给定若干个学习好的图像类别，对输入的新图像序列进行处理，并对其做出一个决策，判断一个已知的类别是否出现在数据中。

2. 基于词袋模型的图像分类

词袋模型（Bag-of-words model）最初用于文本分类中，然后逐步引入到了图像分类任务中。在文本分类中，文本被视为一些不考虑先后顺序的单词集合。而在图像分类中，图像被视为是一些与位置无关的局部区域的集合，因此这些图像中的局部区域就等同于文本中的单词了。在不同的图像中，局部区域的分布是不同的。因此，可以利用提取的局部区域的分布对图像进行识别。图像分类和文本分类的不同点在于，在文本分类的词袋模型算法中，字典是已存在的，不需要通过学习获得；而在图像分类中，词袋模型算法需要通过监督或非监督的学习来获得视觉词典。

词袋模型的基本思想：

1) 提取特征：根据数据集选取特征，然后进行描述，形成特征数据；
2) 学习词袋：利用处理好的特征数据全部合并，再用聚类的方法把特征词分为若干类，此若干类的数目由自己设定，每一个类相当

于一个视觉词；

3) 利用视觉词袋量化图像特征：每一张图像由很多视觉词汇组成，我们利用统计的词频直方图，可以表示图像属于哪一类。

基于词袋模型的图像分类算法一般分为四步：

1) 对图像进行局部特征向量的提取。为了取得很好的分类效果，提取的特征向量需要具备不同程度的不变性，如旋转，缩放，平移等不变性；
2) 利用上一步得到的特征向量集，抽取其中有代表性的向量，作为单词，形成视觉词典；
3) 对图像进行视觉单词的统计，一般判断图像的局部区域和某一单词的相似性是否超过某一阈值。这样即可将图像表示成单词的分布，即完成了图像的表示；
4) 设计并训练分类器，利用图像中单词的分布进行图像分类。

3. 特征描述（以 SIFT 特征为例）

尺度不变特征变换（Scale-invariant feature transform, SIFT）是用于图像处理领域的一种描述。这种描述具有尺度不变性，可在图像中检测出关键点，是一种局部特征描述子。

SIFT 算法的特点：

1) SIFT 特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性；
2) 独特性：信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配；
3) 多量性：即使少数的几个物体也可以产生大量的 SIFT 特征向量；
4) 高速性：经优化的 SIFT 匹配算法甚至可以达到实时的要求；
5) 可扩展性：可以很方便的与其他形式的特征向量进行联合。

4. SVM 分类器

支持向量机(Support Vector Machine, SVM) 的基本模型是定义在特征空间上间隔最大的线性分类器。是一种二分类模型，当采用了核

技巧后，支持向量机就可以用于非线性分类。

SVM 的主要思想可以概括为两点：

1) 它是针对线性可分情况进行分析，对于线性不可分的情况，通过使用非线性映射算法将低维输入空间线性不可分的样本转化为高维特征空间使其线性可分，从而使得高维特征空间采用线性算法对样本的非线性特征进行线性分析成为可能；
2) 它基于结构风险最小化理论之上在特征空间中建构最优分割超平面，使得学习器得到全局最优化，并且在整个样本空间的期望风险以某个概率满足一定上界。

5. K-means 聚类算法

K-means 算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

K-means 算法步骤：

1) 从 n 个数据对象任意选择 k 个对象作为初始聚类中心；
2) 根据每个聚类对象的均值（中心对象），计算每个对象与这些中心对象的距离，并根据最小距离重新对相应对象进行划分；
3) 重新计算每个（有变化）聚类的均值（中心对象）；
4) 循环 2)到 3)直到每个聚类不再发生变化为止。

三、实验目的：

构建不同的特征提取器和分类器组合，分别处理 15 种场景下的图像分类问题，进行图像的分类预测，比较准确率。

四、实验内容：

1. 了解并学习基于词袋模型的场景检测的知识和操作。
2. 利用给定的训练集与测试集图片，使用特征提取的方法提取图像的特征，并构建词袋模型与训练分类器，在测试集上，用训练得到的模型

对图片场景进行分类，计算准确率并输出结果。

五、实验步骤：

1. Tiny 特征提取

完善“student.py”文件中的 get_tiny_images()函数，实现功能包括读取图像、裁剪到指定大小、转换成灰度图、缩小图像、像素归一化等。经过修改完善后的代码如下：

```
def get_tiny_images(image_paths):
    '''
    This feature is inspired by the simple tiny images used as features in
    80 million tiny images: a large dataset for non-parametric object and
    scene recognition. A. Torralba, R. Fergus, W. T. Freeman. IEEE
    Transactions on Pattern Analysis and Machine Intelligence, vol.30(11),
    pp. 1958-1970, 2008. http://groups.csail.mit.edu/vision/TinyImages/
    Inputs:
        image_paths: a 1-D Python list of strings. Each string is a complete
                     path to an image on the filesystem.
    Outputs:
        An n x d numpy array where n is the number of images and d is the
        length of the tiny image representation vector. e.g. if the images
        are resized to 16x16, then d is 16 * 16 = 256.
    To build a tiny image feature, resize the original image to a very small
    square resolution (e.g. 16x16). You can either resize the images to square
    while ignoring their aspect ratio, or you can crop the images into squares
    first and then resize evenly. Normalizing these tiny images will increase
    performance modestly.
    As you may recall from class, naively downsizing an image can cause
    aliasing artifacts that may throw off your comparisons. See the docs for
    skimage.transform.resize for details:
    http://scikit-image.org/docs/dev/api/skimage.transform.html#skimage.transform.resize
    Suggested functions: skimage.transform.resize, skimage.color.rgb2grey,
                         skimage.io.imread, np.reshape
```

```
    '''

    #TODO: Implement this function!
    image_feats = []
    size = 16
    for image_path in tqdm(image_paths, desc="Image-tiny"):
        image = cv2.imread(image_path)
        image = cv2.resize(image, (size, size))
        # column vector
        image_feat = np.resize(image, [size * size])
        image_feat = image_feat.tolist()
        # Normalizing
        mean = np.mean(image_feat)
        image_feat = [(value - mean) for value in image_feat]
        image_feats.append(image_feat)

    return np.array(image_feats)
```

