读取 SIFT 特征代码：

```
function SiftFeat = readsift(imgPath)

src_1 = imgPath;
ext1 = '.dsift'; % extension name of SIFT file
siftDim = 128;


featPath_1 = [src_1, ext1];
fid_1 = fopen(featPath_1, 'rb');
featNum_1 = fread(fid_1, 1, 'int32');
SiftFeat_1 = zeros(siftDim, featNum_1);

for i = 1 : featNum_1 % 逐个读取SIFT特征
    SiftFeat_1(:, i) = fread(fid_1, siftDim, 'uchar');
    paraFeat_1(:, i) = fread(fid_1, 4, 'float32');
end
fclose(fid_1);

%% normalization
SiftFeat_1 = SiftFeat_1 ./ repmat(sqrt(sum(SiftFeat_1.^2)), size(SiftFeat_1, 1), 1);
SiftFeat = SiftFeat_1';

end
```

读取所有图片的 SIFT 特征：

```
clear;
close all;
clc;

num_pic = 1000;%图像数量
K = 10^4;%码本大小
num_sample = K * 15;%随机采样数量

%读取图像SIFT特征
srcFolderPath = './Image';
allFiles = dir(srcFolderPath);
imgCount = 0;
sift_all = [];
for i = 3 : length(allFiles)
    fileName = allFiles(i).name;
    if length(fileName) > 3 && strcmp(fileName(end-9 : end-6), '.jpg') == 1
        imgCount = imgCount + 1;
        imgPath = [srcFolderPath, '/', fileName(1:end-6)];
        fprintf('File %d/%d: %s\n', imgCount, num_pic, imgPath);
        sift = readsift(imgPath);
        sift_all = [sift_all;sift];
        pic_sift{imgCount}=sift;
    end
end

save("sift_data.mat","sift_all","pic_sift")
```

随机采样并训练视觉码本：

```
clear;
close all;
clc;

num_pic = 1000;%图像数量
K = 10^3;%码本大小
num_sample = K * 15;%随机采样数量

load('sift_all.mat')

%随机抽样
[sm,~] = size(sift_all);
sift_chose = sift_all(ceil(rand(1,num_sample)*sm),:);
clear sift_all;

%训练视觉码本
[~,codebook] = kmeans(sift_chose,K,'MaxIter',100,'display','iter');
clear sift_chose;
save("codebook3","codebook")
```

训练视觉码本的 GPU 实现（Python）：

```
import scipy.io as scio
import numpy as np
import torch
from kmeans_pytorch import kmeans

num_pic = 1000  #图像数量
K = 10**5  #码本大小
num_sample = 15*K  #随机采样数量
dataFile = 'sift_all.mat'
data = scio.loadmat(dataFile)['sift_all']
print(data.shape[0])

idx=np.random.randint(0,data.shape[0]-1,num_sample)
select_data=torch.from_numpy(data[idx])

cluster_ids_x, cluster_centers = kmeans(X=select_data, num_clusters=K, device=torch.device('cuda:0'))

print(0)
```

由于当码本大小为 10^5 时，采样数量数量至少为 10^6，数据量过大，因此采用 MiniBatchKMeans 算法对视觉码本进行训练：

```
import scipy.io as scio
import numpy as np
from sklearn.cluster import KMeans, MiniBatchKMeans

level=4.5
num_pic = 1000  # 图像数量
K = round(10 ** level)  # 码本大小
num_sample = 15*K  # 随机采样数量
dataFile = 'sift_all.mat'
data = scio.loadmat(dataFile)['sift_all']
idx = np.random.randint(0, data.shape[0] - 1, num_sample)
select_data = data[idx]
batch_size = 4000
mbk = MiniBatchKMeans(init='k-means++', n_clusters=K, batch_size=batch_size, random_state=1)
mbk.fit(select_data)
scio.savemat("codebook"+str(level)+".mat", {'codebook': mbk.cluster_centers_})
print('finish')
```

利用视觉码本对每幅图象的 SIFT 特征进行量化，表达为视觉单词直方图，并进行归一化，同时对图像数据库进行倒排索引。

```
clear;
close all;
clc;
level=4.5;
num_pic = 1000;%图像数量
K = round(10^level);%码本大小
num_sample = K * 15;%随机采样数量
load(['codebook' num2str(level) '.mat'])
load("pic_sift.mat")

%绘制直方图并归一化
figure('visible','off')

for l=1:2
    Database = [];
    for i = 1:num_pic
        fprintf('Dist %d/%d\n', i, num_pic);
        similarDistances = pdist2(pic_sift{i},codebook);
        [minElements,idx] = min(similarDistances,[],2);
        bins = 0.5:1:K+0.5;
        hist = histogram(idx,bins);
        Features = hist.Values;
        if l==1
            % L1 归一化
            Features = Features./sum(Features);
        else
            % L2 归一化
            Features = Features./sqrt(sum(Features.^2));
        end
        Database = [Database,Features'];
    end
    save(['Database' num2str(level) 'L' num2str(l)],"Database")
    %创建倒排索引
    D=cell(1,K);
    for i=1:num_pic
        for j=1:K
            if Database(j,i)>0
                D{1,j}=[D{1,j};[i,Database(j,i)]];
            end
        end
    end
    save(['Database' num2str(level) 'L' num2str(l) 'D'],"D")
end
clear pic_sift;
```

