

数字媒体技术作业3实验报告

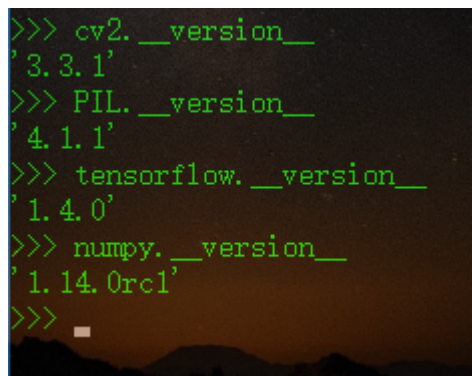
标签：树媒技术基础作业

郭柱明 15331094

语言和模块

- 基于python
- python模块
 - cv2
 - PIL
 - tensorflow
 - numpy

模块版本



```
>>> cv2.__version__
'3.3.1'
>>> PIL.__version__
'4.1.1'
>>> tensorflow.__version__
'1.4.0'
>>> numpy.__version__
'1.14.0rc1'
>>>
```

从纸张图像中截取数字

从纸上中截取数字这部分在文件get_digits_from_paper.py中实现

步骤：

1. 读取图像，转为灰度矩阵，二值化
2. 对二值化后的图像使用opencv的cv2.findContours函数进行轮廓检测，画出轮廓
3. 对画出轮廓的图像再次进行轮廓检测，提取轮廓

4. 对上第一步二值化得到的图像用正方形框出第三步检测出来的轮廓，并且把框出来的部分截取出来，每个数字，在截取的时候，对正方形的大小进行限制以过滤掉噪声

主要代码解释

提取轮廓部分

```
#读取图片为rgb数组，用作第一次轮廓检测
im = cv2.imread('input.jpg')
#转为灰度矩阵，用以二值化
imggray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
#进行二值化
ret, thresh = cv2.threshold(imggray, 169, 255, 0)

#展示二值化后的图像
Image.fromarray(thresh).show()

#使用进行了二值化的矩阵进行轮廓检测
im2, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
#在原图中画出轮廓，画迹宽度为3
cv2.drawContours(im, contours, -1, (0,255,0), 3)
#对画出了轮廓了的图像矩阵进行第二次轮廓检测(这样检测出来的轮廓才是连续的)
#转为灰度矩阵
imggray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
#第二次二值化
ret, thresh = cv2.threshold(imggray, 169, 255, 0)
#第二次轮廓检测，此时获得的轮廓才是连续的，第一次获得的是断断续续的
im2, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
#去掉第一个轮廓，那时整张A4纸的边框
contours = contours[1:]
```

