Igor Martinelli: Assignment 3 Report

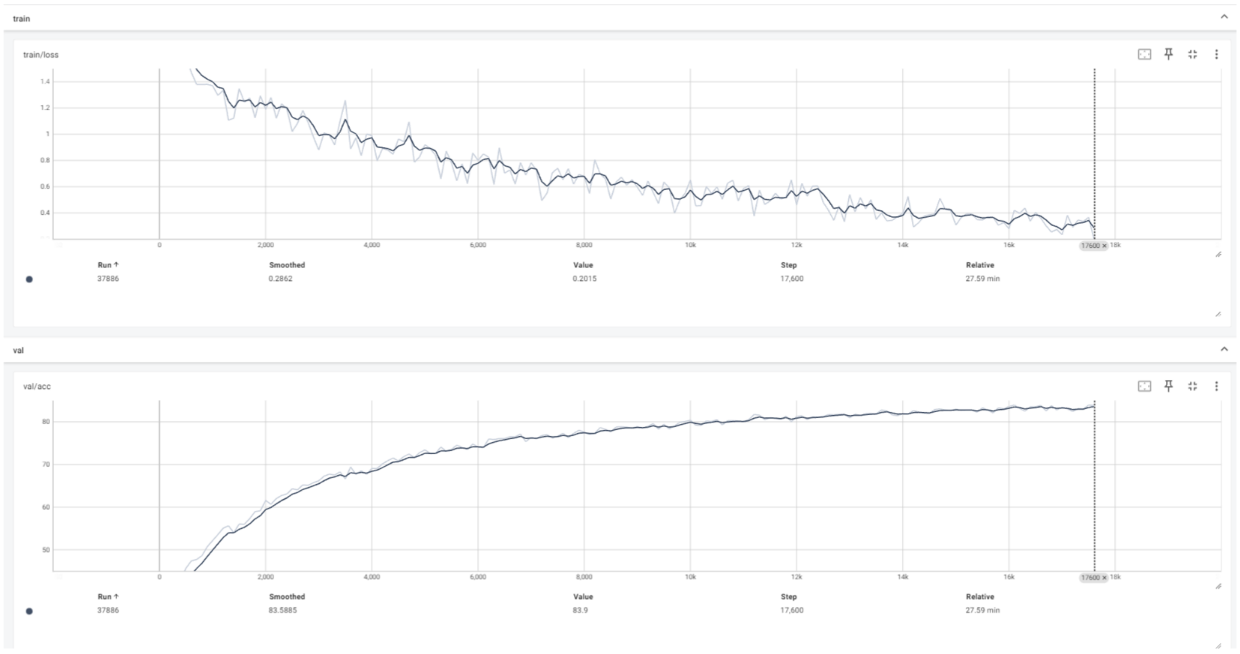
maigor@ethz.ch, 19.916.048.

1 Part 1 - Bag-of-Words Classifier Implementation

In this section, we discuss the implementation details of the Bag-of-Words (BoW) Classifier, specifically focusing on the code structure, functions, and their roles in achieving the outlined tasks.
The grid_points function employs a straightforward approach to generate a regular grid of feature points. Given an input gray scale image, it calculates the step size based on the number of points and the border specified. The resulting grid points are stored in a 2D array.
The descriptors_hog function implements the Histogram of Oriented Gradients (HOG) descriptor for each grid point. It utilizes the Sobel operator for gradient computation, divides the region around each grid point into cells, and constructs a histogram of gradient orientations. The final descriptor is a 128-dimensional vector, capturing the local characteristics of the image around each grid point.
The create_codebook function is responsible for constructing the visual vocabulary or codebook. It iterates through the training images, extracts grid points and HOG descriptors, and utilizes the K-Means clustering algorithm to find representative cluster centers. The resulting vCenters matrix represents the visual words that characterize the training dataset.
The bow_histogram function encodes each image as a histogram of visual words based on the constructed codebook. It assigns each descriptor to the nearest cluster center, creating a histogram that represents the distribution of visual words in the image. The histogram is then used for image classification.
The create_bow_histograms function extends the bag-of-words encoding to process an entire directory of training images. It computes the bag-of-words histogram for each image in the directory, resulting in a matrix where each row corresponds to a training example.
The bow_recognition_nearest function implements the nearest neighbor classification principle. It compares the bag-of-words histogram of a test image with histograms from positive and negative training examples. The label (1 for car or 0 for no car) is assigned based on the label of the nearest neighbor, determined by the Euclidean distance between histograms.
Testing was conducted on positive and negative samples from separate directories. The accuracy for positive samples was calculated successfully. However, for negative samples, the accuracy was found to be 0.86, indicating a strong performance in correctly classifying images without cars.

2 Part 2 - CNN-based Classifier Implementation

This section details the implementation of a simplified version of the VGG image classification network on the CIFAR-10 dataset. The provided code includes the network architecture, training script, and testing procedures.
The Vgg class defines the simplified VGG network architecture. The architecture consists of five convolutional blocks (conv_block1 to conv_block5) followed by a classifier. Each convolutional block involves convolutional layers with ReLU activation and max-pooling. The classifier includes fully connected layers with ReLU activation and dropout.
The training script (train_cifar10_vgg.py) facilitates model training. The script is configured to save trained models, training/validation logs, and tensorboard logs for monitoring the learning process. Key considerations include adjusting parameters in the args variable and monitoring training progress using TensorBoard. In Figure 1 we see the plots of the training loss and validation accuracy over the training steps.



	Run ↑	Smoothed	Value	Step	Relative
●	37886	0.2862	0.2015	17,600	27.59 min

	Run ↑	Smoothed	Value	Step	Relative
●	37886	83.5885	83.9	17,600	27.59 min

Figure 1: Plots of training loss and validation accuracy

To evaluate the model's performance on the CIFAR-10 test set, the testing script (test_-cifar10_vgg.py) is provided. Users must specify the path to their trained model and other relevant parameters in the script. The achieved test accuracy in this implementation was 82.36%, exceeding the minimum requirement of 0.6. This result demonstrates the effectiveness of the implemented VGG classifier on the given dataset.