Computer Vision
Assignment# 5

Submitted by
Shubham Kumar(21-944-491)

15 December, 2021

Contents

1 Bag-of-words Classifier **1**
1.1 Implementation Details of BoW Classifier 1
1.2 Hyperparameters 1
1.3 Results 2

2 CNN-based Classifier **3**
2.1 Implementation Details 3
2.2 Hyperparameters 4
2.3 Results 4

List of Figures

1 BoW Classifier Console Output 2
2 Training loss curve 5
3 Validation accuracy curve 5

1 Bag-of-words Classifier

1.1 Implementation Details of BoW Classifier

The implementation has the following main steps :

- **Codebook Construction**: In this step k centers of the HOG features are computed using K-means. These features are computed over sampled patches of given size. (i.e. $10 \times 10 = 100$ patches for each image, over 4×4 grid with each cell having dimension $4pixels \times 4pixels$). The gradient direction (angle, $[0, 2\pi]$) is divided into 8 bins (as mentioned in the handout) and a histogram is computed which is further used to create the feature vector (128 dimensional, *i.e.* 8 elements for each of the 16 cells). Using the features gathered from all the images, k cluster centers are found, which would later be used to create bag of visual words.
- **BoW histogram construction**: In this step HOG based binned (histogram) features are computed for a test image and each feature vector in this set of features is assigned to one of the cluster centers (based on the proximity) and the count of vectors assigned to each of these centers is used to create BoW histogram. This histogram is used as feature vector for final image classification.
- **Classification**: Such BoW histograms are kept pre-computed for training (positive as well as negative) samples. And to classify a given image, its BoW histogram is computed. The distance between the test image BoW histogram and all the precomputed histograms are calculated. The prediction is determined based on the class of pre-computed histogram to which the test image histogram is closest to.

1.2 Hyperparameters

After trying different values of k and *numiter* , **k=7** and **numiter=8** were found to give the best results. Please also refer table 1

k	numiter	positive sample accuracy	negative sample accuracy
5	6	87.75%	86.00%
6	6	97.95%	98.00%
7	6	95.91%	100.00%
5	8	91.83%	88.00%
6	8	95.92%	96.00%
7	8	100.00%	100.00%

Table 1: Hyperparameter Values and Obtained Accuracies

1.3 Results

Test positive sample accuracy: **100%**
Test negative sample accuracy: **100%**
Please refer to figure [1]

