电 子 科 技 大 学

实 验 报 告

学生姓名：李逢君	学 号：2016060601010

一、实验室名称：立人楼 B105

二、实验项目名称：使用词带模型进行场景识别

三、实验原理：

先从非常简单的方法（微小图像和最近邻分类）开始检查场景识别的任务，然后转向更高级的方法 - 量子化局部特征和支持向量机学习的线性分类器。

词袋模型是受自然语言处理中使用的模型启发的用于图像分类的流行技术。该模型忽略或淡化单词排列（图像中的空间信息），并基于视觉单词频率的直方图进行分类。视觉词“词汇”是通过聚类大量局部特征来建立的。

四、实验目的：

图像识别。特征提取+分类器构建和使用

五、实验内容：

实现 2 种不同的图像表示：

- 微小图像和 SIFT 特征包

以及 2 种不同的分类技术：

- 最近邻和线性 SVM。

在撰写中，特别要求您报告以下组合的性能，并且强烈建议您按以下顺序实现它们：

- 微小的图像表示和最近邻分类器（精度约为 18-25%）。
- 基于 SIFT 特征的词带模型的表示和最近邻分类器（精度约为 50-60%）。
- 基于 SIFT 特征的词带模型的和线性 SVM 分类器（精度约为 60-70%）。

从实现微小图像表示和最近邻分类器开始。它们易于理解，易于实施，并且可以非常快速地运行我们的实验。

微小图像

受到 Torralba，Fergus 和 Freeman 同名作品启发的“微小图像”功能是最简单的图像表示之一。只需将每个图像调整为一个小的固定分辨率（我们建议 16x16）。如果使微小图像的平均值和单位长度为零，则效果稍好一些。这不是特别好的表示，因为它丢弃了所有高频图像

内容，并且对空间或亮度偏移不是特别不变。Torralba，Fergus 和 Freeman 提出了几种对齐方法来缓解后一种缺点，但我们不担心这个项目的对齐方式。我们仅将微小图像用作基线。

Kmeans

最近邻分类器同样易于理解。当负责将测试特征分类到特定类别时，可以简单地找到“最近”的训练示例（L2 距离是足够的度量）并且为测试用例分配最近训练示例的标签。最近邻分类器具有许多期望的特征 - 它不需要训练，它可以学习任意复杂的决策边界，并且它通常支持多类问题。但它很容易受到训练噪音的影响，可以通过基于 K 个最近邻居的投票来缓解（但你不需要这样做）。随着特征维度的增加，最近邻分类器也会受到影响，因为分类器没有机制来了解哪些维度与决策无关。对于高维数据和许多训练示例，最近邻居计算也变慢。
在 15 场景数据库中，微小的图像表示和最近邻分类器一起将获得约 15％至 25％的准确度。相比之下，机会表现约为 7％。

SIFT

在实现基线场景识别之后，现在是时候进行更复杂的图像表示 - 量子化 SIFT 特征的包。在我们将图像表示为特征直方图包之前，我们首先需要建立一个*词汇表*视觉词汇。我们将通过从我们的训练集（10 个或 100 个数千个）中抽取许多局部特征来形成这个词汇表，然后用 kmeans 对它们进行聚类。kmeans 聚类的数量是我们词汇量的大小和我们的特征的大小。例如，您可以首先将许多 SIFT 描述符聚类为 k = 50 个聚类。这将连续的 128 维 SIFT 特征空间划分为 50 个区域。对于我们观察到的任何新的 SIFT 特征，只要我们保存原始聚类的质心，我们就可以确定它属于哪个区域。那些质心是我们的视觉词汇词汇。由于采样和集群许多本地功能可能很慢，因此启动程序代码会保存集群质心并避免在将来的运行中重新计算它们。

