# BOW/CNN classifiers lab report

Ambroise

November 4, 2023

# 1 Bag-of-words Classifier

## 1.1 Local Feature Extraction

### 1.1.1 Feature detection - feature points on a grid

Within the `grid_points` function, the aim is to uniformly compute a grid of feature points from the given input image, avoiding a specified border. The steps involved in computing these grid points are:

- Compute evenly spaced x-coordinates that range from the `border` to the width of the image less the `border`. This ensures that there are `nPointsX` grid points along the x-dimension of the image.
- Similarly, generate evenly spaced y-coordinates that span from the `border` to the height of the image minus the `border`. This ensures there are `nPointsY` grid points along the y-dimension of the image.
- Using the `np.meshgrid` function, the x and y coordinates are combined to produce a 2D grid of points. This results in two matrices, one for the x-coordinates and the other for the y-coordinates of the grid points.
- Flatten these matrices and concatenate them to form a single 2D array named `vPoints`. This array contains pairs of x and y coordinates for each grid point with a shape of `[nPointsX*nPointsY, 2]`.

### 1.1.2 Feature description - histogram of oriented gradients

The `descriptors_hog` function seeks to compute the Histogram of Oriented Gradients (HOG) descriptors for the given grid points in the image. For each grid point, the gradient values around it are captured to form the HOG descriptor. The steps involved in computing these descriptors are:

- For each grid point centered at coordinates `(center_x, center_y)`, a local patch of the image is considered. This patch consists of a 4x4 grid of cells, where each cell has a size of `cellWidth x cellHeight`.
- For every cell in the local patch:
  - The gradient angles of the pixels in the cell are computed using the `np.arctan2` function. This function returns angles in the range `[-pi, pi]`.
  - To convert these angles into a non-negative range, any negative angles are incremented by `2*pi`, bringing the range to `[0, 2*pi]`.
  - A histogram of these angles is then computed. The range `[0, 2*pi]` is divided into `nBins` (which is 8 in this case) to form the histogram bins.
  - The counts of angles falling into each bin are computed using the `np.histogram` function. These counts represent the HOG values for the cell.
  - The computed histogram for the cell is appended to the descriptor for the current grid point.
- After computing the HOG descriptor for all cells around a grid point, this descriptor (a vector) is added to the list of descriptors for the image.

## 1.2 Codebook construction

In the `create_codebook` function, feature descriptors are extracted from a set of training images and then clustered to form a codebook. The steps you implemented are as follows:

- For every image:
  - Grid points are computed using the `grid_points` function, taking into consideration the image's dimensions and a predefined border.
  - The `descriptors_hog` function is then used to determine the Histogram of Oriented Gradients (HOG) descriptor for each grid point.
  - These descriptors are accumulated in the `vFeatures` list.
- The `vFeatures` list is reshaped, preparing the descriptors for clustering.
- K-means clustering is applied on the descriptors to derive cluster centers, which are returned as the function's output.

## 1.3 Bag-of-words Vector Encoding

### 1.3.1 Bag-of-Words histogram

In the `bow_histogram` function, the goal is to compute a Bag of Words (BoW) histogram for a given set of feature vectors from an image, with respect to a set of cluster centers:

- The nearest cluster center is determined for each feature vector in `vFeatures` using the `findnn` function, which returns the closest cluster indices and their respective distances.
- An empty histogram with the same number of bins as cluster centers is initialized.
- Based on the indices of nearest neighbors from `findnn`, the histogram bins are incremented accordingly to create the BoW activation histogram.

### 1.3.2 Processing a directory with training examples

In the `create_bow_histograms` function, the goal is to create the histogram encoding for each image:

- For each image in the specified directory:
  - The image is read and converted to grayscale.
  - Grid points for the grayscale image are computed using the `grid_points` function.
  - HOG descriptors for these grid points are computed using the `descriptors_hog` function.
  - The BoW histogram of the HOG descriptors with respect to the provided cluster centers is computed using the `bow_histogram` function.
  - The computed histogram is appended to the `vBoW` list.
- The list of histograms, `vBoW`, is then converted to a matrix and returned.

