svm_perturbations_and_evaluation.ipynb

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

[ ]:
```
import os
import cv2 as cv
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
from utils import read_img
from sklearn.metrics import f1_score
import joblib
from scipy.spatial.distance import cdist
from sklearn.preprocessing import StandardScaler
import helper
from sklearn.metrics import f1_score
```

[ ]:
```
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')
```

[ ]:

```
# generates Bag-of-Words features using SIFT descriptors for a given list of␣
 ↪image paths and a pre-computed codebook (visual vocabulary).

# initializes a SIFT object, Iterates through each image path, Reads the image,␣
 ↪converts it to grayscale, and detects keypoints and computes SIFT descriptors.
# For each descriptor, it computes distances to the codebook centroids and␣
 ↪assigns it to the nearest centroid.
# Accumulates the histogram of visual words (BoW) for each image. Returns a␣
 ↪matrix of BoW features for all images

scaler = StandardScaler() #scale the features before feeding them to the SVM␣
 ↪classifier
def bag_of_words_SIFT(image_paths, codebook):
    sift = cv.SIFT_create(nfeatures=300)
    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
```

[ ]:

```
codebook_SIFT = joblib.load('Saved_Models_SVM/codebook_SIFT.joblib')
svm_SIFT = joblib.load('Saved_Models_SVM/svm_bow_SIFT.joblib')
```

[ ]:

```
#compute BoW features for the training images using SIFT descriptors and the␣
 ↪pre-computed codebook. Scales the BoW features.
```

```
print('Generating BOW features for training set...')
train_images_SIFT = bag_of_words_SIFT(train_image_paths, codebook_SIFT)
train_images_scaled_SIFT = scaler.fit_transform(train_images_SIFT)
print('Train images:', train_images_SIFT.shape)
```

[ ]:

```
# Initializes a SIFT object, Loops over each image path, Reads the image,␣
 ↪converts it to grayscale, and applies Gaussian pixel noise with increasing␣
 ↪intensity (i).
# Normalizes the pixel values to the range [0, 255], Detects keypoints and␣
 ↪computes SIFT descriptors for the noisy image,
# Computes BoW features for the image by assigning each descriptor to the␣
 ↪nearest centroid in the codebook and accumulating the counts,
# Appends the computed BoW features for the image to the image_features list and␣
 ↪ After processing all images, concatenates the BoW features into a single␣
 ↪array.

def bag_of_words_SIFT_GPN(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.gaussian_pixel_noise(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

[ ]:

```
#list l containing intensity values for Gaussian pixel noise.

l = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

[ ]:

```
#creates a pandas DataFrame df from the accuracies stored in the list res, with␣
 ↪the intensity levels of Gaussian pixel noise (l) as the index
#DataFrame provides a structured representation of accuracies corresponding to␣
 ↪different intensity levels of Gaussian pixel noise,
#making it easier to analyze and visualize the results to understand how the␣
 ↪performance of the SVM classifier varies with different levels of noise.

res = []
for i in range(len(l)):
    print(l[i])
    test_images = bag_of_words_SIFT_GPN(test_image_paths, codebook_SIFT, l[i])
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Gaussian pixel noise ACCURACY.csv")
```

[ ]:

```
def bag_of_words_SIFT_GB(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.gaussian_blur(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

[ ]:

```
res = []
for i in range(10):
    print(i)
    test_images = bag_of_words_SIFT_GB(test_image_paths, codebook_SIFT, i)
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Gaussian blurring ACCURACY.csv")
df
```

[ ]:

```
#Detects keypoints and computes descriptors using SIFT on the contrast-scaled␣
 ↪image, Computes the BoW histogram using the codebook
#and distances between descriptors and codebook centroids, populates the BoW␣
 ↪histogram for each image and Returns the concatenated BoW histograms of all␣
 ↪images

def bag_of_words_SIFT_ICI(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.scale_contrast(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

