第五次上机作业

18029100040

吴程锴

一、作业 34：使用卷积神经网络判别狗的类别：

1.1 代码

```
import torch
import torch.nn as nn
import torch.utils.data as Data
from torch.autograd import Variable
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

LR=0.001
EPOCH=10
train=0

Dog_names=['哈士奇','柯基犬','秋田犬','边境牧羊犬']
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
size=256
data_transform = transforms.Compose([
    transforms.Resize(size),
    transforms.CenterCrop((size, size)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.5,0.5,0.5],
        std=[0.5, 0.5, 0.5])
])
train_dataset = ImageFolder("DogData/",transform = data_transform)
train_loader = Data.DataLoader(dataset=train_dataset, batch_size=10, shuffle=True, num_workers=2)
img, label = train_dataset.__getitem__(600)

```

```
# unloader = torchvision.transforms.ToPILImage()  # .ToPILImage() 把 tensor 或数组转换成图像
# def imshow(tensor, title=None):
#     image = tensor.cpu().clone()    # we clone the tensor to not do changes on it

#     image = image.squeeze(0)
#
#     image = unloader(image)         # tensor 转换成图像
#     plt.imshow(image)
#     if title is not None:
#         plt.title(title)
#     plt.pause(1)                    # 只是延迟显示作用
#
# plt.figure()
# imshow(img, title='Image')

class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(          #3*256*256
                in_channels=3,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2
            ),                  #16*256*256
            nn.ReLU(),          #16*256*256
            nn.MaxPool2d(kernel_size=2)#16*128*128
        )
        self.conv2 = nn.Sequential(#16*128*128
            nn.Conv2d(
                in_channels=16,
                out_channels=32,
                kernel_size=5,
                stride=1,
                padding=2
            ),                      #32*128*128
            nn.ReLU(),              #32*128*128
            nn.MaxPool2d(kernel_size=2)#32*64*64
        )
        self.out = nn.Linear(32*64*64,4)

    def forward(self,x):
        x = self.conv1(x)
```

```
        x = self.conv2(x)
        x = x.view(x.size(0),-1)
        output = self.out(x)
        return output
if __name__ == '__main__':
    cnn = CNN()
    if train==1:
        optimizer = torch.optim.Adam(cnn.parameters(),lr=LR)
        loss_func = nn.CrossEntropyLoss()
        cnn=cnn.to(device)

        for epoch in range(EPOCH):
            for step, (x,y) in enumerate(train_loader):
                x = x.to(device)
                y = y.to(device)
                b_x = Variable(x)
                b_y = Variable(y)

                output = cnn(b_x)
                loss = loss_func(output, b_y)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                print('loss:',float(loss.data))
        torch.save(cnn.state_dict(), 'CNN.pth')


    cnn.load_state_dict(torch.load('CNN.pth'))

    image1 = Image.open('DogData/哈士奇/15_15_N_15_哈士奇 69.jpg')
    image2 = Image.open('DogData/柯基犬/2_2_N_2_柯基犬 21.jpg')
    image3 = Image.open('DogData/秋田犬/13_13_N_13_秋田犬 20.jpg')
    image4 = Image.open('DogData/边境牧羊犬/10_10_N_10_边境牧羊犬 20.jpg')

    image = image1
    image_transformed = data_transform(image)
    image_transformed = image_transformed.unsqueeze(0)
    image_transformed = Variable(image_transformed)
    output = cnn(image_transformed)

    predict_value, predict_idx = torch.max(output, 1)
    print('image1 预测结果:',Dog_names[predict_idx])

    image = image2
    image_transformed = data_transform(image)
```

```
119.     image_transformed = image_transformed.unsqueeze(0)
120.     image_transformed = Variable(image_transformed)
121.     output = cnn(image_transformed)
122.
123.     predict_value, predict_idx = torch.max(output, 1)
124.     print('image1 预测结果:', Dog_names[predict_idx])
125.
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)
131.
132.     predict_value, predict_idx = torch.max(output, 1)
133.     print('image1 预测结果:', Dog_names[predict_idx])
134.
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)
140.
141.     predict_value, predict_idx = torch.max(output, 1)
142.     print('image1 预测结果:', Dog_names[predict_idx])
```