## 1.4 Nearest Neighbor Classification

In the `bow_recognition_nearest` function, the goal is to classify test images in terms of their nearest neighbor in the training set:

- Initialize `DistPos` and `DistNeg` to `None`.
- Find the nearest neighbor in the positive set and store the distance in `DistPos` using the `findnn` function with the given histogram and `vBoWPos`.
- Similarly, find the nearest neighbor in the negative set and store the distance in `DistNeg` using the `findnn` function with the given histogram and `vBoWNeg`.
- If `DistPos` is less than `DistNeg`, set the predicted result, `sLabel`, to 1. Otherwise, set `sLabel` to 0.

### 1.5 results

Different values of k and numiter were tested. The file bow_log.txt shows the obtained logs for different values of k and numiter.The information is summarized in the following table. The best performance since to have been with k=32 and numiter = 150, with 0.9388 accuray on positive test samples and 0.92 on negative ones.

Table 1: Accuracy obtained for different k and numiter values

| k | numiter | Test Pos Sample Accuracy | Test Neg Sample Accuracy |
|---|---|---|---|
| 2 | 10 | 0.2857 | 0.78 |
| 2 | 50 | 0.1429 | 0.82 |
| 2 | 100 | 0.3673 | 0.92 |
| 2 | 150 | 0.1429 | 0.8 |
| 4 | 10 | 0.8980 | 0.92 |
| 4 | 50 | 0.9592 | 0.94 |
| 4 | 100 | 0.9592 | 0.98 |
| 4 | 150 | 0.9592 | 0.94 |
| 8 | 10 | 0.9796 | 0.68 |
| 8 | 50 | 0.9796 | 0.6 |
| 8 | 100 | 0.9796 | 0.58 |
| 8 | 150 | 0.9796 | 0.56 |
| 16 | 10 | 0.8163 | 0.98 |
| 16 | 50 | 0.8367 | 0.9 |
| 16 | 100 | 0.9184 | 0.92 |
| 16 | 150 | 0.8776 | 0.94 |
| 32 | 10 | 0.8776 | 0.74 |
| 32 | 50 | 0.9388 | 0.9 |
| 32 | 100 | 0.8980 | 0.96 |
| 32 | 150 | 0.9388 | 0.92 |
| 64 | 10 | 0.8980 | 0.8 |
| 64 | 50 | 0.9592 | 0.8 |
| 64 | 100 | 0.8776 | 0.92 |
| 64 | 150 | 0.9184 | 0.86 |

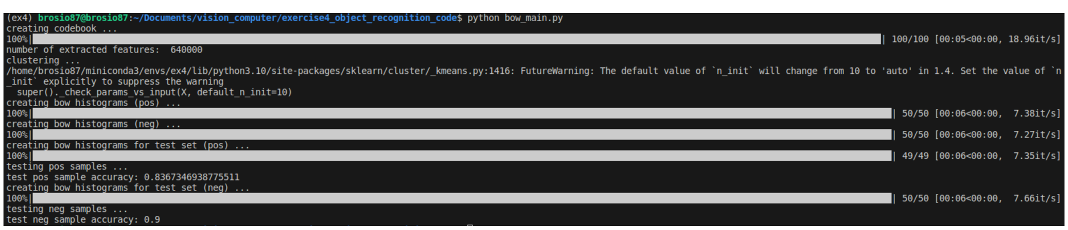


Figure 1: log example with screenshot from the console, k=50, numiter = 300

# 2 CNN-based Classifier

## 2.1 A Simplified version of VGG Network

In the `CNN_Implementation` class:

- In the `init` method:
  - Construct a convolutional block using `nn.Sequential`. A block comprise:
    - A 2D convolutional layer (`nn.Conv2d`) with specified in-channels, out-channels, kernel size, stride, and padding.

* a ReLU activation function (`nn.ReLU`).
    * a pooling layer (`nn.MaxPool2d`) to reduce spatial dimensions.
  - The final block is the classifier, made of a linear layer followed by a ReLu, a Dropout and another linear layer, ending with a vector of length 10, for the 10 classes.

- In the `forward` method:
  - Pass the input tensor through each convolutional block sequentially. The tensor transforms and reduces in spatial dimensions as it goes through the blocks.
  - After passing through all convolutional blocks, flatten the tensor to prepare it for fully connected layers.
  - Pass the flattened tensor through the fully connected layers to produce the final output.

