猫狗分类

撰写人：微电子所-杨登天-202028015926089

目 录

猫狗分类

杨登天-微电子所-202028015926089

1、实验要求

1. 基于 Python 语言和任意一种深度学习框架（实验指导书中使用 Pytorch 框架进行介绍），从零开始一步步完成数据读取、网络构建、模型训练和模型测试等过程，最终实现一个可以进行猫狗图像分类的分类器。
2. 考虑到同学们机器性能的差异，该实验不强制要求使用 Kaggle 猫狗竞赛的原始数据集，大家可以根据自己的实际情况将原始数据集中训练集里的猫狗图像人为重新划分训练集和测试集。原则上要求人为划分的数据集中，训练集图像总数不少于 2000 张，测试集图像总数不少于大于 500，最终模型的准确率要求不低于 75%。鼓励在机器性能满足条件的情况下，使用大的数据集提高猫狗分类的准确率。
3. 按规定时间在课程网站提交实验报告、代码以及 PPT。

2、环境配置

Anaconda Navigator 1.10.0
Python 3.7.10 64-bit
Tensorflow 2.3.0
Visual Studio Code 1.56.2

3、实验分析

3.1 预处理模块(这两部分有参考网上教程，因为并不熟悉图片的处理方式，但学会之后用自己的语言描述)

1) 文件名读入——这一步主要是将存放图片的文件夹内所有图片读入，并用相关函数处理训练集，并将 10000 张图片随机分类，该 10000 张图片也是随机选出来的。

```
def get_files(filedir):
    cat_imag = []
    dog_imag = []
    cat_lab  = []
    dog_lab  = []
    for file in os.listdir(filedir): #os.listdir 将文件名分解
        name = file.split(sep = '.')  # 此时 name 按照 [cat or dog, 数字, jpg] 分
```

```
开
                if name[0] == 'cat':
                    cat_imag.append(filedir + file) # 这一步是为了将 train 文件夹内所有的猫图片以 path 的形式保存
                    cat_lab.append(0)
                elif name[0] == 'dog':
                    dog_imag.append(filedir + file)
                    dog_lab.append(1)
                    # 根据要求，随机选取 2500 张 train 中的图片
                imag = np.hstack((cat_imag, dog_imag))
                lab  = np.hstack((cat_lab, dog_lab))
            imag_lab = np.array([imag,lab])
            imag_lab = imag_lab.transpose()
            # 打乱 imag_lab
            np.random.shuffle(imag_lab)
            imag_rand_list = list(imag_lab[0:10000,0])
            lab_rand_list  = list(imag_lab[0:10000,1])
            lab_rand_list = [int(i) for i in lab_rand_list]
            # 选取 2000 张作为训练集
            imag_rand_list_train = imag_rand_list[0:9500]
            lab_rand_list_train = lab_rand_list[0:9500]
            # 选取 500 张作为测试集
            imag_rand_list_test = imag_rand_list[9500:10000]
            lab_rand_list_test = lab_rand_list[9500:10000]
            for i in range(500):
                test_sample_file = open("D:\\AI computer system Lab\\MyLab\\DLcourse\\Lab2\\New\\test_sample.txt",'a')
                print(imag_rand_list_test[i],file = test_sample_file)
                test_sample_file.close()
                test_label_file = open("D:\\AI computer system Lab\\MyLab\\DLcourse\\Lab2\\New\\test_label.txt",'a')
                if float(lab_rand_list_test[i]) == 1.:
                    print('dog',file = test_label_file)
                elif float(lab_rand_list_test[i]) == 0.:
                    print('cat',file = test_label_file)
                test_label_file.close()
            return imag_rand_list_train,lab_rand_list_train,imag_rand_list_test,lab_rand_list_test


    #train_dir = "D://AI computer system Lab/MyLab/DLcourse/kaggle/train/"
#image_list_train,label_list_train,image_list_test,label_list_test = get_files(train_dir)
```

2) 设置函数，可以按照特定 batch 量取出

```
def get_batch(image,label,image_W,image_H,batch_size,capacity):
    image = tf.cast(image,tf.string)
    label = tf.cast(label, tf.int32)
    input_queue = tf.train.slice_input_producer([image,label])
    label = input_queue[1]
    image_contents = tf.read_file(input_queue[0])
    image = tf.image.decode_jpeg(image_contents,channels =3)
    image = tf.image.resize_image_with_crop_or_pad(image, image_W, image_H)
    image = tf.image.per_image_standardization(image)
    image_batch, label_batch = tf.train.batch([image, label],batch_size = batch_size,
num_threads = 64, capacity = capacity)
    label_batch = tf.reshape(label_batch , [batch_size])
    image_batch = tf.cast(image_batch,tf.float32)
return    image_batch, label_batch
```

