Computer Vision
Lab Assignment Report - Object Recognition

ETH Zürich, 263-5902-00L Computer Vision HS2022
Prof. Marc Pollefeys, Prof. Siyu Tang, Dr. Fisher Yu

Anisha Mohamed Sahabdeen

November 11, 2022

Abstract

In order to perform object recognition, it is possible to adopt a number of different approaches. One of the most intuitive algorithms for the task, known as *bag-of-words*, measures the frequency of occurrence of particular words in a document in order to quickly find documents that match a particular query. However, this technique ignores the spatial correlation between said words as it assumes that the individual instances of generic visual categories have relatively little spatial coherence. On the other hand, one of the most popular techniques nowadays to solve object recognition and classification problems are neural networks, reaching state-of-the-art performances over traditional machine learning approaches. The purpose of this report is to present a Python implementation and evaluate the performances of: (a) a bag-of-words classifier, tested on a binary classification task, and (b) a neural network based on the VGG architecture trained on the CIFAR-10 dataset.

1 Bag-of-words classifier

The task is to implement a bag-of-words classifier that can assess whether or not a test image contains a car. Note that the training dataset contains both positive and negative examples.

Details of the classifier implementation and experimental results are provided below.

1.1 Local feature extraction

In order to fit the classifier, it is first required to identify a set of feature points for each image in the training data (*feature detection*), and then convert the region around each detected keypoint into a more compact and stable descriptor (*feature extraction*).

A simple way to select feature points is to use a regular grid with a certain granularity on the x and y axis on the given input image, leaving a border of B pixels in each image dimension.

This can be achieved through the function `grid_points`, which given the number of desired points on the grid on the x and y dimensions, respectively n_x and n_y, returns the coordinates of $n_x \cdot n_y$ grid-points.

Then, in order to perform feature extraction, it is possible to employ the Histogram of Oriented Gradients (HOG) as feature descriptor. This technique produces a histogram over the gradient orientations in a localized portion of an image. The computation of the HOG descriptors can be broken down into the following steps.

1. **Computation of image gradient.** The image gradients and can be readily obtained by convolving the image with Sobel filters, which are among the most utilized derivative filters nowadays.

$$\mathbf{I}_x = \frac{\partial \mathbf{I}}{\partial x} = \mathbf{I} * \mathbf{F}_x \ , \ \mathbf{F}_x = \mathbf{S}_y * \mathbf{D}_x = \left(\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}\right) * \left(\frac{1}{2}\cdot\begin{bmatrix}1 & 0 & -1\end{bmatrix}\right) = \frac{1}{8}\cdot\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$$

$$\mathbf{I}_y = \frac{\partial \mathbf{I}}{\partial y} = \mathbf{I} * \mathbf{F}_y \ , \ \mathbf{F}_y = \mathbf{S}_x * \mathbf{D}_y = \left(\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}\right) * \left(\frac{1}{2}\cdot\begin{bmatrix}1\\0\\-1\end{bmatrix}\right) = \frac{1}{8}\cdot\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix},$$

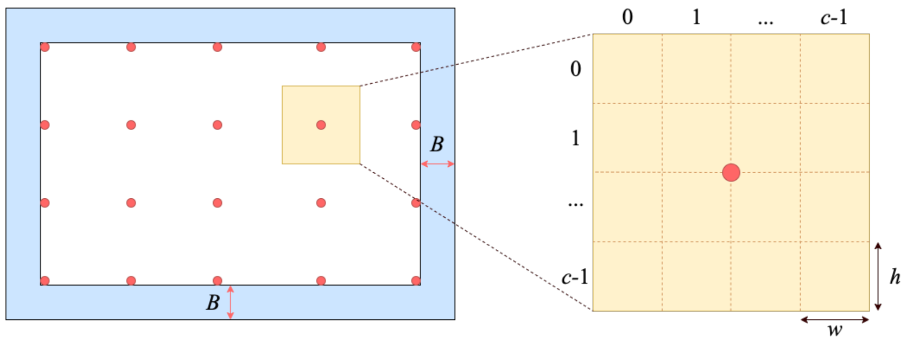

Figure 1: Cell matrix around grid-points.

where $\mathbf{I}$ is the 2-dimensional input matrix, $\mathbf{F}_x$ and $\mathbf{F}_y$ are the Sobel derivative filters, $\mathbf{S}_x$ and $\mathbf{S}_y$ are the smoothing kernels used by the Sobel filters in order to reduce noise, and $\mathbf{D}_x$ and $\mathbf{D}_y$ implement the derivative operators.