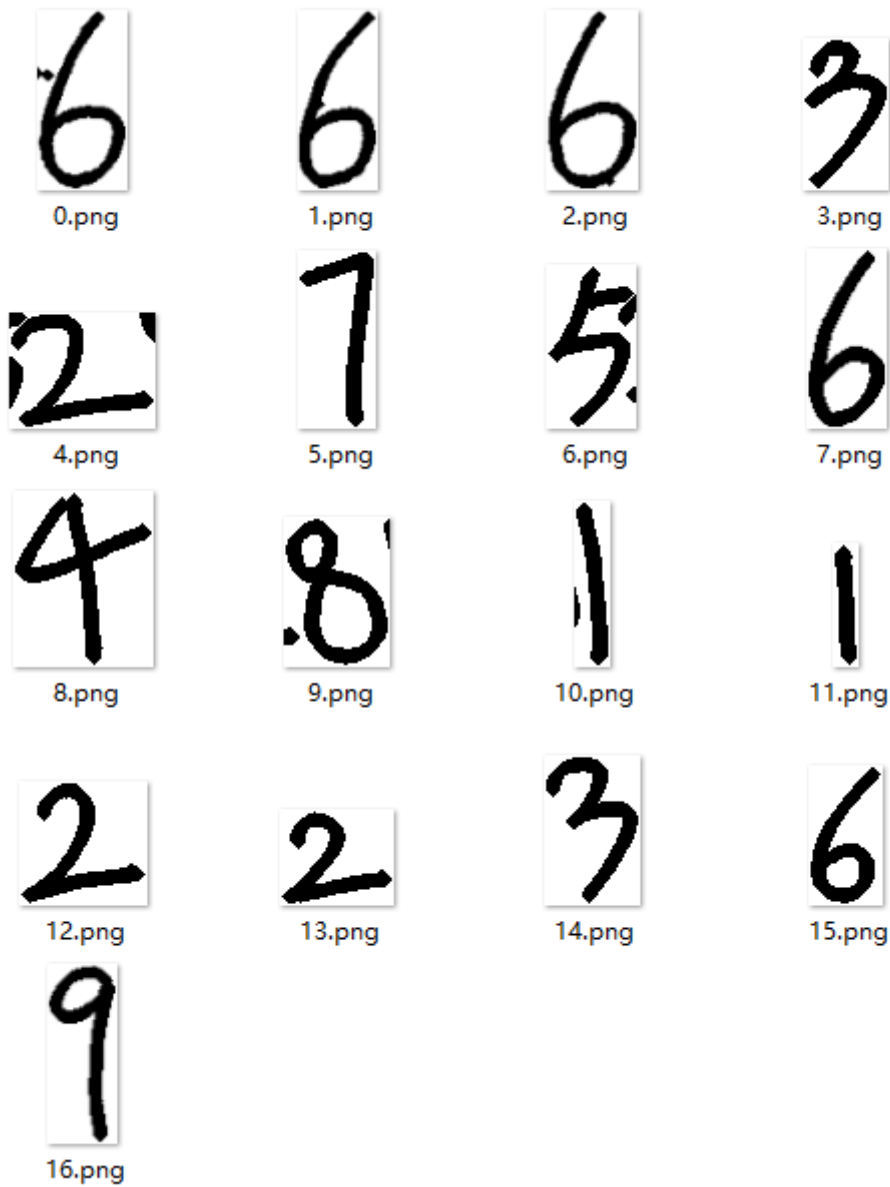
截取数字

```
#新打开一次原图用作圈出轮廓
img = cv2.imread("input.jpg")
#对每个轮廓
index = 0
for c in contours:
    #获取每个轮廓被包围的矩形的横纵坐标和宽高
    x, y, w, h = cv2.boundingRect(c)
    if h < 45 or h > 160:
        continue
    #画出包围每个轮廓的矩形
    #cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
    #展示每个被圈住的轮廓
    Image.fromarray(thresh[y:y+h,x:x+w]).save("digits/" + str(index) + ".png")
    index += 1

#展示画出包围轮廓的矩形的图像
cv2.imshow("Contours", img)
cv2.imwrite("test.png", img)

# 等待键盘输入
cv2.waitKey(0)
#关闭展示窗口
cv2.destroyAllWindows()
```

截取数字运行结果



使用线性回归识别数字

识别数字使用作业3中使用的线性回归模型进行识别

构造线性回归模型，并且训练，输出测试准确率

```

#从mnist数据集中加载训练和测试的数据
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
#x用来存放图像，维度为[None, 784]，None意味着图像数量不限定，784表示每张图表示成784
个点的一维向量
x = tf.placeholder(tf.float32, [None, 784])
#线性回归的参数W，维度为[784, 10]
W = tf.Variable(tf.zeros([784,10]))
#线性回归的参数b，维度为[10]
b = tf.Variable(tf.zeros([10]))
#让W和x矩阵相乘，加上b，然后计算softmax，softmax(x)=normalize(exp(x))
y = tf.nn.softmax(tf.matmul(x,W) + b)
#y_为用来存放正确的y结果
y_ = tf.placeholder("float", [None,10])
#使用交叉熵，以在后面使用梯度下降法根据交叉熵确定最佳的W和b
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
#使用梯度下降法最小化交叉熵
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

#初始化所有变量
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

#训练1000次
for i in range(1000):
    #每次拿100张图像和标签来训练，batch_xs为训练的图像，batch_ys为对应的标签
    batch_xs, batch_ys = mnist.train.next_batch(100)
    #根据最小梯度下降法降低交叉熵
    sess.run(train_step, {x: batch_xs, y_: batch_ys})

#correct_prediction表示下面的预测是否正确
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
#accuracy表示准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
#使用mnist.test里面的数据进行测试，计算准确率
print("accuracy", sess.run(accuracy, feed_dict = {x: mnist.test.images, y_
_: mnist.test.labels}))