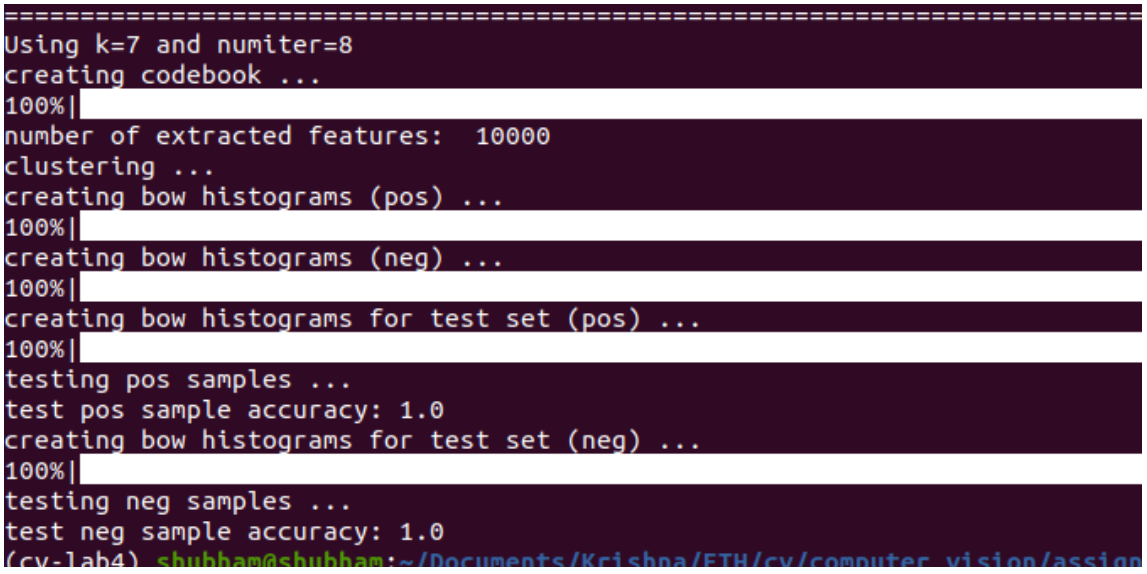

Figure 1: BoW Classifier Console Output

2 CNN-based Classifier

2.1 Implementation Details

Network Architecture: The network has been implemented as per specifications mentioned in the handout. A padding of $(1, 1)$ was used in convolutional layers to preserve the dimension after convolution operations. Dropout probability of 0.5 has been used in the *classifier* block.
The following is the output of *print(model)*:

```
Vgg(
  (conv_block1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block5): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout2d(p=0.5, inplace=False)
    (3): Linear(in_features=512, out_features=10, bias=True)
```

```
  )
)
```

2.2 Hyperparameters

The training was run with the following hyperparameter settings:

Batch Size: 128
FC Layer (num feature elements): 512
Learning rate: 0.001
Number of epochs: 50

params.json


```
{
 "batch_size": 128,
 "fc_layer": 512,
 "log_step": 100,
 "lr": 0.001,
 "num_epoch": 50,
 "root": "data/data_cnn/cifar-10-batches-py",
 "save_dir": "runs",
 "val_step": 100
}
```


2.3 Results

Final training loss: **0.2945**
Final validation accuracy: **85.58%**
Test accuracy: **84.70%**
(The trained model is located at : *runs/37686/last_model.pkl*. The default value of - *-model_path* has been changed to *runs/37686/last_model.pkl*)

Figure 2: Training loss curve

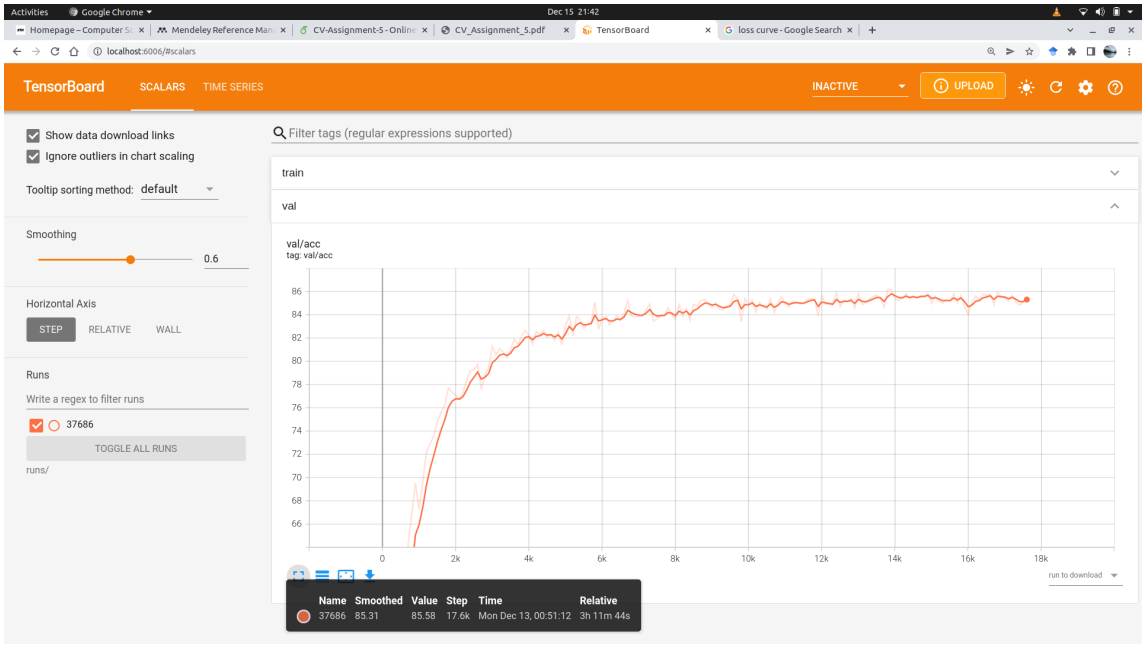

Figure 3: Validation accuracy curve