Lab4

Bag-of-words Classifier

1.Local Feature Extraction

1.1 Get Grid Coordinates

```
    h, w = img.shape
    step_size_x = (w - 2 * border) // nPointsX
    step_size_y = (h - 2 * border) // nPointsY
    cor_x = np.arange(w)[border:w - border:step_size_x][:nPointsX]
    cor_y = np.arange(h)[border:h - border:step_size_y][:nPointsY]
    vPoints = np.array([(x, y) for x in cor_x for y in cor_y])
```

I calculate the step_size in x, y direction respectively, then choose n points between the boarders, at last get the coordinates of the grid.

1.2 Get Cell Set Descriptors

```
        for cell_y in range(-2, 2):
            for cell_x in range(-2, 2):
                start_y = center_y + (cell_y) * h
                end_y = center_y + (cell_y + 1) * h

                start_x = center_x + (cell_x) * w
                end_x = center_x + (cell_x + 1) * w

                # compute the angles
                cell_grad_x = grad_x[start_y:end_y, start_x:end_x]
                cell_grad_y = grad_y[start_y:end_y, start_x:end_x]
                theta = np.arctan2(cell_grad_y, cell_grad_x)
                # compute the histogram
                delta = 2 * np.pi / 8
                edges = [delta * i + 1e-7 for i in range(-nBins // 2, nBins // 2
+ 1)]
                freq, _ = np.histogram(theta, bins=edges)
                desc.extend(freq)

        descriptors.append(desc)
```

For every grid points, there's a cell set around it, and there are 16 pixels in every cell, for every cell, I use the gradients in x, y direction to get the θ (gradient angle), and assign all the angles into a 8 bins histogram (should take care of the numerical accuracy when assigning because of the arctan2 did the approximation), then finally I get a $4 \times 4 \times 8$-d vector, that's a descriptor for a cell set (for a grid point).

2.Codebook construction

```
#for each image
vPoints = grid_points(img, nPointsX, nPointsY, border)
descriptors = descriptors_hog(img, vPoints, cellWidth, cellHeight)
```

I generate the Codebook including all the descriptors from the images (100 including pos and neg), so the shape is $((50+50)*10*10,\ 128)$. Then I use Kmeans to cluster these 10,000 descriptors to k centers. So the codebook shape is $(k, 128)$

3.Bag-of-words Vector Encoding

3.1 Bag-of-words histogram

```
    #descriptors for a image
    k = len(vCenters)
    vFeatures = np.expand_dims(vFeatures, axis=1)
    vCenters = np.expand_dims(vCenters, axis=0)
    distances = np.linalg.norm(vFeatures - vCenters, axis=2)
    # M-dim
    res = np.argmin(distances, axis=1)
    # N-dim
    histo = np.bincount(res, minlength=k)
```

Given every image, I have the descriptors, so according to codebook, I can find the closest centers for these descriptors, then I generate a histogram for these center_id. So the frequency describe the feature of the image, now it can be seen as a new kind of descriptors.

3.2 Processing Training images

```
#for each image
vPoints = grid_points(img, nPointsX, nPointsY, border)
vFeatures = descriptors_hog(img, vPoints, cellWidth, cellHeight)
vBoW_histo = bow_histogram(vFeatures, vCenters)
```

Do the above operations for every pos and neg images in the training set, and get 100 bow-histograms.

4.Nearest Neighbor Classification

```
# Find the nearest neighbor in the positive and negative sets and decide based on
this neighbor
_, DistPos = findnn(histogram, vBoWPos)
_, DistNeg = findnn(histogram, vBoWNeg)
```

Because the bow-histogram can describe the feature of image (a new descriptors), so just do the same operations on the test image, get the bow-histogram for it, and then use the bow-histogram to find the closest bow-histogram in the training set, and according to where the closest bow-histogram from, I can predict the class label of the test image.

5.Results

```
    k = 10  # todo
    numiter = 100  # todo
```

I find these hyper params set reach 100% accuracy .

These results are really good, even reach 100% accuracy, which prove it's a really useful method although old. I think maybe the task is too simple (just checking if there's a back view of cars), so the accuracy is pretty high.

