Eric Tillmann Bill: Assignment 1

erbill@student.ethz.ch

Task 2: Bag-of-words Classifier

In the following, I will briefly describe my implementation for each subtask.

Task 2.1

For the first Task 2.1.1, I calculated the grid points using the Numpy functions `linspace` and `meshgrid` to create `nPointsX`∗`nPointsY` many and evenly placed points of the image.

Then, for Task 2.1.2, I first calculate the angle of its gradients for each pixel of the image using the already given gradients in the x-direction (`grad_x`) and gradients in the y-direction (`grad_y`) by calling the following function:

```
angle = np.arctan2(grad_y, grad_x)
```

Then, in each iteration of the loop, I calculate the histogram of the angles of the current cluster by doing the following:

```
histo = np.histogram(angle[start_y:end_y, start_x:end_x],
                     bins=nBins, range=(-np.pi, np.pi))
```

Finally, I converted the list `desc` to an `np.array` and appended it to `descriptors` after calculating the descriptor for each point in `vPoints`. Similarly, I converted `descriptors` to a `np.array` as the last step and returned it.

Task 2.2

For Task 2.2, while iterating over each image in the codebook, I call the previously implemented functions as follows:

```
vPoints = grid_points(img,nPointsX,nPointsY,border)
desc = descriptors_hog(img, vPoints, cellWidth, cellHeight)
vFeatures.append(desc)
```

Specifically, I first compute the grid points of the current image (`img`) and then compute the descriptor for each of those points (`desc`). Since an implementation of the KMeans algorithm is already provided, I did not modify the function further.

Task 2.3

For Task 2.3.1, I first created an `np.array` named `dist` of dimension `(num_centers, num_features)` to store the distance from each descriptor in `vFeatures` to each of the centers in `vCenters`. I then iterate over each center and calculate the distance using `np.linalg.norm` as follows:

```
for i in range(num_centers):
    dist[i] = np.linalg.norm(vFeatures - vCenters[i], axis=1)
```

The calculated distance is the SSD distance as recommended in the lecture.

Afterward, I used the following line to calculate the final bag-of-words histogram of the given features.

```
histo = np.histogram(np.argmin(dist, axis=0), bins=num_centers)[0]
```

In this line, the `np.argmin` function is first used to select the index of the center to which each feature in `vFeature` is closest. Then, the function `np.histogram` converts this information into a histogram of dimension `(num_centers)`.

For Task 2.3.2, we implemented this function logically very similar to the loop used in `create_codebook()`. In detail, we iterate over all images, calculate for each image its `vPoints` by calling `grid_points()`, afterward we call `descriptors_hog()` with the obtained `vPoints` to obtain the descriptors of each of the `vPoints`. Finally, we call `bow_histogram()` to obtain the bag-of-words histogram for that image and save it.

Task 2.4

Similarly to Task 2.3.1, we calculate the distance between the given data using `np.linalg.norm`. In particular, we first compute the distance between the given histogram and all positive BoW histograms and then take the minimum of these distances. We proceed similarly with the negative histograms.

```
    DistPos = np.linalg.norm(histogram - vBoWPos, axis=1).min()
    DistNeg = np.linalg.norm(histogram - vBoWNeg, axis=1).min()
```

After that, we simply return 1 if the distance between the given histogram and a positive BoW histogram is smaller than the distance to all negative BoW histograms. This can be achieved by simply looking at the minimum distance of the given histogram to all positive histograms (`DistPos`) and the minimum distance to all negative BoW histograms (`DistNeg`) and simply checking if (`DistPos` $>$ `DistNeg`). If this statement is true, we return 1, otherwise 0.

Result

By setting the parameters as follows,

```
    k = 50
    numiter = 300
```

I achieve a positive test accuracy of 81.1% and a negative test accuracy of 100%, which is very good considering the simplicity of the bag-of-words approach. Below is the output generated by running my code (for ease of readability I shortened the loading bars):

```
(ex4) Task 3 python bow_main.py
creating codebook ...
100%|#| 100/100 [00:06<00:00, 15.20it/s]
number of extracted features:  10000
clustering ...
/Users/ericbill/anaconda3/envs/ex4/lib/python3.10/site-packages/sklearn/
    cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init`
    will change from 10 to 'auto' in 1.4. Set the value of `n_init`
    explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
creating bow histograms (pos) ...
100%|#| 50/50 [00:03<00:00, 14.55it/s]
creating bow histograms (neg) ...
100%|#| 50/50 [00:03<00:00, 14.62it/s]
creating bow histograms for test set (pos) ...
100%|#| 49/49 [00:03<00:00, 14.48it/s]
testing pos samples ...
test pos sample accuracy: 0.8163265306122449
creating bow histograms for test set (neg) ...
100%|#| 50/50 [00:03<00:00, 14.67it/s]
testing neg samples ...
test neg sample accuracy: 1.0
```

1 Task 3

For this task, I implemented each layer as suggested by adding the attribute **self.conv_blocki** for each convolution block i. In detail, each of my 5 convolution blocks follows the same implementation and looks similar to the first convolution block:

```
    self.conv_block1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
```

I implemented the last fully connected layer in a similar way:

```
    self.fc_block = nn.Sequential(
            nn.Linear(512, fc_layer),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(fc_layer, classes)
        )
```

Although not specified in the task, I set the probability of the dropout layer to 0.5, since this was done by the others of the actual VGG model.