1.2 结果

每种狗选取一张测试

```
image1 = Image.open('DogData/哈士奇/15_15_N_15_哈士奇69.jpg')
image2 = Image.open('DogData/柯基犬/2_2_N_2_柯基犬21.jpg')
image3 = Image.open('DogData/秋田犬/13_13_N_13_秋田犬20.jpg')
image4 = Image.open('DogData/边境牧羊犬/10_10_N_10_边境牧羊犬20.jpg')
```

```
D:\ProgramData\Anaconda3\python.exe D:/WORK/01学校/03选修/3.1python/第5次上机作业/38.py
image1预测结果: 哈士奇
image1预测结果: 柯基犬
image1预测结果: 秋田犬
image1预测结果: 边境牧羊犬

Process finished with exit code 0
```

二、作业 35：使用 DnCNN 进行图像去噪：

2.1 代码

```
    IMG_H = 40
IMG_W = 40
IMG_C = 1
DEPTH = 17
BATCH_SIZE = 32
EPOCHS = 50
SIGMA = 25
EPSILON = 1e-10

from network import *
from PIL import Image
import scipy.misc as misc
import os


class DnCNN:
    def __init__(self):
        self.clean_img = tf.placeholder(tf.float32, [None, None, None, IMG_C])
        self.noised_img = tf.placeholder(tf.float32, [None, None, None, IMG_C])
        self.train_phase = tf.placeholder(tf.bool)
        dncnn = net("DnCNN")
        self.res = dncnn(self.noised_img, self.train_phase)
        self.denoised_img = self.noised_img - self.res
        self.loss = tf.reduce_mean(tf.reduce_sum(tf.square(self.res - (self.noised_img - self.clean_img)), [1, 2, 3]))
        self.Opt = tf.train.AdamOptimizer(1e-3).minimize(self.loss)
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())

    def train(self):
        filepath = "./TrainingSet//"
        filenames = os.listdir(filepath)
        saver = tf.train.Saver()
        for epoch in range(50):
            for i in range(filenames.__len__()//BATCH_SIZE):
                cleaned_batch = np.zeros([BATCH_SIZE, IMG_H, IMG_W, IMG_C])
                for idx, filename in enumerate(filenames[i*BATCH_SIZE:i*BATCH_SIZE+BATCH_SIZE]):
```

```
                    cleaned_batch[idx, :, :, 0] = np.array(Image.open(filepath+filename))
                noised_batch = cleaned_batch + np.random.normal(0, SIGMA, cleaned_batch.shape)
                self.sess.run(self.Opt, feed_dict={self.clean_img: cleaned_batch, self.noised_img: noised_batch, self.train_phase: True})
                if i % 10 == 0:
                    [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})
                    print("Epoch: %d, Step: %d, Loss: %g"%(epoch, i, loss))
                    compared = np.concatenate((cleaned_batch[0, :, :, 0], noised_batch[0, :, :, 0], denoised_img[0, :, :, 0]), 1)
                    Image.fromarray(np.uint8(compared)).save("./TrainingResults//"+str(epoch)+"_"+str(i)+".jpg")
                if i % 500 == 0:
                    saver.save(self.sess, "./save_para//DnCNN.ckpt")
            np.random.shuffle(filenames)

    def test(self, cleaned_path="./TestingSet//02.png"):
        saver = tf.train.Saver()
        saver.restore(self.sess, "./save_para/DnCNN.ckpt")
        cleaned_img = np.reshape(np.array(misc.imresize(np.array(Image.open(cleaned_path)), [256, 256])), [1, 256, 256, 1])
        noised_img = cleaned_img + np.random.normal(0, SIGMA, cleaned_img.shape)
        [denoised_img] = self.sess.run([self.denoised_img], feed_dict={self.clean_img: cleaned_img, self.noised_img: noised_img, self.train_phase: False})
        compared = np.concatenate((cleaned_img[0, :, :, 0], noised_img[0, :, :, 0], denoised_img[0, :, :, 0]), 1)
        Image.fromarray(np.uint8(compared)).show()


if __name__ == "__main__":
    dncnn = DnCNN()
    dncnn.train()

from ops import *
from config import *
import numpy as np

class net:
    def __init__(self, name):
        self.name = name

    def __call__(self, inputs, train_phase):
```

```
        with tf.variable_scope(self.name):
            inputs = tf.nn.relu(conv("conv0", inputs, 64, 3, 1))
            for d in np.arange(1, DEPTH - 1):
                inputs = tf.nn.relu(batchnorm(conv("conv_" + str(d + 1), inputs, 64, 3, 1), train_phase, "bn" + str(d)))
            inputs = conv("conv" + str(DEPTH - 1), inputs, IMG_C, 3, 1)
            return inputs

import tensorflow as tf


def batchnorm(x, train_phase, scope_bn):
    #Batch Normalization
