Lab Assignment Report - Object Recognition

Yaqi Qin*

11 November 2022

1 Bag-of-words Classifier

1.1 Local feature extraction

Feature detection. To compute an nPointsX $\times$ nPointsY regular grid leaving a border of 8 pixels, I first use the `numpy.linspace` function to get the coordinates of the y-axis and x-axis respectively, which are evenly spaced numbers (round to integers) within the range of $[border, h-border-1]$ and $[border, w-border-1]$. Then I use the `numpy.meshgrid` to construct the coordinates matrices and wrap them as 2d- coordinates which is a 100×2 numpy array.

```
h_points = np.linspace(border, h - border - 1, num = nPointsY, dtype = int)
w_points = np.linspace(border, w - border - 1, num = nPointsX, dtype = int)
hc, wc = np.meshgrid(h_points, w_points)
vPoints = np.array((wc.ravel(), hc.ravel())).T  # numpy array, [nPointsX*nPointsY, 2]
```

HOG feature description. For each cell around each vPoint:

- Compute pixel gradients. Use the same `numpy.meshgrid` trick to get the x and y coordinates of all the pixels and then retrieve their Sobel gradients along the x and y axis.
- Compute angles. For each pixel, the orientation is $arctan(y/x)$, computed using `numpy.arctan2` to ensure the output range is $[-\pi, \pi]$.
- Compute HOG. Use `numpy.histgram` to construct the HOG with fixed number of bins $(= 8)$ in the range of $[-\pi, \pi]$.
- Concatenate HOG for each vPoints (finally into a 128×1 vector).

```
cell_orients = np.arctan2(cell_grad_y,  cell_grad_x) # orientation for each pixel
cell_hist, _ = np.histogram(cell_orients, bins = nBins, range = (-np.pi, np.pi)) # cell HOG
desc.extend(cell_hist.tolist())  # concat the HOD of current cell to the descriptor i
```

1.2 Codebook construction

Feature extractions for all images. For each image in both positive and negative training sets, use the `grid_points()` function to extract vPoints. Use the `descriptors_hog()` function to extract HOG descriptor (100×128 matrix) for the image. Wrap all the features from all images into a $(100 \times nImages) \times 128$ matrix.

```
vPoints = grid_points(img, nPointsX, nPointsY, border) # [n_vPoints, 2]
vFeatures.append(descriptors_hog(img, vPoints, cellWidth, cellHeight))
```

*Student ID: 21-946-819

Cluster the features using K-Means. Use `sklear.cluster.KMeans` function to perform the cluster algorithm on the extracted feature matrix. The resulting cluster centers ($k \times 128$) are the visual words we need.

```
kmeans_res = KMeans(n_clusters=k, max_iter=numiter).fit(vFeatures)
vCenters = kmeans_res.cluster_centers_  # [k, 128]
```

1.3 Bag-of-words vector encoding

Bag-of-Words histogram. For each feature vector of a image, first get the cluster id with the minimum distance (l^2) from it using the `finnn()` function. Then the BOW histogram over visual words is computed using the `numpy.bincount` with minlength set to be k.

```
k = vCenters.shape[0]
cluster_id, _ = findnn(vFeatures, vCenters) # get the cluster assignment for each feature vectors
# set minlength to ensure the histgram is counted for all k cluster bins [k,]
histo = np.bincount(cluster_id, minlength = k)
```

Processing a directory with training examples. Given all the functions written so far, the BOW histagram of all the images from one training set is simply for each image, extracting feature points, and then building feature vectors as before. Finally, construct the BOW histagram using the `bow_histogram()` function.

```
vPoints = grid_points(img, nPointsX, nPointsY, border) # [n_vPoints, 2]
vFeatures = descriptors_hog(img, vPoints, cellWidth, cellHeight) # [n_vPoints, 128]
vBoW.append(bow_histogram(vFeatures, vCenters))
```

1.4 Nearest Neighbor Classification.

For each testing image, first use the `findnn()` function to find the nearest distance to the positive and negative training images over their BOW histogram encoding respectively. The the classification label is the class with the minimum nearest distance.

```
_, DistPos = findnn(histogram, vBoWPos)
_, DistNeg = findnn(histogram, vBoWNeg)
if (DistPos < DistNeg):
    sLabel = 1
else:
    sLabel = 0
```

1.5 Result

For testing, I set the max iteration steps to be 100, and compare the accuracy with different k numbers. The results are listed in Table 1. We can see that, in this simple classification task, $k = 10$ suffices to yield quite good accuracy on both positive (0.94) and negative samples (0.96). As k continues to increase, either positive accuracy or the negative accuracy decreases.

k	Positive Sample Accuracy	Negative Sample Accuracy
8	0.84	0.9
10	**0.94**	**0.96**
15	0.96	0.88
20	0.92	0.94
25	0.92	0.9

Table 1: Classification results with different k and max iter = 100.

2 CNN-based Classifier

2.1 A Simplified version of VGG network

The network structure is formed by 5 convolution layers and one linear classification layer. **conv_block1.**

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

conv_block2.

```
self.conv_block2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size = 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
```

conv_block3.

```
self.conv_block3 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size = 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
```

conv_block4.

```
self.conv_block4 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size = 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
```

conv_block5.

```
self.conv_block5 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size = 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
```

classifier. Here `nn.Flatten()` is performed at first to flatten all the 512 channels of the previous conv layer into one dimension.

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

2.2 Result

With the default parameter setting, I train the network for 50 epochs on CPU. The training loss and validation accuracy are shown in Figure 1.

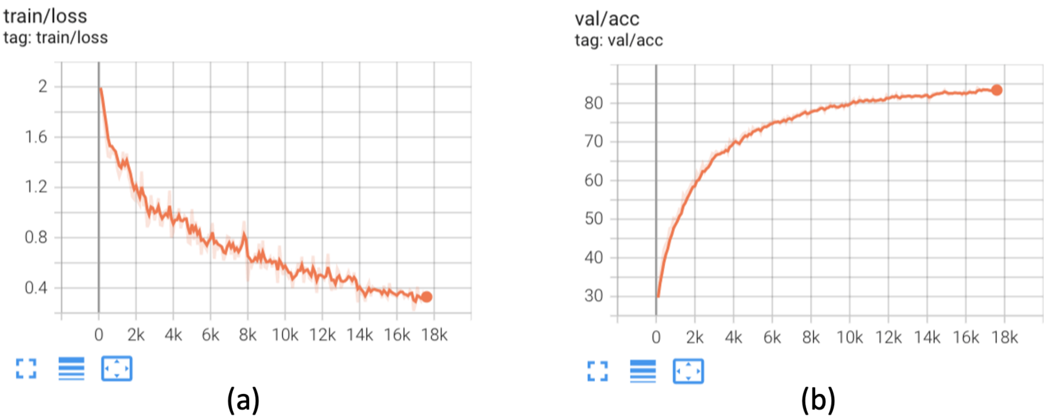

Figure 1: The screen shot of the Tensorboard with train loss (a) and validation accuracy (b). The parameters are the default setting, and the number of epochs is 50.

The accuracy for the testing images is 82.3%.

```
(cvas3) qyq@yaqis-MacBook-Pro exercise4_object_recognition_code % python test_cifar10_vgg.py --model_path='runs/13218/last_model.pkl'
[INFO] test set loaded, 10000 samples in total.
79it [00:09,  7.96it/s]
test accuracy: 82.3
```

Figure 2: The screen shot of the test output with accuracy of 82.3%.