2. SIFT 特征提取与词袋构建

完善“student.py”文件中的 build_vocabulary()函数，遍历训练集的所有图片，提取每张图片的 128 维的 SIFT 特征，保存训练集所有图片的 SIFT 特征，进行聚类，最后返回簇中心。经过修改完善后的代码如下：

```
def build_vocabulary(image_paths, vocab_size):
    '''''
    This function should sample HOG descriptors from the training images,
    cluster them with kmeans, and then return the cluster centers.
    Inputs:
        image_paths: a Python list of image path strings
         vocab_size: an integer indicating the number of words desired for t
he
                     bag of words vocab set
    Outputs:
        a vocab_size x (z*z*9) (see below) array which contains the cluster
        centers that result from the K Means clustering.
    You'll need to generate HOG features using the skimage.feature.hog() fun
ction.
    The documentation is available here:
    http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.featur
e.hog
```

```
    However, the documentation is a bit confusing, so we will highlight some
    important arguments to consider:
        cells_per_block: The hog function breaks the image into evenly-sized
            blocks, which are further broken down into cells, each made of
            pixels_per_cell pixels (see below). Setting this parameter tells the
            function how many cells to include in each block. This is a tuple of
            width and height. Your SIFT implementation, which had a total of
            16 cells, was equivalent to setting this argument to (4,4).
        pixels_per_cell: This controls the width and height of each cell
            (in pixels). Like cells_per_block, it is a tuple. In your SIFT
            implementation, each cell was 4 pixels by 4 pixels, so (4,4).
        feature_vector: This argument is a boolean which tells the function
            what shape it should use for the return array. When set to True,
            it returns one long array. We recommend setting it to True and
            reshaping the result rather than working with the default value,
            as it is very confusing.
    It is up to you to choose your cells per block and pixels per cell. Choose
    values that generate reasonably-sized feature vectors and produce good
    classification results. For each cell, HOG produces a histogram (feature
    vector) of length 9. We want one feature vector per block. To do this we
    can append the histograms for each cell together. Let's say you set
    cells_per_block = (z,z). This means that the length of your feature vector
    for the block will be z*z*9.
    With feature_vector=True, hog() will return one long np array containing every
    cell histogram concatenated end to end. We want to break this up into a
    list of (z*z*9) block feature vectors. We can do this using a really nifty numpy
    function. When using np.reshape, you can set the length of one dimension to
    -1, which tells numpy to make this dimension as big as it needs to be to
```

```
    accomodate to reshape all of the data based on the other dimensions. So if
    we want to break our long np array (long_boi) into rows of z*z*9 feature
    vectors we can use small_bois = long_boi.reshape(-1, z*z*9).
    The number of feature vectors that come from this reshape is dependent on
    the size of the image you give to hog(). It will fit as many blocks as it
    can on the image. You can choose to resize (or crop) each image to a consistent size
    (therefore creating the same number of feature vectors per image), or you
    can find feature vectors in the original sized image.
    ONE MORE THING
    If we returned all the features we found as our vocabulary, we would have an
    absolutely massive vocabulary. That would make matching inefficient AND
    inaccurate! So we use K Means clustering to find a much smaller (vocab_size)
    number of representative points. We recommend using sklearn.cluster.KMeans
    to do this. Note that this can take a VERY LONG TIME to complete (upwards
    of ten minutes for large numbers of features and large max_iter), so set
    the max_iter argument to something low (we used 100) and be patient. You
    may also find success setting the "tol" argument (see documentation for
    details)
    '''

    #TODO: Implement this function!
    # cluster_SIFT_features = []
    # sift = cv2.xfeatures2d.SIFT_create()
    # for image_path in tqdm(image_paths, desc="Imaging-SIFT"):
    #     image = cv2.imread(image_path)
    #     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #     locations, SIFT_features = sift.detectAndCompute(gray, None)
    #     temp = SIFT_features.tolist()
    #     cluster_SIFT_features += temp
    # cluster_SIFT_features = random.sample(cluster_SIFT_features, 400 * 3)
```

```
    # kmeans = KMeans(n_clusters=vocab_size, max_iter=100).fit(cluster_SIFT_features)
    # cluster_centers = kmeans.cluster_centers_
    # return np.array(cluster_centers)


    bag_of_features = []

    print("Extract SIFT features")
    #pdb.set_trace()
    for path in tqdm(image_paths,desc='build_vocabulary'):
        img = np.asarray(Image.open(path),dtype='float32')
        frames, descriptors = dsift(img, step=[5,5], fast=True)
        bag_of_features.append(descriptors)
    bag_of_features = np.concatenate(bag_of_features, axis=0).astype('float32')
    #pdb.set_trace()
    print("Compute vocab")
    start_time = time()
    vocab = kmeans(bag_of_features, vocab_size, initialization="PLUSPLUS")

    end_time = time()
    print("It takes ", (start_time - end_time), " to compute vocab.")

    return vocab
```

3. 根据词袋获取测试图片的直方特征表示

完善“student.py”文件中的 get_bags_of_words ()函数，对图片进行遍历，获取图片的所有 SIFT 特征，并分别计算其与簇中心的距离，选取最短距离的 bag 索引。使用直方图统计这些 SIFT 特征落在不同 bag 的次数，最后将直方图频数归一化，也就是计算直方图的频率，将其作为图片的新的特征并返回。经过修改完善后的代码如下：

```
def get_bags_of_words(image_paths):
    '''''
    This function should take in a list of image paths and calculate a bag of
    words histogram for each image, then return those histograms in an array.
    Inputs:
        image_paths: A Python list of strings, where each string is a complete
```

```
                        path to one image on the disk.
    Outputs:
        An nxd numpy matrix, where n is the number of images in image_paths
and
        d is size of the histogram built for each image.
    Use the same hog function to extract feature vectors as before (see
    build_vocabulary). It is important that you use the same hog settings fo
r
    both build_vocabulary and get_bags_of_words! Otherwise, you will end up
    with different feature representations between your vocab and your test
    images, and you won't be able to match anything at all!
    After getting the feature vectors for an image, you will build up a
    histogram that represents what words are contained within the image.
    For each feature, find the closest vocab word, then add 1 to the histogr
am
    at the index of that word. For example, if the closest vector in the voc
ab
    is the 103rd word, then you should add 1 to the 103rd histogram bin. You
r
    histogram should have as many bins as there are vocabulary words.
    Suggested functions: scipy.spatial.distance.cdist, np.argsort,
                         np.linalg.norm, skimage.feature.hog
    '''

    # vocab = np.load('vocab.npy')
    # print('Loaded vocab from file.')

    # #TODO: Implement this function!
    # vocab_size = len(image_paths)
    # tree = KDTree(vocab)
    # cluster_SIFT_features = []
    # sift = cv2.xfeatures2d.SIFT_create()
    # for image_path in tqdm(image_paths, desc='SIFT'):
    #     image_bag = [0] * vocab_size
    #     image = cv2.imread(image_path)
    #     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #     locations, SIFT_features = sift.detectAndCompute(gray, None)
    #     temp = SIFT_features.tolist()
    #     nearest_dist, nearest_ind = tree.query(temp, k=1)
    #     for index in nearest_ind:
    #         image_bag[int(index)] += 1
    #     cluster_SIFT_features.append(image_bag)
    # return np.array(cluster_SIFT_features)

