Atividade 03

Com base no código de Image Retrieval apresentado durante a aula, modifique o código para que a imagem seja representada com os keypoints definidos em grid ao invés de usar detector de keypoints.

1. Use a função da última atividade prática para definir kepoints em grid e keypoints de forma aleatória.
2. Execute o código e calcule a acurácia média.
3. Compare a acurácia média com a acurácia média do código apresentado durante a aula.
4. Escreva uma pequena discussão (máximo 5 linhas) explicado o resultado encontrado.
5. Use a partição de validação para encontrar o melhor tamanho do grid através do teste do cotovelo.

In [1]:
```
pip install -U scikit-learn & pip install tqdm
```

```
Requirement already satisfied: tqdm in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (4.65.0)
Requirement already satisfied: scikit-learn in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: numpy>=1.17.3 in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (from scikit-learn) (1.24.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages (from scikit-learn) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

Functions for random and grid-based keypoint detection

Used in last activity

In [2]:
```
import random
import cv2 as cv


def random_keypoints(image, number_keypoints):
    keypoints =[]
    height, width = image.shape
    for i in range(number_keypoints):
        keypoint = cv.KeyPoint()
        h = random.randint(0, height - 1)
        w = random.randint(0, width - 1)
        keypoint.pt = (h,w)
        keypoint.size = 40
        keypoints.append(keypoint)
    return keypoints


def grid_keypoints(image, grid_size):
    keypoints = []
    height, width = image.shape
    keypoints = []
    for i in range(width):
        if(i%grid_size == 0):
            for j in range(height): #(0,0), (0,grid_size), ... , (grid_size,0), (grid_size,grid_size), ...
                if(j % grid_size == 0):
                    keypoint = cv.KeyPoint()
                    keypoint.pt = (i, j)
                    keypoint.size = grid_size/2
                    keypoints.append(keypoint)
    return keypoints
```

Code used in class with modifications

In [3]:
```
%matplotlib inline

import matplotlib.pyplot as plt

import cv2
import numpy as np

from tqdm import trange
import random
```

Image Display

In [4]:
```
def show_image_and_keypoints( image , kps ) :
    cv2.drawKeypoints( image, kps, image, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS )

    plt.figure(figsize = (10,10))
    plt.imshow(image, aspect='auto')
    plt.axis('off')
    plt.title('Keypoints and descriptors.')
    plt.show()
```

In [5]:
```
def show_top_images ( dataset_path, indices , id_test , ids , labels ) :

    label = (ids[id_test] - 1) // 80
    name = dataset_path + '/jpg/' + str(label) + '/image_' + str(ids[id_test]).zfill(4) + '.jpg'

    image = cv2.imread( name )
    image = cv2.cvtColor( image , cv2.COLOR_BGR2RGB )

    top = 0
    show_image_label(top, image, labels[id_test], ids[id_test] )

    accuracy = 0

    for i in indices[0] :
        label_i = labels[i]
        name = dataset_path + '/jpg/' + str(label_i) + '/image_' + str(ids[i]).zfill(4) + '.jpg'

        image = cv2.imread( name )
        image = cv2.cvtColor( image , cv2.COLOR_BGR2RGB )

        show_image_label(top, image, label_i, ids[i] )
        top = top + 1


def show_image_label ( top, image, label , image_id ) :

    plt.figure(figsize = (5,5))
    plt.imshow(image, aspect='auto')
    plt.axis('off')
    plt.title(f'{top} - Image id {image_id} with label {label}.')
    plt.show()
```

Generate descriptors

In [6]:
```
def detect_and_describe_keypoints ( image, algorithm_descriptor='orb', algorithm_detector='orb', grid_size=15) :

    image_gray = cv2.cvtColor( image , cv2.COLOR_BGR2GRAY )

    if algorithm_descriptor == 'sift' :
        keypoint = sift = cv2.xfeatures2d.SIFT_create()

    elif algorithm_descriptor == 'orb' :
        keypoint = cv2.ORB_create()

