

实验三 基于 tensorflow 的 captcha 注册码识别实验

1 实验思路与代码实现

1.1 生成注册码图片

为了简化模型训练的过程，本次实验只采用数字验证码。每次从 0-9 十个数字中随机取 4 轮，生成一个数字字符，然后通过 captcha.image 生成数字验证码图片并转换为像素矩阵输出。

```
from captcha.image import ImageCaptcha
import numpy as np
from PIL import Image
import random

number = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

#从 number 列表的 10 个数中 4 次随机取数
def random_captcha_text(char_set=number, captcha_size=4):
    captcha_text = []
    for i in range(captcha_size):
        c = random.choice(char_set)
        captcha_text.append(c)
    return captcha_text

# 生成 4 个随机数字的验证码图片
def gen_captcha_text_and_image():
    image = ImageCaptcha()
    captcha_text = random_captcha_text()
    captcha_text = ''.join(captcha_text) # 连接字符串
    captcha = image.generate(captcha_text)
    captcha_image = Image.open(captcha)
    captcha_image = np.array(captcha_image)
    return captcha_text, captcha_image
```

每一步训练通过 get_next_batch 方法批量获取验证码图片，并将 gen_captcha_text_and_image 方法生成的验证码图片转换为灰度图。标签值转换为 ASCII 码组成的向量。

```
def get_next_batch(batch_size=128):
    batch_x = np.zeros([batch_size, IMAGE_HEIGHT * IMAGE_WIDTH])
    batch_y = np.zeros([batch_size, MAX_CAPTCHA * CHAR_SET_LEN])#4*10
```

```

# 有时生成图像大小不是(60, 160, 3)
def wrap_gen_captcha_text_and_image():
    while True:
        text, image = gen_captcha_text_and_image()
        if image.shape == (60, 160, 3):
            return text, image

for i in range(batch_size):
    text, image = wrap_gen_captcha_text_and_image()
    image = convert2gray(image)
    batch_x[i, :] = image.flatten() / 255 # (image.flatten()-128)/128
mean 为 0
    batch_y[i, :] = text2vec(text)

return batch_x, batch_y

```

1.2 定义卷积神经网络

本次实验的网络结构含有一个输入层，三个卷积层，一个全连接层，一个输出层。

第一层卷积层输入为 $1*60*160$ ，输出为 $32 \text{ 维} * 60 * 160$ 。

第二层卷积核大小 3×3 ，输入维度 32，输出维度 64。

第三层卷积核大小为 3×3 ，输入维度 64，输出维度 64。

每一个池化层都是步长为 2，取最大值。

第四层全连接层将 $8 * 64 * 20$ 的输入投射到 1 维 1024 的向量上。

最后输出层输出的为一个 4×10 的矩阵，存放的是四位验证码的预测值对应 10 个数字的索引。

```

# 定义卷积神经网络

def crack_captcha_cnn(w_alpha=0.01, b_alpha=0.1):
    x = tf.reshape(X, shape=[-1, IMAGE_HEIGHT, IMAGE_WIDTH, 1])
    # 3 层卷积网络
    w1 = tf.Variable(w_alpha * tf.random_normal([3, 3, 1, 32]))#
    b1 = tf.Variable(b_alpha * tf.random_normal([32]))

    conv1 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(x, w1,
strides=[1, 1, 1, 1], padding='SAME'), b1))
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1,
2, 2, 1], padding='SAME')
    conv1 = tf.nn.dropout(conv1, keep_prob)

    w2 = tf.Variable(w_alpha * tf.random_normal([3, 3, 32, 64]))

```

```

    b2 = tf.Variable(b_alpha * tf.random_normal([64]))
    conv2 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(conv1, w2,
strides=[1, 1, 1, 1], padding='SAME'), b2))
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1,
2, 2, 1], padding='SAME')
    conv2 = tf.nn.dropout(conv2, keep_prob)

    w3 = tf.Variable(w_alpha * tf.random_normal([3, 3, 64, 64]))
    b3 = tf.Variable(b_alpha * tf.random_normal([64]))
    conv3 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(conv2, w3,
strides=[1, 1, 1, 1], padding='SAME'), b3))
    conv3 = tf.nn.max_pool(conv3, ksize=[1, 2, 2, 1], strides=[1,
2, 2, 1], padding='SAME')
    conv3 = tf.nn.dropout(conv3, keep_prob)

    # Fully connected layer
    w4 = tf.Variable(w_alpha * tf.random_normal([8 * 20 * 64,
1024]))
    b4 = tf.Variable(b_alpha * tf.random_normal([1024]))
    dense = tf.reshape(conv3, [-1, w4.get_shape().as_list()[0]])
    dense = tf.nn.relu(tf.add(tf.matmul(dense, w4), b4))
    dense = tf.nn.dropout(dense, keep_prob)

    w_o = tf.Variable(w_alpha * tf.random_normal([1024, MAX_CAPTCHA
* CHAR_SET_LEN]))
    b_o = tf.Variable(b_alpha * tf.random_normal([MAX_CAPTCHA *
CHAR_SET_LEN]))
    out = tf.add(tf.matmul(dense, w_o), b_o)
    return out