```

```
46.
47.     with open('vocab.pkl', 'rb') as v:
48.         vocab = pickle.load(v)
49.         image_feats = np.zeros((len(image_paths),len(vocab)))
50.
51.     for i, path in tqdm(enumerate(image_paths), desc='get_bags_of_words'):
52.
53.         image = np.asarray(Image.open(path), dtype = 'float32')
54.         frames, descriptors = dsift(image, step=[9,9], fast=True)
55.
56.         dist = distance.cdist(vocab, descriptors, 'euclidean')
57.         mdist = np.argmin(dist, axis = 0)
58.         histo, bins = np.histogram(mdist, range(len(vocab)+1))
59.         if np.linalg.norm(histo) == 0:
60.             image_feats[i, :] = histo
61.         else:
62.             image_feats[i, :] = histo / np.linalg.norm(histo)
63.
64.     return image_feats
```

4. 分类器构建

1) SVM 分类器

完善“student.py”文件中的 svm_classify()函数，训练 SVM 分类器并用训练得到的分类器预测结果。经过修改完善后的代码如下：

```
1.  def svm_classify(train_image_feats, train_labels, test_image_feats):
2.      '''''
3.      This function will predict a category for every test image by training
4.      15 many-versus-one linear SVM classifiers on the training data, then
5.      using those learned classifiers on the testing data.
6.      Inputs:
7.          train_image_feats: An nxd numpy array, where n is the number of trai
    ning
8.                             examples, and d is the image descriptor vector si
    ze.
9.          train_labels: An nx1 Python list containing the corresponding ground

10.                         truth labels for the training data.
11.         test_image_feats: An mxd numpy array, where m is the number of test
12.                             images and d is the image descriptor vector size.
13.     Outputs:
```

```
        An mx1 numpy array of strings, where each string is the predicted label
        for the corresponding image in test_image_feats
    We suggest you look at the sklearn.svm module, including the LinearSVC
    class. With the right arguments, you can get a 15-class SVM as described

    above in just one call! Be sure to read the documentation carefully.
    '''

    # TODO: Implement this function!
    clf = svm.SVC(C=100, gamma='scale', decision_function_shape="ovr")
    clf.fit(train_image_feats, train_labels)
    predicted_categories = clf.predict(test_image_feats)

    return np.array(predicted_categories)

    # svc = svm.SVC(random_state=0)
    # param_C = [0.001 , 0.01 , 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
    # param_gamma = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0]
    # param_grid = [{'C': param_C,
    #                'gamma': param_gamma,
    #                'kernel': ['rbf']}]

    # gs = GridSearchCV(estimator = svc,
    #                   param_grid= param_grid,
    #                   scoring='accuracy',
    #                  )

    # gs = gs.fit(train_image_feats, train_labels)

    # print(f'Best Training Score = {gs.best_score_:.3f} with parameters {gs.best_params_}')

    # classifier = gs.best_estimator_
    # classifier.fit(train_image_feats, train_labels)


    # pred_label = classifier.predict(test_image_feats)
    # return  np.array(pred_label)
```

2) KNN 分类器

完善“student.py”文件中的nearest_neighbor_classify()函数，计算测试集每个图片样本与训练集每个图片样本的距离，

取距离最短的k个图片样本的类别的众数作为训练集图片样本的类别。经过修改完善后的代码如下：

```
def nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats):
    '''''
    This function will predict the category for every test image by finding
    the training image with most similar features. You will complete the given
    partial implementation of k-nearest-neighbors such that for any arbitrary
    k, your algorithm finds the closest k neighbors and then votes among them
    to find the most common category and returns that as its prediction.
    Inputs:
        train_image_feats: An nxd numpy array, where n is the number of training
                           examples, and d is the image descriptor vector size.
        train_labels: An nx1 Python list containing the corresponding ground
                      truth labels for the training data.
        test_image_feats: An mxd numpy array, where m is the number of test
                          images and d is the image descriptor vector size.
    Outputs:
        An mx1 numpy list of strings, where each string is the predicted label
        for the corresponding image in test_image_feats
    The simplest implementation of k-nearest-neighbors gives an even vote to
    all k neighbors found - that is, each neighbor in category A counts as one
    vote for category A, and the result returned is equivalent to finding the
    mode of the categories of the k nearest neighbors. A more advanced version
    uses weighted votes where closer matches matter more strongly than far ones.
    This is not required, but may increase performance.
    Be aware that increasing k does not always improve performance - even
    values of k may require tie-breaking which could cause the classifier to
    arbitrarily pick the wrong class in the case of an even split in votes.
    Additionally, past a certain threshold the classifier is considering so
```

```
    many neighbors that it may expand beyond the local area of logical matches
    and get so many garbage votes from a different category that it mislabels
    the data. Play around with a few values and see what changes.
    Useful functions:
        scipy.spatial.distance.cdist, np.argsort, scipy.stats.mode
    '''


    # Gets the distance between each test image feature and each train image feature
    # e.g., cdist

    #TODO:
    # 1) Find the k closest features to each test image feature in euclidean space
    # 2) Determine the labels of those k features
    # 3) Pick the most common label from the k
    # 4) Store that label in a list

    k = 7
    distances = cdist(test_image_feats, train_image_feats, 'euclidean')
    predicted_categories = []
    for i in range(len(distances)):
        min_distance_index = distances[i].argsort()
        k_neighbor = min_distance_index[:k]
        labels = [train_labels[x] for x in k_neighbor]
        label = max(labels, key=labels.count)
        predicted_categories.append(label)

    return np.array(predicted_categories)

    # use sklearn
    # classifier = KNeighborsClassifier(n_neighbors=k, metric="manhattan")
    # classifier.fit(train_image_feats, train_labels)
    # predicted_categories = classifier.predict(test_image_feats)
    # return np.array([predicted_categories])
```

