Computer Vision
ETH Zurich, Fall 2023
Hardik Shah

Exercise 4: Object Recognition

Hardik Shah

The exercise on Object Recognition is divided into two parts:

1. **Bag-of-Words** image classifier. This method involves classical image processing and clustering techniques to achieve binary classification(in this task).
2. **CNN-based** image classification on CIFAR 10, where we use convolutional neural networks for multi-label image classification using the famous VGG network.

1 Bag-of-Words Classifier

This section presents the implementation and results of a Bag-of-Words (BoW) image recognition system. The goal of this system is to classify images into positive (with back of a car) or negative (without back of a car) categories. The implementation involves several key steps, including feature extraction, codebook creation using K-Means clustering, and recognition based on BoW histograms. The system's performance was evaluated using testing dataset, and the results are discussed in the subsequent sections.

1.1 Feature Extraction

The feature extraction process involves extracting local feature points from input grayscale images. These feature points are obtained by dividing the images into a grid(`10x10`) and computing Histogram of Oriented Gradients (HOG) descriptors for each grid point. The HOG descriptors capture gradient information and provide a compact representation of local image patterns.
Each HOG descriptor consists of `4x4` cells around the feature point, with one HOG computed for each cell. A histogram consists of `8` bins, and hence one HOG descriptor is `4x4x8=128` dimensional. The following steps describe the overall pipeline of feature extraction:

1. `grid_points(img, nPointsX, nPointsY, border):`
   (a) The function calculates grid point coordinates by evenly dividing the input grayscale image (`img`) into `nPointsX` by `nPointsY` grid cells, leaving a border of `border` pixels on each side. This is to ensure that the feature descriptor does not overshoot the image boundary.
   (b) It uses `np.linspace` to generate evenly spaced points along the rows and columns, and then combines these points to form the grid.
2. `descriptors_hog(img, vPoints, cellWidth, cellHeight):`

(a) Sobel operators (`cv2.Sobel`) are applied to the input image (img) to compute gradient values in the x and y directions.

(b) For each grid point in `vPoints`, the function extracts a local region using `cellWidth x cellHeight` pixels for `4x4` cells around a feature.

(c) Gradient orientations (`np.arctan2(grad_y, grad_x)`) are calculated for the pixels in the local region. `np.arctan2` returns the arctangent of the quotient of its arguments, providing the angle information.

(d) Histograms with 8 bins are created using `np.histogram` to represent the distribution of gradient orientations in each local region.

(e) These histograms are concatenated to form a single 128-dimensional descriptor for the grid point.

1.2 Codebook Creation

The codebook is created using K-Means clustering. The local descriptors extracted from training images are clustered into k centers, forming the codebook. Effectively, each cluster represents a 'word' in our vocabulary, and we assume that each image in our set of training/testing images must be a combination of these 'words'. The choice of the number of clusters (k) is critical and affects the recognition performance. To determine the optimal k value, a hyperparameter search was performed, and the results are presented later. For codebook creation, we use both negative and positive examples from our set of training images.

The `create_codebook(nameDirPos, nameDirNeg, k, numiter)` function creates the codebook by performing K-Means clustering on local descriptors extracted from positive and negative training images. The function reads positive and negative training images, extracts local descriptors using the previously defined functions in Section 1.1, and performs K-Means clustering (`KMeans` from `scikit-learn`) to create k cluster centers. The resulting `vCenters` matrix represents the codebook i.e. a `128` dimensional feature for each of the `k` words in our vocabulary.

1.3 Bag-of-Words Histograms

For each image, BoW histograms are constructed by assigning each local descriptor to the nearest cluster center in the codebook. These histograms represent the distribution of visual words (cluster centers) in the image. The histograms for positive and negative training images are computed separately. The following steps give an outline of the implementation in code:

1. `bow_histogram(vFeatures, vCenters):` Computes BoW histograms for a set of local descriptors using the given codebook.

   (a) Pairwise distances between `vFeatures` (local descriptors) and `vCenters` (cluster centers) are calculated using broadcasting and element-wise operations.

