一、实验项目名称：

基于词袋模型的场景识别

二、实验原理：

本次实验主要使用了两种特征提取算法（Tiny images feature 和 Bag of sift）及两种分类算法（k-Nearest Neighbor 和 SVM）进行场景识别。接下来将分别介绍四种算法的原理：

（1） 特征提取——Tiny images feature

Tiny images feature 方法首先将原始图片的大小调整，例如调整到 16*16，接着将图片转换为向量并归一化处理得到一个目标图像特征向量，然后利用这个特征向量代表该类别以进行后面的分类操作。

（2） 特征提取——Bag of sift

该算法分为以下三步：

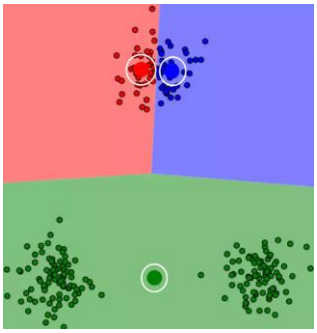
a) 利用 SIFT 算法提取图片特征点，得到视觉词汇

b) 用 K-means 对 SIFT 特征点（视觉词汇）进行聚类，构造词典，得到 k 个聚类中心，每个聚类中心看作一个用于计算的标准词汇，k 个聚类中心组成词典

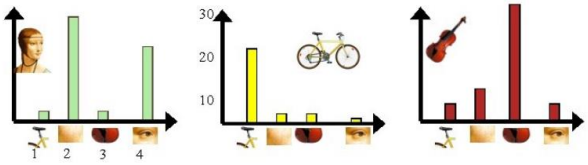
c) 利用词汇字典的中标准词汇（聚类中心）表示图像。对于每个输入图像，利用 SIFT 算法从图像中提取多个特征点，将这些特征点用词汇字典中的标准词汇近似代替，通过统计词汇字典中每个标准词汇在图像中出现的次数，得到一个 k 维特征向量，该向量表征了图片中各标准词汇的出现次数。

以人脸，自行车，吉他三个目标类为例，首先从三个图像中利用 SIFT 算法提取出视觉词汇如下：

接着将所有视觉词汇集合在一起，利用 K-means 算法构造词汇字典：



再将输入图像用 SIFT 算法得到的特征点用词汇字典中近似词汇代替，并统计词典中每个词汇出现的次数，得到一个 k 维特征向量。如下：



（3） 分类算法——k-Nearest Neighbor

kNN 是通过测量不同特征值之间的距离进行分类，如果一个样本在特征空间中的 k 个最邻近的样本中的大多数属于某一个类别，则该样本也划分为这个类别。在本实验中，计算输入图片的特征向量与训练集中所有图片特征的距离，筛选出距离最近的 k 个图片，他们中出现次数最多的类别作为图片的类别。

（4） 分类算法——SVM

SVM 即支持向量机，是一种 2 元分类器，通过最大化正负类之间分割线的距离得到分类方法。由于 svm 只能进行 2 元分类，因此对 n 个类别进行分类时需要训练 n 个分类器，用 n 个分类器判断样本属于该类别的概率，取概率最大值作为样本的最终类别。

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：

本次实验实现了 Tiny images feature 特征提取，Bag of SIFT 特征提取，k-Nearest Neighbor 图片分类，SVM 分类器四个部分模块及主函数模块。

（1）Tiny images feature 特征提取由 get_tiny_images.m 实现

（2）Bag of SIFT 特征提取
build_vocabulary.m 实现词袋中标准词汇的选择
get_bags_of_sifts.m 实现词袋模型的构建

