Object Detection Report

Nicole Damblon

Computer Vision

1 Bag-of-Words Classification

The goal of this assignment is to detect objects in a given image. In the first part a Bag-of-Words classifier is used to represent images in compact features and classify them based on the distance to the features of the new image. The second part utilizes a CNN for object detection on the CIFAR-10 dataset, it predicts probabilities for 10 different object classes within an image.

Calculate grid points. To calculate a grid of $10x10$ points, the numpy function `meshgrid(x,y` is used, where x and y are two arrays, each containing 10 equidstant points throughout the image size. An $8 - pixel$ margin in the $x-$ and $y - direction$ is applied.

Compute HoG-descriptor. A Histogram-of-Oriented Gradients is chosen to describe the single patches of the image. The main part of the functionality described below is pictured in figure 1.

1. For every pixel, the angle and magnitude of the gradient is computed with `cv2.cartToPolar(grad_x, grad_y, angleInDegrees=True)`
2. Patches that contain $4x4$ cells around each gridpoint are extracted.
3. Each of those cells consists of 16 pixels.
4. For every cell an 8-bin histogram is computed as a list with 8 entries.
5. The bins of the histogram are intervals of size 45, all 8 representing an angle of 360 degrees. The last bin spans from 315 - 0 degree.
6. For every pixel in a cell, each angle is matched to one of the bins.
7. For each match, the magnitude of the angle is added to the corresponding bins of the histogram proportionally.
8. The 16 histograms describing one cell are concatenated.
9. One matrix per image consisting of one descriptor per grid point is returned.

```
            # compute the histogram

            # size of each bin 360/8 = 45
            # last bin is between 360 and 0 because here 0 = 360
            bins = [0,45,90,135,180,225,270,315,0]
            hist = [0,0,0,0,0,0,0,0]
            for i in range(1,nBins+1):
                for y in range(start_y,end_y):
                    for x in range(start_x,end_x):
                        # first, angle of each gradient gets assigned to corresponding bin in the histogram
                        # second, magnitude of each gradient is split proportional between lower and upper bin and added to the bins value
                        if bins[i-1] <= abs(angle[y,x]) <= bins[i]:
                            prop = (angle[y,x] - bins[i-1]) / 45
                            hist[i-1] += mag[y,x]*prop
                            hist[i] += mag[y,x]*(1-prop)
            # concatenate 16 histograms, each has length 8
            desc = np.append(desc,hist)
            # flatten array
            desc = np.ravel(desc) #[128]
    descriptors.append(desc) # [nPointsX*nPointsY, 128], descriptor for the current image (100 grid points)

descriptors = np.asarray(descriptors)
return descriptors #[100,128]
```

Fig. 1. Main Loop of the HoG-Descriptor: Create the 8-bin histogram based on magnitude and angle of the gradient to describe an image patch.

Create the codebook of visual words. In order to create the codebook, fist the gridpoints and the corresponding HoG-descripors are computed with the functions explained above. These features are computed for all images (the dataset consists of 100 images), each feature being a matrix of size $[100, 128]$. All features are concatenated, yielding 10000 features that form the feature space. To find the visual words, the feature space is clustered into k cluster centers, each cluster representing one visual word. The parameters of K-means $k = 30$ and $num\_iter = 60$ are found empirically.

The BoW-histogram. To assign every feature of an image to a visual word or cluster center, the index of each visual word that has the closest distance to a feature is computed by the function `idx, dist = findnn(vFeatures,vCenters)`. Now the number of visual words occuring in one image is counted in one histogram per image, where the bins are represented by the indices of the visual words, with the numpy function `np.histogram(idx, bins=idx_centers, density=False)`. The function `create_bow_histograms(nameDir, vCenters)` combines the functionality of creating the grid points for the features, calculating the HoG-descriptor and with its output the BoW-histrograms for each descriptor.

BoW-classification. The last step is classifying the image in terms of the labels "car" or "no car". The decision is based on the distance of each BoW-histogram of the test image (or new image) to the BoW-histogram of the positive ad negative training images. If the distance to a positive instance is shorter than the distance to a negative image, the classifier predicts a positive label "*car*".

Accuracy. The BoW-Classifier reaches an accuracy of 0.9183673469387755 for positive samples and an accuracy of 0.8 for negative samples. Since the initialization of the cluster centers in K-means when creating the visual vocabulary introduces randomness this results can vary.

2 CNN Classifier

The VGG simplified module consists of 5 convolutional blocks that each has a conv layer, ReLU and MaxPooling layer. After the fifth conv block, the output is reshaped in the forward pass so the classifier can be applied on the input. The classifier consists of two linear layers, ReLU and a dropout layer for regularization. The final output are probability scores for each of the 10 classes. After 50 epochs of training with a batch size of 128 a test accuracy of 82.73 is achieved. The results are logged in a tensorboard representation for training and validation phase, see figure 2 and 3. Interestingly, a BoW-classifier trained on only 100 images with a simple decision rule performs similar in terms of accuracy to a CNN trained on a way larger data set.