5. main 函数

用于保存程序执行结果，代码如下：

```
#!/usr/bin/python
import numpy as np
```

```
3.  import os
4.  import pickle
5.
6.  from helpers import get_image_paths
7.  from student import get_tiny_images, build_vocabulary, get_bags_of_words, \

8.      svm_classify, nearest_neighbor_classify
9.  from create_results_webpage import create_results_webpage
10.
11. def projSceneRecBoW():
12.     '''''
13.     For this project, you will need to report performance for three
14.     combinations of features / classifiers. We recommend that you code them
    in
15.     this order:
16.         1) Tiny image features and nearest neighbor classifier
17.         2) Bag of word features and nearest neighbor classifier
18.         3) Bag of word features and linear SVM classifier
19.     The starter code is initialized to 'placeholder' just so that the starte
    r
20.     code does not crash when run unmodified and you can get a preview of how

21.     results are presented.
22.
23.     Interpreting your performance with 100 training examples per category:
24.      accuracy  =   0 -> Something is broken.
25.      accuracy ~= .07 -> Your performance is equal to chance.
26.                         Something is broken or you ran the starter code unch
    anged.
27.      accuracy ~= .20 -> Rough performance with tiny images and nearest
28.                         neighbor classifier. Performance goes up a few
29.                         percentage points with K-NN instead of 1-NN.
30.      accuracy ~= .20 -> Rough performance with tiny images and linear SVM
31.                         classifier. Although the accuracy is about the same
    as
32.                         nearest neighbor, the confusion matrix is very diffe
    rent.
33.      accuracy ~= .40 -> Rough performance with bag of word and nearest
34.                         neighbor classifier. Can reach .60 with K-NN and
35.                         different distance metrics.
36.      accuracy ~= .50 -> You've gotten things roughly correct with bag of
37.                         word and a linear SVM classifier.
38.      accuracy >= .70 -> You've also tuned your parameters well. E.g. number
39.                         of clusters, SVM regularization, number of patches
```

```
                        sampled when building vocabulary, size and step for
                        dense features.
    accuracy >= .80 -> You've added in spatial information somehow or you've
                        added additional, complementary image features. This
                        represents state of the art in Lazebnik et al 2006.
    accuracy >= .85 -> You've done extremely well. This is the state of the
                        art in the 2010 SUN database paper from fusing many
                        features. Don't trust this number unless you actually
                        measure many random splits.
    accuracy >= .90 -> You used modern deep features trained on much larger
                        image databases.
    accuracy >= .96 -> You can beat a human at this task. This isn't a
                        realistic number. Some accuracy calculation is broken
                        or your classifier is cheating and seeing the test
                        labels.
    '''

    # Step 0: Set up parameters, category list, and image paths.
    # Uncomment various feature and classifier combinations to test them.

    # FEATURE = 'tiny image'
    FEATURE = 'bag of words'
    # FEATURE = 'placeholder'

    CLASSIFIER = 'nearest neighbor'
    # CLASSIFIER = 'support vector machine'
    # CLASSIFIER = 'placeholder'

    # This is the path the script will look at to load images from.
    data_path = '../data/'

    # This is the list of categories / directories to use. The categories are
    # somewhat sorted by similarity so that the confusion matrix looks more
    # structured (indoor and then urban and then rural).
    categories = ['Kitchen', 'Store', 'Bedroom', 'LivingRoom', 'Office',
           'Industrial', 'Suburb', 'InsideCity', 'TallBuilding', 'Street',
```

```
            'Highway', 'OpenCountry', 'Coast', 'Mountain', 'Forest']

    # This list of shortened category names is used later for visualization.
    abbr_categories = ['Kit', 'Sto', 'Bed', 'Liv', 'Off', 'Ind', 'Sub',
        'Cty', 'Bld', 'St', 'HW', 'OC', 'Cst', 'Mnt', 'For']

    # Number of training examples per category to use. Max is 100. For
    # simplicity, we assume this is the number of test cases per category as
    # well.
    num_train_per_cat = 100

    # This function returns string arrays containing the file path for each train
    # and test image, as well as string arrays with the label of each train and
    # test image. By default all four of these arrays will be 1500x1 where each
    # entry is a string.
    print('Getting paths and labels for all train and test data.')
    train_image_paths, test_image_paths, train_labels, test_labels = \
        get_image_paths(data_path, categories, num_train_per_cat)
    #   train_image_paths  1500x1   list
    #   test_image_paths   1500x1   list
    #   train_labels       1500x1   list
    #   test_labels        1500x1   list

    ############################################################################
    ## Step 1: Represent each image with the appropriate feature
    # Each function to construct features should return an N x d matrix, where
    # N is the number of paths passed to the function and d is the
    # dimensionality of each image representation. See the starter code for
    # each function for more details.
    ############################################################################

    print('Using %s representation for images.' % FEATURE)

    if FEATURE.lower() == 'tiny image':
        print('Loading tiny images...')
```

```
111.         # YOU CODE get_tiny_images (see student.py)
112.         train_image_feats = get_tiny_images(train_image_paths)