（3）k-Nearest Neighbor 图片分类由 nearest_neighbor_classify.m 实现

（4）SVM 分类器由 svm_classify.m 实现

（5）主函数 project3.m 实现每次实验不同算法的选取，实现了以下几种“特征提取+分类器”组合来实现图像识别

a) Tiny + Nearest Neighbor
b) Tiny + SVM

c) Bags of SIFT + Nearest Neighbor
d) Bags of SIFT+SVM

五、实验步骤：

以上几个模块的代码实现如下：

1) get_tiny_images.m

```
function image_feats = get_tiny_images(image_paths)

row = size(image_paths, 1);
image_feats = zeros(row, 16*16);
for i = 1 : row
    image = imread(image_paths{i});
    image_feats(i,:) = reshape(imresize(image, [16 16]), 1, 16*16);
    temp = image_feats(i,:) - mean(image_feats(i,:));
    image_feats(i,:) = temp./norm(temp);
end
```

2) build_vocabulary.m

```
function vocab = build_vocabulary( image_paths, vocab_size )

row = size(image_paths, 1);
descriptors_num = 8;
descriptors = zeros(128, row * descriptors_num);
for i=1:row
    img = im2single(imread(image_paths{i}));
    [~,DESCRS] = vl_dsift(img);
    DESCRS = DESCRS(:,1:descriptors_num);
    descriptors(:,descriptors_num * (i-1) + 1 : descriptors_num * i) = DESCRS;
end

[C, ~] = vl_kmeans(descriptors, vocab_size);
vocab = single(C');
```

3) get_bags_of_sifts.m

```
function image_feats = get_bags_of_sifts(image_paths)

load('vocab.mat')
vocab = vocab';
vocab_size = size(vocab, 2);
image_size=size(image_paths,1);
image_feats=zeros(image_size,vocab_size);
for i=1:1:image_size
    image=single(imread(image_paths{i}));
```

```
    [~, features] = vl_dsift(image,'fast','step',5);
    features=single(features) ;
    D=vl_alldist2(features,vocab);
    feature_size=size(features,2);
    feature_hist=zeros(1,vocab_size);
    for j=1:1:feature_size
        min_dist=min(D(j,:));
        feature_index=find(D(j,:)==min_dist);
        feature_hist(1,feature_index)=feature_hist(1,feature_index)+1;
    end
    image_feats(i,:)=feature_hist/norm(feature_hist);
end
```

4) nearest_neighbor_classify.m

```
function predicted_categories = nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats)

test_num = size(test_image_feats,1);
distances = vl_alldist2(train_image_feats', test_image_feats');
labels = unique(train_labels);
labels_num = size(labels, 1);
[~, indices] = sort(distances, 1);
labels_count = zeros(labels_num, test_num);
for i = 1:test_num
    for j = 1:labels_num
        max20_labels = train_labels(indices(1:20, i));
        labels_count(j,i) = sum(strcmp(labels(j), max20_labels));
    end
end
[~, I] = max(labels_count,[],1);
predicted_categories = labels(I);
end
```

5) svm_classify.m

```
function predicted_categories = svm_classify(train_image_feats, train_labels, test_image_feats)

train_num = size(train_image_feats, 1);
test_num = size(test_image_feats, 1);
categories = unique(train_labels);
categories_num = length(categories);
W1 = zeros(categories_num, size(test_image_feats, 2));
B2 = zeros(categories_num, 1);
for i=1:categories_num
    labels = ones(train_num,1).*-1;
    labels(strcmp(categories{i}, train_labels)) = 1;
```

```
    [W, B] = vl_svmtrain(train_image_feats', labels, 0.001);
    W1(i,:) = W';
    B2(i) = B;
end

confidences = W1*test_image_feats'+repmat(B2,1,test_num);
[~, indices] = max(confidences);
predicted_categories = categories(indices);
end
```

