Vidyavardhini's College of Engineering & Technology
Department of Computer Engineering

Experiment No. 9
To Perform Creating and Training an Object Detector
Name of Student: - 09_Amruta Poojary
Date of Performance: 4/10/2023
Date of Submission: 11/10/2023

Aim: To Perform 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 a pre-trained detector makes it easy to build a quick prototype and OpenCv provides a readily available face detection and people detection functionality. However in the industry we may be tasked to deal with problems of detecting very specific objects, such as registration plates, book covers or whatever thing may be most important to your employer or client. Thus, the question is, how do we come up with our own classifiers? There are many popular approaches and one answer lies in SVMs and the BoW technique.

Bag of words: -

These vectors can be conceptualized as a histogram representation of documents or as a descriptor vector that can be used to train classifiers. For example, a document can be classified as spam or not spam based on such a representation. Indeed, spam filtering is one of the many real-world applications of BoW.

BOW in computer vision: -

We can take the following approach to build a classifier:-

1. Take a sample dataset of images.

2. For each image in the dataset, extract descriptors (with SIFT, SURF, ORB, or a similar algorithm).

3. Add each descriptor vector to the BoW trainer.

4. Cluster the descriptors into k clusters whose centers (centroids) are our visual words.

At the end of this process, we have a dictionary of visual words ready to be used. As you can imagine, a large dataset will help make our dictionary richer in visual words. Up to a point, the more words, the better! Having trained a classifier, we should proceed to test it. The good news is that the test process is conceptually very similar to the training process outlined previously. Given a test image, we can extract descriptors and quantize them (or reduce their dimensionality) by calculating a histogram of their distances to the centroids. Based on this, we can attempt to recognize visual words, and locate them in the image.

Detecting cars: -

Detecting cars using the Bag of Words (BoW) approach in computer vision involves a step-by-step process. First, you gather a dataset containing both car and non-car images with proper labels. Then, you extract local features, such as Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT), from the images. Subsequently, you create a vocabulary of visual words by clustering these features, often using K-means. Once the vocabulary is established, you encode the features of each image using it, creating histograms of visual words. Afterward, you train a machine learning classifier, such as a Support Vector

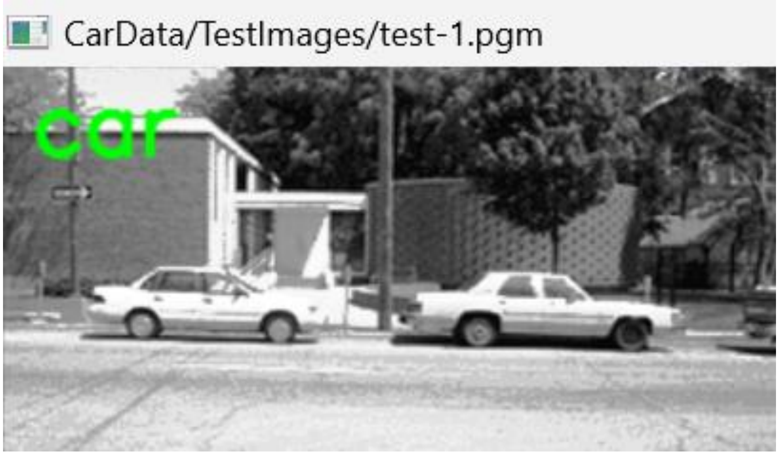
Machine or Random Forest, on the encoded features to distinguish between car and non-car images. Evaluation is crucial, which involves metrics like accuracy and F1-score. While BoW can be effective for object detection, it's worth noting that modern deep learning techniques, especially Convolutional Neural Networks (CNNs), have largely surpassed BoW in terms of accuracy and are more suitable for complex tasks like car detection. Nevertheless, BoW can still be useful in simpler scenarios or as a learning exercise in computer vision.

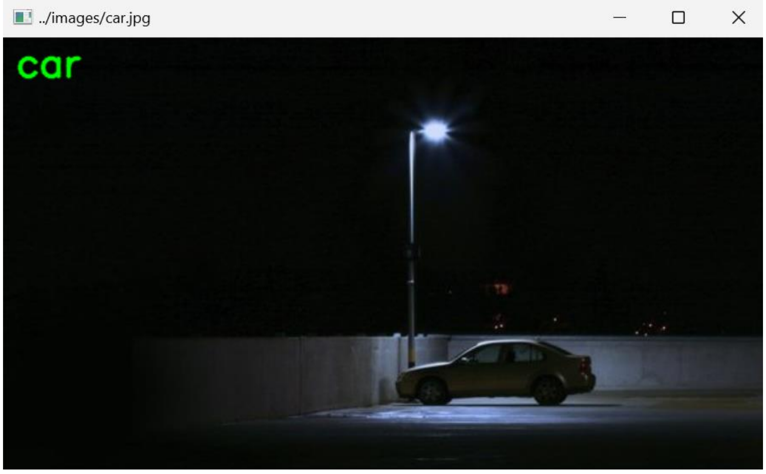
Code :-

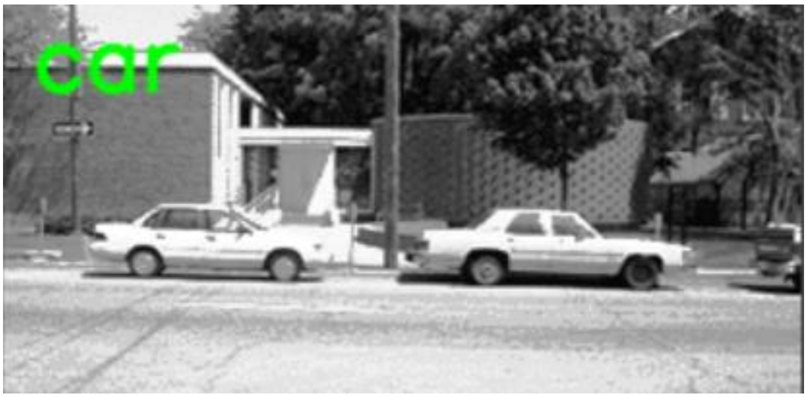
```
import cv2
import numpy as np
import os

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
```