```

1.3 训练网络模型参数

本次实验采用交叉熵作为模型训练的代价函数，设置模型训练次数的上限是 2500 次，每一百步输出一次准确率。当模型训练的准确率达到 0.9 时，自动保存模型退出循环。每一步训练的 batch_size 设为 64。dropout 比例设为 0.75。

```

def train_crack_captcha_cnn():
output = crack_captcha_cnn()
    #softmax 交叉熵
    loss =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=output
, labels=Y))
    optimizer =
tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

    predict = tf.reshape(output, [-1, MAX_CAPTCHA, CHAR_SET_LEN])

```

```

#返回最大值索引
max_idx_p = tf.argmax(predict, 2)
max_idx_l = tf.argmax(tf.reshape(Y, [-1, MAX_CAPTCHA,
CHAR_SET_LEN]), 2)
correct_pred = tf.equal(max_idx_p, max_idx_l)
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
saver = tf.train.Saver()
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2500):
        batch_x, batch_y = get_next_batch(64)
        _, loss_ = sess.run([optimizer, loss], feed_dict={X:
batch_x, Y: batch_y, keep_prob: 0.75})
        #print(step, loss_)
        if step % 100 == 0:
            batch_x_test, batch_y_test = get_next_batch(100)
            acc = sess.run(accuracy, feed_dict={X: batch_x_test,
Y: batch_y_test, keep_prob: 1.})
            print("step=",step,"acc=",acc)
            if acc > 0.9:
                model_path = os.getcwd() + os.sep + str(acc) +
"captcha.model"
                saver.save(sess, model_path, global_step=step)
                break

```

1.4 测试网络效果

每次测试时测试模块会测试 100 个验证码用例，最后输出正确结果数量的比例。测试模块对正确的定义是只要预测结果与标签值有一位不相同就定义为预测不正确。

```

def crack_captcha_test():
    output = crack_captcha_cnn()
    saver = tf.train.Saver()
    with tf.Session() as sess:
        saver.restore(sess, "./0.935captcha.model-1600")
        predict = tf.argmax(tf.reshape(output, [-1, MAX_CAPTCHA,
CHAR_SET_LEN]), 2)
        count = 0
        all_count = 100
        for i in range(all_count):
            text, image = gen_captcha_text_and_image()
            gray_image = convert2gray(image)
            captcha_image = gray_image.flatten() / 255
            text_list = sess.run(predict, feed_dict={X: [captcha_image],

```

```

keep_prob: 1})
    predict_text = text_list[0].tolist()
    predict_text = str(predict_text)
    predict_text = predict_text.replace("[", "").replace("]",
    "").replace(", ", "").replace(" ", "")
    if text == predict_text:
        count += 1
        check_result = ", 预测结果正确"
    else:
        f = plt.figure()
        ax = f.add_subplot(111)
        ax.text(0.1, 0.9, text, ha='center', va='center',
transform=ax.transAxes)
        plt.imshow(image)
        plt.show()
        check_result = ", 预测结果不正确"
    print("正确: {} 预测: {}".format(text, predict_text) +
check_result)
    print("正确率:", count, "/", all_count)