6) project3.m

```
%FEATURE = 'tiny image';
FEATURE = 'bag of sift';
%FEATURE = 'placeholder';

CLASSIFIER = 'nearest neighbor';
%CLASSIFIER = 'support vector machine';
%CLASSIFIER = 'placeholder';

data_path = 'C:\Users\cy\Documents\Tencent Files\786678234\FileRecv\proj1-3
(1)\proj3\data\';

categories = {'Kitchen', 'Store', 'Bedroom', 'LivingRoom', 'Office', ...
        'Industrial', 'Suburb', 'InsideCity', 'TallBuilding', 'Street', ...
        'Highway', 'OpenCountry', 'Coast', 'Mountain', 'Forest'};

abbr_categories = {'Kit', 'Sto', 'Bed', 'Liv', 'Off', 'Ind', 'Sub', ...
    'Cty', 'Bld', 'St', 'HW', 'OC', 'Cst', 'Mnt', 'For'};

num_train_per_cat = 100;

fprintf('Getting paths and labels for all train and test data\n')
[train_image_paths, test_image_paths, train_labels, test_labels] = ...
    get_image_paths(data_path, categories, num_train_per_cat);

fprintf('Using %s representation for images\n', FEATURE)

switch lower(FEATURE)
    case 'tiny image'

        train_image_feats = get_tiny_images(train_image_paths);
        test_image_feats  = get_tiny_images(test_image_paths);

    case 'bag of sift'
```

```
        if exist('vocab.mat', 'file')
            fprintf('calculating train_image_feats\n');
            vocab_size=600;
            vocab=build_vocabulary(train_image_paths, vocab_size);
            save('vocab.mat','vocab');
        end
        train_image_feats = get_bags_of_sifts(train_image_paths);
        test_image_feats    = get_bags_of_sifts(test_image_paths);

    case 'placeholder'
        train_image_feats = [];
        test_image_feats = [];

    otherwise
        error('Unknown feature type')
end

fprintf('Using %s classifier to predict test set categories\n', CLASSIFIER)

switch lower(CLASSIFIER)
    case 'nearest neighbor'

        predicted_categories = nearest_neighbor_classify(train_image_feats,
train_labels, test_image_feats);

    case 'support vector machine'

        predicted_categories = svm_classify(train_image_feats, train_labels,
test_image_feats);

    case 'placeholder'

        random_permutation = randperm(length(test_labels));
        predicted_categories = test_labels(random_permutation);

    otherwise
        error('Unknown classifier type')
end
create_results_webpage( train_image_paths, ...
                        test_image_paths, ...
                        train_labels, ...
                        test_labels, ...
                        categories, ...
```

```
                                    abbr_categories, ...
                                    predicted_categories)
```

7) get_image_paths.m

```
function [train_image_paths, test_image_paths, train_labels, test_labels] = ...
      get_image_paths(data_path, categories, num_train_per_cat)

num_categories = length(categories);

train_image_paths = cell(num_categories * num_train_per_cat, 1);
test_image_paths    = cell(num_categories * num_train_per_cat, 1);

train_labels = cell(num_categories * num_train_per_cat, 1);
test_labels    = cell(num_categories * num_train_per_cat, 1);

for i=1:num_categories
    images = dir( fullfile(data_path, 'train', categories{i}, '*.jpg'));
    for j=1:num_train_per_cat
        train_image_paths{(i-1)*num_train_per_cat + j} = fullfile(data_path, 'train',
categories{i}, images(j).name);
        train_labels{(i-1)*num_train_per_cat + j} = categories{i};
    end

    images = dir( fullfile(data_path, 'test', categories{i}, '*.jpg'));
    for j=1:num_train_per_cat
        test_image_paths{(i-1)*num_train_per_cat + j} = fullfile(data_path, 'test',
categories{i}, images(j).name);
        test_labels{(i-1)*num_train_per_cat + j} = categories{i};
    end
end
```

