<https://www.jianshu.com/p/75a8eb564de7>

# **tensorflow 构建CNN识别自己的数据（类似于MNSIT）**

**在学习了通过构建CNN来识别MNIST之后，我们都大概理解了tensorflow的构建过程，那么怎么才用CNN来识别自己的数据呢？**  
想想都有意思，那就请认真的看下面的文章

说明：

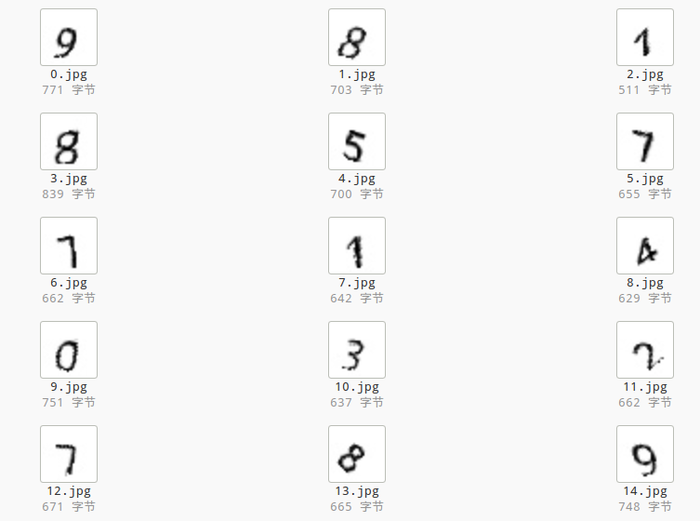
1. 图片数据是通过代码自己生成的不用下载，下面先讲的就是怎么生成图片，生成的时候需要用到 字体文件和cv2库 我在下文中都给出了地址和安装方式。
2. 生成了图片之后，训练的时候是image是从文件夹读的，label是从文本读的，如果想改，完全可以按照规则改成自己的数据，不过记得再完成卷积之后的全连接层的时候一定要改按照图片的大小改输入。
3. 有什么问题欢迎留言。

**解释基本都在代码中，基本都有注释，欢迎留言**

**完整代码和image样本：如果把data下面的训练图片和测试图片都下载了的话，可以不同再生成图片，可直接训练，再研究如何调用图片即可**[https://coding.net/u/centaur/p/Share/git/tree/master/custom\_image\_tutorials](https://link.jianshu.com?t=https://coding.net/u/centaur/p/Share/git/tree/master/custom_image_tutorials" \t "_blank)

## 生成数据

先看一下生成的数据如下图



自己生成的数据.png

这里的数据类似于MNSIT的形式，只含有十个数字。

MNIST是手写的数字，这里为了模仿这个效果对图片做了一定的扭曲和旋转  
下面做一个对比

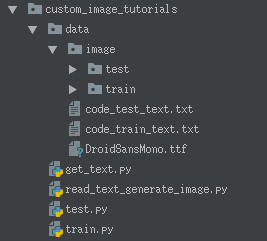


mnist数据.png



自己构建的数据.png

**因为文件中多个地方都用到了相对于代码存储位置，这里大家先看一下我的目录结构**



目录结构.png

* data下面的image分别是测试和训练的样本 路径是"data/image/train/"和"data/image/train/"
* 下面是两个文本 分别是训练和测试的label，文本中9#8#1#8#5#7#7#1#4#0#3#2#7#8#9#8#5#3#1#8#4#1#1#5#2#7#9#5#2#7#0#1#7......每两个标签之间通过#连接，图片是按照0,1,2...的顺序命名的，这样正好把image和label对应了起来，在训练的时候容易取。
* DroidSansMono.ttf是生成图片中字的字体，可做更改，代码中也要记着更改。
* get\_text.py是生成两个label文本的代码，其中可指定生成训练和测试样本的数量
* read\_text\_generate\_image.py是通过读label文本然后生成对应的图片。**再生成图片的时候一定要记着把字体放在data/目录下面**[下载地址](https://link.jianshu.com?t=https://coding.net/u/centaur/p/Share/git/blob/master/custom_image_tutorials/data/DroidSansMono.ttf" \t "_blank)
* train.py是训练的代码

#### 生成测试样本和训练样本的label。

**生成label的代码比较简单就不做赘述**

|  |
| --- |
| import random  import os  number = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']  path\_dir = "data/"  # 不存在直接创建  if not os.path.exists(path\_dir):  os.makedirs(path\_dir)  # 随机的返回一个字符，这里不只限于生成一个字符（后期扩展性）  def random\_number\_text(char\_set=number, code\_size=1):  code\_text = []  for each in range(code\_size):  c = random.choice(char\_set)  code\_text.append(c)  return code\_text  def write\_labels(size, name):  code\_list = []  for each in range(size):  number\_list = random\_number\_text()  code = ''.join(number\_list) # 用引号中的东西去连接list的两个条目  code\_list.append(code)  code\_text = '#'.join(code\_list)  print(code\_text)  f = open(path\_dir + name, 'w')  f.write(code\_text)  f.close()  def main():  # 可指定大小  trian\_size = 5000  test\_size = 1000  train\_label\_name = "code\_train\_text.txt"  test\_label\_name = "code\_test\_text.txt"  write\_labels(trian\_size, train\_label\_name)  write\_labels(test\_size, test\_label\_name)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

#### 根据样本生成image

**再生成图片的时候一定要记着把字体放在data/目录下面**[下载地址](https://link.jianshu.com?t=https://coding.net/u/centaur/p/Share/git/blob/master/custom_image_tutorials/data/DroidSansMono.ttf" \t "_blank)

有人安装opencv 请用pip install opencv-python这个指令

如果你的电脑是Windows还可能报下面这个错误:

|  |
| --- |
| import cv2  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  ImportError: DLL load failed: 找不到指定的模块。 |

