第五次上机作业

**18029100040**

**吴程锴**

# 作业 34：使用卷积神经网络判别狗的类别：

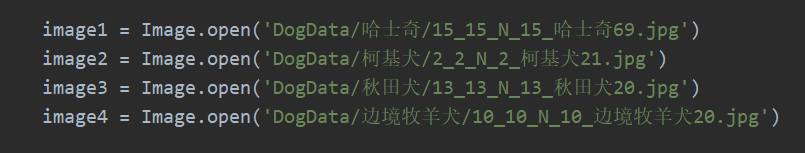
## 代码

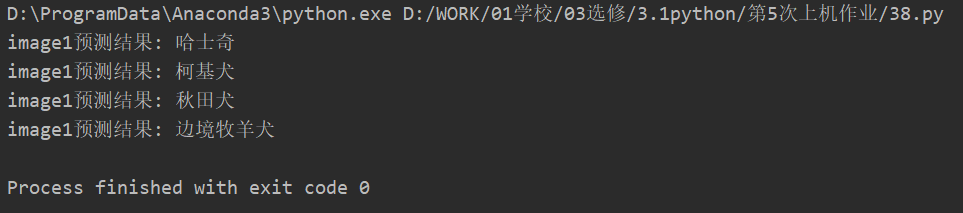
1. **import** torch
2. **import** torch.nn as nn
3. **import** torch.utils.data as Data
4. **from** torch.autograd **import** Variable
5. **import** torchvision
6. **from** torchvision **import** transforms
7. **from** torchvision.datasets **import** ImageFolder
8. **from** PIL **import** Image
9. **import** matplotlib.pyplot as plt
10. **import** numpy as np
12. LR=0.001
13. EPOCH=10
14. train=0
16. Dog\_names=['哈士奇','柯基犬','秋田犬','边境牧羊犬']
17. device = torch.device("cuda" **if** torch.cuda.is\_available() **else** "cpu")
18. size=256
19. data\_transform = transforms.Compose([
20. transforms.Resize(size),
21. transforms.CenterCrop((size, size)),
22. transforms.ToTensor(),
23. transforms.Normalize(
24. mean=[0.5,0.5,0.5],
25. std=[0.5, 0.5, 0.5])
26. ])
27. train\_dataset = ImageFolder("DogData/",transform = data\_transform)
28. train\_loader = Data.DataLoader(dataset=train\_dataset, batch\_size=10, shuffle=True, num\_workers=2)
29. img, label = train\_dataset.\_\_getitem\_\_(600)
31. # unloader = torchvision.transforms.ToPILImage()  # .ToPILImage() 把tensor或数组转换成图像
32. # def imshow(tensor, title=None):
33. #     image = tensor.cpu().clone()    # we clone the tensor to not do changes on it
34. #     image = image.squeeze(0)
35. #
36. #     image = unloader(image)         # tensor转换成图像
37. #     plt.imshow(image)
38. #     if title is not None:
39. #         plt.title(title)
40. #     plt.pause(1)                    # 只是延迟显示作用
41. #
42. # plt.figure()
43. # imshow(img, title='Image')
45. **class** CNN(nn.Module):
46. **def** \_\_init\_\_(self):
47. super(CNN,self).\_\_init\_\_()
48. self.conv1 = nn.Sequential(
49. nn.Conv2d(          #3\*256\*256
50. in\_channels=3,
51. out\_channels=16,
52. kernel\_size=5,
53. stride=1,
54. padding=2
55. ),                  #16\*256\*256
56. nn.ReLU(),          #16\*256\*256
57. nn.MaxPool2d(kernel\_size=2)#16\*128\*128
58. )
59. self.conv2 = nn.Sequential(#16\*128\*128
60. nn.Conv2d(
61. in\_channels=16,
62. out\_channels=32,
63. kernel\_size=5,
64. stride=1,
65. padding=2
66. ),                      #32\*128\*128
67. nn.ReLU(),              #32\*128\*128
68. nn.MaxPool2d(kernel\_size=2)#32\*64\*64
69. )
70. self.out = nn.Linear(32\*64\*64,4)
72. **def** forward(self,x):
73. x = self.conv1(x)
74. x = self.conv2(x)
75. x = x.view(x.size(0),-1)
76. output = self.out(x)
77. **return** output
78. **if** \_\_name\_\_ == '\_\_main\_\_':
79. cnn = CNN()
80. **if** train==1:
81. optimizer = torch.optim.Adam(cnn.parameters(),lr=LR)
82. loss\_func = nn.CrossEntropyLoss()
83. cnn=cnn.to(device)
85. **for** epoch **in** range(EPOCH):
86. **for** step, (x,y) **in** enumerate(train\_loader):
87. x = x.to(device)
88. y = y.to(device)
89. b\_x = Variable(x)
90. b\_y = Variable(y)
92. output = cnn(b\_x)
93. loss = loss\_func(output, b\_y)
94. optimizer.zero\_grad()
95. loss.backward()
96. optimizer.step()
97. **print**('loss:',float(loss.data))
98. torch.save(cnn.state\_dict(), 'CNN.pth')