8) create_results_webpage.m

```
function create_results_webpage( train_image_paths, test_image_paths, train_labels,
test_labels, categories, abbr_categories, predicted_categories)

fprintf('Creating results_webpage/index.html, thumbnails, and confusion matrix\n')

num_samples = 2;
thumbnail_height = 75; %pixels

delete('results_webpage/thumbnails/*.jpg')

[success,message,messageid] = mkdir('results_webpage');
[success,message,messageid] = mkdir('results_webpage/thumbnails');
fclose('all');
fid = fopen('results_webpage/index.html', 'w+t');
```

```
num_categories = length(categories);

confusion_matrix = zeros(num_categories, num_categories);
for i=1:length(predicted_categories)
	row = find(strcmp(test_labels{i}, categories));
	column = find(strcmp(predicted_categories{i}, categories));
	confusion_matrix(row, column) = confusion_matrix(row, column) + 1;
end

num_test_per_cat = length(test_labels) / num_categories;
confusion_matrix = confusion_matrix ./ num_test_per_cat;
accuracy = mean(diag(confusion_matrix));
fprintf(	'Accuracy (mean of diagonal of confusion matrix) is %.3f\n', accuracy)

fig_handle = figure;
imagesc(confusion_matrix, [0 1]);
set(fig_handle, 'Color', [.988, .988, .988])
axis_handle = get(fig_handle, 'CurrentAxes');
set(axis_handle, 'XTick', 1:15)
set(axis_handle, 'XTickLabel', abbr_categories)
set(axis_handle, 'YTick', 1:15)
set(axis_handle, 'YTickLabel', categories)

visualization_image = frame2im(getframe(fig_handle));

imwrite(visualization_image, 'results_webpage/confusion_matrix.png')

fprintf(fid,'<!DOCTYPE html>\n');
fprintf(fid,'<html>\n');
fprintf(fid,'<head>\n');
fprintf(fid,'<link
href=''http://fonts.googleapis.com/css?family=Nunito:300|Crimson+Text|Droid+Sa
ns+Mono'' rel=''stylesheet'' type=''text/css''>\n');
fprintf(fid,'<style type="text/css">\n');

fprintf(fid,'body {\n');
fprintf(fid,'    margin: 0px;\n');
fprintf(fid,'    width: 100%%;\n');
fprintf(fid,'    font-family: ''Crimson Text'', serif;\n');
fprintf(fid,'    background: #fcfcfc;\n');
fprintf(fid,'}\n');
fprintf(fid,'table td {\n');
fprintf(fid,'    text-align: center;\n');
```

```
fprintf(fid,'    vertical-align: middle;\n');
fprintf(fid,'}\n');
fprintf(fid,'h1 {\n');
fprintf(fid,'    font-family: ''Nunito'', sans-serif;\n');
fprintf(fid,'    font-weight: normal;\n');
fprintf(fid,'    font-size: 28px;\n');
fprintf(fid,'    margin: 25px 0px 0px 0px;\n');
fprintf(fid,'    text-transform: lowercase;\n');
fprintf(fid,'}\n');
fprintf(fid,'.container {\n');
fprintf(fid,'    margin: 0px auto 0px auto;\n');
fprintf(fid,'    width: 1160px;\n');
fprintf(fid,'}\n');

fprintf(fid,'</style>\n');
fprintf(fid,'</head>\n');
fprintf(fid,'<body>\n\n');

fprintf(fid,'<div class="container">\n\n\n');
fprintf(fid,'<center>\n');
fprintf(fid,'<h1>CS 143 Project 3 results visualization</h1>\n');
fprintf(fid,'<img src="confusion_matrix.png">\n\n');
fprintf(fid,'<br>\n');
fprintf(fid,'Accuracy (mean of diagonal of confusion matrix) is %.3f\n', accuracy);
fprintf(fid,'<p>\n\n');

fprintf(fid,'<table border=0 cellpadding=4 cellspacing=1>\n');
fprintf(fid,'<tr>\n');
fprintf(fid,'<th>Category name</th>\n');
fprintf(fid,'<th>Accuracy</th>\n');
fprintf(fid,'<th colspan=%d>Sample training images</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>Sample true positives</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>False positives with true label</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>False negatives with wrong predicted label</th>\n',
num_samples);
fprintf(fid,'</tr>\n');

for i = 1:num_categories
    fprintf(fid,'<tr>\n');

    fprintf(fid,'<td>'); %category name
    fprintf(fid,'%s', categories{i});
    fprintf(fid,'</td>\n');
```

```
    fprintf(fid,'<td>'); %category accuracy
    fprintf(fid,'%.3f', confusion_matrix(i,i));
    fprintf(fid,'</td>\n');