把数据库每一幅图像分别作为查询图像，计算平均检索精度

```
clear;
close all;
clc;
level=4.5;
num_pic = 1000;%图像数量
K = round(10^level);%码本大小
num_sample = K * 15;%随机采样数量
method = ['顺','倒'];
for l=1:2
    load(['Database' num2str(level) 'L' num2str(l) '.mat'])
    load(['Database' num2str(level) 'L' num2str(l) 'D.mat'])
    %进行检索测试
    for mode=0:1
        total_true=0;
        tic;
        for search_id=1:num_pic
            if mode==0
                %顺排表实现
                s=zeros(num_pic,1);
                for i =1:num_pic
                    for j =1:K
                        s(i) = s(i)+(Database(j,search_id)-Database(j,i))^2;
                    end
                end
            else
                %倒排表实现
                s=zeros(num_pic,1)+2;
                for j =1:K
                    if Database(j,search_id)>0
                        for m=1:size(D{1,j},1)
                            id = D{1,j}(m,1);
                            val = D{1,j}(m,2);
                            s(id)=s(id)-2*Database(j,search_id)*val;
                        end
                    end
                end
            end
            temp_result=sort(s);
            for i =1:4
                temp_find = find(s==temp_result(i));
                if ceil(temp_find/4)==ceil(search_id/4)
                    total_true=total_true+1;
                end
            end
        end
        fprintf('码本大小：10^%d 归一化方式：L%d 排序方法：%s\n', level,l,method(mode+1));
        fprintf('平均用时%f秒\n', toc/num_pic);
        fprintf('平均准确率%f\n\n', total_true/4/num_pic);
    end
end
```

计算结果如图所示：

码本数量	归一化方式	查询方式	准确率	平均用时/s	索引数据库大小/KB
10^3	L1	顺排	0.5975	0.002236	696
10^3.5	L1	顺排	0.5163	0.007001	1259
10^4	L1	顺排	0.4088	0.021568	2235
10^4.5	L1	顺排	0.3458	0.068406	3988
10^5	L1	顺排	0.3283	0.212822	7571
10^3	L2	顺排	0.7030	0.002203	699
10^3.5	L2	顺排	0.7048	0.007004	1264
10^4	L2	顺排	0.7120	0.021815	2246
10^4.5	L2	顺排	0.7710	0.067806	4013
10^5	L2	顺排	0.7453	0.211953	7623
10^3	L1	倒排	0.2610	0.015673	3303
10^3.5	L1	倒排	0.4515	0.014929	5032
10^4	L1	倒排	0.5693	0.009163	6700
10^4.5	L1	倒排	0.6758	0.004835	7928
10^5	L1	倒排	0.5775	0.003483	8895
10^3	L2	倒排	0.7030	0.015673	3569
10^3.5	L2	倒排	0.7048	0.014807	5352
10^4	L2	倒排	0.7120	0.009379	7038
10^4.5	L2	倒排	0.7710	0.004832	8208
10^5	L2	倒排	0.7453	0.003417	9169

从码本大小上分析，当码本越大时，索引数据库也越大，顺排平均索引时间也就越长，而倒排平均时间用时越短。

level	索引数据库大小/KB			
	L1 顺排	L2 顺排	L1 倒排	L2 倒排
3	696	699	3303	3569
3.5	1259	1264	5032	5352
4	2235	2246	6700	7038
4.5	3988	4013	7928	8208
5	7571	7623	8895	9169

level	平均检索时间/s			
	L1 顺排	L2 顺排	L1 倒排	L2 倒排
3	0.002236	0.002203	0.015673	0.015673
3.5	0.007001	0.007004	0.014929	0.014807
4	0.021568	0.021815	0.009163	0.009379
4.5	0.068406	0.067806	0.004835	0.004832
5	0.212822	0.211953	0.003483	0.003417

从索引方式上来看，倒排索引数据库比顺排的要大，而当码本大小逐渐变大时，差距逐渐减小。

从归一化方式上来讲，L_2 的归一方式比 L_1 具有相对稳定且更高的检索准确率。

level	准确率			
	L1 顺排	L2 顺排	L1 倒排	L2 倒排
3	0.5975	0.7030	0.2610	0.7030
3.5	0.5163	0.7048	0.4515	0.7048
4	0.4088	0.7120	0.5693	0.7120
4.5	0.3458	0.7710	0.6758	0.7710
5	0.3283	0.7453	0.5775	0.7453