```
%matplotlib inline

import matplotlib.pyplot as plt

import cv2
import numpy as np

from tqdm import trange
import random

from skimage.feature import local_binary_pattern as lbp


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_
```

Image Display

```
def show_image( image ) :
    plt.figure(figsize = (10,10))
    plt.imshow(image, aspect='auto')
    plt.axis('off')
    plt.show()


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()


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

```
def detect_and_describe_keypoints ( image, algorithm='orb' ) :

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

    y_size, x_size, _  = image.shape

    kps = []

    if algorithm == 'sift' :
        descriptor = sift = cv2.SIFT_create()
        kps = sift.detect( image_gray, None )
    elif algorithm == 'orb' :
        descriptor = orb = cv2.ORB_create()
        kps = orb.detect( image_gray, None )
    else :
        descriptor = cv2.ORB_create()

        if algorithm == 'random':
          for i in range(3000):
            keypoint = cv2.KeyPoint()
            keypoint.pt = (random.randint(0, x_size), random.randint(0, y_size))
            keypoint.size = 40
            kps.append(keypoint)
        elif algorithm == 'grid':
          x_grid = x_size // 50
          y_grid = y_size // 50

          for i in range(0, x_size, x_grid):
              for j in range(0, y_size, y_grid):
                  keypoint = cv2.KeyPoint()
                  keypoint.pt = (i + x_grid // 2, j + y_grid // 2)
                  keypoint.size = 40
```

```
                    kps.append(keypoint)
        else:
          print('Error:Algorithm not defined.')
          return None

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

    return kps, descs


def create_bovw_descriptors (image, dictionary, algorithm='orb') :

    descs = detect_and_describe_keypoints( image, algorithm=algorithm )[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


def local_binary_pattern(image, radius=3, points=8, method='uniform'):

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

  _lbp = lbp(image, points, radius, method=method)

  descs, bins = np.histogram(_lbp.ravel(), bins=np.arange(257))

  descs = np.expand_dims(descs, axis=0)

  return descs


def create_lbp_descriptors(image, dictionary) :

    descs = local_binary_pattern( image )
```

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

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

    return desc_lbp


from sklearn.cluster import MiniBatchKMeans

def create_dictionary_kmeans ( vocabulary , num_cluster ) :

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

```
import scipy.io
import tqdm

def create_vocabulary ( dataset_path , representation='bovw', algorithm='orb', show_image=False , debug=False ) :

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

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

if representation == 'lbp':
    train_descs = np.ndarray( shape=(0,256) , dtype=float )
elif algorithm == 'orb' :
    train_descs = np.ndarray( shape=(0,32) , dtype=float )
elif algorithm == 'sift' :
    train_descs = np.ndarray( shape=(0,128) , dtype=float )
else:
    train_descs = np.ndarray( shape=(0,32) , dtype=float )

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

    if representation == 'lbp':
      descs = local_binary_pattern(image)
      train_descs = np.concatenate((train_descs, descs), axis=0)

      if show_image:
        show_image(image)
    elif representation == 'bovw':

      kps, descs = detect_and_describe_keypoints ( image, algorithm=algorithm )
      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]) )
            else:
              print('Error: Representation not defined.')
              return None

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

        return train_descs
```

Dictionary

```
def represent_dataset( dataset_path, dictionary , representation='bovw', algorithm='orb' ) :

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

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

    space = []
    labels = []

    for id in tqdm.tqdm(ids, desc='Processing test 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

            if representation == 'lbp':
                desc = create_lbp_descriptors(image, dictionary)
            elif representation == 'bovw':
              desc = create_bovw_descriptors(image, dictionary, algorithm=algorithm)
            else:
              print('Error: Representation not defined.')
              return None

            space.append(desc)
            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


    from sklearn.neighbors import NearestNeighbors

    def run_test ( space , labels , dictionary , dataset_path, representation='bovw', algorithm='orb', top=10 ) :
        knn = NearestNeighbors(n_neighbors=top+1).fit(space)