Since I don't have access to GPUs and my laptop is not the fastest, I only ran the model for a total of 10 epochs and achieved an accuracy of 67.91% on the test set. Below is the output of running **`python test_cifar10_vgg.py`**:

```
(ex4) Task 3 python test_cifar10_vgg.py
[INFO] test set loaded, 10000 samples in total.
79it [00:14,  5.33it/s]
test accuracy: 67.91
```

Below are two screenshots of my model's training performance from Tensorboard.

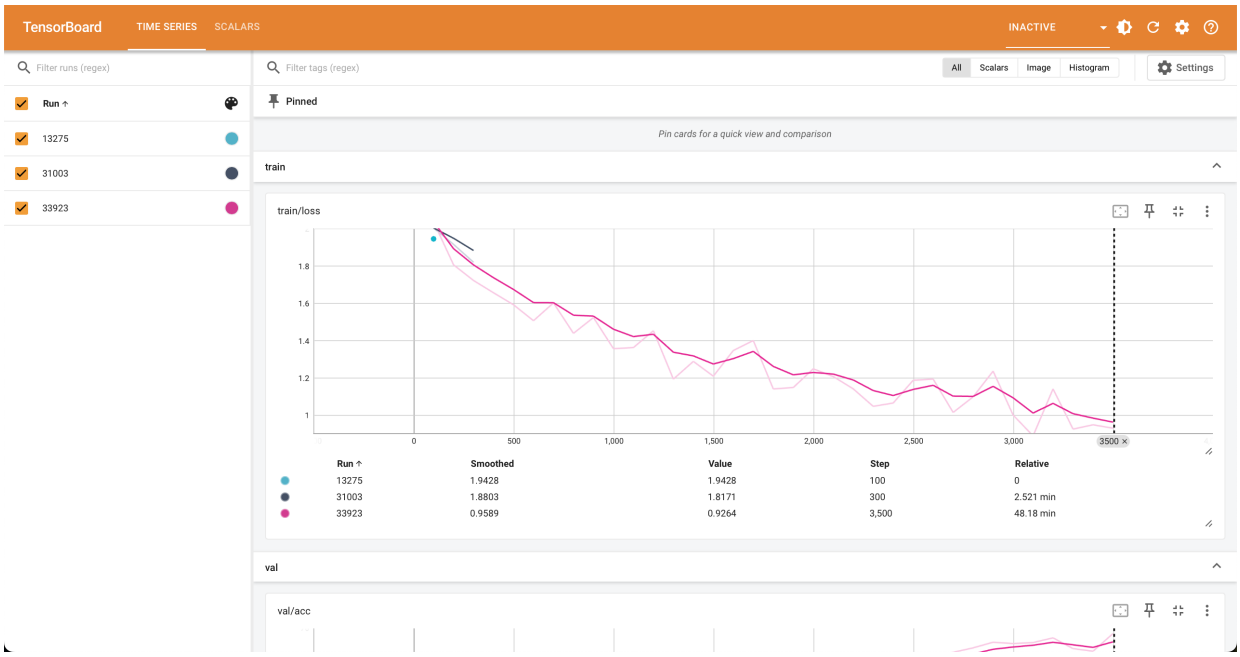

Figure 1: Screenshot of the Training Loss

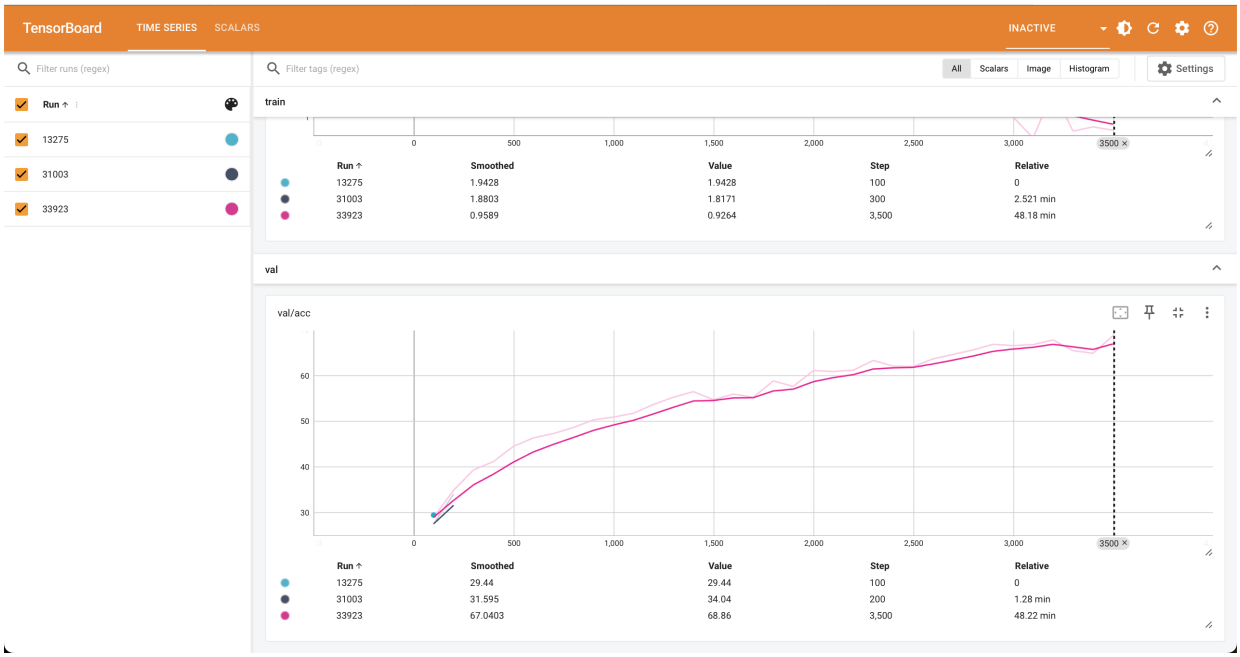

Figure 2: Screenshot of the Validation Accuracy

Below you will find the training log of my best model. It achieves a validation accuracy of 68.86%:

```
2023-10-27 15:55:16,057 INFO Let the games begin
2023-10-27 15:56:13,455 INFO [Step 99/ Epoch 0]  Loss: 2.0346
2023-10-27 15:56:34,527 INFO val acc:28.96
2023-10-27 15:56:34,618 INFO [*] best model saved

2023-10-27 15:57:35,504 INFO [Step 199/ Epoch 0]  Loss: 1.8033
2023-10-27 15:57:57,201 INFO val acc:34.9
2023-10-27 15:57:57,272 INFO [*] best model saved

2023-10-27 15:58:53,036 INFO [Step 299/ Epoch 0]  Loss: 1.7193
2023-10-27 15:59:15,468 INFO val acc:39.38
2023-10-27 15:59:15,528 INFO [*] best model saved

2023-10-27 16:00:27,019 INFO [Step 47/ Epoch 1]  Loss: 1.6534
2023-10-27 16:00:48,472 INFO val acc:41.22
2023-10-27 16:00:48,551 INFO [*] best model saved

2023-10-27 16:01:43,818 INFO [Step 147/ Epoch 1]  Loss: 1.5890
2023-10-27 16:02:05,709 INFO val acc:44.56
2023-10-27 16:02:05,789 INFO [*] best model saved

2023-10-27 16:03:02,198 INFO [Step 247/ Epoch 1]  Loss: 1.5035
2023-10-27 16:03:25,326 INFO val acc:46.4
2023-10-27 16:03:25,380 INFO [*] best model saved

2023-10-27 16:04:44,866 INFO [Step 347/ Epoch 1]  Loss: 1.5992
2023-10-27 16:05:08,633 INFO val acc:47.36
2023-10-27 16:05:08,711 INFO [*] best model saved