101. cnn.load\_state\_dict(torch.load('CNN.pth'))
103. image1 = Image.open('DogData/哈士奇/15\_15\_N\_15\_哈士奇69.jpg')
104. image2 = Image.open('DogData/柯基犬/2\_2\_N\_2\_柯基犬21.jpg')
105. image3 = Image.open('DogData/秋田犬/13\_13\_N\_13\_秋田犬20.jpg')
106. image4 = Image.open('DogData/边境牧羊犬/10\_10\_N\_10\_边境牧羊犬20.jpg')
108. image = image1
109. image\_transformed = data\_transform(image)
110. image\_transformed = image\_transformed.unsqueeze(0)
111. image\_transformed = Variable(image\_transformed)
112. output = cnn(image\_transformed)
114. predict\_value, predict\_idx = torch.max(output, 1)
115. **print**('image1预测结果:',Dog\_names[predict\_idx])
117. image = image2
118. image\_transformed = data\_transform(image)
119. image\_transformed = image\_transformed.unsqueeze(0)
120. image\_transformed = Variable(image\_transformed)
121. output = cnn(image\_transformed)
123. predict\_value, predict\_idx = torch.max(output, 1)
124. **print**('image1预测结果:', Dog\_names[predict\_idx])
126. image = image3
127. image\_transformed = data\_transform(image)
128. image\_transformed = image\_transformed.unsqueeze(0)
129. image\_transformed = Variable(image\_transformed)
130. output = cnn(image\_transformed)
132. predict\_value, predict\_idx = torch.max(output, 1)
133. **print**('image1预测结果:', Dog\_names[predict\_idx])
135. image = image4
136. image\_transformed = data\_transform(image)
137. image\_transformed = image\_transformed.unsqueeze(0)
138. image\_transformed = Variable(image\_transformed)
139. output = cnn(image\_transformed)
141. predict\_value, predict\_idx = torch.max(output, 1)
142. **print**('image1预测结果:', Dog\_names[predict\_idx])

## 结果

每种狗选取一张测试





# 作业 35：使用 DnCNN 进行图像去噪：

## 代码

1. IMG\_H = 40
2. IMG\_W = 40
3. IMG\_C = 1
4. DEPTH = 17
5. BATCH\_SIZE = 32
6. EPOCHS = 50
7. SIGMA = 25
8. EPSILON = 1e-10
10. **from** network **import** \*
11. **from** PIL **import** Image
12. **import** scipy.misc as misc
13. **import** os