    #adding random and grid algorithms
    if algorithm_detector == 'sift' or algorithm_detector == 'orb':
        kps = keypoint.detect( image_gray, None )
    elif algorithm_detector == 'random':
        kps = random_keypoints(image_gray, 300)
    elif algorithm_detector == 'grid':
        kps = grid_keypoints(image_gray, grid_size)

    else :
        print('Error: algorithm not defined')
        return None

    # Describing Keypoints
    kps, descs = keypoint.compute( image_gray, kps )

    return kps, descs
```

In [7]:
```
def create_bovw_descriptors (image, dictionary, algorithm_descriptor='orb', algorithm_detector='orb', grid_size=15) :

    descs = detect_and_describe_keypoints( image, algorithm_descriptor, algorithm_detector, grid_size)[1]

    predicted = dictionary.predict(np.array(descs, dtype=np.double))

    desc_bovw = np.histogram(predicted, bins=range(0, dictionary.n_clusters+1))[0]

    return desc_bovw
```

Dictionary

In [8]:
```
from sklearn.cluster import MiniBatchKMeans
# from sklearn.cluster import KMeans

def create_dictionary_kmeans ( vocabulary , num_cluster ) :

    print( ' -> [I] Dictionary Info:\n',
        '\nTrain len: ', len(vocabulary),
        '\nDimension: ', len(vocabulary[0]),
        '\nClusters: ', num_cluster
        )

#     dictionary = KMeans( n_clusters=num_cluster )
    dictionary = MiniBatchKMeans( n_clusters=num_cluster, batch_size=1000)

    print ( 'Learning dictionary by Kmeans...')
    dictionary = dictionary.fit( vocabulary )
    print ( 'Done.')

    return dictionary
```

Data

In [9]:
```
import scipy.io
import tqdm

def create_vocabulary ( dataset_path , algorithm_descriptor='orb', algorithm_detector='orb', grid_size=15,
                      show_image=False ,  debug=False ) :

    mat = scipy.io.loadmat( dataset_path+'/datasplits.mat' )

    ids = mat['trn1'][0] #  'val1' or 'tst1'

    if algorithm_descriptor == 'orb' :
        train_descs = np.ndarray( shape=(0,32) , dtype=float )
    elif algorithm_descriptor == 'sift':
        train_descs = np.ndarray( shape=(0,128) , dtype=float )
    else :
        print('Error:Algorithm not defined.')
        return None
    cont = 0

    for id in tqdm.tqdm(ids, desc='Processing train set') :

        label = (id - 1) // 80
        name = dataset_path + '/jpg/' + str(label) + '/image_' + str(id).zfill(4) + '.jpg'

        image = cv2.imread( name )

        if image is None:
            print(f'Reading image Error. Path: {name}')
            return None

        kps, descs = detect_and_describe_keypoints ( image, algorithm_descriptor, algorithm_detector, grid_size )

        train_descs = np.concatenate((train_descs, descs), axis=0)

        if show_image :
            show_image_and_keypoints(image, kps)

        if debug :
            print( name )
            print( 'Number of keypoints: ', len(kps) )
            print( 'Number of images: ', len(ids) )
            print( 'Descriptor size: ', len(descs[0]) )
            print( type(descs[0]) )

    print( ' -> [I] Image Loader Info:\n',
      '\nTrain len: ', len(train_descs),
      '\nNumber of images: ', len(ids),
      '\nDescriptor size: ', len(descs[0])
      )

    return train_descs
```

In [10]:
```
def represent_dataset( dataset_path, dictionary , algorithm_descriptor='orb', algorithm_detector='orb', ids='tst1'
                     , grid_size=15 ) :


    mat = scipy.io.loadmat( dataset_path+'/datasplits.mat' )

    ids = mat[ids][0] #  'trn1' or 'val1'

    space = []
    labels = []

    for id in tqdm.tqdm(ids, desc='Processing train set') :

        label = (id - 1) // 80
        name = dataset_path + '/jpg/' + str(label) + '/image_' + str(id).zfill(4) + '.jpg'

        image = cv2.imread( name )

        desc_bovw = create_bovw_descriptors(image, dictionary, algorithm_descriptor, algorithm_detector, grid_size)

        space.append(desc_bovw)
        labels.append(label)

    print( ' -> [I] Space Describing Info:\n',
        '\nNumber of images: ', len(space),
        '\nNumber of labels: ', len(labels),
        '\nDimension: ', len(space[0])
        )

    return space , labels
```

Experimental evaluation

In [11]:
```
from sklearn.neighbors import NearestNeighbors

def run_test ( space , labels , dictionary , dataset_path,algorithm_descriptor='orb', algorithm_detector='orb',ids='tst1',
             grid_size=15, top=10 ) :
    knn = NearestNeighbors(n_neighbors=top+1).fit(space)