113.         test_image_feats  = get_tiny_images(test_image_paths)
114.         print('Tiny images loaded.')
115.
116.     elif FEATURE.lower() == 'bag of words':
117.         # Because building the vocabulary takes a long time, we save the ge
    nerated
118.         # vocab to a file and re-load it each time to make testing faster.
    If
119.         # you need to re-generate the vocab (for example if you change its
    size
120.         # or the length of your feature vectors), simply delete the vocab.n
    py
121.         # file and re-run main.py
122.         # if not os.path.isfile('vocab.npy'):
123.         #     print('No existing visual word vocabulary found. Computing on
    e from training images.')
124.
125.         #     #Larger values will work better (to a point), but are slower
    to compute
126.         #     vocab_size = 400
127.
128.         #     # YOU CODE build_vocabulary (see student.py)
129.         #     vocab = build_vocabulary(train_image_paths, vocab_size)
130.         #     np.save('vocab.npy', vocab)
131.
132.         # YOU CODE get_bags_of_words.m (see student.py)
133.         # train_image_feats = get_bags_of_words(train_image_paths)
134.         # You may want to write out train_image_features here as a *.npy an
    d
135.         # load it up later if you want to just test your classifiers withou
    t
136.         # re-computing features
137.
138.         # test_image_feats  = get_bags_of_words(test_image_paths)
139.         # Same goes here for test image features.
140.
141.         if os.path.isfile('vocab.pkl') is False:
142.             print('No existing visual word vocabulary found. Computing one
    from training images\n')
143.             vocab_size = 400   ### Vocab_size is up to you. Larger values w
    ill work better (to a point) but be slower to comput.
144.             vocab = build_vocabulary(train_image_paths, vocab_size)
```

```
145.             with open('vocab.pkl', 'wb') as handle:
146.                 pickle.dump(vocab, handle, protocol=pickle.HIGHEST_PROTOCOL
    )
147.
148.         if os.path.isfile('train_image_feats.pkl') is False:
149.             # YOU CODE get_bags_of_sifts.py
150.             train_image_feats = get_bags_of_words(train_image_paths);
151.             with open('train_image_feats.pkl', 'wb') as handle:
152.                 pickle.dump(train_image_feats, handle, protocol=pickle.HIGH
    EST_PROTOCOL)
153.         else:
154.             with open('train_image_feats.pkl', 'rb') as handle:
155.                 train_image_feats = pickle.load(handle)
156.
157.         if os.path.isfile('test_image_feats.pkl') is False:
158.             test_image_feats  = get_bags_of_words(test_image_paths);
159.             with open('test_image_feats.pkl', 'wb') as handle:
160.                 pickle.dump(test_image_feats, handle, protocol=pickle.HIGHE
    ST_PROTOCOL)
161.         else:
162.             with open('test_image_feats.pkl', 'rb') as handle:
163.                 test_image_feats = pickle.load(handle)
164.
165.
166.     elif FEATURE.lower() == 'placeholder':
167.         train_image_feats = []
168.         test_image_feats = []
169.
170.     else:
171.         raise ValueError('Unknown feature type!')
172.
173.     ##########################################################################
    #####
174.     ## Step 2: Classify each test image by training and using the appropria
    te classifier
175.     # Each function to classify test features will return an N x 1 string a
    rray,
176.     # where N is the number of test cases and each entry is a string indica
    ting
177.     # the predicted category for each test image. Each entry in
178.     # 'predicted_categories' must be one of the 15 strings in 'categories',

179.     # 'train_labels', and 'test_labels'. See the starter code for each func
    tion
```

```
    # for more details.
    ##############################################################################
    #####

    print('Using %s classifier to predict test set categories.' % CLASSIFIE
    R)

    if CLASSIFIER.lower() == 'nearest neighbor':
        # YOU CODE nearest_neighbor_classify (see student.py)
        predicted_categories = nearest_neighbor_classify(train_image_feats,
     train_labels, test_image_feats)

    elif CLASSIFIER.lower() == 'support vector machine':
        # YOU CODE svm_classify (see student.py)
        predicted_categories = svm_classify(train_image_feats, train_labels
    , test_image_feats)

    elif CLASSIFIER.lower() == 'placeholder':
        #The placeholder classifier simply predicts a random category for e
    very test case
        random_permutation = np.random.permutation(len(test_labels))
        predicted_categories = [test_labels[i] for i in random_permutation]


    else:
        raise ValueError('Unknown classifier type')

    ##############################################################################
    #####
    ## Step 3: Build a confusion matrix and score the recognition system
    # You do not need to code anything in this section.

    # If we wanted to evaluate our recognition method properly we would tra
    in
    # and test on many random splits of the data. You are not required to d
    o so
    # for this project.

    # This function will recreate results_webpage/index.html and various im
    age
    # thumbnails each time it is called. View the webpage to help interpret

    # your classifier performance. Where is it making mistakes? Are the
    # confusions reasonable?
```

```
    ##########################################################################
#####

    create_results_webpage( train_image_paths, \
                            test_image_paths, \
                            train_labels, \
                            test_labels, \
                            categories, \
                            abbr_categories, \
                            predicted_categories)

if __name__ == '__main__':
    projSceneRecBoW()
```

