电 子 科 技 大 学

实 验 报 告

学生姓名：关文聪　　学 号：2016060601008　　指导教师：徐行

一、 实验项目名称：

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

二、 实验原理：

图像词袋特征表示（例如比较流行的图像 SIFT 特征）：

SIFT，即尺度不变特征变换（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 算法具有如下一些特点：

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.区分性（Distinctiveness）好，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。

SIFT 特征匹配主要包括 2 个阶段：

第一阶段：SIFT 特征的生成，即从多幅图像中提取对尺度缩放、旋转、亮度变化无关的特征向量。

第二阶段：SIFT 特征向量的匹配。

SIFT 特征的生成一般包括以下几个步骤：1. 构建尺度空间，检测极值点，获得尺度不变性。2. 特征点过滤并进行精确定位。3. 为特征点分配方向值。4. 生成特征描述子。

词袋模型：

Bag-of-words model (BoW model) 最早出现在神经语言程序学(NLP)和信息检索（IR）领域. 该模型忽略掉文本的语法和语序, 用一组无序的单词(words)来表达一段文字或一个文档. 近年来, BoW 模型被广泛应用于计算机视觉中. 与应用于文本的 BoW 类比, 图像的特征(feature)被当作单词(Word),把图像“文字化”之后,有助于大规模的图像检索.也有人把简写为 Bag-of-Feature model(BOF model)或 Bag-of-Visual-Word(BOVW model)。

基本思想与关键步骤：

1、提取特征：根据数据集选取特征（如比较流行的 SIFT 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。

分类器构建：

最近邻分类器(Nearest Neighbor Classifier)：

通过计算测试集中样本点到训练集中每一个样本点的距离，找到训练集中距离该样本点最近的 K（K 一般为奇数 ）个样本，然后根据这 K 个样本的标签来确定测试集中样本的标签，确定方法为 K 个样本中出现最多的标签即为测试集中样本点的标签。样本之间的距离衡量标准常用的有欧氏距离、马氏距离等。

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。其基本原理是（以二维数据为例）：如果训练数据分布在二维平面上的点，它们按照其分类聚集在不同的区域。SVM 算法就是找一个超平面，并且它到离他最近的训练样本的距离要最大。即最优分割超平面最大化训练样本边界。

三、实验目的：

图像识别。特征提取+分类器构建和使用。

四、实验内容：

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。

1. 图像词袋特征表示：

   1.1 tiny 特征

   1.2 SIFT 特征

2. 分类器构建

   2.1 Nearest neighbor classifier：距离计算+归类

   2.2 SVM：一对多的策略

3. 词袋构建

   3.1 将从训练集获取的特征进行 k-means 聚类。K 的取值决定了后续的效果。

五、实验步骤：

1. 图像的 resize 和 tiny 特征提取：

修改完善“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：

```python
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
    '''

    dim = 8
    # 设定 dim 为 8 表示将读取的图像 resize 成 8*8 的小图像
    tiny_images = np.empty((len(image_paths), dim * dim), dtype=float)
    for i in range(0, len(image_paths)):
        # 读取图像
        im = imread(image_paths[i])
        # 裁剪到指定大小
        mdim = min(im.shape)
        im = im[:mdim, :mdim]
        # 转换成灰度图
        im = rgb2grey(im)
        # 缩小图像
        im = resize(im, (dim, dim), anti_aliasing=True)
        # 将图像像素归一化
```

```python
        im = im / np.sum(im)
        tiny_images[i, :] = np.reshape(im, (dim * dim))

    return tiny_images


cells_per_block = (4, 4)
pixels_per_cell = (4, 4)
orients = 9


def get_hogs(image, cells_per_block, pixels_per_cell, orients):
    # 缩小图像以进行调试
    scale = 0.4
    image = resize(image, (int(image.shape[0] * scale), int(image.shape[1] * scale)), anti_aliasing=True)

    out = hog(image, orients, cells_per_block=cells_per_block, pixels_per_cell=pixels_per_cell, feature_vector=True,
              block_norm='L1')

    return out.reshape(-1, cells_per_block[0] * cells_per_block[1] * orients)
```