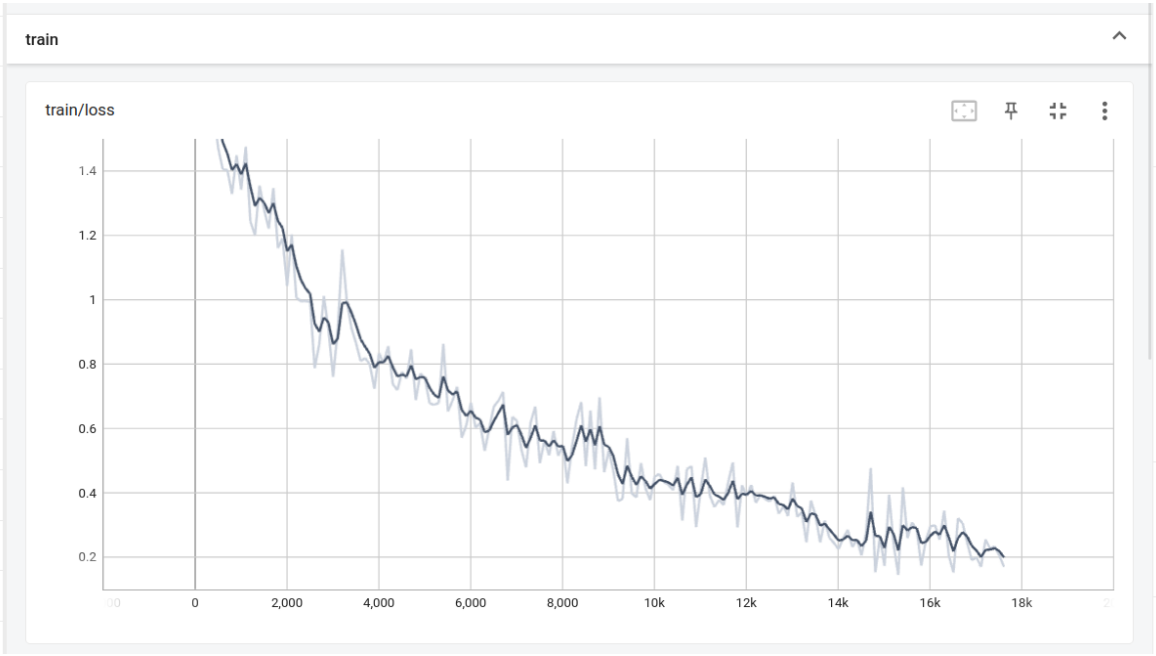

Fig. 2. Tensorboard: Loss during training.

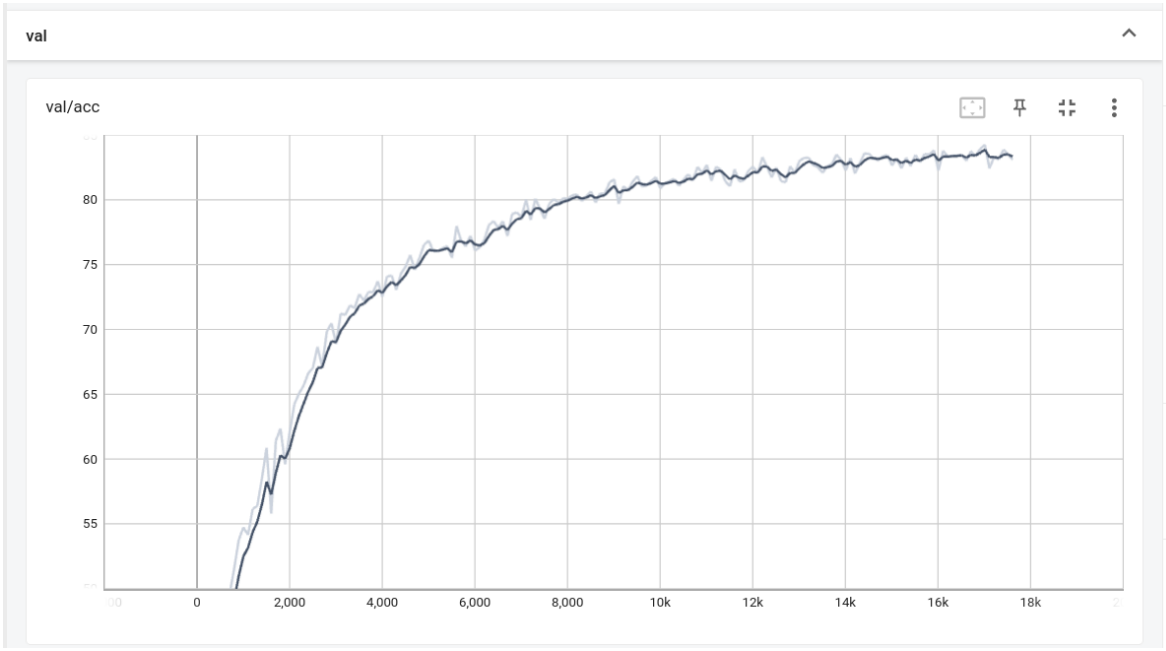

Fig. 3. Tensorboard: Accuracy on the validation set.