六、实验数据及结果分析：

图像特征提取器和分类器的组合有五种，故结果有五个文件夹，具体为：

1. **placeholderplaceholder_results_webpage**

   无特征提取器，无分类器，准确率为 6.9%。

图 1 无特征提取器，无分类器结果图

2. **tiny imagenearest neighbor_results_webpage**

   特征提取器为 Tiny，分类器为 KNN，准确率为 18.9%。

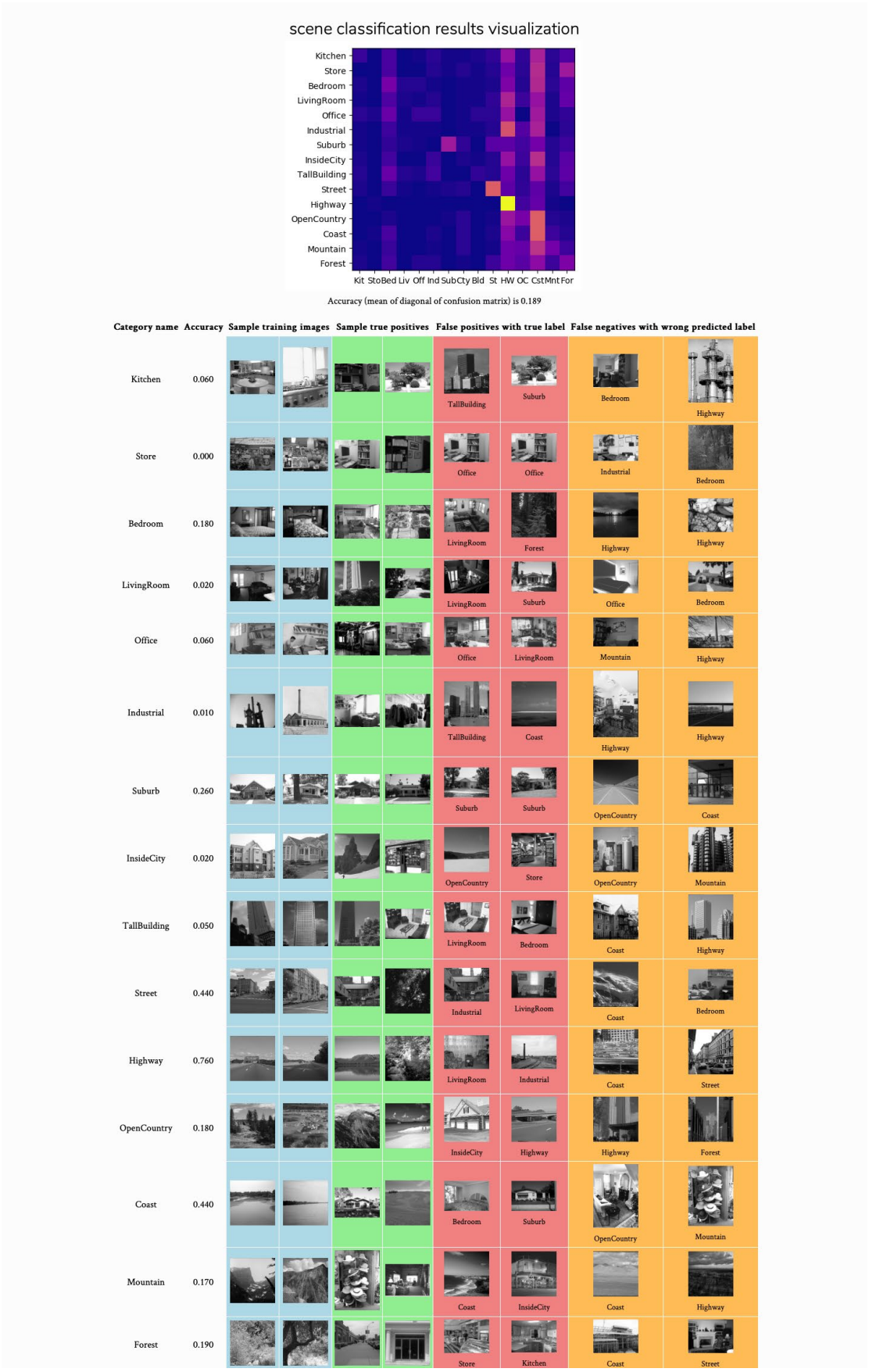
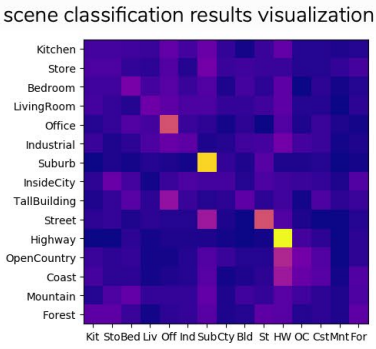


图 2 Tiny 特征提取器，KNN 分类器结果图

3. tiny imagesupport vector machine_results_webpage

特征提取器为 Tiny，分类器为 SVM，准确率为 22.3%。



Accuracy (mean of diagonal of confusion matrix) is 0.223

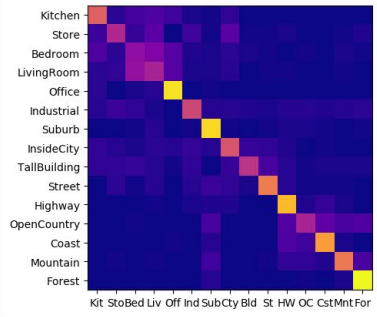
Category name	Accuracy	Sample training images	Sample true positives	False positives with true label	False negatives with wrong predicted label
Kitchen	0.080			Mountain, Street	Store, Forest
Store	0.090			Mountain, Kitchen	Forest, Office
Bedroom	0.170			Kitchen, TallBuilding	Kitchen, Office
LivingRoom	0.150			Bedroom, Mountain	Forest, Coast
Office	0.380			Bedroom, Bedroom	Highway, Industrial
Industrial	0.120			Street, InsideCity	Highway, TallBuilding
Suburb	0.650			Mountain, Kitchen	Coast, LivingRoom
InsideCity	0.080			Coast, Industrial	Coast, Store
TallBuilding	0.120			Store, LivingRoom	Industrial, Industrial
Street	0.380			Suburb, InsideCity	Suburb, Suburb
Highway	0.710			OpenCountry, Kitchen	OpenCountry, OpenCountry
OpenCountry	0.160			Mountain, Highway	Suburb, Highway
Coast	0.110			Industrial, Kitchen	OpenCountry, Office
Mountain	0.020			Office, Bedroom	Kitchen, Highway
Forest	0.120			Street, Mountain	Bedroom, Bedroom

图 3 Tiny 特征提取器，SVM 分类器结果图

4. bag of wordsnearest neighbor_results_webpage

特征提取器为 bag of words，分类器为 KNN，准确率为 58.1%。

scene classification results visualization



Accuracy (mean of diagonal of confusion matrix) is 0.581

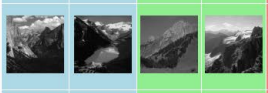
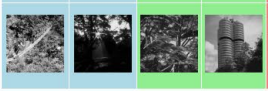
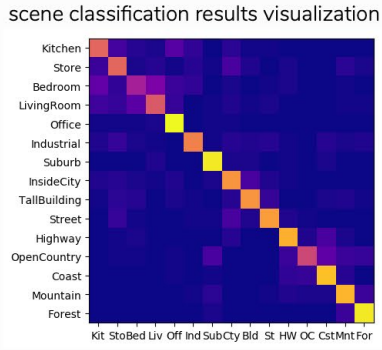
Category name	Accuracy	Sample training images	Sample true positives	False positives with true label	False negatives with wrong predicted label
Kitchen	0.550			Kitchen	Office
Store	0.370			Store	OpenCountry
Bedroom	0.280			LivingRoom	Industrial
LivingRoom	0.350			LivingRoom	Coast
Office	0.880			Kitchen	OpenCountry
Industrial	0.470			Store	Suburb
Suburb	0.860			Suburb	Highway
InsideCity	0.510			InsideCity	Highway
TallBuilding	0.410			Industrial	LivingRoom
Street	0.630			TallBuilding	OpenCountry
Highway	0.790			Street	OpenCountry
OpenCountry	0.350			OpenCountry	Industrial
Coast	0.720			Coast	InsideCity
Mountain	0.620			TallBuilding	Highway
Forest	0.930			Forest	Office