    %collect num_samples random paths to images of each type.
    %Training examples.
    train_examples = train_image_paths(strcmp(categories{i}, train_labels));
    %True positives. There might not be enough of these if the classifier
    %is bad
    true_positives = test_image_paths(strcmp(categories{i}, test_labels) & ...
                                      strcmp(categories{i},
predicted_categories));

    false_positive_inds = ~strcmp(categories{i}, test_labels) & ...
                          strcmp(categories{i}, predicted_categories);
    false_positives    = test_image_paths(false_positive_inds);
    false_positive_labels = test_labels(false_positive_inds);



    false_negative_inds = strcmp(categories{i}, test_labels) & ...
                          ~strcmp(categories{i}, predicted_categories);
    false_negatives = test_image_paths( false_negative_inds );
    false_negative_labels = predicted_categories(false_negative_inds);

    %Randomize each list of files
    train_examples    = train_examples( randperm(length(train_examples)));
    true_positives    = true_positives( randperm(length(true_positives)));

    false_positive_shuffle = randperm(length(false_positives));
    false_positives = false_positives(false_positive_shuffle);
    false_positive_labels = false_positive_labels(false_positive_shuffle);

    false_negative_shuffle = randperm(length(false_negatives));
    false_negatives = false_negatives(false_negative_shuffle);
    false_negative_labels = false_negative_labels(false_negative_shuffle);



    train_examples      =      train_examples(      1:min(length(train_examples),
num_samples));
    true_positives    = true_positives( 1:min(length(true_positives), num_samples));
    false_positives = false_positives(1:min(length(false_positives),num_samples));
    false_positive_labels                                                    =
false_positive_labels(1:min(length(false_positive_labels),num_samples));
    false_negatives = false_negatives(1:min(length(false_negatives),num_samples));
```

```
    false_negative_labels                                                    =
false_negative_labels(1:min(length(false_negative_labels),num_samples));

    for j=1:num_samples
        if(j <= length(train_examples))
            tmp = imread(train_examples{j});
            height = size(tmp,1);
            rescale_factor = thumbnail_height / height;
            tmp = imresize(tmp, rescale_factor);
            [height, width] = size(tmp);

            [pathstr,name, ext] = fileparts(train_examples{j});
            imwrite(tmp, ['results_webpage/thumbnails/' categories{i} '_' name
'.jpg'], 'quality', 100)
            fprintf(fid,'<td bgcolor=LightBlue>');
            fprintf(fid,'<img src="%s" width=%d height=%d>', ['thumbnails/'
categories{i} '_' name '.jpg'], width, height);
            fprintf(fid,'</td>\n');
        else
            fprintf(fid,'<td bgcolor=LightBlue>');
            fprintf(fid,'</td>\n');
        end
    end

    for j=1:num_samples
        if(j <= length(true_positives))
            tmp = imread(true_positives{j});
            height = size(tmp,1);
            rescale_factor = thumbnail_height / height;
            tmp = imresize(tmp, rescale_factor);
            [height, width] = size(tmp);

            [pathstr,name, ext] = fileparts(true_positives{j});
            imwrite(tmp, ['results_webpage/thumbnails/' categories{i} '_' name
'.jpg'], 'quality', 100)
            fprintf(fid,'<td bgcolor=LightGreen>');
            fprintf(fid,'<img src="%s" width=%d height=%d>', ['thumbnails/'
categories{i} '_' name '.jpg'], width, height);
            fprintf(fid,'</td>\n');
        else
            fprintf(fid,'<td bgcolor=LightGreen>');
            fprintf(fid,'</td>\n');
        end
    end
```