   (b) `np.linalg.norm` is used to compute the Euclidean distances between the descriptors and cluster centers.

(c) `np.argmin` is employed to find the indices of the nearest cluster centers for each descriptor.

(d) `np.histogram` is used to create histograms based on these indices. The bins parameter is set to the number of cluster centers (`len(vCenters)`) to ensure each index falls into the appropriate bin.

(e) The resulting histogram represents the BoW histogram for a set of local descriptors.

2. `create_bow_histograms(nameDir, vCenters):`

(a) The function reads images from the specified directory (`nameDir` and extracts local descriptors for each image using the functions in Section 1.1

(b) Uses `bow_histogram` to compute BoW histograms for each image. The resulting `vBoW` matrix represents the BoW histograms for all images.

1.4 Recognition and Evaluation

The recognition process involves finding the nearest neighbor in both positive and negative sets based on the test image's BoW histogram. The image is classified as positive (with back of a car) if its nearest neighbor in the positive set is closer than the nearest neighbor in the negative set.

The `bow_recognition_nearest(histogram, vBoWPos, vBoWNeg)` function takes a BoW histogram (`histogram`) of a test image and compares it with BoW histograms of positive (`vBoWPos`) and negative (`vBoWNeg`) training images. It calculates the distances and classifies the test image as positive or negative based on the nearest neighbor.

1.5 Hyperparameter Optimization and Results

A hyperparameter search was conducted to determine the optimal number of clusters (k) for K-Means clustering. The search ranged from 1 to 20 clusters. The results indicated that the system achieved a maximum accuracy of **83.6%** for positive samples and **100%** accuracy for negative samples when `k=5`.

The accuracy results for different values of k are visualized in the plot provided in Figure 1. It is evident that the accuracy saturates at a certain point, emphasizing the importance of selecting an appropriate number of clusters for effective representation of visual words.

The techniques implemented in the previous sections are fundamental to the Bag-of-Words image recognition system, enabling accurate representation of local image features and efficient classification based on visual words.

*Figure 2 shows the logging of running `bow_main.py`.