2. 词袋模型构建与读取：

分别修改完善“student.py”文件中的“build_vocabulary”函数与“get_bags_of_words”函数，进行词袋模型的构建和获取。经过修改完善后的代码如下：

```python
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)
    '''

    # TODO: Implement this function!

    global cells_per_block
    global pixels_per_cell
    global orients

    out = np.empty((0, cells_per_block[0] * cells_per_block[1] * orients))
    # vocab = None
    for i in range(0, len(image_paths)):
        # 读取图像
        im = imread(image_paths[i])
        # 转换为灰度图
        im = rgb2grey(im)

        out = np.concatenate((out, get_hogs(im, cells_per_block, pixels_per_cell, orients)))

    # Kmeans 聚类，得到聚类簇的中心为词袋模型中的词
    kmeans = KMeans(vocab_size, max_iter=100, tol=1e-4, random_state=0).fit(out)
    vocab = kmeans.cluster_centers_

    print(vocab)
    print(vocab.shape)

    return vocab


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
    '''
    global cells_per_block
    global pixels_per_cell
    global orients

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

    # TODO: Implement this function!

    hist = np.zeros((len(image_paths), len(vocab)))
    for i in range(0, len(image_paths)):
        # 读取图像
        im = imread(image_paths[i])
        # 转换为灰度图
        im = rgb2grey(im)

        out = get_hogs(im, cells_per_block, pixels_per_cell, orients)

        # 创建直方图
        for j in range(0, len(out)):
            dist = cdist(np.array([out[j]]), vocab, metric='cosine')
```

```python
            bin = np.argsort(dist)[0][0]
            hist[i, bin] = hist[i, bin] + 1

        # 标准化直方图
        norm = np.linalg.norm(hist[i])
        if norm != 0:
            hist[i] = hist[i] / norm

    print(hist)
    print(hist.shape)

    return hist
```

3. 分类器构建：

（1）训练 SVM 分类器：修改完善“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：

```python
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.
    '''

    # 训练 SVM 分类器
    classifier = LinearSVC('l2', multi_class='ovr', random_state=0, max_iter=100)
    # 用训练得到的 SVM 分类器预测结果
    classifier.fit(train_image_feats, train_labels)

    print('*******')
    print(train_image_feats.shape)
    print(test_image_feats[0].shape)

    out = [''] * len(test_image_feats)
    for i in range(0, len(test_image_feats)):
        out[i] = classifier.predict(test_image_feats[i].reshape(1, -1))

    return np.array(out)
```

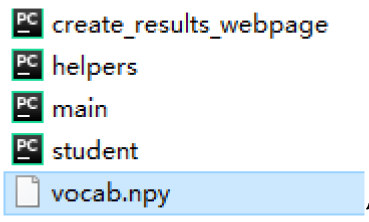
（2）训练最近邻分类器：修改完善“student.py”文件中的“nearest_neighbor_classify”函数，要注意到 k 值的选取会影响模型的结果，需要进行尝试。经过修改完善后的代码如下：

```
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
```

```python
                      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
    '''

    k = 9

    # Gets the distance between each test image feature and each train image feature
    # e.g., cdist
    distances = cdist(test_image_feats, train_image_feats, 'cosine')

    # 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

    labels = [None] * len(test_image_feats)
    for i in range(0, len(test_image_feats)):
        nn = np.argsort(distances[i])
        nn_label = np.array(train_labels)
        nn_label = nn_label[nn[:k]]
        mode, cnt = stats.mode(nn_label)
        labels[i] = mode[0]

    labels = np.array(labels)
    return labels
```

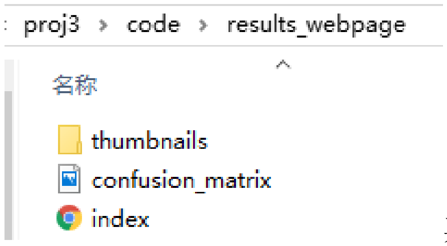
六、实验数据及结果分析：

运行“main.py”文件，程序构建词袋模型并进行训练，训练集数据共有 15 个场景类别，每类 100 张图片，总共训练 1500 张图片（注：训练时间可能会比较长，需要等待）。上述步骤完成后，在“code”目录下生成“vocab.npy”文件，