    mat = scipy.io.loadmat( dataset_path+'/datasplits.mat' )

    ids = mat[ids][0] #  'trn1' or 'val1'

    accuracy_t = 0

    for id_test in tqdm.tqdm(ids, desc='running the test phase') :

        label = (id_test - 1) // 80
        name = dataset_path + '/jpg/' + str(label) + '/image_' + str(id_test).zfill(4) + '.jpg'

        image = cv2.imread( name )

        desc_bovw = create_bovw_descriptors(image, dictionary, algorithm_descriptor, algorithm_detector, grid_size)

        indices = knn.kneighbors(desc_bovw.reshape(1, -1))[1]

        labels_top = [ labels[i] for i in indices[0] ]

        accuracy = sum( np.equal(labels_top, label) )
        accuracy =( (accuracy-1)/(top) ) * 100
        accuracy_t = accuracy_t + accuracy

    print(f'Average accuracy in the test set: {accuracy_t/len(ids):5.2f}%')
```

In [12]:
```
def retrieve_single_image ( space , labels , dictionary , dataset_path, algorithm_descriptor='orb', algorithm_detector='orb',
                          grid_size = 15 , top=10 ) :
    knn = NearestNeighbors(n_neighbors=top+1).fit(space)

    mat = scipy.io.loadmat( dataset_path+'/datasplits.mat' )

    ids = mat['tst1'][0] #  'trn1' or 'val1'

    id_test = random.randrange( len(ids) )

    label = (ids[id_test] - 1) // 80
    name = dataset_path + '/jpg/' + str(label) + '/image_' + str(ids[id_test]).zfill(4) + '.jpg'

    image = cv2.imread( name )

    desc_bovw = create_bovw_descriptors(image, dictionary, algorithm_descriptor, algorithm_detector, grid_size)

    distances, indices = knn.kneighbors(desc_bovw.reshape(1, -1))

    show_top_images(dataset_path, indices, id_test, ids, labels)

    labels_top = [ labels[i] for i in indices[0] ]

    accuracy = sum( np.equal( label , labels_top ) )
    accuracy =( (accuracy-1)/(top) ) * 100

    print(f'Accuracy for image id {ids[id_test]}: {accuracy:5.2f}%')

    print(name)
    print(f'Image: {ids[id_test]} with label {labels[id_test]}')
    print(f'Closest image: {ids[indices[0][0]]} with distance {distances[0][0]} and label {labels[indices[0][0]]}')
    print('Distances: ',distances)
    print('Indices: ',indices)
    print('Labels: ',labels_top)
```

Execution ORB detector and SIFT detector(during class)

In [13]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'sift'
algorithm_detector = 'sift'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector )
```

```
Processing train set:   0%|          | 0/680 [00:00<?, ?it/s][ WARN:0@16.414] global shadow_sift.hpp:13 SIFT_create DEPRECATED: cv.xfeatures2d.SIFT_create() is deprecated due SIFT tra
nfer to the main repository. https://github.com/opencv/opencv/issues/16736
Processing train set: 100%|██████████| 680/680 [01:58<00:00,  5.76it/s]
 -> [I] Image Loader Info:

Train len:  1233814
Number of images:  680
Descriptor size:  128
```

In [14]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,ids='tst1' )
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector, ids='tst1')
```

```
 -> [I] Dictionary Info:

Train len:  1233814
Dimension:  128
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:28<00:00, 11.80it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:49<00:00,  6.86it/s]
Average accuracy in the test set: 21.09%
```

In [15]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'orb'
algorithm_detector = 'orb'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector )
```

```
Processing train set: 100%|██████████| 680/680 [00:12<00:00, 53.88it/s]
 -> [I] Image Loader Info:

Train len:  333473
Number of images:  680
Descriptor size:  32
```

In [16]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,ids='tst1' )
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector, ids='tst1')
```

```
 -> [I] Dictionary Info:

Train len:  333473
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:03<00:00, 86.51it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:22<00:00, 14.97it/s]
Average accuracy in the test set: 17.00%
```

Execution with random detector

Selected 300 random keypoints for each image, like in short activity 2

In [17]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'orb'
algorithm_detector = 'random'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector )
```

```
Processing train set: 100%|██████████| 680/680 [00:04<00:00, 165.97it/s]
 -> [I] Image Loader Info:

Train len:  132363
Number of images:  680
Descriptor size:  32
```