```

2 实验结果

图 2.1 为生成的验证码图片。

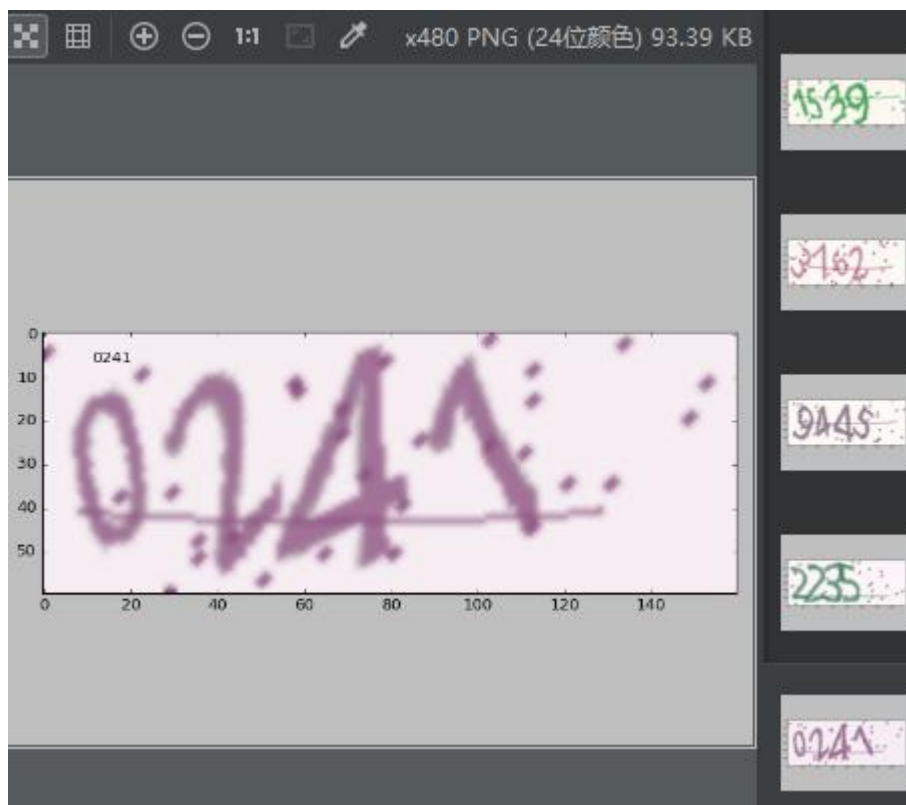
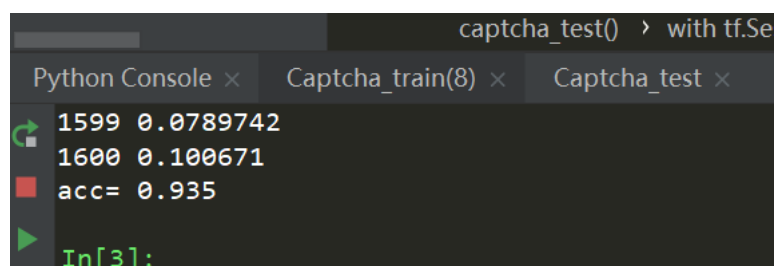


图 2.1

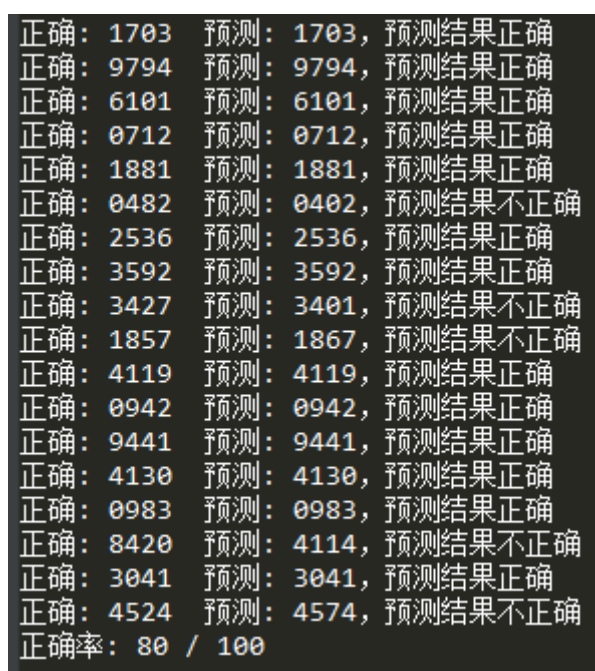
训练结果如图 2.2 所示，大约训练 1600 次后，模型的准确率达到了 0.93。



```
captcha_test() > with tf.Se
Python Console x  Captcha_train(8) x  Captcha_test x
1599 0.0789742
1600 0.100671
acc= 0.935
In[3]:
```

图 2.2

测试结果如图 2.3 所示，但是测试效果只有 0.8 的准确率，测试的准确率比训练要差不少，这是否存在过拟合呢。



```
正确: 1703 预测: 1703, 预测结果正确
正确: 9794 预测: 9794, 预测结果正确
正确: 6101 预测: 6101, 预测结果正确
正确: 0712 预测: 0712, 预测结果正确
正确: 1881 预测: 1881, 预测结果正确
正确: 0482 预测: 0402, 预测结果不正确
正确: 2536 预测: 2536, 预测结果正确
正确: 3592 预测: 3592, 预测结果正确
正确: 3427 预测: 3401, 预测结果不正确
正确: 1857 预测: 1867, 预测结果不正确
正确: 4119 预测: 4119, 预测结果正确
正确: 0942 预测: 0942, 预测结果正确
正确: 9441 预测: 9441, 预测结果正确
正确: 4130 预测: 4130, 预测结果正确
正确: 0983 预测: 0983, 预测结果正确
正确: 8420 预测: 4114, 预测结果不正确
正确: 3041 预测: 3041, 预测结果正确
正确: 4524 预测: 4574, 预测结果不正确
正确率: 80 / 100
```

图 2.3

3 问题记录

第一个问题是在训练完准确率达到 0.935 的模型后，我发现我无法读取模型进行测试，每次运行测试代码总会重新开始训练，我一直以为是我的模型保存错误，然后最后问题却是出在我使用了没有定义的变量上。

还有就是原本在训练时，我设计每一步训练后都输出 loss 值，但是每一步训练都调用 print 函数会占用较多时间，最后我选择删除了