Report for Lab exercise 4 for Computer Vision

Bag-of-words Classifier

Local Feature Extraction

- grid_points

```
# just create two linspace and combine them into a matrix
    h = img.shape[0]
    w = img.shape[1]
    x_cord = np.floor(np.linspace(border, h-border-1, 10))
    y_cord = np.floor(np.linspace(border, w-border-1, 10))
    iter = 0
    for i in range(nPointsX):
        for j in range(nPointsY):
            vPoints[iter][0] = x_cord[i]
            vPoints[iter][1] = y_cord[j]
            iter = iter + 1
```

- descriptors_hog

```
# compute the magnitude and angle for each pixel, and add to the histgram by
finding the scope where the angle is. each desc has 8 pin hist. 16 pixel
makes up the 128 pin dictionary feature
                magnitude = np.zeros((h,w))
                angle = np.zeros((h,w))
                hist = np.zeros(8) #-180 -135 -90 -45 0 45 90 135 180
                for x in range(start_x,end_x):
                    for y in range(start_y,end_y):
                        m = x - start_x
                        n = y - start_y
                        magnitude[m][n] = np.sqrt(grad_x[x][y] ** 2 +
grad_y[x][y] ** 2)
                        angle[m][n] = np.arctan2(grad_y[x][y], grad_x[x][y])
                        if -np.pi < angle[m][n] < -0.75*np.pi:
                            hist[0] += magnitude[m][n]
                        elif -0.75*np.pi < angle[m][n] < -0.5*np.pi:
                            hist[1] += magnitude[m][n]
                        elif -0.5*np.pi < angle[m][n] < -0.25*np.pi:
                            hist[2] += magnitude[m][n]
                        elif -0.25*np.pi < angle[m][n] < 0:
                            hist[3] += magnitude[m][n]
                        elif 0 < angle[m][n] < 0.25*np.pi:
                            hist[4] += magnitude[m][n]
                        elif 0.25*np.pi < angle[m][n] < 0.5*np.pi:
                            hist[5] += magnitude[m][n]
                        elif 0.5*np.pi < angle[m][n] < 0.75*np.pi:
                            hist[6] += magnitude[m][n]
                        elif 0.75*np.pi < angle[m][n] < np.pi:
                            hist[7] += magnitude[m][n]
                desc.extend(hist) # add 8 elements in desc
                # compute the histogram
                ...
        descriptors.append(desc) # 128 elements in desc
```

Codebook Construction

- create_codebook

```
# simply use the two functions in Local Feature Extract part
    for i in tqdm(range(nImgs)):
        vPoints = grid_points(img,nPointsX,nPointsY,border) # 100*2
        descriptors_eachimg =
descriptors_hog(img,vPoints,cellWidth,cellHeight) # 100*128
        vFeatures.append(descriptors_eachimg)
    # vFeatures: n_imgs * 100 * 128
```

Bags-of-words Vector Encoding

- bow_histogram

```
# We have a new image who has M feature vector.
# For each feature, we find its nearest counterpart vector in the codebook,
add to histogram
# After processing all feature in one image, we get its histogram
distribution in our codebook
    M = vFeatures.shape[0]
    N = vCenters.shape[0]
    histo = np.zeros(N)
    for index_fea in range(M):
        dist = float('inf')
        for index_center in range(N):
            if np.linalg.norm(vFeatures[index_fea]-vCenters[index_center]) <
dist:
                dist = np.linalg.norm(vFeatures[index_fea]-
vCenters[index_center])
                histo_index = index_center
        histo[histo_index] += 1

    return histo
```

- create_bow_histograms

```
# For all test image, we first compute the features for each of them.
# Then we use bow_hisogram above to get histogram distribution in codebook
for each of them
    for i in tqdm(range(nImgs)):
        # print('processing image {} ...'.format(i + 1))
        img = cv2.imread(vImgNames[i])  # [172, 208, 3]
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # [h, w]

        vPoints = grid_points(img, nPointsX, nPointsY, border)  # 100*2
        descriptors_eachimg = descriptors_hog(img, vPoints, cellWidth,
cellHeight)  # 100*128
        vBoW.append(bow_histogram(descriptors_eachimg, vCenters))
```