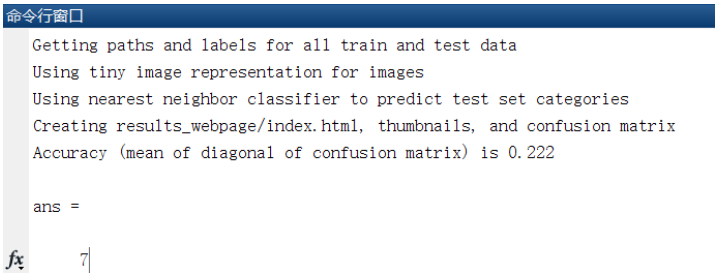
```
    for j=1:num_samples
        if(j <= length(false_positives))
            tmp = imread(false_positives{j});
            height = size(tmp,1);
            rescale_factor = thumbnail_height / height;
            tmp = imresize(tmp, rescale_factor);
            [height, width] = size(tmp);

            [pathstr,name, ext] = fileparts(false_positives{j});
            imwrite(tmp, ['results_webpage/thumbnails/' false_positive_labels{j}
'_' name '.jpg'], 'quality', 100)
            fprintf(fid,'<td bgcolor=LightCoral>');
            fprintf(fid,'<img src="%s" width=%d height=%d>', ['thumbnails/'
false_positive_labels{j} '_' name '.jpg'], width, height);
            fprintf(fid,'<br><small>%s</small>', false_positive_labels{j});
            fprintf(fid,'</td>\n');
        else
            fprintf(fid,'<td bgcolor=LightCoral>');
            fprintf(fid,'</td>\n');
        end
    end

    for j=1:num_samples
        if(j <= length(false_negatives))
            tmp = imread(false_negatives{j});
            height = size(tmp,1);
            rescale_factor = thumbnail_height / height;
            tmp = imresize(tmp, rescale_factor);
            [height, width] = size(tmp);

            [pathstr,name, ext] = fileparts(false_negatives{j});
            imwrite(tmp, ['results_webpage/thumbnails/' categories{i} '_' name
'.jpg'], 'quality', 100)
            fprintf(fid,'<td bgcolor=#FFBB55>');
            fprintf(fid,'<img src="%s" width=%d height=%d>', ['thumbnails/'
categories{i} '_' name '.jpg'], width, height);
            fprintf(fid,'<br><small>%s</small>', false_negative_labels{j});
            fprintf(fid,'</td>\n');
        else
            fprintf(fid,'<td bgcolor=#FFBB55>');
            fprintf(fid,'</td>\n');
        end
    end
```

```
        fprintf(fid,'</tr>\n');
end

fprintf(fid,'<tr>\n');
fprintf(fid,'<th>Category name</th>\n');
fprintf(fid,'<th>Accuracy</th>\n');
fprintf(fid,'<th colspan=%d>Sample training images</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>Sample true positives</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>False positives with true label</th>\n', num_samples);
fprintf(fid,'<th colspan=%d>False negatives with wrong predicted label</th>\n',
num_samples);
fprintf(fid,'</tr>\n');

fprintf(fid,'</table>\n');
fprintf(fid,'</center>\n\n\n');
fprintf(fid,'</div>\n')

fprintf(fid,'</body>\n');
fprintf(fid,'</html>\n');
fclose(fid);
```