```

构造resize_img函数，用以将截取出来的数字图像进行补255，补为正方形，主要是为了防止数字在resize为(28, 28)的时候变形

```

#img_arr是2维的，转为正方形
def resize_img(img_arr):
    heigh, width = img_arr.shape
    if heigh >= width:
        mark = True
        new_arr = numpy.zeros(shape = (heigh, heigh))
    else:
        mark = False
        new_arr = numpy.zeros(shape = (width, width))
    #如果高>宽 转为(heigh, heigh)
    if mark:
        i1 = int((heigh - width) / 2)
        new_arr[:, 0:i1] = 255
        new_arr[:, i1:i1 + width] = img_arr
        new_arr[:, i1 + width:] = 255
    #如果宽>高 转为(width, width)
    else:
        i1 = int((width - heigh) / 2)
        new_arr[0:i1, :] = 255
        new_arr[i1:i1 + heigh, :] = img_arr
        new_arr[i1 + heigh:, :] = 255
    return new_arr

```

识别截取出来的数字

```

#图像中正确的数字
r = [6, 6, 6, 3, 2, 7, 5, 6, 4, 8, 1, 1, 2, 2, 3, 6, 9]
#用于记录检测准确的数量
count = 0
#拿文件夹test_digits里面的0-9图片进行测试
for i in range(17):
    #读取图像为灰度图
    img = Image.open("digits/" + str(i) + ".png").convert("L")

    arr = numpy.array(img)
    #通过补255来填补为正方形，防止数字变形
    arr = resize_img(arr)
    img = Image.fromarray(arr)
    #将补零之后的图像转为(28, 28)
    img = img.resize((28, 28), Image.ANTIALIAS)
    #获取灰度矩阵
    arr = numpy.array(img, dtype = "float32")
    #图片二值化
    ret, arr = cv2.threshold(255 - arr, 90, 255, cv2.THRESH_BINARY)
    #像素值归一化和展平为一维向量
    arr = (arr / 255).flatten()
    #进行测试，输出预测结果
    narr = numpy.zeros((1, 784))
    narr[0] = arr
    y_r = sess.run(y, {x: narr})
    rec_r = sess.run(tf.argmax(y_r, 1))[0]
    if rec_r == r[i]:
        count += 1
    print("检测值:", rec_r, "真实值:", r[i])

print("准确率:", count / 17)

```

运行结果

```
C:\WINDOWS\system32\cmd.exe

G:\大三上\数图基础\作业\数字媒体技术作业4>python digital_recognition.py
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
accuracy 0.9178
检测值: 2 真实值: 6
检测值: 6 真实值: 6
检测值: 6 真实值: 6
检测值: 7 真实值: 3
检测值: 2 真实值: 2
检测值: 3 真实值: 7
检测值: 5 真实值: 5
检测值: 6 真实值: 6
检测值: 7 真实值: 4
检测值: 5 真实值: 8
检测值: 1 真实值: 1
检测值: 8 真实值: 1
检测值: 2 真实值: 2
检测值: 2 真实值: 2
检测值: 3 真实值: 3
检测值: 6 真实值: 6
检测值: 3 真实值: 9
准确率: 0.5882352941176471

G:\大三上\数图基础\作业\数字媒体技术作业4>
```

因为使用的是线性回归模型，实际在这次测试纸上的数字时，只有58.8%的准确率，本来想试一下卷积神经网络的，但是因为刚考完试就到ddl了，所以没有时间了。