16. **class** DnCNN:
17. **def** \_\_init\_\_(self):
18. self.clean\_img = tf.placeholder(tf.float32, [None, None, None, IMG\_C])
19. self.noised\_img = tf.placeholder(tf.float32, [None, None, None, IMG\_C])
20. self.train\_phase = tf.placeholder(tf.bool)
21. dncnn = net("DnCNN")
22. self.res = dncnn(self.noised\_img, self.train\_phase)
23. self.denoised\_img = self.noised\_img - self.res
24. self.loss = tf.reduce\_mean(tf.reduce\_sum(tf.square(self.res - (self.noised\_img - self.clean\_img)), [1, 2, 3]))
25. self.Opt = tf.train.AdamOptimizer(1e-3).minimize(self.loss)
26. self.sess = tf.Session()
27. self.sess.run(tf.global\_variables\_initializer())
29. **def** train(self):
30. filepath = "./TrainingSet//"
31. filenames = os.listdir(filepath)
32. saver = tf.train.Saver()
33. **for** epoch **in** range(50):
34. **for** i **in** range(filenames.\_\_len\_\_()//BATCH\_SIZE):
35. cleaned\_batch = np.zeros([BATCH\_SIZE, IMG\_H, IMG\_W, IMG\_C])
36. **for** idx, filename **in** enumerate(filenames[i\*BATCH\_SIZE:i\*BATCH\_SIZE+BATCH\_SIZE]):
37. cleaned\_batch[idx, :, :, 0] = np.array(Image.open(filepath+filename))
38. noised\_batch = cleaned\_batch + np.random.normal(0, SIGMA, cleaned\_batch.shape)
39. self.sess.run(self.Opt, feed\_dict={self.clean\_img: cleaned\_batch, self.noised\_img: noised\_batch, self.train\_phase: True})
40. **if** i % 10 == 0:
41. [loss, denoised\_img] = self.sess.run([self.loss, self.denoised\_img], feed\_dict={self.clean\_img: cleaned\_batch, self.noised\_img: noised\_batch, self.train\_phase: False})
42. **print**("Epoch: %d, Step: %d, Loss: %g"%(epoch, i, loss))
43. compared = np.concatenate((cleaned\_batch[0, :, :, 0], noised\_batch[0, :, :, 0], denoised\_img[0, :, :, 0]), 1)
44. Image.fromarray(np.uint8(compared)).save("./TrainingResults//"+str(epoch)+"\_"+str(i)+".jpg")
45. **if** i % 500 == 0:
46. saver.save(self.sess, "./save\_para//DnCNN.ckpt")
47. np.random.shuffle(filenames)
49. **def** test(self, cleaned\_path="./TestingSet//02.png"):
50. saver = tf.train.Saver()
51. saver.restore(self.sess, "./save\_para/DnCNN.ckpt")
52. cleaned\_img = np.reshape(np.array(misc.imresize(np.array(Image.open(cleaned\_path)), [256, 256])), [1, 256, 256, 1])
53. noised\_img = cleaned\_img + np.random.normal(0, SIGMA, cleaned\_img.shape)
54. [denoised\_img] = self.sess.run([self.denoised\_img], feed\_dict={self.clean\_img: cleaned\_img, self.noised\_img: noised\_img, self.train\_phase: False})
55. compared = np.concatenate((cleaned\_img[0, :, :, 0], noised\_img[0, :, :, 0], denoised\_img[0, :, :, 0]), 1)
56. Image.fromarray(np.uint8(compared)).show()

59. **if** \_\_name\_\_ == "\_\_main\_\_":
60. dncnn = DnCNN()
61. dncnn.train()
63. **from** ops **import** \*
64. **from** config **import** \*
65. **import** numpy as np
67. **class** net:
68. **def** \_\_init\_\_(self, name):
69. self.name = name
71. **def** \_\_call\_\_(self, inputs, train\_phase):
72. with tf.variable\_scope(self.name):
73. inputs = tf.nn.relu(conv("conv0", inputs, 64, 3, 1))
74. **for** d **in** np.arange(1, DEPTH - 1):
75. inputs = tf.nn.relu(batchnorm(conv("conv\_" + str(d + 1), inputs, 64, 3, 1), train\_phase, "bn" + str(d)))
76. inputs = conv("conv" + str(DEPTH - 1), inputs, IMG\_C, 3, 1)
77. **return** inputs
79. **import** tensorflow as tf


