Assignment 5

Computer Vision ETH, Fall 2021

Quentin Guignard

17th December, 2021

2 Bag of Words Classifier

In this section we will firstly explain our implementation of our bags of words and then we will show our results.

2.1 Explanation

The bag of words classifier consists of classifying images in function of local image features. The goal is to get a vocabulary (visual codebook) which is a set of words that we'll represent as vectors. Theses words are the possible words of the things we aim to distinguish in the image. In order to get this codebook we need a training set and process its data by processing the following steps. The first step is as follows, for each image in the training set, we will extract a set of local features. We divide the image into a grid with N grid cells. We then divide each cell as 16 subcells. From each cell we will extract a descriptor made out of the concatenation of the gradients orientations histograms of each subcells. Every subcell yield a $8 - D$ vector where the components represents the count of each possible gradient orientations (360 divided by 8 in our case gave the ranges for each bucket). At the end of this step, we get $128 - D$ (16 subcells for each grid cell) descriptors vectors for all N grid cell in the image. Therefore, we end up with $N * 128$ descriptors for each image. The second steps is finding the visual words that are contained in the training images. We will apply a K-means clustering algorithm for this task. Thy hyper-parameter k needs to be (as far as we understand) equal to the amount of the possible different visual features that we aim to express in our codebook. Equivalently, k should be equal to the number of words we aim to detect. This means that for cars images, we need to count the visual elements that can appear on the car (wheel, window etc...) as well as the features that could be located around the car (road, sky essentially). Once we have found our clusters we can compute the Bag of Words of some images. We use the descriptors of each image and assign the descriptors to the nearest cluster (nearest word). This yields a histogram of words which we call bag of words.

To sum up, we find the code book with the K-means algorithm and using the descriptors of the training images. We then compute a bag of words for each training images (positives and negatives). After that, for each testing image we compute its bag of words and find its nearest distance to the positive training samples and to the negative training samples. We return the label associated with the smallest distance.

Now that we have seen the big lines, let's cover the intuition behind the bag of words (BOW) with histogram of gradients (HOG). In this method, each image is represented as a vector. This vector is the concatenation of all gradients orientation histograms of all local regions of the image (the concatenation of all N descriptors made out of $128 - D$ vectors). The intuition behind it is that the features of an image do have a visual structure or representation through the pixel intensities (particular change in the intensity around some local region). This structure imply the presence of local gradients with particular

orientations depending of the local feature. Therefore, the same feature will tend to yield the same local gradients orientation every time it appears on some image. Therefore, by counting the amount of time the same orientations appear locally (the sampling is done on all grid points), we can make a vector representation of these occurrences. Then, the images with similar features will tend to have similar local gradients orientation vectors and so will tend to lie in the same neighborhood in some high dimensional euclidean space. Similar features will tend to lie in the same neighborhood, this is why finding the k clusters of the descriptors is equivalent to find all the words presents in the image.

2.2 Implementation

We first implemented the grid point function where the idea is simply to get a grid with all the indices of the central positions of a local descriptor. This was done by creating a grid with Numpy (See code grid_points). The next function we had to implement was descriptors_hog which should compute the N $128 - D$ descriptors for some image. We implemented this function as a loop around each grid position. Then for each grid position we wanted to compute the histogram of the gradients orientation for each subcells (4x4) and then concatenate all the 16 $8D$ vectors to get the $128D$ descriptor. We computed the gradient orientation in degrees. A quick look at the Polar Coordinates Wikipedia page gave us the forgotten formula from our analysis lecture:

$$\theta = arctan2(y, x)$$

We then incremented by 1 each corresponding bucket of a subcell histogram. For this task, we divided the buckets evenly among 360 degrees (see code).

The next function that we implemented was the codebook construction that simply retrieved all the descriptors from all input image and find the clusters. After that we implemented a method to build an histogram of clusters (words) for some input image. For this function we did not used findnn because we did not see it was there. We implemented it in Numpy (see code : bow_histogram). Our implementation is using the squared distance instead of the norm and is fully vectorized except to fill the bins. The next function we implemented was create_bow_histograms which would simply compute for all input images the grid points, the descriptors and the bag of words for the freshly computed descriptors. At the end, we implemented a function bow_recognition_nearest that should be used after training to find in which group (negative or positive) our testing samples is the closest. We again used squared distances and some numpy functions to get the resulting distances (see code). We then returned the label associated to the smallest one.

In order to find k we came up with the following reasoning: We have 100 grid points which imply 100 possible regions that will be associated with a word. By looking at the dataset we decided that the cars were occupying on average just under a fifth of the image (even less something) and from here we came up with $k = 17$. For the num_iter, we decided to use a big value since it was fast and we empirically found that the values where not changing much from 100. We selected 10000 iterations. That's all for the implementation. Our final results on the testing set gave 89.8 percent accuracy for the positive samples and 86 percents for the negative samples.

3 Object recognition

For this part, we implemented the neural network as described in the hand in. It was similar to all the previous exercises (see code). Here are the results:

Figure 1: Training loss accross the training process

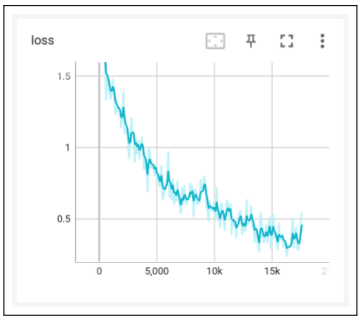
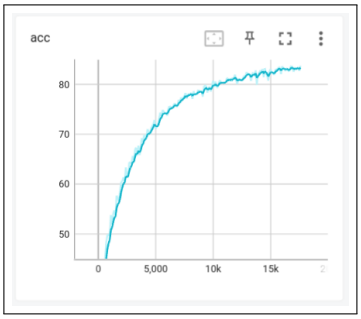


Figure 2: Training accuracy accross the training process



The achieved training accuracy after 49 epochs was 82.5 percents. The achieved testing accuracy on the cifar-10 dataset was 81.4 percents.

```
[INFO] test set loaded, 10000 samples in total.
79it [00:02, 28.19it/s]
test accuracy: 81.43
```