3.2 网络层模块

```
# 结构
# conv1     卷积层 1
# pooling1   池化层 1
# conv2   卷积层 2
# pooling2   池化层 2
# conv3   卷积层 3
# pooling3 池化层 3
# conv4   卷积层 4
# pooling4   池化层 4
# local3 全连接层 1
# local4 全连接层 2
# softmax 全连接层 3
```

3.3 训练并保存模型模块

采用 adam 优化器+多线程+模型保存

3.4 输出模块

```
training()
plt.figure(1)
```

```
    plt.title("Training loss--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Training loss")
    plt.plot(step_box,train_loss_box)
    plt.show()
    plt.figure(2)
    plt.title("Training acc--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Training acc")
    plt.plot(step_box,train_accu_box)
    plt.show()
```

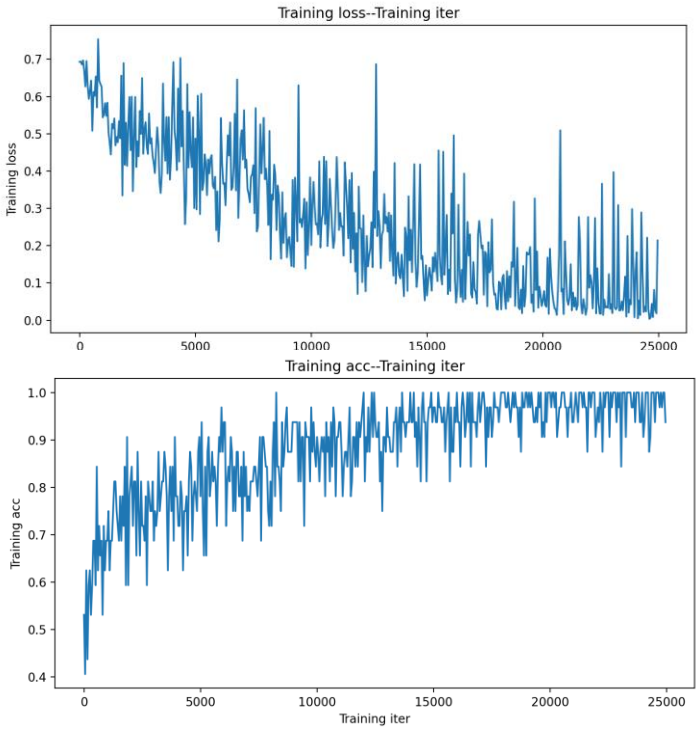
3.5 调用模型模块

```
def evaluate(train):
image_ori = getimag(train)

    with tf.Graph().as_default():
        BATCH_SIZE = 1
        CLASS = 2
        x = tf.placeholder(tf.float32, shape=[208, 208, 3])
        image = tf.cast(image_ori, tf.float32)
        image = tf.image.per_image_standardization(image)
        image = tf.reshape(image, [1, 208, 208, 3])
        logit = trainCD.inference(image, BATCH_SIZE, CLASS)
        logit = tf.nn.softmax(logit)

        logs_train_dir = "D://AI computer system
Lab/MyLab/DLcourse/Lab2/New/saveNet/"
        saver = tf.train.Saver()
        with tf.Session() as sess:
            ckpt = tf.train.get_checkpoint_state(logs_train_dir)
            saver.restore(sess, ckpt.model_checkpoint_path)
            prediction = sess.run(logit, feed_dict={x: image_ori})
            max_index = np.argmax(prediction)

            return max_index
# 测试
cat_num = 0
dog_num = 0
for i in np.arange(500):
    train = test_box[i]
    max_index = evaluate(train)
```

```
        if max_index == 0 and cat_or_dog_box[i] == 1.: #  都是猫
            cat_num += 1.
        elif max_index == 1 and cat_or_dog_box[i] == 0.: #都是狗
            dog_num += 1.
        print("第",i,"次已完成！")
real_accuracy = (cat_num + dog_num)/500.0
print("准确率是",real_accuracy)
```

4、实验结果

经过 25000 次训练之后的损失函数和正确率情况如下



经过模型调用之后的情况如下:0.78>0.75

```
第 485 次已完成!
第 486 次已完成!
第 487 次已完成!
第 488 次已完成!
第 489 次已完成!
第 490 次已完成!
第 491 次已完成!
第 492 次已完成!
第 493 次已完成!
第 494 次已完成!
第 495 次已完成!
第 496 次已完成!
第 497 次已完成!
第 498 次已完成!
第 499 次已完成!
准确率是0.78
```

5、实验心得

本次实验最大的收获在于多线程的学习，原来代码的问题跑了第一轮几乎需要等待 2 个小时才会出现一次结果，后来实在受不了了，于是查阅网络上别人怎么加速猫狗分类的方法，才发现使用多线程和线程队列的方法可以实现加速，于是便套用了这个思路，大获成功!

参考网址为：详解 Tensorflow 数据读取有三种方式（next_batch） - 闲汉 - 博客园 (cnblogs.com)