现在我们准备将我们的训练和测试图像表示为视觉词的直方图。对于每个图像，我们将密集地采样许多 SIFT 描述符。我们只计算在视觉词汇词汇表中落入每个群集的 SIFT 描述符的数量，而不是存储数百个 SIFT 描述符。这是通过为每个 SIFT 特征找到最近邻居 kmeans 质心来完成的。因此，如果我们有 50 个视觉单词的词汇表，并且我们在图像中检测到 220 个 SIFT 特征，那么我们的 SIFT 表示将是 50 维的直方图，其中每个 bin 计算 SIFT 描述符分配给该群集的次数和总和应该归一化，以便图像大小不会显着改变特征量的包。

现在，您应该测量当与最近邻居分类器配对时，您的 SIFT 表示包的工作情况。SIFT 表示包（簇数，采样密度，采样比例，SIFT 参数等）有*许多*设计决策和自由参数，因此精度可能在 50％到 60％之间变化。

SVM

最后一项任务是训练 1-vs-all 线性 SVMS 以在 SIFT 特征空间中进行操作。线性分类器是最简单的学习模型之一。特征空间由学习的超平面划分，测试用例根据它们落在哪个超平面上进行分类。尽管这个模型的表达远远不如最近邻分类器，但它通常表现得更好。例如，也许在我们的 SIFT 表示中，50 个视觉单词中的 40 个是无信息的。他们根本无法帮助我们决定图像是“森林”还是“卧室”。也许它们代表在所有类型的场景中出现的平滑补丁，渐变或阶梯边缘。来自最近邻分类器的预测仍将受到这些频繁视觉词的严重影响，而线性分类器可以知道特征向量的那些维度不太相关，因此在做出决策时会降低它们的权重。有许多方法可以学习线性分类器，但我们会找到支持向量机的线性决策边界。您不必实现支持向量机。

然而，线性分类器本质上是二元的，我们有 15 路分类问题。要确定测试用例属于哪 15 个类别，您将训练 15 个二进制，1 对所有 SVM。1-vs-all 意味着每个分类器将被训练以识别’森林’与’非森林’，’厨房’与’非厨房’等。所有 15 个分类器将在每个测试用例和分类器上进行评估是最自信的积极的“胜利”。例如，如果’厨房’ 分类器返回-0.2 的分数（其中 0 在决策

边界上），并且'forest'分类器返回-0.3 的分数，并且所有其他分类器甚至更负，测试用例将被归类为厨房即使没有一个分类器将测试用例放在决策边界的正面。

在学习 SVM 时，你有一个自由参数'lambda'来控制模型的正规化程度。你的准确性对 lambda 非常敏感，所以一定要测试很多值。看到　尽管没有一个分类器将测试用例放在决策边界的正面，但测试用例将被归类为厨房。在学习 SVM 时，你有一个自由参数'lambda'来控制模型的正规化程度。你的准确性对 lambda 非常敏感，所以一定要测试很多值。看到　尽管没有一个分类器将测试用例放在决策边界的正面，但测试用例将被归类为厨房。在学习 SVM 时，你有一个自由参数'lambda'来控制模型的正规化程度。你的准确性对 lambda 非常敏感，所以一定要测试很多值。

现在，您可以评估与 1-vs-all 线性 SVM 配对的 SIFT 表示包。根据参数，准确度应为 60％至 70％。起动代码，从开始 student.py 包含输入，输出建议策略的五大功能，您将实施更具体的指导和：get_tiny_images()，nearest_neighbor_classify()，build_vocabulary()，get_bags_of_sifts()，和 svm_classify()。utils 文件夹包含在笔记本上可视化结果所需的代码，您无需编辑该代码。

六、实验器材（环境配置）：

Windows：

CPU: Intel® Core™ i7-6700HQ CPU @ 2.60GHz

Cache: L1:256KB

L2:1.0MB,

L3:6.0MB

开发环境：python 3.6、opencv_python-3.3.0.10、opencv_contrib_python-3.3.0.10

七、实验步骤及操作：

1. 获取微小图片

get_tiny_images ()实现

该函数接收一个参数，图片集的路径，这是一个长度为 1500 的数组。遍历图片，直接使用 resize 函数将其大小为 16*16，并进行归一化，汇总微小图片集，再将微小图片集返回。具体实现如下：

```
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="Imaging-SIFT"):
        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 特征提取和词袋构建

build_vocabulary ()实现

该函数接收两个参数，分别是训练集图片集的路径，这是一个长度为 1500 的数组，另一个是待构建词袋的大小。遍历图片，提取 sift 特征，这是一个 128 维特征。把训练集所有图片的 sift 特征保存起来，进行聚类，簇中心共有 vocab_size 个，最后将簇中心返回。另外由于构建字典的时间较长，可将簇中心保存在文件中，方便下次使用。具体实现如下：