2. **Cell division.** The HOG feature descriptor analyzes the gradient orientations in localized regions of an image. By dividing the region surrounding each keypoint into $c \times c$ non-overlapping cells, each of $h \times w$ pixels, it will be possible to compute a histogram for each one of the cells. The resulting $c \times c$ histograms will be then concatenated to form a one-dimensional descriptor for each feature point.

3. **Histogram computation.** Despite there being several methods to calculate the histogram for a given cell, the most straightforward is to simply divide the range of possible values for gradient orientations in a number of regular intervals (bins), and produce a frequency histogram by counting the occurrences of the gradient orientation values in each bin.

   First, calculate the gradient orientation $\theta_\mathbf{p}$ at each pixel $\mathbf{p}$ of every cell relative to a given grid-point.

$$\theta_\mathbf{p} = \mathsf{atan2}(\left(\frac{I_y(\mathbf{p})}{I_x(\mathbf{p})}\right), \mathbf{p} \in \bigcup_{i=1}^{c^2} C_i$$

   where C_i represents the set of pixels composing one of the c^2 cells relative to the given grid-point.

   Note that the function `atan2` is the 2-argument arctangent which returns a correct and unambiguous value for the angle θ, in the range $(-\pi, \pi]$. This function is implemented by the `numpy` library function `np.arctan2`.

   Now compute a frequency histogram using b bins, where each bin will have a width of $\frac{2\pi}{b}$. The resulting histogram for a given cell can therefore be represented by a vector of length b.

4. **Vector concatenation.** By repeating the previous step for each cell relative to a given feature point, c^2 vectors of length b are obtained. The final HOG descriptor for a given point will be given by the concatenation of the c^2 vectors, resulting in a one-dimensional vector of length $c^2 \cdot b$.

   Given that the total number of keypoints in an input image is $n_x \cdot n_y$, the final feature matrix should be $(n_x \cdot n_y) \times (c^2 \cdot b)$-dimensional.

The function `descriptors_hog` implements the HOG descriptor, where the parameters c and b are set respectively to 4 and 8, so that the function returns a $(n_x \cdot n_y) \times 128$-dimensional feature matrix.

1.2 Codebook construction

In order to train the bag-of-words classifier to recognise meaningful words that can be associated with a given object category (i.e., cars), it is first needed to build a vocabulary of such visual words, also known as *appearance codebook*.

For this purpose, it is possible to first extract the descriptors of all images in the training dataset and then apply the k-means clustering algorithm so to obtain k words, represented by k vectors of the same length as the descriptor verctors. This is done through the function `create_codebook`. Note that the descriptors of both positive and negative observations are to be employed at this stage, meaning that some of the computed words will relate to the object category of interest, while others will not.

k-means clustering can be performed using the `KMeans` function from the `sklearn.cluster` package library as follows:

```
kmeans_res = KMeans(n_clusters=k, max_iter=numiter).fit(vFeatures),
```

where `vFeatures` is the $(N \times (n_x \cdot n_y) \times 128)$-dimensional descriptor matrix of the training set, where N is the total number of training images, and `k` and `numiter` are scalars.

It is important to observe that, since the k-means algorithm finds a local rather than a global optimum, the obtained results will depend on the initial (random) cluster assignment of each training image. Thus, it is important to run the algorithm multiple times from different random initial configurations and select the best solution, that is, the one that minimizes the within-cluster variation. The `KMeans` functions runs the algorithm with different centroid seeds `n_init` times, set by deafult to 10, and for each run the maximum number of iterations allowed to find the local optimum is `max_iter`.

The cluster centroids can be obtained as follows:

```
vCenters = kmeans_res.cluster_centers_  ,
```

where `vCenters` is a $k \times 128$-dimensional array, representing the k words of the appearance codebook.

1.3 Bag-of-words encoding

During this stage, each observation from the training dataset needs to be encoded as a *bag-of-words* histogram, that is, a histogram over the visual words previously obtained.

For this purpose, given an input image, represented by a a sequence of feature descriptors, each of those descriptors is to be assigned to a word from the codebook, so that a frequency histogram can be then drawn by counting the number of assignments to each of the words.