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

        ids = mat['tst1'][0] #  'tst1' or '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 )

        if representation == 'lbp':
            desc = create_lbp_descriptors(image, dictionary)
        elif representation == 'bovw':
          desc = create_bovw_descriptors(image, dictionary, algorithm=algorithm)
        else:
          print('Error: Representation not defined.')
          return None

        indices = knn.kneighbors(desc.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}%')
```

Experimental evaluation

```
def retrieve_single_image ( space , labels , dictionary , dataset_path, representation='bovw', algorithm='orb', 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 )

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

if representation == 'lbp':
    desc = create_lbp_descriptors(image, dictionary)
elif representation=='bovw':
    desc = create_bovw_descriptors(image, dictionary, algorithm=algorithm)
else:
  print('Error: Representation not defined.')
  return None

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

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

labels_top = [ int(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[0])
print('Labels: ',labels_top)
```

```
print( Labels: ",labels_top)
```

Sift

```
dataset_path = '/content/drive/MyDrive/Disciplinas/8º Período - 25 1/INF492/practices/practice-1/flowers_classes'
algorithm = 'sift'
num_cluster = 100

vocabulary = create_vocabulary( dataset_path, algorithm=algorithm )

dictionary = create_dictionary_kmeans( vocabulary , num_cluster=num_cluster )

space, labels = represent_dataset ( dataset_path , dictionary, algorithm=algorithm )
```

```
Processing train set: 100%|██████████| 680/680 [05:54<00:00,  1.92it/s]
 -> [I] Image Loader Info:

Train len:  1233814
Number of images:  680
Descriptor size:  128
 -> [I] Dictionary Info:

Train len:  1233814
Dimension:  128
Clusters:  100
Learning dictionary by Kmeans...
Done.
Processing test set: 100%|██████████| 340/340 [01:29<00:00,  3.79it/s] -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
```

```
run_test( space, labels, dictionary, dataset_path, algorithm=algorithm )
```

```
running the test phase: 100%|██████████| 340/340 [01:25<00:00,  3.98it/s]Average accuracy in the test set: 20.94%
```

```
retrieve_single_image( space, labels, dictionary, dataset_path , algorithm=algorithm)
```

Orb

```
dataset_path = '/content/drive/MyDrive/Disciplinas/8º Período - 25 1/INF492/practices/practice-1/flowers_classes'# ***
algorithm = 'orb'
num_cluster = 100

vocabulary = create_vocabulary( dataset_path, algorithm=algorithm )

dictionary = create_dictionary_kmeans( vocabulary , num_cluster=num_cluster )

space, labels = represent_dataset ( dataset_path , dictionary, algorithm=algorithm )
```

```
Processing train set: 100%|██████████| 680/680 [00:31<00:00, 21.61it/s]
 -> [I] Image Loader Info:

Train len:  333473
Number of images:  680
Descriptor size:  32
 -> [I] Dictionary Info:

Train len:  333473
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
Done.
Processing test set: 100%|██████████| 340/340 [00:10<00:00, 31.16it/s] -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
```

```
Dimension:  100
```

```
run_test( space, labels, dictionary, dataset_path, algorithm=algorithm )
```

```
running the test phase: 100%|██████████| 340/340 [00:10<00:00, 32.79it/s]Average accuracy in the test set: 18.03%
```

```
retrieve_single_image( space, labels, dictionary, dataset_path , algorithm=algorithm)
```

Random

```
dataset_path = '/content/drive/MyDrive/Disciplinas/8º Período - 25 1/INF492/practices/practice-1/flowers_classes'
algorithm = 'random'
num_cluster = 100

vocabulary = create_vocabulary( dataset_path, algorithm=algorithm )

dictionary = create_dictionary_kmeans( vocabulary , num_cluster=num_cluster )

space, labels = represent_dataset ( dataset_path , dictionary, algorithm=algorithm )
```

