Aim: To Creating and Training an Object Detector

Objective: Bag of Words BOW in computer version Detecting cars in a scene.

Theory :

Creating and Training an object detector:-

Using built-in features makes it easy to come up with a quick prototype for an application. and we're all very grateful to the OpenCV developers for making great features, such as face detection or people detection readily available (truly, we are). However, whether you are a hobbyist or a computer vision professional, it's unlikely that you will only deal with people and faces:

Bag-of –words:-

Bag-of-words (BOW) is a concept that was not mitially intended for computer vision, rather, we use an evolved version of this concept in the context of computer vision. So, let's first talk about its basic version, which-as you may have guessed-originally belongs to the field of language analysis and information retrieval. BOW is the technique by which we assign a count weight to each word in a series of documents; we then represent these documents with vectors that represent these
set of counts. Let's look at an example:

Document 1: like OpenCV and I like Python
Document 2: like C++ and Python
Document 3: don't like artichokes

BOW in Computer Vision :-

We are by now familiar with the concept of image features. We've used feature extractors, such as SIFT, and SURF, to extract features from images so that we could match these features in another image. We've also familiarized ourselves with the concept of codebook, and we know about SVM, a model that can be fed a set of features and utilizes complex algorithms to classify train data, and can predict the classification of new data.

So, the implementation of a BOW approach will involve the following steps:

1. Take a sample dataset.

2. For each image in the dataset, extract descriptors (with SIFT, SURF, and so on).

3. Add each descriptor to the BOW trainer.

4. Cluster the descriptors to k clusters (okay, this sounds obscure, but bear with me) whose centers (centroids) are our visual words.

Detecting ears

There is no virtual limit to the type of objects you can detect in your images and videos. However, to obtain an acceptable level of accuracy, you need a sufficiently large dataset. containing train images that are identical in size. This would be a time-consuming operation if we were to do it all by ourselves

Example – car detection in a scene

We are now ready to apply all the concepts we leamed so far to a real-life example, and create a car detector application that scans an image and draws rectangles around cars.

Let's summarize the process before diving into the code:

1. Obtain a train dataset.

2. Create a BOW trainer and create a visual vocabulary.

3. Train an SVM with the vocabulary.

4. Attempt detection using sliding windows on an image pyramid of a test image.

5. Apply non-maximum suppression to overlapping boxes.

6. Output the result.

Code :-

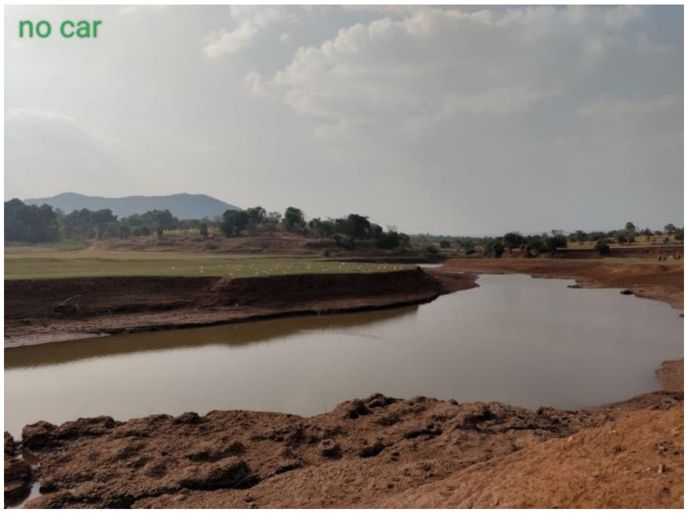
```
import cv2
import numpy as np
import os

if not os.path.isdir('CarData'):
    exit(1)

BOW_NUM_TRAINING_SAMPLES_PER_CLASS = 10
SVM_NUM_TRAINING_SAMPLES_PER_CLASS = 110

BOW_NUM_CLUSTERS = 40

sift = cv2.SIFT_create()

FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)

bow_kmeans_trainer = cv2.BOWKMeansTrainer(BOW_NUM_CLUSTERS)
bow_extractor = cv2.BOWImgDescriptorExtractor(sift, flann)

def get_pos_and_neg_paths(i):
    pos_path = 'CarData/TrainImages/pos-%d.pgm' % (i+1)
    neg_path = 'CarData/TrainImages/neg-%d.pgm' % (i+1)
    return pos_path, neg_path

def add_sample(path):
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    keypoints, descriptors = sift.detectAndCompute(img, None)
    if descriptors is not None:
        bow_kmeans_trainer.add(descriptors)

for i in range(BOW_NUM_TRAINING_SAMPLES_PER_CLASS):
    pos_path, neg_path = get_pos_and_neg_paths(i)
    add_sample(pos_path)
    add_sample(neg_path)

voc = bow_kmeans_trainer.cluster()
bow_extractor.setVocabulary(voc)
```

```
def extract_bow_descriptors(img):
    features = sift.detect(img)
    return bow_extractor.compute(img, features)

training_data = []
training_labels = []
for i in range(SVM_NUM_TRAINING_SAMPLES_PER_CLASS):
    pos_path, neg_path = get_pos_and_neg_paths(i)
    pos_img = cv2.imread(pos_path, cv2.IMREAD_GRAYSCALE)
    pos_descriptors = extract_bow_descriptors(pos_img)
    if pos_descriptors is not None:
        training_data.extend(pos_descriptors)
        training_labels.append(1)
    neg_img = cv2.imread(neg_path, cv2.IMREAD_GRAYSCALE)
    neg_descriptors = extract_bow_descriptors(neg_img)
    if neg_descriptors is not None:
        training_data.extend(neg_descriptors)
        training_labels.append(-1)

svm = cv2.ml.SVM_create()

svm.train(np.array(training_data), cv2.ml.ROW_SAMPLE,
          np.array(training_labels))

for test_img_path in ['CarData/TestImages/test-0.pgm',
                      'CarData/TestImages/test-1.pgm',
                      'images/car.jpg',
                      'images/haying.jpg',
                     ]:
    img = cv2.imread(test_img_path)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    descriptors = extract_bow_descriptors(gray_img)
    prediction = svm.predict(descriptors)
    if prediction[1][0][0] == 1.0:
        text = 'car'
        color = (0, 255, 0)
    else:
        text = 'not car'
        color = (0, 0, 255)
    cv2.putText(img, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
                color, 2, cv2.LINE_AA)
    cv2.imshow(test_img_path, img)
cv2.waitKey(0)
```

OUTPUT :-

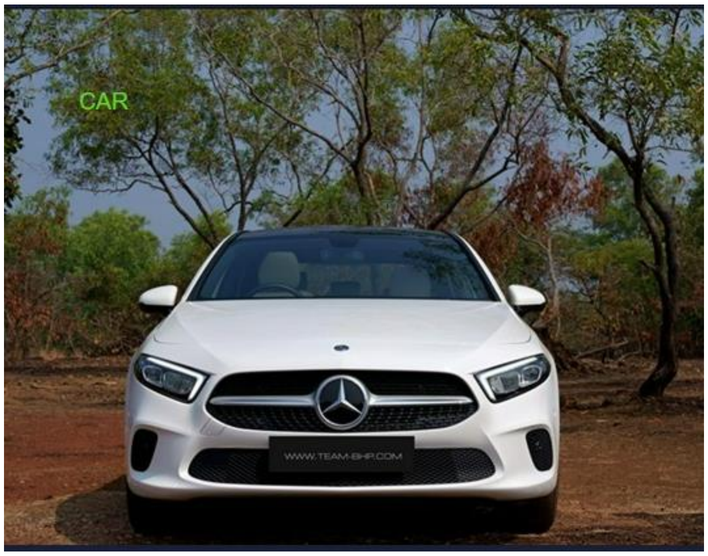
Input Image :-

Output Image :-



Input Image :-

Output Image :-



Conclusion :-

Object recognition is steps to collect the related data for identifying object in digital organization.

BOM model create a histogram of Bow model creates a visual wordy that return an image then this histogram is used to train.