    #Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift[J]. 2015:448-456.
    with tf.variable_scope(scope_bn, reuse=tf.AUTO_REUSE):
        beta = tf.get_variable(name='beta', shape=[x.shape[-1]], initializer=tf.constant_initializer([0.]), trainable=True)
        gamma = tf.get_variable(name='gamma', shape=[x.shape[-1]], initializer=tf.constant_initializer([1.]), trainable=True)
        batch_mean, batch_var = tf.nn.moments(x, [0, 1, 2], name='moments')
        ema = tf.train.ExponentialMovingAverage(decay=0.5)

        def mean_var_with_update():
            ema_apply_op = ema.apply([batch_mean, batch_var])
            with tf.control_dependencies([ema_apply_op]):
                return tf.identity(batch_mean), tf.identity(batch_var)

        mean, var = tf.cond(train_phase, mean_var_with_update,
                            lambda: (ema.average(batch_mean), ema.average(batch_var)))
        normed = tf.nn.batch_normalization(x, mean, var, beta, gamma, 1e-3)
    return normed

def InstanceNorm(inputs, name):
    with tf.variable_scope(name):
        mean, var = tf.nn.moments(inputs, axes=[1, 2], keep_dims=True)
        scale = tf.get_variable("scale", shape=mean.shape[-1], initializer=tf.constant_initializer([1.]))
        shift = tf.get_variable("shift", shape=mean.shape[-1], initializer=tf.constant_initializer([0.]))
        return (inputs - mean) * scale / tf.sqrt(var + 1e-10) + shift

def conv(name, inputs, nums_out, ksize, strides, padding="SAME", is_SN=False):
```

```
    with tf.variable_scope(name):
        W = tf.get_variable("W", shape=[ksize, ksize, int(inputs.shape[-1]), nums_out], initializer=tf.truncated_normal_initializer(stddev=0.02))
        b = tf.get_variable("b", shape=[nums_out], initializer=tf.constant_initializer(0.))
        if is_SN:
            return tf.nn.conv2d(inputs, spectral_norm(name, W), [1, strides, strides, 1], padding) + b
        else:
            return tf.nn.conv2d(inputs, W, [1, strides, strides, 1], padding) + b

def uconv(name, inputs, nums_out, ksize, strides, padding="SAME"):
    with tf.variable_scope(name):
        w = tf.get_variable("W", shape=[ksize, ksize, nums_out, int(inputs.shape[-1])], initializer=tf.truncated_normal_initializer(stddev=0.02))
        b = tf.get_variable("b", [nums_out], initializer=tf.constant_initializer(0.))
        # inputs = tf.image.resize_nearest_neighbor(inputs, [H*strides, W*strides])

        # return tf.nn.conv2d(inputs, w, [1, 1, 1, 1], padding) + b
    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


def fully_connected(name, inputs, nums_out):
    with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
        W = tf.get_variable("W", [int(inputs.shape[-1]), nums_out], initializer=tf.truncated_normal_initializer(stddev=0.02))
        b = tf.get_variable("b", [nums_out], initializer=tf.constant_initializer(0.))
        return tf.matmul(inputs, W) + b


def spectral_norm(name, w, iteration=1):
    #Spectral normalization which was published on ICLR2018,please refer to "https://www.researchgate.net/publication/318572189_Spectral_Normalization_for_Generative_Adversarial_Networks"
    #This function spectral_norm is forked from "https://github.com/taki0112/Spectral_Normalization-Tensorflow"
    w_shape = w.shape.as_list()
    w = tf.reshape(w, [-1, w_shape[-1]])
    with tf.variable_scope(name, reuse=False):
        u = tf.get_variable("u", [1, w_shape[-1]], initializer=tf.truncated_normal_initializer(), trainable=False)
```

```
    u_hat = u
    v_hat = None

    def l2_norm(v, eps=1e-12):
        return v / (tf.reduce_sum(v ** 2) ** 0.5 + eps)

    for i in range(iteration):
        v_ = tf.matmul(u_hat, tf.transpose(w))
        v_hat = l2_norm(v_)
        u_ = tf.matmul(v_hat, w)
        u_hat = l2_norm(u_)
    sigma = tf.matmul(tf.matmul(v_hat, w), tf.transpose(u_hat))
    w_norm = w / sigma
    with tf.control_dependencies([u.assign(u_hat)]):
        w_norm = tf.reshape(w_norm, w_shape)
    return w_norm

def leaky_relu(x, slope=0.2):
    return tf.maximum(x, slope*x)
```

2.2 结果