```
Processing train set: 100%|██████████| 680/680 [01:11<00:00,  9.58it/s]
 -> [I] Image Loader Info:

Train len:  1603815
Number of images:  680
Descriptor size:  32
 -> [I] Dictionary Info:

Train len:  1603815
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
Done
```

```
Done.
Processing test set: 100%|██████████| 340/340 [00:15<00:00, 22.28it/s] -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
```

```
run_test( space, labels, dictionary, dataset_path, algorithm=algorithm )
```

```
running the test phase: 100%|██████████| 340/340 [00:12<00:00, 28.06it/s]Average accuracy in the test set: 11.94%
```

```
retrieve_single_image( space, labels, dictionary, dataset_path , algorithm=algorithm)
```

Grid 50x50

```
dataset_path = '/content/drive/MyDrive/Disciplinas/8º Período - 25 1/INF492/practices/practice-1/flowers_classes'
algorithm = 'grid'
num_cluster = 100

vocabulary = create_vocabulary( dataset_path, algorithm=algorithm )

dictionary = create_dictionary_kmeans( vocabulary , num_cluster=num_cluster )

space, labels = represent_dataset ( dataset_path , dictionary, algorithm=algorithm )
```

```
Processing train set: 100%|██████████| 680/680 [01:01<00:00, 10.97it/s]
 -> [I] Image Loader Info:

Train len:  1406770
Number of images:  680
Descriptor size:  32
 -> [I] Dictionary Info:
```

```
Train len:  1406770
Dimension:  32
Clusters:  100
Learning dictionary by Kmeans...
Done.
Processing test set: 100%|██████████| 340/340 [00:06<00:00, 53.06it/s] -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
```

```
run_test( space, labels, dictionary, dataset_path, algorithm=algorithm )
```

```
running the test phase: 100%|██████████| 340/340 [00:08<00:00, 38.97it/s]Average accuracy in the test set: 12.00%
```

```
retrieve_single_image( space, labels, dictionary, dataset_path , algorithm=algorithm)
```

LBP

```
dataset_path = '/content/drive/MyDrive/Disciplinas/8º Período - 25 1/INF492/practices/practice-1/flowers_classes'

# Fui revisar a atividade e fiquei na dúvida se seria necessário realizar essa etapa do vocabulary e do dictionary para
vocabulary = create_vocabulary( dataset_path, representation='lbp' )

dictionary = create_dictionary_kmeans( vocabulary , num_cluster=num_cluster )

space, labels = represent_dataset ( dataset_path , dictionary, representation='lbp' )
```

```
Processing train set: 100%|██████████| 680/680 [00:57<00:00, 11.81it/s]
 -> [I] Image Loader Info:

Train len:  680
```

```
Train len:  680
Number of images:  680
Descriptor size:  256
 -> [I] Dictionary Info:

Train len:  680
Dimension:  256
Clusters:  100
Learning dictionary by Kmeans...
Done.
Processing test set: 100%|██████████| 340/340 [00:25<00:00, 13.44it/s] -> [I] Space Describing Info:

Number of images:  340
Number of labels:  340
Dimension:  100
```

```
run_test( space, labels, dictionary, dataset_path, representation='lbp' )
```

```
running the test phase: 100%|██████████| 340/340 [00:30<00:00, 11.28it/s]Average accuracy in the test set:  7.00%
```

```
retrieve_single_image( space, labels, dictionary, dataset_path , representation='lbp')
```

Discussão

SIFT e FAST obtiveram os melhores resultados, o que faz sentido considerando que eles obtém informações locais mais detalhadas. Já as abordagens Random e Grid apresentaram resultados piores, fato que se justifica por esses métodos selecionarem muitos keypoint que podem não agregar nenhuma informação útil. E o teste com LBP gerou os piores resultados, possivelmente pelo fato de ele seguir uma abordagem global, tornando-se sensível a estruturas complexas nas imagens.