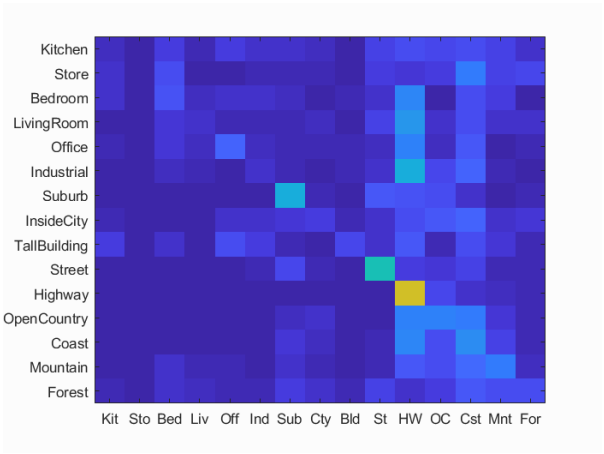
六、实验数据及结果分析：

（1） Tiny + Nearest Neighbor

命令行结果如下所示，可见精度为 0.222



confusion_matrix 结果如下：



Webpage 结果如下：

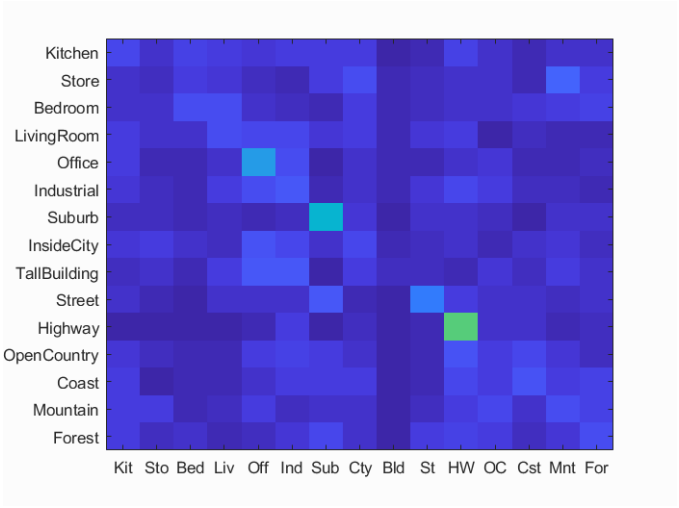
（2） Tiny + SVM

命令行结果如下所示，可见精度为 0.196

命令行窗口

```
>> proj3
Getting paths and labels for all train and test data
Using tiny image representation for images
Using support vector machine classifier to predict test set categories
Creating results_webpage/index.html, thumbnails, and confusion matrix
Accuracy (mean of diagonal of confusion matrix) is 0.196
```

confusion_matrix 结果如下：



Webpage 结果如下：

（3） Bags of SIFT + Nearest Neighbor

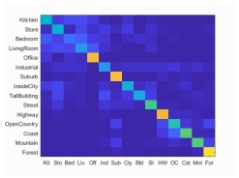
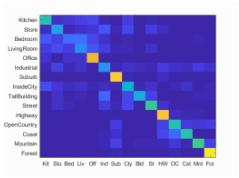
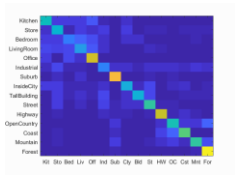
当 k=20，step=5，vocab_size=500 时，运行结果精度为 0.551 。

confusion_matrix 结果如下：

Webpage 结果如下：

a) 讨论 Bag of SIFT 中 vocal_size 的选取对结果的影响

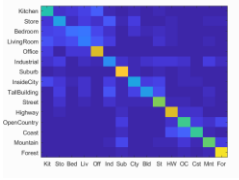
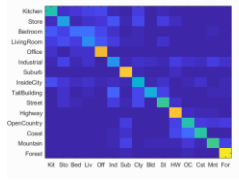
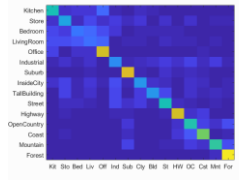
当 vocal_size 分别取 400,500,600 时(控制 k=1,step=5)，结果如下：