2023-10-27 16:06:47,134 INFO [Step 95/ Epoch 2]  Loss: 1.4356
2023-10-27 16:07:10,879 INFO val acc:48.68
2023-10-27 16:07:10,968 INFO [*] best model saved

2023-10-27 16:08:06,973 INFO [Step 195/ Epoch 2]  Loss: 1.5205
2023-10-27 16:08:28,037 INFO val acc:50.36
2023-10-27 16:08:28,123 INFO [*] best model saved

2023-10-27 16:09:20,899 INFO [Step 295/ Epoch 2]  Loss: 1.3535
2023-10-27 16:09:42,042 INFO val acc:50.98
2023-10-27 16:09:42,124 INFO [*] best model saved

2023-10-27 16:10:50,991 INFO [Step 43/ Epoch 3]  Loss: 1.3592
2023-10-27 16:11:12,729 INFO val acc:51.8
2023-10-27 16:11:12,819 INFO [*] best model saved

2023-10-27 16:12:07,534 INFO [Step 143/ Epoch 3]  Loss: 1.4495
2023-10-27 16:12:29,508 INFO val acc:53.74
2023-10-27 16:12:29,589 INFO [*] best model saved

2023-10-27 16:13:37,443 INFO [Step 243/ Epoch 3]  Loss: 1.1900
2023-10-27 16:14:02,619 INFO val acc:55.32
2023-10-27 16:14:02,709 INFO [*] best model saved

2023-10-27 16:15:11,199 INFO [Step 343/ Epoch 3]  Loss: 1.2850
```

```
2023-10-27 16:15:37,561 INFO val acc:56.56
2023-10-27 16:15:37,663 INFO [*] best model saved

2023-10-27 16:16:46,149 INFO [Step 91/ Epoch 4]  Loss: 1.2056
2023-10-27 16:17:07,122 INFO val acc:54.78
2023-10-27 16:18:06,652 INFO [Step 191/ Epoch 4]  Loss: 1.3436
2023-10-27 16:18:28,381 INFO val acc:56.02
2023-10-27 16:19:24,454 INFO [Step 291/ Epoch 4]  Loss: 1.3968
2023-10-27 16:19:45,643 INFO val acc:55.32
2023-10-27 16:20:54,130 INFO [Step 39/ Epoch 5]  Loss: 1.1372
2023-10-27 16:21:15,572 INFO val acc:58.92
2023-10-27 16:21:15,662 INFO [*] best model saved

2023-10-27 16:22:11,200 INFO [Step 139/ Epoch 5]  Loss: 1.1448
2023-10-27 16:22:32,545 INFO val acc:57.7
2023-10-27 16:23:27,549 INFO [Step 239/ Epoch 5]  Loss: 1.2453
2023-10-27 16:23:48,948 INFO val acc:61.18
2023-10-27 16:23:49,037 INFO [*] best model saved

2023-10-27 16:24:41,656 INFO [Step 339/ Epoch 5]  Loss: 1.2038
2023-10-27 16:25:02,695 INFO val acc:60.98