In [18]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,ids='tst1' )
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector, ids='tst1')
```

```
 -> [I] Dictionary Info:

Train len:  132363
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:01<00:00, 211.89it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:07<00:00, 46.91it/s]
Average accuracy in the test set:  3.65%
```

Using RANDOM method to detect keypoints and ORB to describe them, with a number of cluster of 100, we were able to achieve an average accuracy of 3.65%

In [19]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'sift'
algorithm_detector = 'random'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector )
```

```
Processing train set: 100%|██████████| 680/680 [01:09<00:00,  9.79it/s]
 -> [I] Image Loader Info:

Train len:  204000
Number of images:  680
Descriptor size:  128
```

In [20]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,ids='tst1' )
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector, ids='tst1')
```

```
 -> [I] Dictionary Info:

Train len:  204000
Dimension:  128
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:21<00:00, 15.93it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:31<00:00, 10.95it/s]
Average accuracy in the test set: 17.15%
```

Using RANDOM method to detect keypoints and SIFT to describe them, with a number of cluster of 100, we were able to achieve an average accuracy of 17.15%

Execution with grid detector

Use the validation partition to find the best grid size through the elbow test

As the grid size increases, the number of detectable keypoints decreases. To perform the elbow test, I experimented with seven different grid sizes.

Grid Size	Accuracy
150x150	15.02%
100x100	16.21%
90x90	16.53%
60x60	17.03%
45x45	18.56%
30x30	16.62%
15x15	16.06%

Based on the results, it is evident that the highest accuracy is achieved with a grid size of 45x45. And, as we move beyond this size, the accuracy starts to decline. Therefore, we can infer that 45x45 is the optimal grid size.

In [21]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'sift'
algorithm_detector = 'grid'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector, grid_size=45 )
```

```
Processing train set: 100%|██████████| 680/680 [00:16<00:00, 41.12it/s]
 -> [I] Image Loader Info:

Train len:  119412
Number of images:  680
Descriptor size:  128
```

In [25]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,
                                  ids='tst1', grid_size=45)
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector,
         ids='tst1', grid_size=45)
```

```
 -> [I] Dictionary Info:

Train len:  119412
Dimension:  128
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:05<00:00, 61.48it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:16<00:00, 20.73it/s]
Average accuracy in the test set: 18.06%
```

Using GRID method to detect keypoints and SIFT to describe them, with a number of cluster of 100 and grid size of 45x45, we were able to achieve an average accuracy of 18.06%

In [26]:
```
dataset_path = 'flowers_classes'
algorithm_descriptor = 'orb'
algorithm_detector = 'grid'
vocabulary = create_vocabulary( dataset_path, algorithm_descriptor, algorithm_detector, grid_size=45 )
```

```
Processing train set: 100%|██████████| 680/680 [00:03<00:00, 218.31it/s]
 -> [I] Image Loader Info:

Train len:  89000
Number of images:  680
Descriptor size:  32
```

In [27]:
```
num_clusters = 100
dictionary = create_dictionary_kmeans( vocabulary , num_clusters )
space, labels = represent_dataset ( dataset_path , dictionary, algorithm_descriptor, algorithm_detector,
                                  ids='tst1', grid_size=45)
run_test ( space , labels , dictionary , dataset_path, algorithm_descriptor, algorithm_detector,
         ids='tst1', grid_size=45)
```

```
 -> [I] Dictionary Info:

Train len:  89000
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
/home/luisa/Documents/faculdade/optativas/inf 492/inf492/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Done.
Processing train set: 100%|██████████| 340/340 [00:01<00:00, 283.31it/s]
 -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
running the test phase: 100%|██████████| 340/340 [00:04<00:00, 81.18it/s]
Average accuracy in the test set:  8.12%
```

Using GRID method to detect keypoints and SIFT to describe them, with a number of cluster of 100 and grid size of 45x45, we were able to achieve an average accuracy of 8.12%

Conclusion

The accuracy achieved through the SIFT and ORB detectors used in class is significantly higher than that achieved through the random and grid detector methods. This can be attributed to the fact that the latter methods often fail to identify a unique, informative, and unambiguous part of the image as a keypoint. Moreover, the SIFT descriptor consistently outperforms the ORB descriptor with all keypoint detection methods.