Vocab_size	400	500	600
精度	0.558	0.551	0.544
confusion_matrix			

可见，随着 vocab_size 的增大，精度不断下降。

b) 讨论 SIFT 中 step 的选取对结果的影响

当 step 分别取 3,5,7 时(控制 vocab_size=500,k=1)，结果如下：

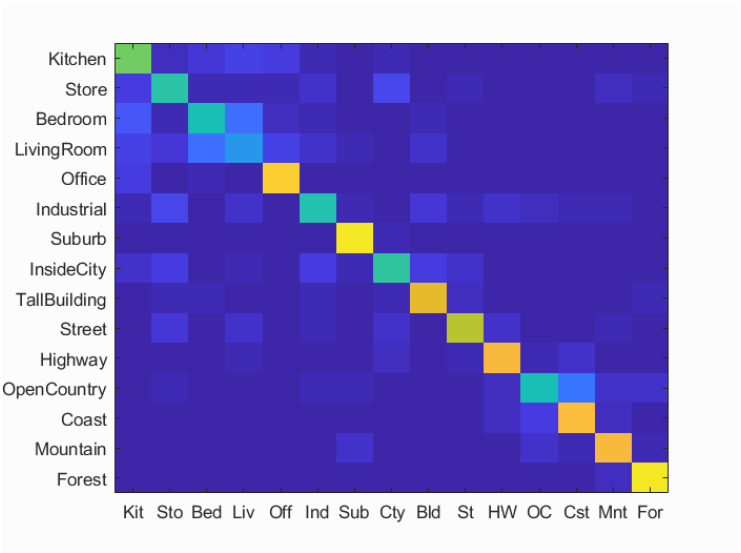
Step	3	5	7
精度	0.547	0.551	0.519
confusion_matrix			

可见，随着 step 的增大，精度先增加后降低，因而应当会存在一个最优的 step 使得精度最高。

（4） Bags of SIFT+SVM

当 lambda=0.0001 时运行结果精度为 0.693

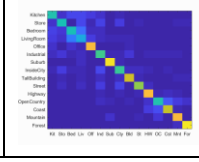
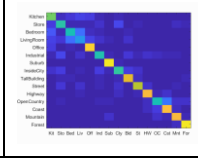
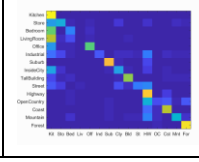
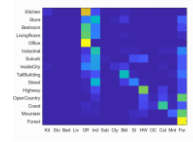
confusion_matrix 结果如下：



webpage 结果如下：

讨论 SVM 中 lambda 取值对结果的影响。

当 lambda 分别取 0.00001、0.0001、0.01、1、10 时，结果如下：

Lambda	0.00001	0.0001	0.01	1
精度	0.675	0.693	0.514	0.324
confusion_matrix				

可见，当 lambda 增大时，精度先增后减，因此 lambda 应有一个最优取值。

七、实验结论：

由以上结果可见，本次实验完成了使用不同方法进行场景识别的目的。此外，通过控制变量法还能得到以下几个小结论：

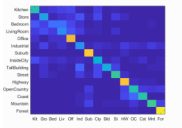
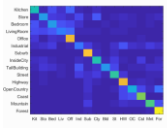
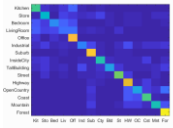
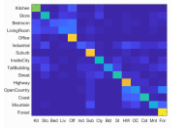
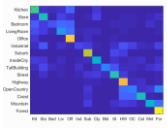
- Tiny images feature 算法无论是与 SVM 还是 k-Nearest Neighbor 结合效果都不好。

- Bag of sift 与 SVM 的结合效果最好，在最好情况精度达到了 0.693。
- 随着随着 vocab_size 的增大，精度不断下降。
- 随着 step 的增大，精度先增加后降低。
- 随着 lambda 的增大，精度先增后减。

八、对本实验过程及方法的改进建议：

本次实验仅对 vocab_size、step 和 lambda 进行了敏感性分析，除此之外还可以对 kNN 中的 k 取值进行敏感性分析。

利用 Bag of SIFT 和 kNN 进行图像识别，当 k 取 1，5，10，20，40 时（控制 vocab_size=500,step=5），实验结果如下：

K	1	5	10	20	40
精度	0.551	0.548	0.557	0.526	0.502
confusion_matrix					

可见，随着 k 的增加，精度先降低再上升后又逐步降低，因而这中间应当存在最优的 k 值使得结果精度最高。