```
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 the
                     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() function.
    The documentation is available here:
    http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.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)
    '''
    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图特征表示

get_bags_of_words ()实现

该函数接收一个参数，图片集的路径，这是一个长度为 1500 的数组。另外需要获取构建的词袋模型簇中心集。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，相对于其作为图片的新的特征，最后函数将新的特征进行返回。另外由于 sift 提出与直方统计的时间也较长，可将得到的图片特征保存在文件中，方便分类器调参的时候使用。具体实现如下：

```
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 for
    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 histogram
    at the index of that word. For example, if the closest vector in the vocab
    is the 103rd word, then you should add 1 to the 103rd histogram bin. Your
    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
    '''

    with open('vocab.pkl', 'rb') as v:
        vocab = pickle.load(v)
        image_feats = np.zeros((len(image_paths),len(vocab)))

    for i, path in tqdm(enumerate(image_paths), desc='get_bags_of_words'):

        image = np.asarray(Image.open(path), dtype = 'float32')
        frames, descriptors = dsift(image, step=[9,9], fast=True)

        dist = distance.cdist(vocab, descriptors, 'euclidean')
        mdist = np.argmin(dist, axis = 0)
        histo, bins = np.histogram(mdist, range(len(vocab)+1))
        if np.linalg.norm(histo) == 0:
            image_feats[i, :] = histo
        else:
            image_feats[i, :] = histo / np.linalg.norm(histo)
    return image_feats
```

4. svm 分类器

Svm_classify ()实现

经过调参最后 svm 分类器使用高斯核函数的准确率比线性 svm 更高，由于 svm 是二分类，所以要使用 ovr

也就是一对多，对每类进行 yes or no 的分类，进而完成多分类。

```
def svm_classify(train_image_feats, train_labels, test_image_feats):
    '''
    This function will predict a category for every test image by training
    15 many-versus-one linear SVM classifiers on the training data, then
    using those learned classifiers on the testing data.
    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 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)
```

5. knn 分类器

nearest_neighbor_classify ()实现

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。

```
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

由于 get_bags_of_words()和 build_vocabulary ()执行太慢了，不利于 svm 和 knn 分类器超参数调整，所以可以对他们的执行结果进行保存。

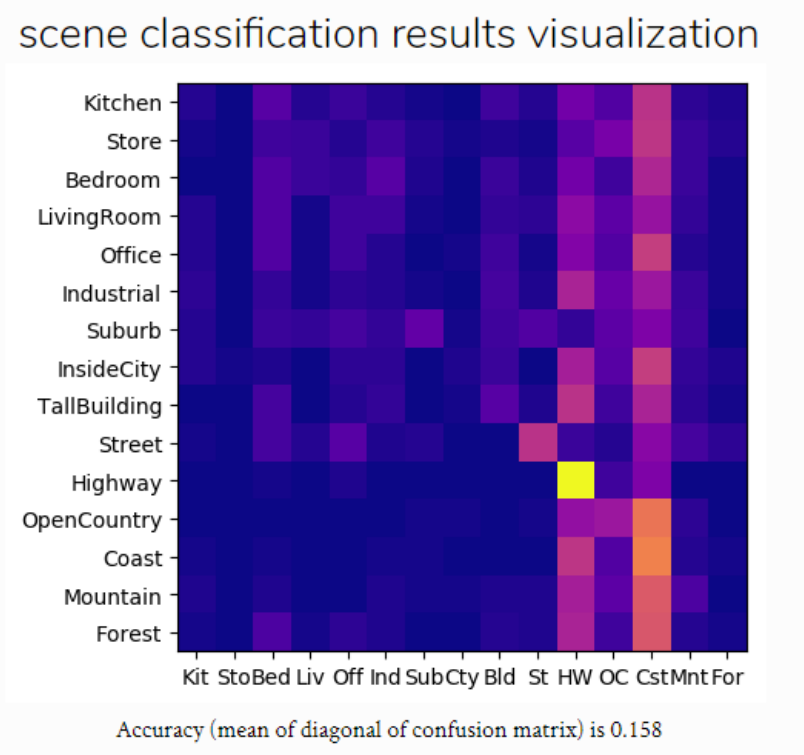
```
#!/usr/bin/python
import numpy as np
import os
import pickle