CNN-based

1.Simplified VGG Network

First I implement the simplified VGG model network according to the guidance.

```
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv_block2 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv_block3 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv_block4 = nn.Sequential(
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv_block5 = nn.Sequential(
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3,
padding=1),
            nn.ReLU(),
```

```
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.flatten = nn.Flatten()
        self.classifier = nn.Sequential(
            nn.Linear(512 * 1 * 1, self.fc_layer),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(self.fc_layer, self.classes)
        )
```

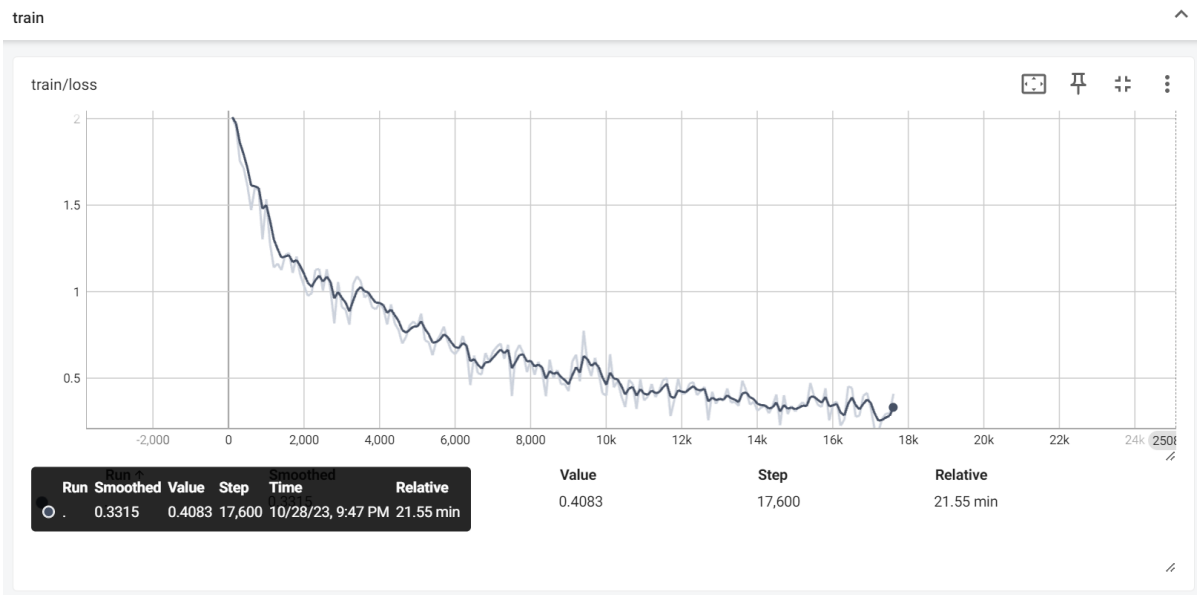
```
    def forward(self, x):
        """
        :param x: input image batch tensor, [bs, 3, 32, 32]
        :return: score: predicted score for each class (10 classes in total),
[bs, 10]
        """
        x_1 = self.conv_block1(x)
        x_2 = self.conv_block2(x_1)
        x_3 = self.conv_block3(x_2)
        x_4 = self.conv_block4(x_3)
        x_5 = self.conv_block5(x_4)
        x_flatten = self.flatten(x_5)
        score = self.classifier(x_flatten)
        return score
```

2.Results

Below are the log and tensorboard graph for training and test.Finally I reach a test accuracy of 83.65%.

2.1 Training

```
2023-10-28 21:47:09,699 INFO [Step 303/ Epoch 48]  Loss: 0.1883
2023-10-28 21:47:14,702 INFO val acc:84.1
2023-10-28 21:47:22,515 INFO [Step 51/ Epoch 49]  Loss: 0.2636
2023-10-28 21:47:27,599 INFO val acc:83.9
2023-10-28 21:47:29,193 INFO [Step 151/ Epoch 49]  Loss: 0.2927
2023-10-28 21:47:34,173 INFO val acc:84.5
2023-10-28 21:47:34,953 INFO [Step 251/ Epoch 49]  Loss: 0.2940
2023-10-28 21:47:39,747 INFO val acc:83.74
2023-10-28 21:47:40,544 INFO [Step 351/ Epoch 49]  Loss: 0.4083
2023-10-28 21:47:45,488 INFO val acc:83.76
```



2.2 Test

```
[INFO] test set loaded, 10000 samples in total.
79it [00:06, 12.41it/s]
test accuracy: 83.65
```