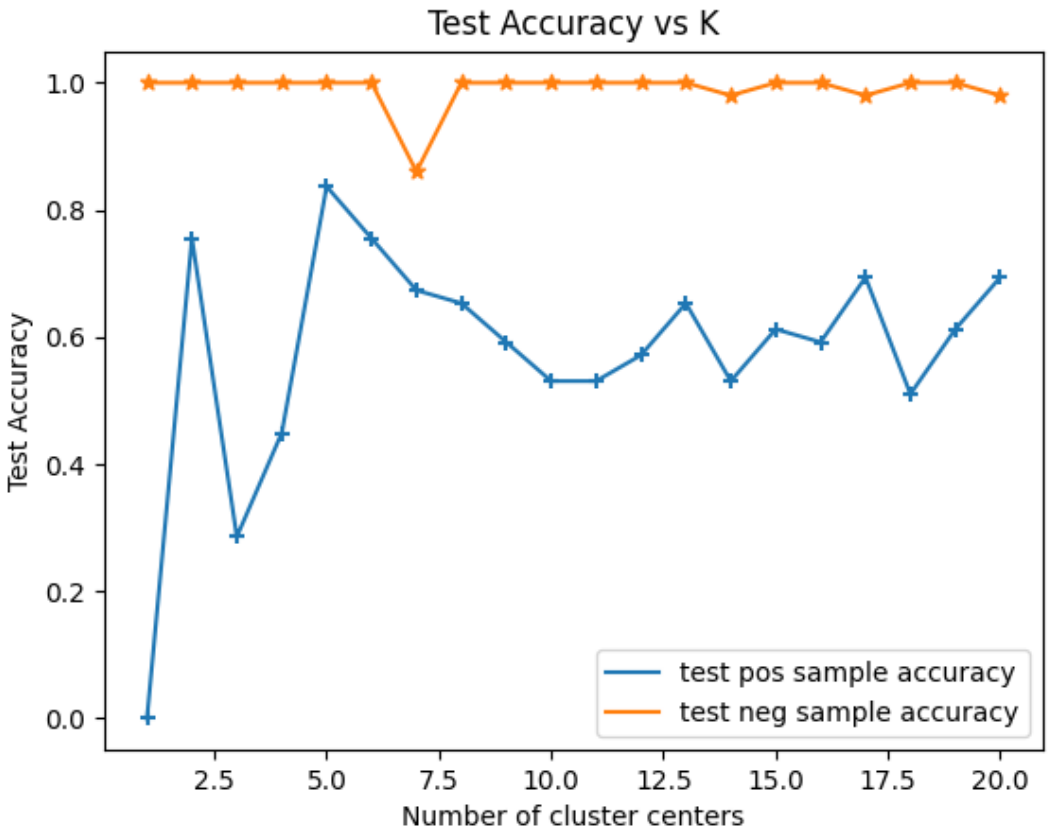

Figure 1: Accuracy results for different values of k (number of clusters)

```
(ex4) administrator@Inspiron-5490:~/Documents/Computer_Vision/Assignments/Assignment_3/code$ python bow_main.py
creating codebook ...
100%|| 100/100 [00:03<00:00, 25.70it/s]
number of extracted features:  1000
clustering ...
creating bow histograms (pos) ...
100%|| 50/50 [00:01<00:00, 26.98it/s]
creating bow histograms (neg) ...
100%|| 50/50 [00:01<00:00, 28.91it/s]
creating bow histograms for test set (pos) ...
100%|| 49/49 [00:02<00:00, 21.53it/s]
testing pos samples ...
test pos sample accuracy: 0.8367346938775511
creating bow histograms for test set (neg) ...
100%|| 50/50 [00:01<00:00, 29.54it/s]
testing neg samples ...
test neg sample accuracy: 1.0
(ex4) administrator@Inspiron-5490:~/Documents/Computer_Vision/Assignments/Assignment_3/code$
```

Figure 2: Logging of running `bow_main.py`

2 CNN-based Classifier

The task at hand involves building and training a Convolutional Neural Network (CNN) for image classification using the CIFAR-10 dataset. The CNN architecture, training procedure, and evaluation metrics are discussed below.

2.1 Model Architecture

The CNN architecture utilized for this task is a simplified version of the VGG network. The model consists of multiple convolutional blocks followed by max-pooling layers. The convolutional layers are activated by ReLU functions. The architecture is as follows:

1. Convolutional Blocks:
   (a) `Conv2d (in_channels=3, out_channels=64, kernel_size=3, padding=1)`
   `ReLU`
   `MaxPool2d (kernel_size=2, stride=2)`
   (b) `Conv2d (in_channels=64, out_channels=128, kernel_size=3, padding=1)`
   `ReLU`
   `MaxPool2d (kernel_size=2, stride=2)`
   (c) `Conv2d (in_channels=128, out_channels=256, kernel_size=3, padding=1)`
   `ReLU`
   `MaxPool2d (kernel_size=2, stride=2)`
   (d) `Conv2d (in_channels=256, out_channels=512, kernel_size=3, padding=1)`
   `ReLU`
   `MaxPool2d (kernel_size=2, stride=2)`
   (e) `Conv2d (in_channels=512, out_channels=512, kernel_size=3, padding=1)`
   `ReLU`
   `MaxPool2d (kernel_size=2, stride=2)`
2. Classifier
   (a) Flatten the output from the last convolutional layer using `nn.Flatten()`.
   (b) Fully Connected (Linear) layer with `ReLU` activation `(in_features=512, out_features=512)`
   (c) Add a `Dropout (p=0.5)` layer.
   (d) Fully Connected (Linear) layer `(in_features=512, out_features=10)` for classification (10 classes).

2.2 Training Procedure

Below were the values of the hyperparamters used:

1. Batch Size = 128
2. Learning Rate = 0.0001
3. Number of Training Epochs = 30

The model is trained using the Adam optimizer and cross-entropy loss. During training, the loss is calculated, and backpropagation is performed to update the model's parameters. Training progress is logged, and the best model is saved based on validation accuracy.