from helpers import get_image_paths
from student import get_tiny_images, build_vocabulary, get_bags_of_words, \
    svm_classify, nearest_neighbor_classify
from create_results_webpage import create_results_webpage

def projSceneRecBoW():
    '''''
    For this project, you will need to report performance for three
    combinations of features / classifiers. We recommend that you code them in
    this order:
        1) Tiny image features and nearest neighbor classifier
        2) Bag of word features and nearest neighbor classifier
        3) Bag of word features and linear SVM classifier
    The starter code is initialized to 'placeholder' just so that the starter
    code does not crash when run unmodified and you can get a preview of how
    results are presented.

    Interpreting your performance with 100 training examples per category:
     accuracy  =   0 -> Something is broken.
     accuracy ~= .07 -> Your performance is equal to chance.
                        Something is broken or you ran the starter code unchanged.
     accuracy ~= .20 -> Rough performance with tiny images and nearest
                        neighbor classifier. Performance goes up a few
                        percentage points with K-NN instead of 1-NN.
     accuracy ~= .20 -> Rough performance with tiny images and linear SVM
                        classifier. Although the accuracy is about the same as
                        nearest neighbor, the confusion matrix is very different.
     accuracy ~= .40 -> Rough performance with bag of word and nearest
                        neighbor classifier. Can reach .60 with K-NN and
                        different distance metrics.
     accuracy ~= .50 -> You've gotten things roughly correct with bag of
                        word and a linear SVM classifier.
     accuracy >= .70 -> You've also tuned your parameters well. E.g. number
                        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'
```