## 2.2 CNN results

Information about the training is stored in the file runs/66192. For the logging of runing the test, it can be seen in the file test_logging.txt (in addition to the screenshot below). The following results were observed:

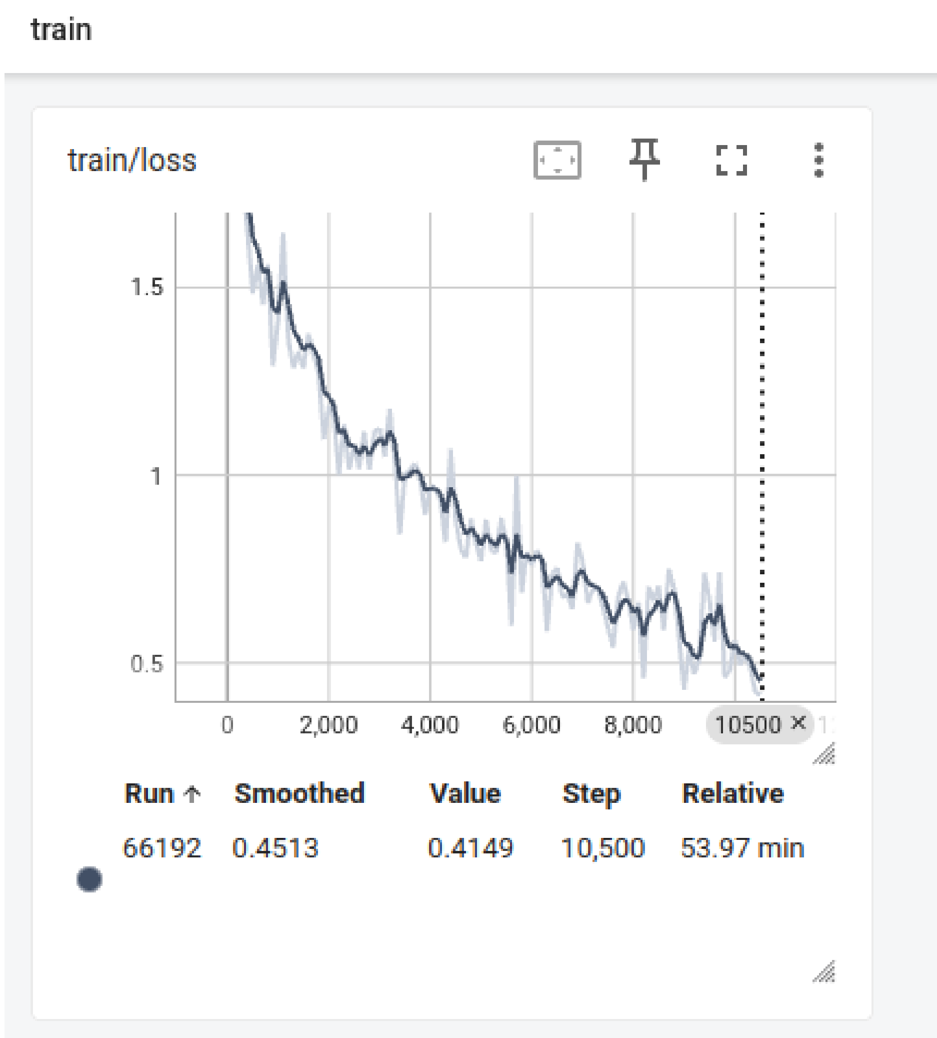


Figure 2: training loss evolution

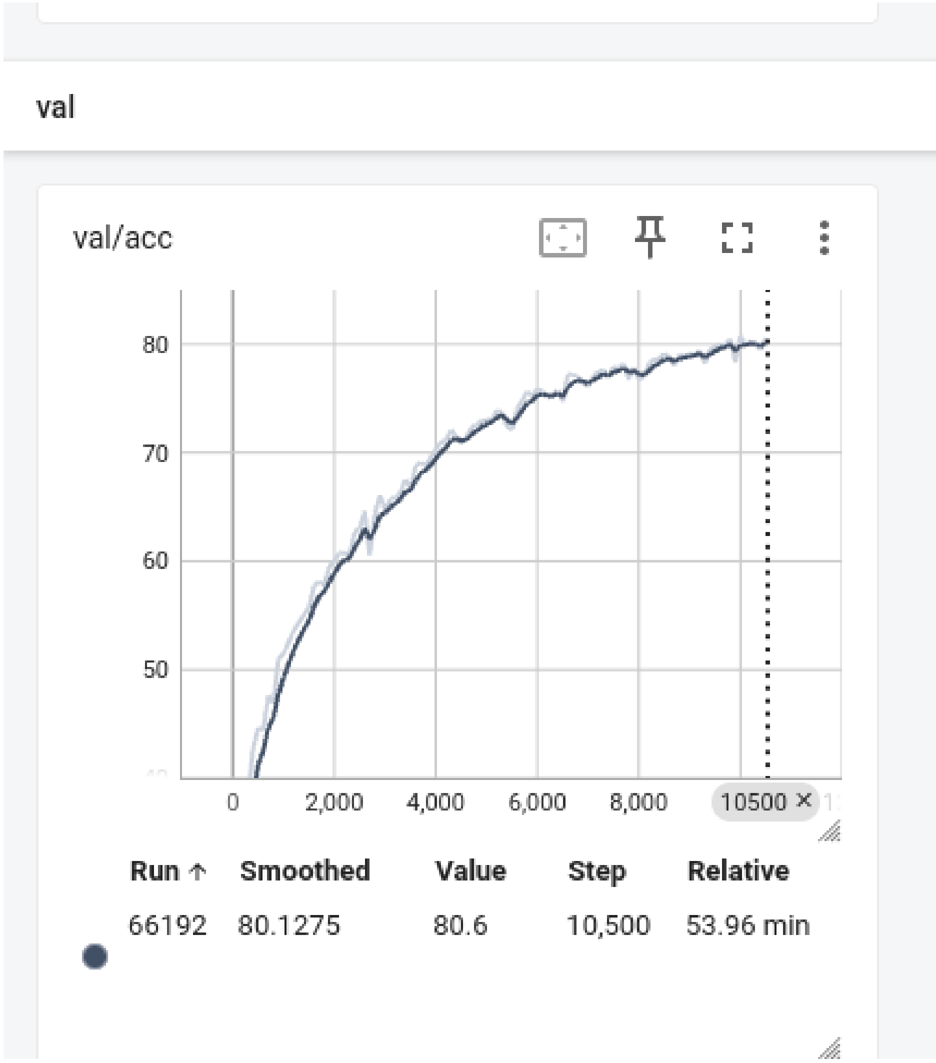


Figure 3: val accuracy evolution

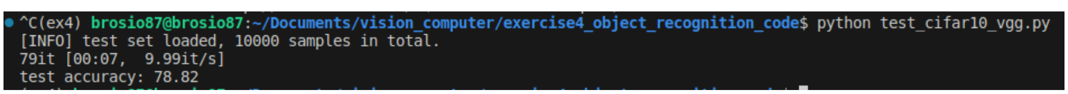


Figure 4: test accuracy

The performance of our Convolutional Neural Network (CNN) classifier can be visualized through its training loss, validation accuracy and test accuracy.

#### 2.2.1 Training Loss

From Figure 2, the training loss shows a consistent decline over the training steps. Starting from a value above 1.5, the loss rapidly decreases and then begins to stabilize around a value of 0.4149 after 10,500 steps. The rapid initial decline suggests that the model is quickly learning the representations from the data, and as it continues training, it fine-tunes and gradually optimizes its performance. The relatively stable value towards the latter stages indicates convergence, ensuring that the model has adequately learned from the training data.

### 2.2.2 Validation Accuracy

The validation accuracy graph increases with a logarithmic pattern (big increase at the beginning and slower after). It quickly climbs and begins to saturate around 80%. This indicates that the model not only learned the training data but was also able to generalize its knowledge to unseen validation data effectively. The high validation accuracy signifies the robustness of the model in handling novel data samples.

### 2.2.3 test Accuracy

Testing this model, I obtained a 78.82 accuracy on test data as can be seen in Figure 4.