In particular, a descriptor $\mathbf{v}$ is assigned to a word $\mathbf{w}^* \in W$ if their pairwise dissimilarity measure is the smallest among the set of all words W in the vocabulary, where $|W| = k$.

$$\mathsf{assign}(\mathbf{v}) = \left\{ \mathbf{w}^* \mid d(\mathbf{v}, \mathbf{w}^*) = \min_{\mathbf{w} \in W} d(\mathbf{v}, \mathbf{w}) \right\}$$

The chosen dissimilarity measure for this task is the Euclidian distance, which can be readily calculated using the `norm` function of the `numpy.linalg` library package.

Thus, after computing the histogram over the set of k words, each image will be represented by a vector of length k, where each element of the vector represents the frequency of that word in the given image.

The bag-of-words encoding is performed by the function `bow_histogram`, which takes as input the $((n_x \cdot n_y) \times 128)$-dimensional descriptor matrix of an image and the $(k \times 128)$-dimensional matrix representing the sequence of words in the codebook, and returns a vector of length k representing the bag-of-words encoding of the image.

1.4 Nearest neighbour classification

By performing the bag-of-words encoding on every image in the positive and negative training sets (through the function `create_bow_histograms`), it is then possible to perform binary classification on new test images by first extracting their feature descriptors and then finding the nearest neighbour among the descriptors of the positive and negative sets, so that the prediction is made according to the label of that neighbour. This last step is implemented in the function `bow_recognition_nearest`.

1.5 Experimental results

The training dataset that was employed contained 50 positive samples (images with a car) and 50 negative samples (images without a car). For the feature detection parameter setup, the border parameter B was set to 8, and n_x and n_y, respectively the number of desired points on the grid on the x and y axis, were each set to 10. Thus, the feature descriptor matrix for an input image would have the following dimensions: $(n_x \cdot n_y) \times 128 = 100 \times 128$.

After running the experiment several times, it was found that the number of clusters k needed to achieve desirable results ($\geq 90\%$ accuracy on both positive and negative test dataset) was as small as $k = 4$, meaning that the codebook only included 4 different words. Note that because the number of words is so small, each word might not be associated with a different visual human-recognizable object category, meaning that descriptors belonging to different visual categories might have been clustered together.

	Accuracy	
Experiment no°	Positive test-set	Negative test-set
1	0.900	1.0
2	0.900	1.0
3	0.918	1.0

Table 1: Experiment results. The experiment was run 3 times on the test set with the parameters specified above.

For k=4, the distributions of the mean bag-of-words histograms of the positive and negative sets was also recorded, in order to investigate whether some words resulted more significant than others when discriminating between the two labels.

Exp. no°	Train Set	w_1	w_2	w_3	w_4
1	Positive	7.56±3.956	18.26±6.154	33.9±7.658	40.28±7.856
	Negative	8.24±1.882	30.0±0	21.52±5.308	40.24±5.117
2	Positive	7.56±3.956	18.26±6.154	33.9±7.658	40.28±7.856
	Negative	8.24±1.882	30.0±0	21.52±5.308	40.24±5.117
3	Positive	7.5±3.885	18.26±6.154	34.24±7.677	40.0±7.759
	Negative	8.22±1.847	30.0±0	21.78±5.304	40.0±5.117

Table 2: Bag-of-words histograms of positive train set and negative train set: mean ± standard deviation.

Observe from Table 2 that the distributions of the histograms do not vary as much as expected between the positive and negative training set, as a matter of fact both mean histograms present a peak in w_4 with comparable values. However, there are some noticeable differences in the concentrations of w_2 and w_3, which are most likely going to be the discriminatory elements in this binary classification task. Because of the poor difference in the mean histograms, it is reasonable to assume that the high accuracy scores were due to similarity between the training set and the test set, inferring that the model would not generalize well on images drawn from other contexts.

2 CNN-based classifier

Here the task is to implement a simplified version of the VGG image classification network ([1]) on the CIFAR-10 dataset.

Details of the network architecture and experimental results are presented below.

2.1 A simplified version of the VGG network

The VGG architecture is the basis of many ground-breaking object recognition models. It achieved top-5 test accuracy of 92,7% on ImageNet, a dataset consisting of more than 14 million images belonging to nearly 1000 classes. It introduced significant improvements over other well-performing convolutional networks by replacing large kernel sizes with multiple with 3×3 kernels one after another, increasing the depth of the network itself.

A simplified version of the VGG network is presented in Table 3.