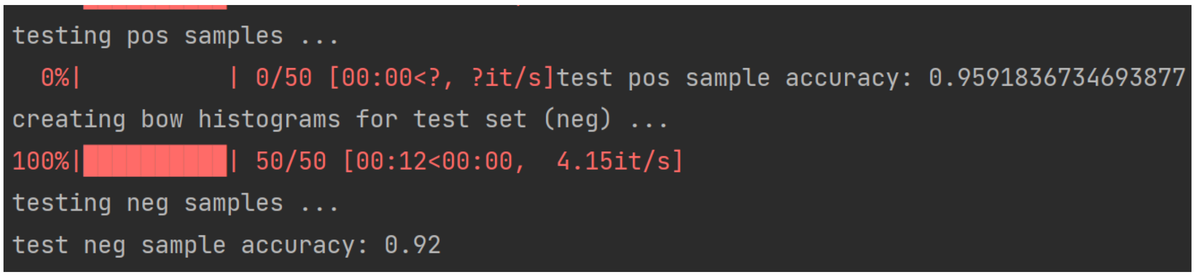
Nearest Neighbor Classification

- bow_recognition_nearest

```
# We construct two bow hist for positive and negative sets respectively.
# Then for a new image, find the nearest vector in both positive and
negative bow hist, if it's closer in positive one, we define it's positive,
vice versa.
    DistPos, DistNeg = np.Inf, np.Inf
    for i in range(vBoWPos.shape[0]):
        DistPos = np.minimum(DistPos, np.linalg.norm(histogram -
vBoWPos[i]))
    for i in range(vBoWNeg.shape[0]):
        DistNeg = np.minimum(DistNeg, np.linalg.norm(histogram -
vBoWNeg[i]))
```

Result

By setting k = 15, max iteration number = 50, we got positive accuracy 0.959 and negative accuracy 0.92.



```
testing pos samples ...
  0%|          | 0/50 [00:00<?, ?it/s]test pos sample accuracy: 0.9591836734693877
creating bow histograms for test set (neg) ...
100%|██████████| 50/50 [00:12<00:00,  4.15it/s]
testing neg samples ...
test neg sample accuracy: 0.92
```

CNN-based Classifier

VGG Network

Code implementation

- Initial function of VGG

```
# I divide VGG network into 6 blocks(5 conv blocks and 1 classifier blocks). By
computing the pixel number changes after each blocks, the parameters of stride
can be computed as 1
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.conv5 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
        self.ReLU = nn.ReLU()
        self.MaxPool2d = nn.MaxPool2d(2, 2)
        self.conv_block1 = nn.Sequential(
            self.conv1,
            self.ReLU,
            self.MaxPool2d,
        )
        self.conv_block2 = nn.Sequential(
            self.conv2,
            self.ReLU,
            self.MaxPool2d,
        )
        self.conv_block3 = nn.Sequential(
            self.conv3,
            self.ReLU,
            self.MaxPool2d,
        )
        self.conv_block4 = nn.Sequential(
            self.conv4,
            self.ReLU,
            self.MaxPool2d,
        )
        self.conv_block5 = nn.Sequential(
            self.conv5,
            self.ReLU,
            self.MaxPool2d,
        )

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

- Forward Function

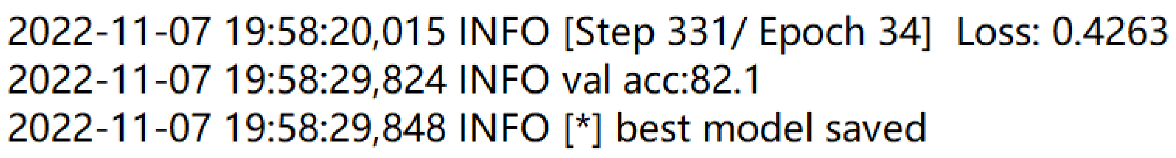
```
# We just concatenate the six blocks above together in forward func.
# The most important thing to do is to flatten output into 1*512 shape between
block5 and classifier.
# And we should notice that the input 'x' is a whole batch with size of 128. So
we need to use a for loop to divide the computing into 128 parts to get 128
scores and concatenate them.
        score = torch.zeros((len(x), 10))
        for i in range(len(x)):
            out = self.conv_block1(x[i])
            out = self.conv_block2(out)
```