```
66.     # CLASSIFIER = 'placeholder'
67.
68.     # This is the path the script will look at to load images from.
69.     data_path = '../data/'
70.
71.     # This is the list of categories / directories to use. The categories are
72.     # somewhat sorted by similarity so that the confusion matrix looks more
73.     # structured (indoor and then urban and then rural).
74.     categories = ['Kitchen', 'Store', 'Bedroom', 'LivingRoom', 'Office',
75.            'Industrial', 'Suburb', 'InsideCity', 'TallBuilding', 'Street',
76.            'Highway', 'OpenCountry', 'Coast', 'Mountain', 'Forest']
77.
78.     # This list of shortened category names is used later for visualization.
79.     abbr_categories = ['Kit', 'Sto', 'Bed', 'Liv', 'Off', 'Ind', 'Sub',
80.         'Cty', 'Bld', 'St', 'HW', 'OC', 'Cst', 'Mnt', 'For']
81.
82.     # Number of training examples per category to use. Max is 100. For
83.     # simplicity, we assume this is the number of test cases per category as
84.     # well.
85.     num_train_per_cat = 100
86.
87.     # This function returns string arrays containing the file path for each train
88.     # and test image, as well as string arrays with the label of each train and
89.     # test image. By default all four of these arrays will be 1500x1 where each
90.     # entry is a string.
91.     print('Getting paths and labels for all train and test data.')
92.     train_image_paths, test_image_paths, train_labels, test_labels = \
93.         get_image_paths(data_path, categories, num_train_per_cat)
94.     #   train_image_paths  1500x1   list
95.     #   test_image_paths   1500x1   list
96.     #   train_labels       1500x1   list
97.     #   test_labels        1500x1   list
98.
99.     ################################################################################
100.    ## Step 1: Represent each image with the appropriate feature
101.    # Each function to construct features should return an N x d matrix, where
102.    # N is the number of paths passed to the function and d is the
103.    # dimensionality of each image representation. See the starter code for
104.    # each function for more details.
105.    ################################################################################
106.
107.    print('Using %s representation for images.' % FEATURE)
108.
109.    if FEATURE.lower() == 'tiny image':
110.        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 generated
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.npy
121.        # file and re-run main.py
122.
123.        if os.path.isfile('vocab.pkl') is False:
124.            print('No existing visual word vocabulary found. Computing one from training images\n')
125.            vocab_size = 400
126.            vocab = build_vocabulary(train_image_paths, vocab_size)
127.            with open('vocab.pkl', 'wb') as handle:
128.                pickle.dump(vocab, handle, protocol=pickle.HIGHEST_PROTOCOL)
129.
130.        if os.path.isfile('train_image_feats.pkl') is False:
131.            # YOU CODE get_bags_of_sifts.py
132.            train_image_feats = get_bags_of_words(train_image_paths);
133.            with open('train_image_feats.pkl', 'wb') as handle:
134.                pickle.dump(train_image_feats, handle, protocol=pickle.HIGHEST_PROTOCOL)
135.        else:
136.            with open('train_image_feats.pkl', 'rb') as handle:
137.                train_image_feats = pickle.load(handle)
138.
139.        if os.path.isfile('test_image_feats.pkl') is False:
140.            test_image_feats  = get_bags_of_words(test_image_paths);
141.            with open('test_image_feats.pkl', 'wb') as handle:
142.                pickle.dump(test_image_feats, handle, protocol=pickle.HIGHEST_PROTOCOL)
143.        else:
144.            with open('test_image_feats.pkl', 'rb') as handle:
145.                test_image_feats = pickle.load(handle)
146.
147.
148.    elif FEATURE.lower() == 'placeholder':
149.        train_image_feats = []
150.        test_image_feats = []
151.
152.    else:
```

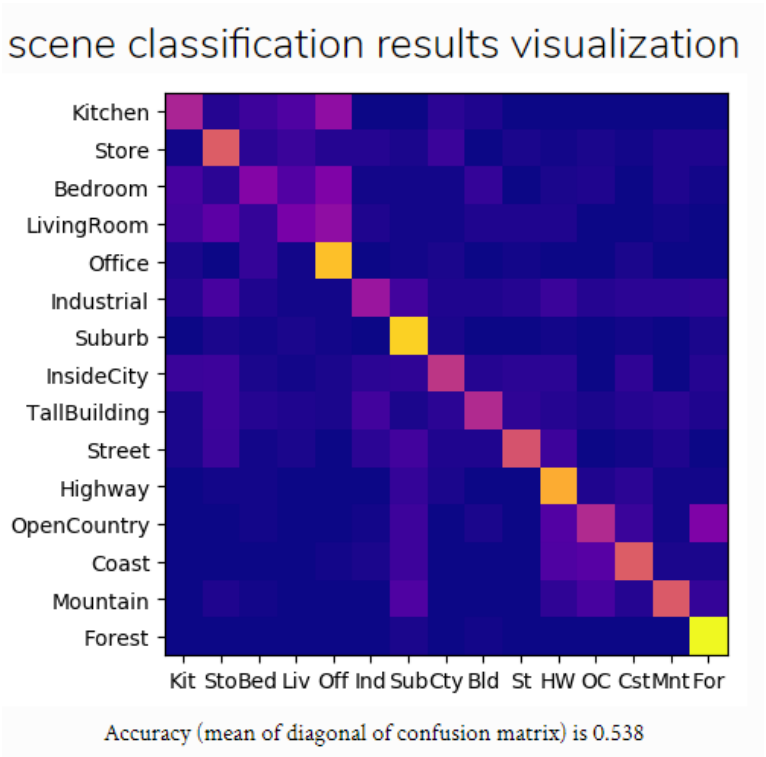
```
        raise ValueError('Unknown feature type!')