[ ]:

```
l = [1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.1, 1.15, 1.20, 1.25]
```

[ ]:

```
res = []
for i in range(len(l)):
    print(l[i])
    test_images = bag_of_words_SIFT_ICI(test_image_paths, codebook_SIFT, l[i])
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Image Contrast Increase ACCURACY.csv")
df
```

[ ]:

```
l = [ 1.0, 0.95, 0.90, 0.85, 0.80, 0.60, 0.40, 0.30, 0.20, 0.10]
```

[ ]:

```
res = []
for i in range(len(l)):
    print(l[i])
    test_images = bag_of_words_SIFT_ICI(test_image_paths, codebook_SIFT, l[i])
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Image Contrast Decrease ACCURACY.csv")
df
```

[ ]:

```
def bag_of_words_SIFT_IB(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.change_brightness(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

[ ]:

```
l = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

[ ]:

```
res = []
for i in range(len(l)):
    print(l[i])
    test_images = bag_of_words_SIFT_IB(test_image_paths, codebook_SIFT, l[i])
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Image Brightness Increase ACCURACY.
 ↪csv")
df
```

[ ]:

```
def bag_of_words_SIFT_IB(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.change_brightness(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

```
[ ]: l = [0, -5, -10, -15, -20, -25, -30, -35, -40, -45]
```

```
[ ]: res = []
     for i in range(len(l)):
         print(l[i])
         test_images = bag_of_words_SIFT_IB(test_image_paths, codebook_SIFT, l[i])
         test_images_scaled = scaler.transform(test_images)
         test_predictions = svm_SIFT.predict(test_images_scaled)
         res.append(accuracy_score(test_labels, test_predictions))
     d = {'Accuracy': res}
     df = pd.DataFrame(data=d, index = l)
     df.to_csv("Robustness_Testing_Results_SVM/Image Brightness Decrease ACCURACY.
      ↪csv")
     df
```

```
[ ]: def bag_of_words_SIFT_OII(image_paths, codebook, i):
         sift = cv.SIFT_create(nfeatures=300)
         codebook_size = codebook.shape[0]
         image_features = []
         for image_path in image_paths:
             img = read_img(image_path, mono=True)
             img = helper.occlusion(img, i)
             img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
             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
```

```
[ ]: l = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

```
[ ]: res = []
     for i in range(len(l)):
         print(l[i])
         test_images = bag_of_words_SIFT_OII(test_image_paths, codebook_SIFT, l[i])
         test_images_scaled = scaler.transform(test_images)
         test_predictions = svm_SIFT.predict(test_images_scaled)
         res.append(accuracy_score(test_labels, test_predictions))
     d = {'Accuracy': res}
     df = pd.DataFrame(data=d, index = l)
     df.to_csv("Robustness_Testing_Results_SVM/Occlusion of the Image Increase␣
      ↪ACCURACY.csv")
     df
```

[ ]:

```
def bag_of_words_SIFT_SPN(image_paths, codebook, i):
    sift = cv.SIFT_create(nfeatures=300)
    codebook_size = codebook.shape[0]
    image_features = []
    for image_path in image_paths:
        img = read_img(image_path, mono=True)
        img = helper.salt_and_pepper(img, i)
        img = cv.normalize(img, None, 0, 255, cv.NORM_MINMAX).astype('uint8')
        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
```

[ ]:

```
l =[0.00, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18]
```

[ ]:

```
res = []
for i in range(len(l)):
    print(l[i])
    test_images = bag_of_words_SIFT_SPN(test_image_paths, codebook_SIFT, l[i])
    test_images_scaled = scaler.transform(test_images)
    test_predictions = svm_SIFT.predict(test_images_scaled)
    res.append(accuracy_score(test_labels, test_predictions))
d = {'Accuracy': res}
df = pd.DataFrame(data=d, index = l)
df.to_csv("Robustness_Testing_Results_SVM/Salt and Pepper Noise ACCURACY.csv")
df
```