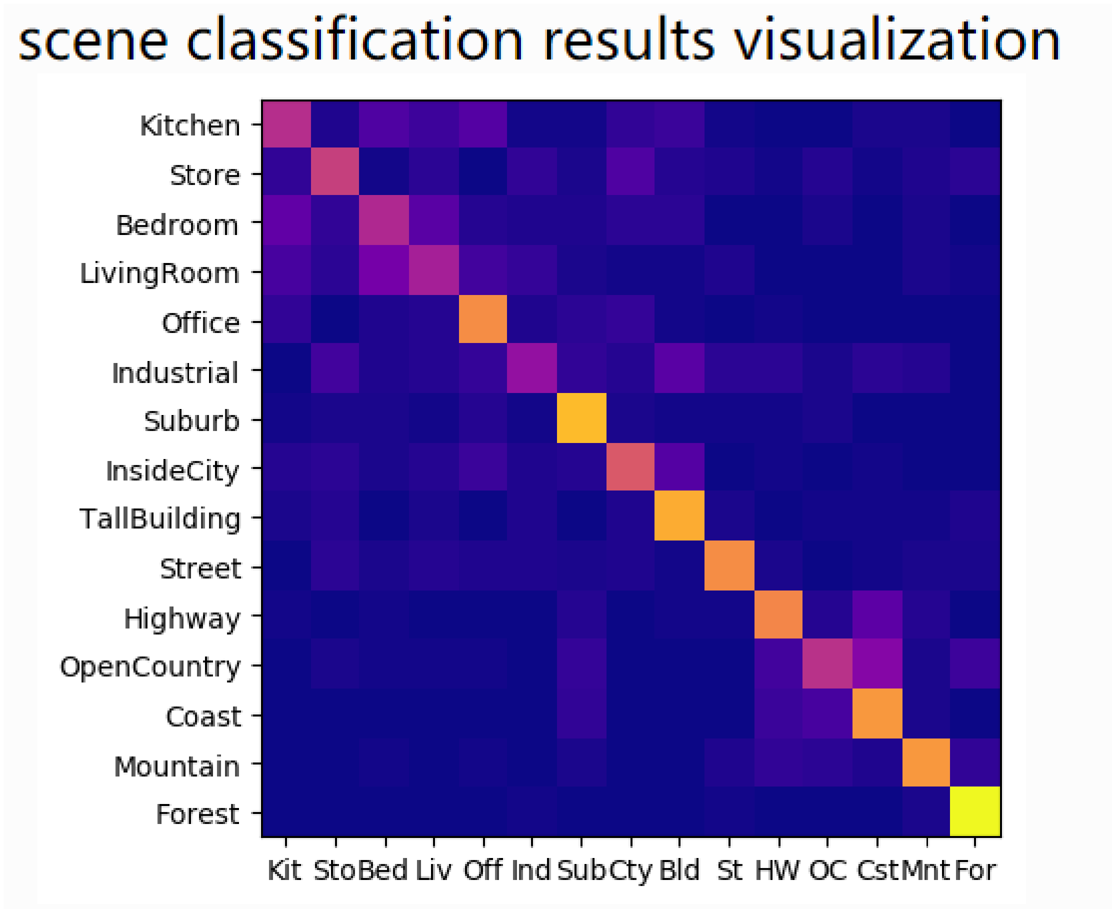

create_results_webpage
helpers
main
student
vocab.npy

它的规模为 1500*300，即词袋模型包含了训练集 1500 张图片，词袋模型中的视觉个数为 300。

接下来会应用模型，使用测试集中的图片数据进行测试。测试集数据同样包含 15 个场景类别，但测试集的图片数量要多于 1500 张。测试完成后，在“code”目录下会生成“results_webpage”文件夹：