    ##################################################################################
    ## Step 2: Classify each test image by training and using the appropriate classifier
    # Each function to classify test features will return an N x 1 string array,
    # where N is the number of test cases and each entry is a string indicating
    # the predicted category for each test image. Each entry in
    # 'predicted_categories' must be one of the 15 strings in 'categories',
    # 'train_labels', and 'test_labels'. See the starter code for each function
    # for more details.
    ##################################################################################

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

    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 every 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 train
    # and test on many random splits of the data. You are not required to do so
    # for this project.

    # This function will recreate results_webpage/index.html and various image
    # 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()
```

八、实验数据及结果分析：

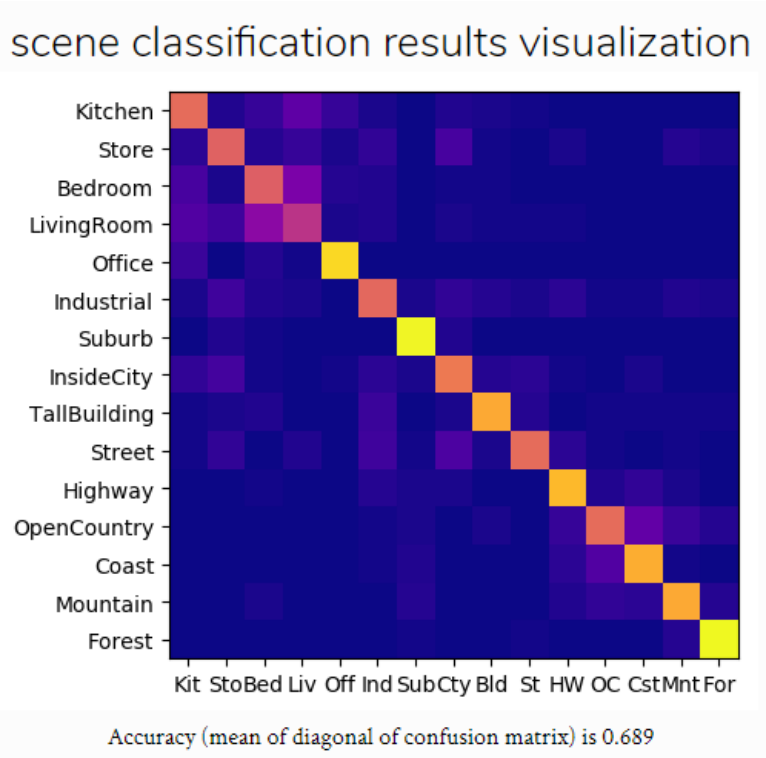
- 微小的图像表示和最近邻分类器，欧式距离，k=7，精度为 15.8%



- 基于 SIFT 特征的词带模型的表示和最近邻分类器，欧式距离，k=7，精度为 53.8%



- 基于 SIFT 特征的词带模型的和线性 SVM 分类器，精度为 68.9%



九、实验结论：

实现了基于特征提取+分类器的图像识别。

十、总结及心得体会：

分类器有一个 GridSearchCV 函数可以网格搜索最优值进行调参，不需要手动调参。
SIFT 获取在 python 中比较曲折，高版本的 cv 因为版权问题已经没有 sift 的相关函数，因此需要下载指定版本 opencv_python-3.3.0.10、opencv_contrib_python-3.3.0.10。
另外在 python 的 sklearn 的 svm 分类中并没有 lambda 这个参数，这个是 svm 损失函数的正则化参数，不像实验指导书中说的那么简单，使用 matlab 的同学只需要将 lambda 调到 0.0001 就能够获得比较精确的效果，但是 python 由于实现方式不同需要大量的调参。

十一、对本实验过程及方法、手段的改进建议：

鼓励学弟学妹使用并行的方式加速 sift 特征提取和词袋构建以及统计直方图。

报告评分：

指导教师签字：