Figure 3 shows the train loss and validation accuracy curves generated during training VGG, using tensorboard. A validation accuracy of **80.38%** (best) was achieved at the end of training and the corresponding model weights were saved.

*Due to hardware constraints, not many hyperparameters could be tuned.

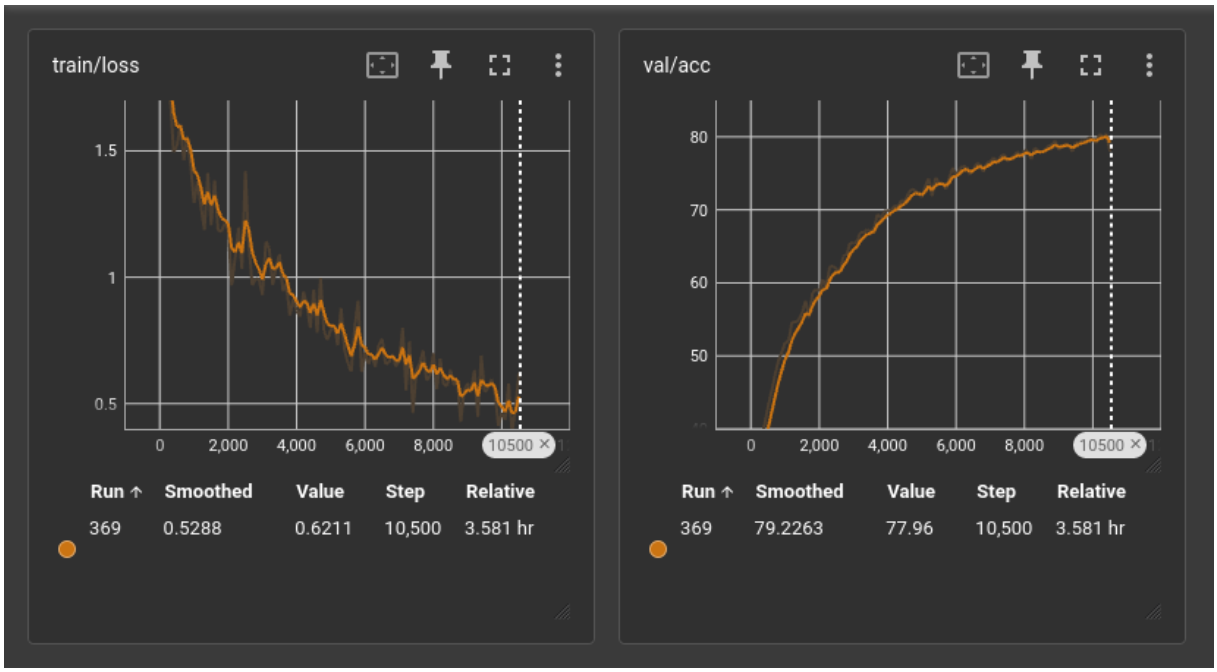

Figure 3: Training curves for VGG - Train loss and Validation Accuracy (best=80.38%)

2.3 Testing Procedure

The trained model is loaded, and the test dataset is fed through the network. Predictions are made for the test images, and accuracy is calculated by comparing the predicted labels with the ground truth labels. Figure 4 shows the logging of running `test cifar10 vgg.py`, which displays the test accuracy of the trained model.

The VGG model achieved a validation accuracy of **80.38%** and a test accuracy of **79.25%**. These results demonstrate the effectiveness of the trained model in accurately classifying images within the CIFAR-10 dataset.

```
(ex4) administrator@Inspiron-5490:~/Documents/Computer_Vision/Assignments/Assignment_3/code$ python test_cifar10_vgg.py
[INFO] test set loaded, 10000 samples in total.
79it [00:36,  2.14it/s]
test accuracy: 79.25
```

Figure 4: Test accuracy of VGG trained for 30 epochs (79.25%)

The CNN-based image classification system demonstrates the power of deep learning in recognizing complex patterns within images. The VGG-inspired architecture, coupled with training and testing procedures, results in accurate image classification.