```python
            out = self.conv_block3(out)
            out = self.conv_block4(out)
            out = self.conv_block5(out)
            out = torch.flatten(out, 1)
            out = torch.reshape(out, (1, 512))
            out = self.classifier(out)
            score[i] = out
        return score
```

Training and Testing

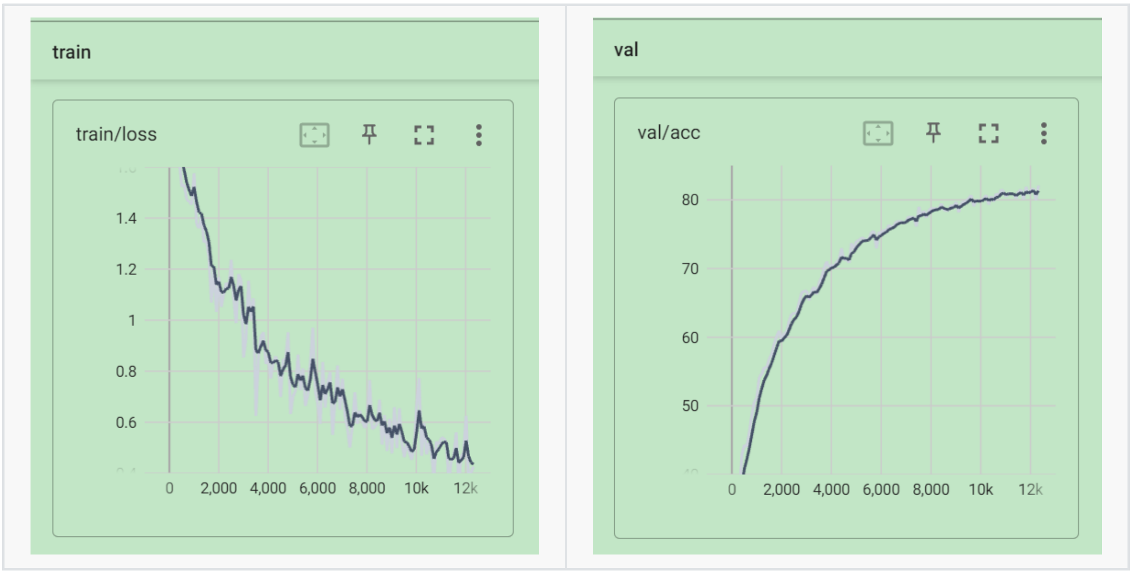
Training accuracy and Tensorboard:

- Run_log:

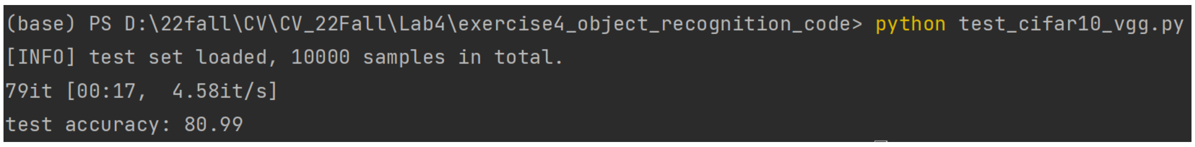


```
2022-11-07 19:58:20,015 INFO [Step 331/ Epoch 34]  Loss: 0.4263
2022-11-07 19:58:29,824 INFO val acc:82.1
2022-11-07 19:58:29,848 INFO [*] best model saved
```

- Tensorboard



Testing accuracy



```
(base) PS D:\22fall\CV\CV_22Fall\Lab4\exercise4_object_recognition_code> python test_cifar10_vgg.py
[INFO] test set loaded, 10000 samples in total.
79it [00:17,  4.58it/s]
test accuracy: 80.99
```