2023-10-27 16:26:12,315 INFO [Step 87/ Epoch 6]  Loss: 1.1377
2023-10-27 16:26:33,754 INFO val acc:61.26
2023-10-27 16:26:33,825 INFO [*] best model saved

2023-10-27 16:27:28,691 INFO [Step 187/ Epoch 6]  Loss: 1.0437
2023-10-27 16:27:50,162 INFO val acc:63.4
2023-10-27 16:27:50,243 INFO [*] best model saved

2023-10-27 16:28:44,918 INFO [Step 287/ Epoch 6]  Loss: 1.0615
2023-10-27 16:29:06,425 INFO val acc:62.16
2023-10-27 16:30:16,939 INFO [Step 35/ Epoch 7]  Loss: 1.1837
2023-10-27 16:30:38,366 INFO val acc:62.06
2023-10-27 16:31:31,132 INFO [Step 135/ Epoch 7]  Loss: 1.1901
2023-10-27 16:31:52,295 INFO val acc:63.76
2023-10-27 16:31:52,387 INFO [*] best model saved

2023-10-27 16:32:47,638 INFO [Step 235/ Epoch 7]  Loss: 1.0109
2023-10-27 16:33:09,052 INFO val acc:64.74
2023-10-27 16:33:09,151 INFO [*] best model saved

2023-10-27 16:34:03,140 INFO [Step 335/ Epoch 7]  Loss: 1.0947
2023-10-27 16:34:24,278 INFO val acc:65.72
2023-10-27 16:34:24,369 INFO [*] best model saved

2023-10-27 16:35:36,587 INFO [Step 83/ Epoch 8]  Loss: 1.2326
2023-10-27 16:35:58,359 INFO val acc:66.96
2023-10-27 16:35:58,450 INFO [*] best model saved

2023-10-27 16:36:56,838 INFO [Step 183/ Epoch 8]  Loss: 0.9972
2023-10-27 16:37:18,308 INFO val acc:66.66
2023-10-27 16:38:12,751 INFO [Step 283/ Epoch 8]  Loss: 0.8850
2023-10-27 16:38:34,028 INFO val acc:66.88
2023-10-27 16:39:47,005 INFO [Step 31/ Epoch 9]  Loss: 1.1376
2023-10-27 16:40:10,102 INFO val acc:67.92
2023-10-27 16:40:10,190 INFO [*] best model saved
```

```
2023-10-27 16:41:19,676 INFO [Step 131/ Epoch 9]  Loss: 0.9216
2023-10-27 16:41:43,568 INFO val acc:65.56
2023-10-27 16:42:56,736 INFO [Step 231/ Epoch 9]  Loss: 0.9446
2023-10-27 16:43:21,079 INFO val acc:64.98
2023-10-27 16:44:24,509 INFO [Step 331/ Epoch 9]  Loss: 0.9264
2023-10-27 16:44:47,499 INFO val acc:68.86
2023-10-27 16:44:47,594 INFO [*] best model saved
```