train_svm.ipynb

[ ]:
```
%run helper.py
%run utils.py
%run lab3_utils.py
```

[ ]:
```
import os
import cv2
import argparse
from sklearn.svm import LinearSVC
from skimage import feature
import pandas as pd
import os
from skimage.transform import resize
from skimage.io import imread
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
train_image_paths = []
train_labels = []
# get all the image folder paths
Categories=['american_football', 'baseball', 'basketball', 'billiard_ball',␣
↪'bowling_ball', 'cricket_ball', 'football', 'golf_ball',
            'hockey_ball', 'hockey_puck', 'rugby_ball', 'shuttlecock',␣
↪'table_tennis_ball', 'tennis_ball', 'volleyball']
datadir='dataset/train'

for i in Categories:

    print(f'loading... category : {i}')
    path=os.path.join(datadir,i)
    for img in os.listdir(path):
        train_image_paths.append(os.path.join(path,img))
        train_labels.append(Categories.index(i))
    print(f'loaded category:{i} successfully')
```

[ ]:
```
test_image_paths = []
test_labels = []
# get all the image folder paths
Categories=['american_football', 'baseball', 'basketball', 'billiard_ball',␣
↪'bowling_ball', 'cricket_ball', 'football', 'golf_ball',
            'hockey_ball', 'hockey_puck', 'rugby_ball', 'shuttlecock',␣
↪'table_tennis_ball', 'tennis_ball', 'volleyball']
datadir='dataset/test'

for i in Categories:
```

```
    print(f'loading... category : {i}')
    path=os.path.join(datadir,i)
    for img in os.listdir(path):
        test_image_paths.append(os.path.join(path,img))
        test_labels.append(Categories.index(i))
    print(f'loaded category:{i} successfully')
```

[ ]:
```
import os

import cv2 as cv
import joblib
import numpy as np
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

from lab3_utils import get_image_paths
#from utils import *
from utils import read_img

#Implements a Bag of Words (BoW) approach using SIFT features for image␣
↪classification, using a Support Vector Machine (SVM) classifier.

SIFT_MAX_FEATURES = 300 #Change this parameter to 100, 200, 300, 400 and delete␣
↪this feature to perform the hyperparameter tuning

IMAGE_CATEGORIES = ['american_football', 'baseball', 'basketball',␣
↪'billiard_ball', 'bowling_ball', 'cricket_ball', 'football', 'golf_ball',
           'hockey_ball', 'hockey_puck', 'rugby_ball', 'shuttlecock',␣
↪'table_tennis_ball', 'tennis_ball', 'volleyball']

# build a codebook by clustering feature descriptors extracted from a set of␣
↪images.
# Iterate through each image in image_paths.
# For each image, extract SIFT features descriptors and concatenate the␣
↪extracted descriptors into a container array.
# Use KMeans clustering to group the descriptors into clusters (default is 150),␣
↪effectively creating the codebook.

def build_codebook(image_paths, num_tokens=150): #Change num_tokens to 8, 50,␣
↪100, 150, 200 to perform the hyperparameter tuning
    sift = cv.SIFT_create(nfeatures  = SIFT_MAX_FEATURES)
    container = []
    for image_path in image_paths:
```

```
        img = read_img(image_path, mono=True)
        keypoints, descriptors = sift.detectAndCompute(img, None)
        if descriptors is not None:
            container.append(descriptors)
    container = np.concatenate(container)
    print(container.shape)
    print('Training KMeans...')
    kmeans = KMeans(n_clusters=num_tokens)
    kmeans.fit(container)
    print('Done')
    return kmeans.cluster_centers_


def bag_of_words(image_paths, codebook):
    sift = cv.SIFT_create(nfeatures  = SIFT_MAX_FEATURES)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        keypoints, descriptors = sift.detectAndCompute(img, None)
        bow = np.zeros(codebook_size)
        if descriptors is not None:
            distances = cdist(descriptors, codebook)
            for d in distances:
                bow[np.argmin(d)] += 1
        image_features.append(bow.reshape(1, codebook_size))
    image_features = np.concatenate(image_features)
    return image_features

# If a pre-trained codebook exists, it is loaded. Otherwise, a new codebook is␣
 ↪built and persisted to disk
if os.path.exists('Saved_Models_SVM/codebook_SIFT.joblib'):
    codebook = joblib.load('Saved_Models_SVM/codebook_SIFT.joblib')
else:
    codebook = build_codebook(train_image_paths)
    print('Persisting codebook...')
    joblib.dump(codebook, 'Saved_Models_SVM/codebook_SIFT.joblib')
    print('Done')

# Standardize features by setting the mean as 0 and scaling to unit variance.
scaler = StandardScaler()

print('Generating BOW features for training set...')
train_images = bag_of_words(train_image_paths, codebook)
train_images_scaled = scaler.fit_transform(train_images)
print('Train images:', train_images.shape)
```

```
print('Generating BOW features for test set...')
test_images = bag_of_words(test_image_paths, codebook)
test_images_scaled = scaler.transform(test_images)
print('Test images:', test_images.shape)

if os.path.exists('Saved_Models_SVM/svm_bow_SIFT.joblib'):
    print('Loading existing linear SVM model...')
    svm = joblib.load('Saved_Models_SVM/svm_bow_SIFT.joblib')
else:
    print('Training a linear SVM...')
    svm = SVC(gamma='scale')
    svm.fit(train_images_scaled, train_labels)
    joblib.dump(svm, 'Saved_Models_SVM/svm_bow_SIFT.joblib')
print('Done')
```

[ ]:
```
# calculates the classification accuracy of the SVM classifier on the test set.

test_predictions = svm.predict(test_images_scaled)
accuracy = accuracy_score(test_labels, test_predictions)
print('Classification accuracy of SVM with BOW and SIFT features:', accuracy)
```

[ ]:
```
#calculates the F1 score of the SVM classifier on the test set specifying␣
 ↪average='macro' to compute the score for each class and then take the␣
 ↪unweighted average.

from sklearn.metrics import f1_score
f1_score(test_labels, test_predictions, average='macro')
```

[ ]:
```

```