图 4 bag of words 特征提取器，KNN 分类器结果图

5. bag of wordssupport vector machine_results_webpage

特征提取器为 bag of words，分类器为 SVM，准确率为 68.9%。



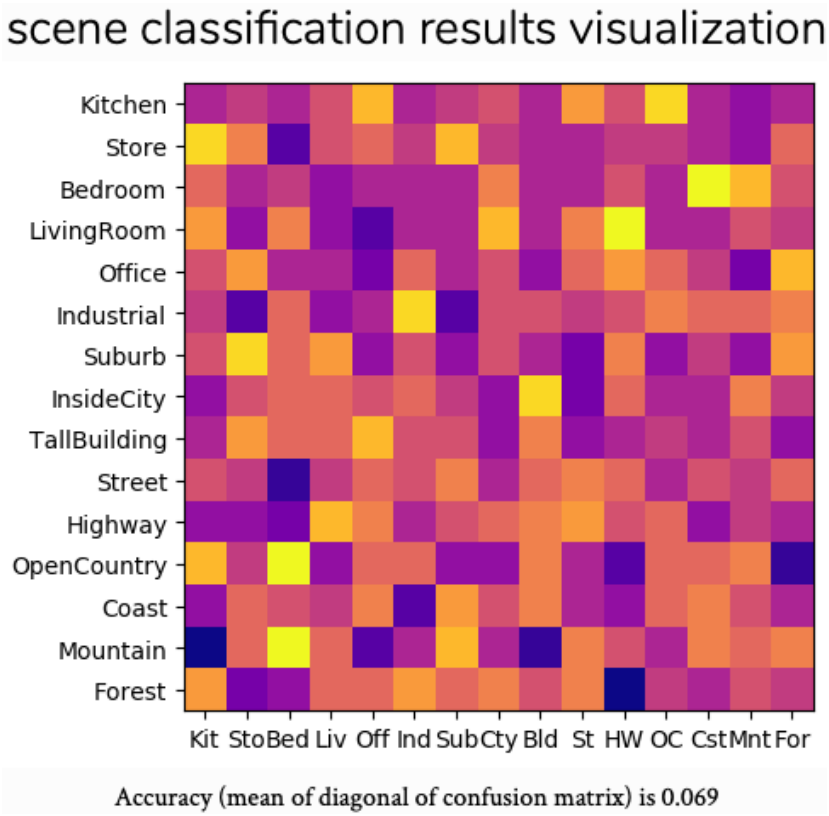
Accuracy (mean of diagonal of confusion matrix) is 0.689

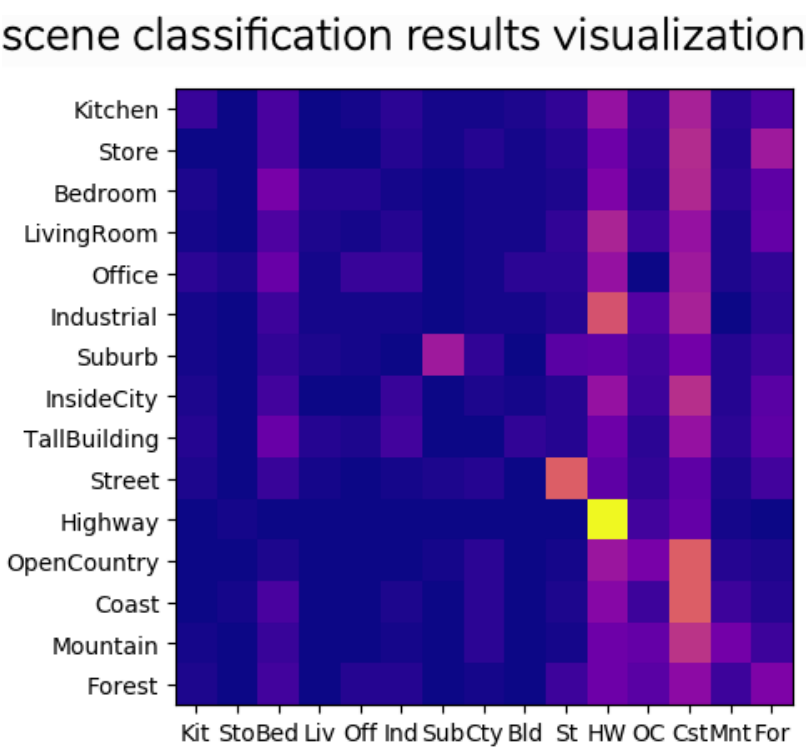
Category name	Accuracy	Sample training images	Sample true positives	False positives with true label	False negatives with wrong predicted label
Kitchen	0.570			Bedroom, Bedroom	Store, Office
Store	0.570			TallBuilding, InsideCity	Industrial, Industrial
Bedroom	0.340			LivingRoom, LivingRoom	LivingRoom, Office
LivingRoom	0.530			Bedroom, Office	Bedroom, Kitchen
Office	0.930			Kitchen, Kitchen	Bedroom, InsideCity
Industrial	0.650			Bedroom, Store	Mountain, Coast
Suburb	0.900			OpenCountry, OpenCountry	LivingRoom, Coast
InsideCity	0.700			Street, Street	TallBuilding, Kitchen
TallBuilding	0.700			InsideCity, Street	Forest, Bedroom
Street	0.720			Bedroom, Industrial	Highway, InsideCity
Highway	0.770			Mountain, OpenCountry	InsideCity, Coast
OpenCountry	0.470			Coast, Coast	Highway, Highway
Coast	0.810			Highway, OpenCountry	Suburb, OpenCountry
Mountain	0.780			OpenCountry, Forest	Coast, Highway
Forest	0.900			Industrial, Store	Suburb, Mountain