proj3 › code › results_webpage
名称
thumbnails
confusion_matrix
index

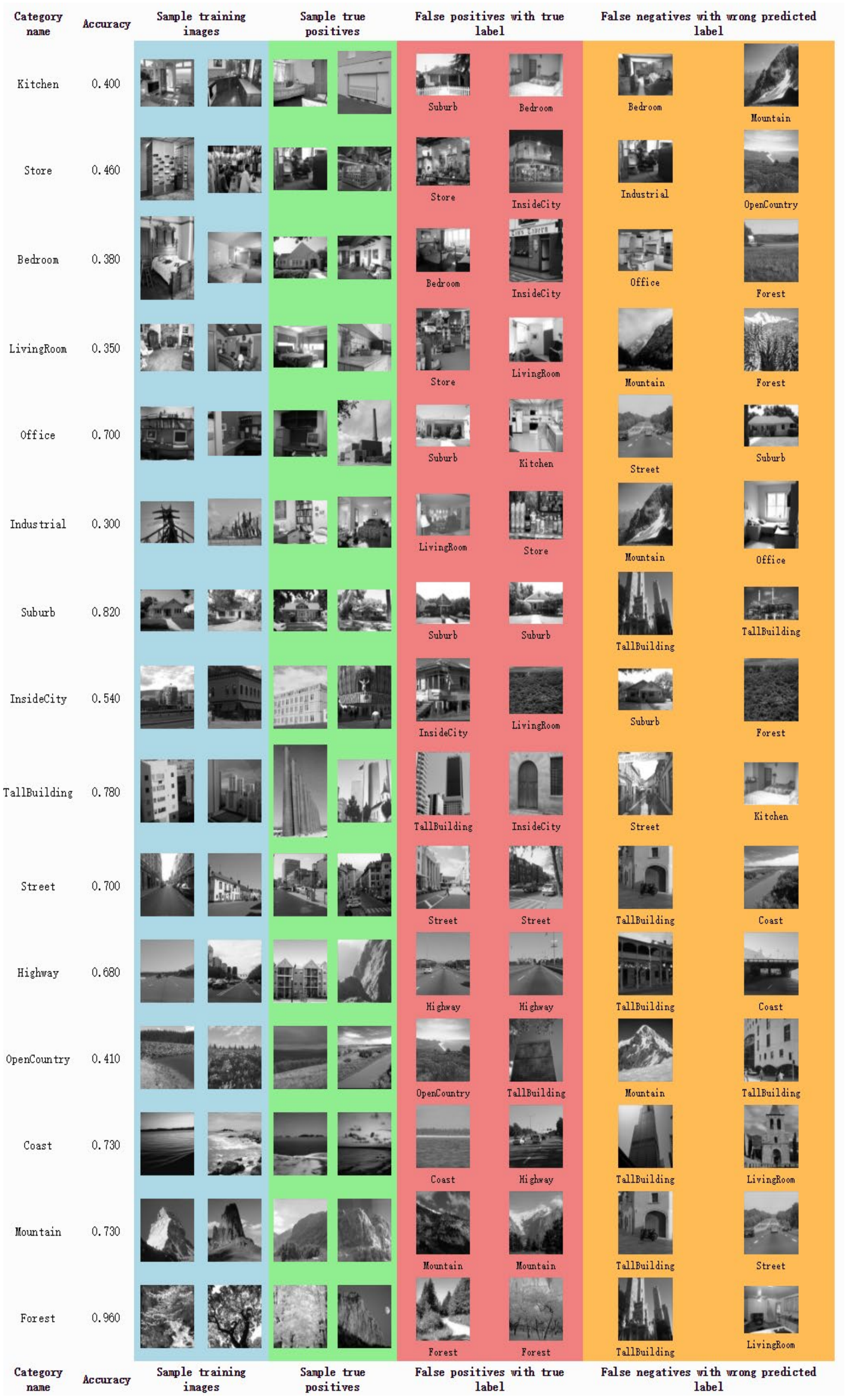
其中存放了输出的测试结果，如下图所示：



Accuracy (mean of diagonal of confusion matrix) is 0.596

结果的精度（混淆矩阵的对角线的平均值）为 59.6%

对于具体每类场景的识别分类精度如下图所示：



Category name	Accuracy	Sample training images	Sample true positives	False positives with true label		False negatives with wrong predicted label	
Kitchen	0.400			Suburb	Bedroom	Bedroom	Mountain
Store	0.460			Store	InsideCity	Industrial	OpenCountry
Bedroom	0.380			Bedroom	InsideCity	Office	Forest
LivingRoom	0.350			Store	LivingRoom	Mountain	Forest
Office	0.700			Suburb	Kitchen	Street	Suburb
Industrial	0.300			LivingRoom	Store	Mountain	Office
Suburb	0.820			Suburb	Suburb	TallBuilding	TallBuilding
InsideCity	0.540			InsideCity	LivingRoom	Suburb	Forest
TallBuilding	0.780			TallBuilding	InsideCity	Street	Kitchen
Street	0.700			Street	Street	TallBuilding	Coast
Highway	0.680			Highway	Highway	TallBuilding	Coast
OpenCountry	0.410			OpenCountry	TallBuilding	Mountain	TallBuilding
Coast	0.730			Coast	Highway	TallBuilding	LivingRoom
Mountain	0.730			Mountain	Mountain	TallBuilding	Street
Forest	0.960			Forest	Forest	TallBuilding	LivingRoom

Category name | Accuracy | Sample training images | Sample true positives | False positives with true label | False negatives with wrong predicted label

由以上结果可以看出，虽然总体的准确率可以达到接近 60%，但是，对于不同具体场景识别分类的准确率差异是巨大的！其中，分类最准确的场景“Forest”的准确率达到了 96%，这是非常精确的识别与分类，比较准确的场景“Suburb”也达到了 82%。但是，类似“Industrial”“LivingRoom”“Bedroom”“Kitchen”等场景识别分类的准确率就只有 30%-40%，效果比较差。

七、实验结论：

在本次实验中，主要实现了图像的特征提取和图像的分类，并综合应用了 KMeans 聚类算法、最近邻分类器、SVM 分类器等算法，构建了视觉词袋模型并实现了场景预测。但是，模型的分类识别准确率并不是很高，综合分析，可能有几个原因。一是训练学习的样本数量太少，少于测试数据，可能有一些特征未能被学习到。二是模型的参数需要调整，比如 KMeans 聚类的 k 的取值，还比如距离的选择，是选取欧氏距离还是其它距离等，不同的取值对于结果也有影响。此外，不同场景本身的差异性也影响了模型的判断。

八、总结及心得体会：

本次实验是一次综合性很强的实验，融合了课堂上所学的图像特征、聚类算法、分类算法、词袋模型等等知识，通过动手操作，锻炼了动手能力，也对课堂上所学知识有了更深刻的理解和认识。模型应该还有不小的提升空间，后续可以继续对模型算法和参数进一步优化。

九、对本实验过程及方法的改进建议：

可以提供更详细的文档和实验指导，例如实验指导书等。

模型的训练时间过长，可以考虑减小数据的规模。

报告评分：

指导教师签字：