83. **def** batchnorm(x, train\_phase, scope\_bn):
84. #Batch Normalization
85. #Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift[J]. 2015:448-456.
86. with tf.variable\_scope(scope\_bn, reuse=tf.AUTO\_REUSE):
87. beta = tf.get\_variable(name='beta', shape=[x.shape[-1]], initializer=tf.constant\_initializer([0.]), trainable=True)
88. gamma = tf.get\_variable(name='gamma', shape=[x.shape[-1]], initializer=tf.constant\_initializer([1.]), trainable=True)
89. batch\_mean, batch\_var = tf.nn.moments(x, [0, 1, 2], name='moments')
90. ema = tf.train.ExponentialMovingAverage(decay=0.5)
92. **def** mean\_var\_with\_update():
93. ema\_apply\_op = ema.apply([batch\_mean, batch\_var])
94. with tf.control\_dependencies([ema\_apply\_op]):
95. **return** tf.identity(batch\_mean), tf.identity(batch\_var)
97. mean, var = tf.cond(train\_phase, mean\_var\_with\_update,
98. **lambda**: (ema.average(batch\_mean), ema.average(batch\_var)))
99. normed = tf.nn.batch\_normalization(x, mean, var, beta, gamma, 1e-3)
100. **return** normed
102. **def** InstanceNorm(inputs, name):
103. with tf.variable\_scope(name):
104. mean, var = tf.nn.moments(inputs, axes=[1, 2], keep\_dims=True)
105. scale = tf.get\_variable("scale", shape=mean.shape[-1], initializer=tf.constant\_initializer([1.]))
106. shift = tf.get\_variable("shift", shape=mean.shape[-1], initializer=tf.constant\_initializer([0.]))
107. **return** (inputs - mean) \* scale / tf.sqrt(var + 1e-10) + shift
109. **def** conv(name, inputs, nums\_out, ksize, strides, padding="SAME", is\_SN=False):
110. with tf.variable\_scope(name):
111. W = tf.get\_variable("W", shape=[ksize, ksize, int(inputs.shape[-1]), nums\_out], initializer=tf.truncated\_normal\_initializer(stddev=0.02))
112. b = tf.get\_variable("b", shape=[nums\_out], initializer=tf.constant\_initializer(0.))
113. **if** is\_SN:
114. **return** tf.nn.conv2d(inputs, spectral\_norm(name, W), [1, strides, strides, 1], padding) + b
115. **else**:
116. **return** tf.nn.conv2d(inputs, W, [1, strides, strides, 1], padding) + b
118. **def** uconv(name, inputs, nums\_out, ksize, strides, padding="SAME"):
119. with tf.variable\_scope(name):
120. w = tf.get\_variable("W", shape=[ksize, ksize, nums\_out, int(inputs.shape[-1])], initializer=tf.truncated\_normal\_initializer(stddev=0.02))
121. b = tf.get\_variable("b", [nums\_out], initializer=tf.constant\_initializer(0.))
122. # inputs = tf.image.resize\_nearest\_neighbor(inputs, [H\*strides, W\*strides])
123. # return tf.nn.conv2d(inputs, w, [1, 1, 1, 1], padding) + b
124. **return** tf.nn.conv2d\_transpose(inputs, w, [tf.shape(inputs)[0], int(inputs.shape[1])\*strides, int(inputs.shape[2])\*strides, nums\_out], [1, strides, strides, 1], padding=padding) + b

127. **def** fully\_connected(name, inputs, nums\_out):
128. with tf.variable\_scope(name, reuse=tf.AUTO\_REUSE):
129. W = tf.get\_variable("W", [int(inputs.shape[-1]), nums\_out], initializer=tf.truncated\_normal\_initializer(stddev=0.02))
130. b = tf.get\_variable("b", [nums\_out], initializer=tf.constant\_initializer(0.))
131. **return** tf.matmul(inputs, W) + b

134. **def** spectral\_norm(name, w, iteration=1):
135. #Spectral normalization which was published on ICLR2018,please refer to "https://www.researchgate.net/publication/318572189\_Spectral\_Normalization\_for\_Generative\_Adversarial\_Networks"
136. #This function spectral\_norm is forked from "https://github.com/taki0112/Spectral\_Normalization-Tensorflow"
137. w\_shape = w.shape.as\_list()
138. w = tf.reshape(w, [-1, w\_shape[-1]])
139. with tf.variable\_scope(name, reuse=False):
140. u = tf.get\_variable("u", [1, w\_shape[-1]], initializer=tf.truncated\_normal\_initializer(), trainable=False)
141. u\_hat = u
142. v\_hat = None
144. **def** l2\_norm(v, eps=1e-12):
145. **return** v / (tf.reduce\_sum(v \*\* 2) \*\* 0.5 + eps)
147. **for** i **in** range(iteration):
148. v\_ = tf.matmul(u\_hat, tf.transpose(w))
149. v\_hat = l2\_norm(v\_)
150. u\_ = tf.matmul(v\_hat, w)
151. u\_hat = l2\_norm(u\_)
152. sigma = tf.matmul(tf.matmul(v\_hat, w), tf.transpose(u\_hat))
153. w\_norm = w / sigma
154. with tf.control\_dependencies([u.assign(u\_hat)]):
155. w\_norm = tf.reshape(w\_norm, w\_shape)
156. **return** w\_norm
158. **def** leaky\_relu(x, slope=0.2):
159. **return** tf.maximum(x, slope\*x)

## 结果