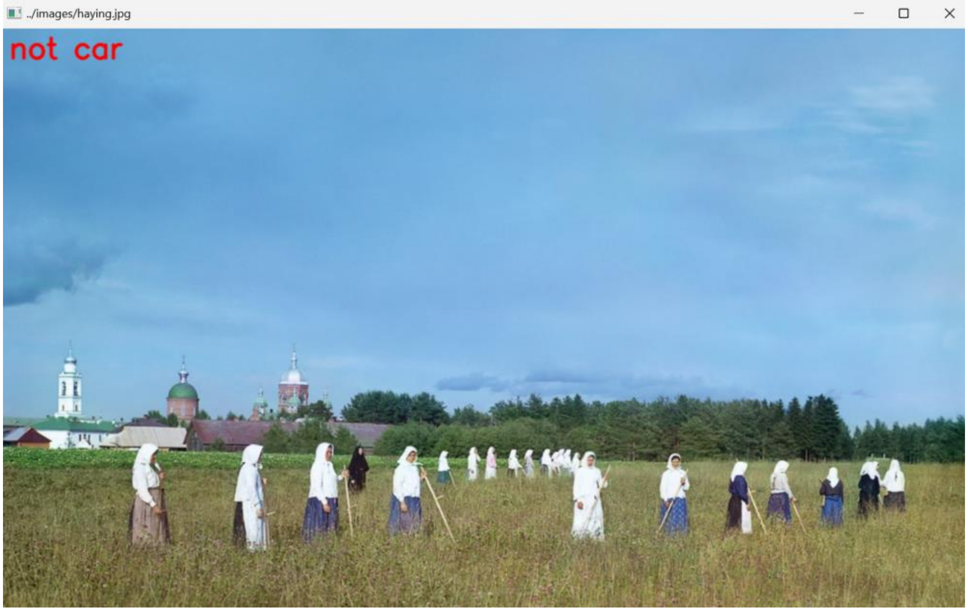
```
def add_sample(path):
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    keypoints, descriptors = sift.detectAndCompute(img, None)
    if descriptors is not None:
        bow_kmeans_trainer.add(descriptors)

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',
                      '../images/statue.jpg',
                      '../images/woodcutters.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 :-



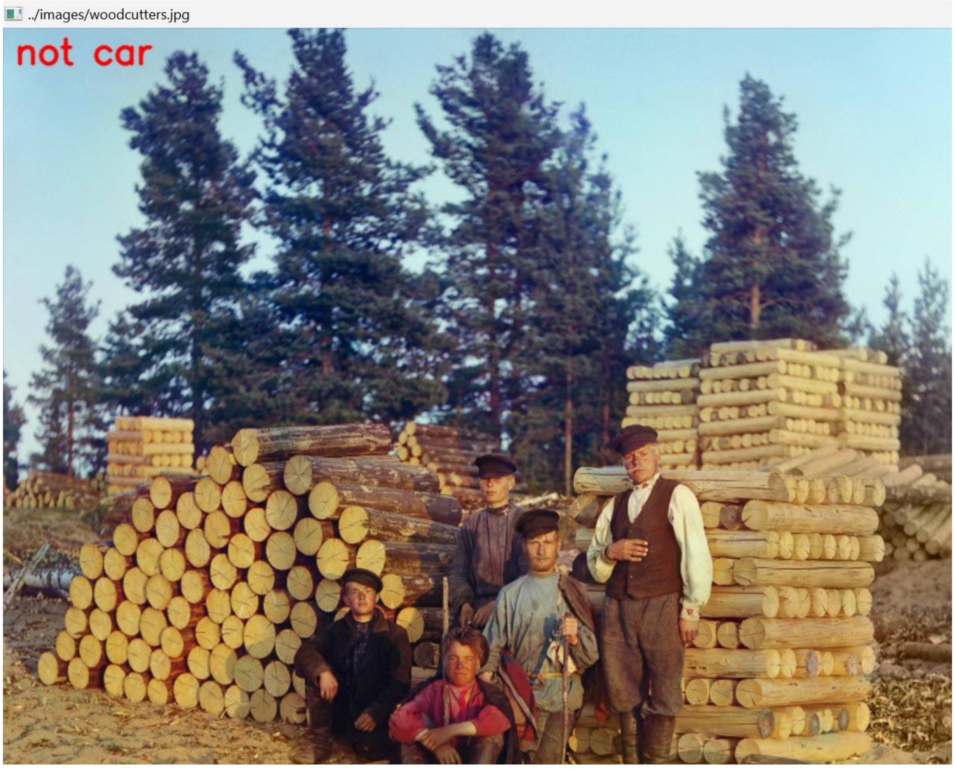






../images/haying.jpg

not car



../images/woodcutters.jpg

not car

Conclusion: -

In computer vision, the bag-of-words model (BoW model) sometimes called bag-of-visual-words model can be applied to image classification or retrieval, by treating image features as words. in the bag-of-words model for computer vision, visual features are represented as a bag of words. Bow works by first performing Feature extraction then codebook generation & then feature vector generation. By using the Bow model, a car detection from scene program was written which identifies whether a particular scene contains a car or not