图 5 bag of words 特征提取器，SVM 分类器结果图

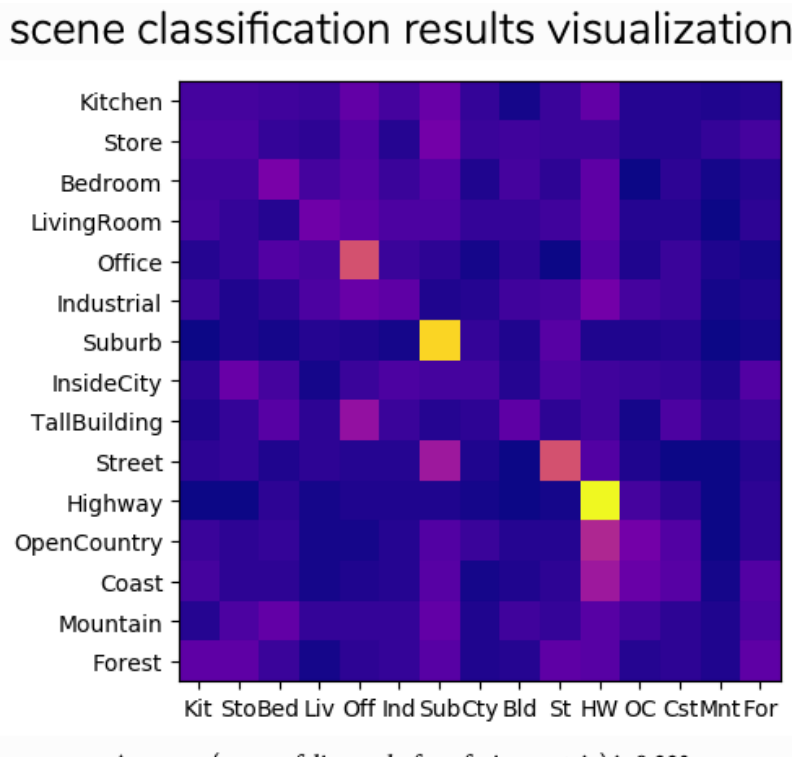
七、实验结论：

五种组合下的混淆矩阵如图 6 所示（从上到下依次为随机生成的、Tiny 特征和 KNN 分类器、Tiny 特征和 SVM 分类器、bag of words 和 KNN 分类器以及 bag of words 和 SVM 分类器）：

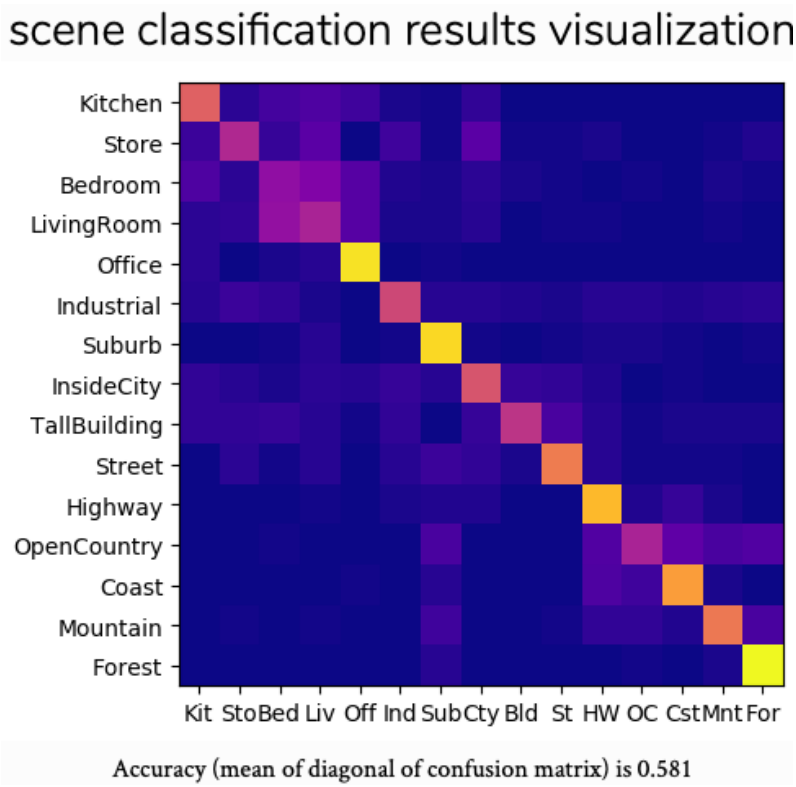




Accuracy (mean of diagonal of confusion matrix) is 0.189



Accuracy (mean of diagonal of confusion matrix) is 0.223



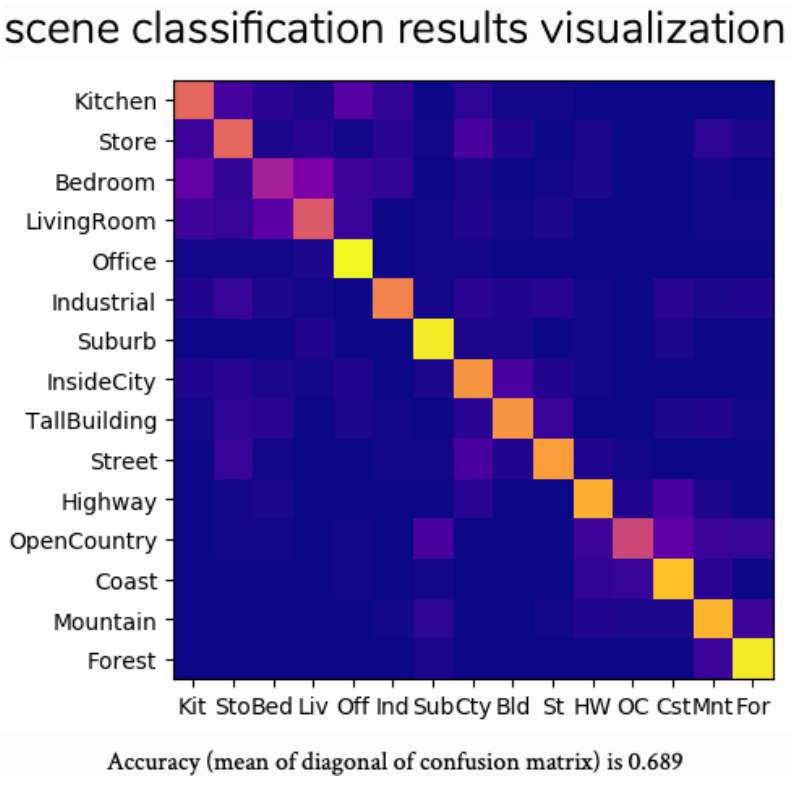


图 6 五种情况下的混淆矩阵

观察五种情况下的混淆矩阵的对角线可知，从上到下的混淆矩阵的对角线越来越明显，意味着分类效果越来越好，这一点从准确率越来越高也得到了佐证。对于每一种组合生成的混淆矩阵而言，其小格颜色越亮代表对应的类别得到了很

好的分类，越暗代表该类别没有得到很好的分类。

八、总结及心得体会：

本实验基本实现了图像特征和分类器构建的各种组合，用来处理 15 种场景下的图像分类问题，取得了较好的实验效果。

首先对原始图像进行了读取和预处理，接着进行对图像特征分别进行了微小化处理和词袋模型构建，最后分别在 KNN 分类器和 SVM 分类器进行了图像的分类预测。从分类结果可以看出，分类效果基本属于较为正常的范围。

九、对本实验过程及方法的改进建议：

提供实验指导书。

报告评分：

指导教师签字：