Block Name	Layers	Output size
conv_block1	ConvReLU(k=3) + MaxPool2d(k=2)	[bs, 64, 16, 16]
conv_block2	ConvReLU(k=3) + MaxPool2d(k=2)	[bs, 128, 16, 16]
conv_block3	ConvReLU(k=3) + MaxPool2d(k=2)	[bs, 256, 16, 16]
conv_block4	ConvReLU(k=3) + MaxPool2d(k=2)	[bs, 512, 16, 16]
conv_block5	ConvReLU(k=3) + MaxPool2d(k=2)	[bs, 512, 16, 16]
classifier	Linear+ReLU+Dropout+Linear	[bs, 10]

Table 3: Simplified VGG architecture.

2.2 Training and testing

The dataset was trained and tested on CIFAR-10, a dataset which includes 10 image classes, with 50000 training images, from 5000 images were used for validation, and 10000 testing images, each of resolution 32 $\times$ 32.

As for the training hyperparameters, the model was trained on a M1 CPU for 40 epochs, with batch size of 128 and a constant learning rate of 0.0001. Figure 2 and Figure 3 show respectively the training loss and validation accuracy of such training.

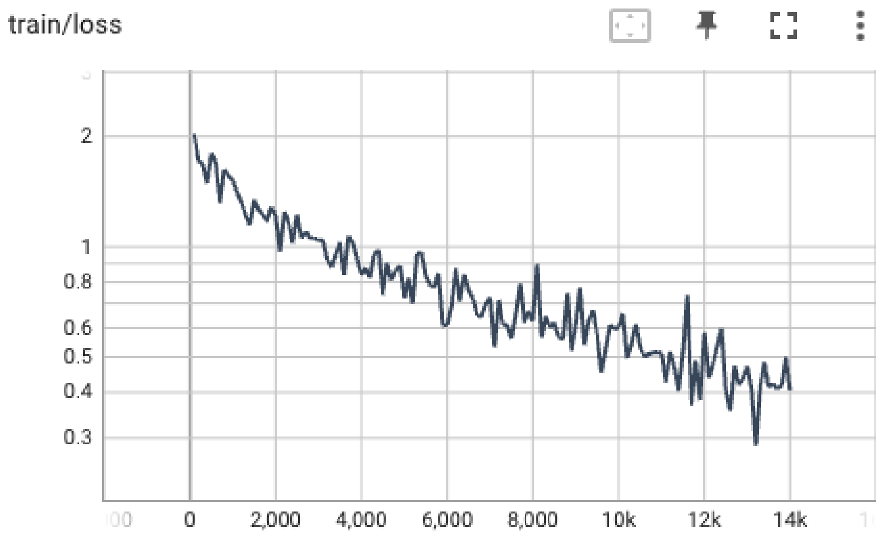

Figure 2: Training loss

The trained model achieved an accuracy of 81.54 % on the test set.

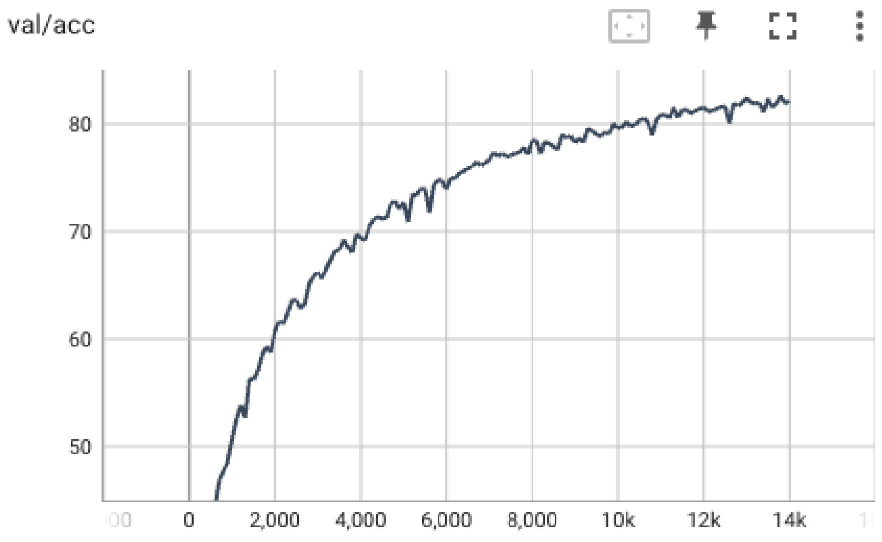

Figure 3: Validation accuracy

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition.

[2] Richard Szeliski. Computer vision algorithms and applications, 2011.