不要惊慌，这是因为opencv需要依赖MSVCP140.dll 这个C++类库，而python 3.5 以上的版本不包括这个类库，你的系统正好是windows 家庭版啥的，也没有这个类库，因此去下载

不要惊慌 [解决方案](https://link.jianshu.com?t=http://www.cnblogs.com/xiezhengcai/p/python.html" \t "_blank)  
**生成图片需要用到cv2 这个库 记着安装**pip install opencv-python  
read\_text\_generate\_image.py

|  |
| --- |
| import os  import random  from PIL import Image, ImageDraw, ImageFont  import numpy as np  import cv2  # 超参数 需要的文件先创建  image\_path\_train = "data/image/train/"  if not os.path.exists(image\_path\_train):  os.makedirs(image\_path\_train)  image\_path\_test = "data/image/test/"  if not os.path.exists(image\_path\_test):  os.makedirs(image\_path\_test)  # 指定字体  DEFAULT\_FONTS = "data/DroidSansMono.ttf"  # 生成图片的大小  WIDHT = 28  HEIGHT = 28  # 把label中的内容返回list调用  def get\_content\_from\_file(label\_name):  content = open("data/" + label\_name, "r", encoding="utf-8")  code\_text = content.read()  return code\_text.split("#")  # 用opencv 转为灰度 这里需要用到cv2  def convert2gray(img):  if len(img.shape) > 2:  gray = np.mean(img, -1)  # 上面的转法较快，正规转法如下  # r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]  # gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b  return gray  else:  return img  # 再目录dir\_path下生成 名字为i.jpg 内容为c的图片  def generate\_image(i, c, dir\_path):  path = dir\_path + str(i) + ".jpg"  print(path)  color = (0, 0, 0) # 字体颜色  background = (255, 255, 255) # 背景颜色  print(str(i) + "要存的字符是" + c)  # 创建函数  image = create\_image\_one\_char(c, color, background)  image = convert2gray(np.array(image)) # 转为灰度  cv2.imwrite(path, image)# 用cv2存起来  # 更加内容c 生成扭曲和旋转的image  def create\_image\_one\_char(c, color, background):  # 自定义字体  font = ImageFont.truetype(DEFAULT\_FONTS, 30)  im = Image.new('RGBA', (WIDHT, HEIGHT), background)  drawAvatar = ImageDraw.Draw(im)  w, h = im.size  drawAvatar.text((4, -3), c, fill=color, font=font) # 在图片上写下内容  del drawAvatar # 释放对象  # 旋转 整个图片旋转  im = im.crop(im.getbbox())  im = im.rotate(random.uniform(-30, 30), Image.BILINEAR, expand=1)  # 扭曲  # 随机生成 几个坐标 为了得到相对扭曲的四个角的坐标  dx = w \* random.uniform(0.1, 0.4)  dy = h \* random.uniform(0.2, 0.5)  x1 = int(random.uniform(-dx, dx))  y1 = int(random.uniform(-dy, dy))  x2 = int(random.uniform(-dx, dx))  y2 = int(random.uniform(-dy, dy))  w2 = w + abs(x1) + abs(x2)  h2 = h + abs(y1) + abs(y2)  data = (  x1, y1,  -x1, h2 - y2,  w2 + x2, h2 + y2,  w2 - x2, -y1,  )  im = im.resize((w2, h2))    # 变量data是一个8元组(x0,y0,x1,y1,x2,y2,x3,y3)，它包括源四边形的左  # 上，左下，右下和右上四个角。 通过四个角去拉扯一张图片  im = im.transform((WIDHT, HEIGHT), Image.QUAD, data)  image = Image.new('RGB', (WIDHT, HEIGHT), background)  # 把旋转乱了的图片贴在一个正规的图片上  image.paste(im, (0, 0), im)  return image  # 超参数  train\_label\_name = "code\_train\_text.txt"  test\_label\_name = "code\_test\_text.txt"  # 根据label名和文件夹位置生成图片  def write\_image(label\_name, dir\_path):  code\_list = get\_content\_from\_file(label\_name)  for each in range(len(code\_list)):  generate\_image(each, code\_list[each], dir\_path)  def main():  # 分别处理 训练样本和测试样本  write\_image(train\_label\_name, image\_path\_train)  write\_image(test\_label\_name, image\_path\_test)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

### **CNN训练**

#### **网络模型**

****网络模型是一个两层的卷积和池化 加上两个全连接层****

|  |
| --- |
| def code\_cnn():  # 第一个卷积层  W\_conv1 = weigth\_variable([5, 5, 1, 32])  b\_conv1 = weigth\_variable([32])  h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1) # 28\*28\*32  h\_pool1 = max\_pool\_2x2(h\_conv1) # 14\*14\*32  # 第二个卷积层  W\_conv2 = weigth\_variable([5, 5, 32, 64])  b\_conv2 = weigth\_variable([64])  h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2) # 14\*14\*64  h\_pool2 = max\_pool\_2x2(h\_conv2) # 7\*7\*64  h\_pool2 = tf.nn.dropout(h\_pool2, keep\_prob)  # 三层全连接层  W\_fc1 = weigth\_variable([7 \* 7 \* 64, 1024])  b\_fc1 = bias\_varibale([1024])  # [n\_samples, 7, 7, 64] ->> [n\_samples, 7\*7\*64]  h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7 \* 7 \* 64])  h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)  h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob) # 防止过度拟合  # 第四层全连接层  W\_fc2 = weigth\_variable([1024, 10])  b\_fc2 = bias\_varibale([10])  prediction = tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)  return prediction |

#### **取数据训练**

既然把数据存成了图片那么就需要把图片读出来转成tensor

##### **1. 把image和label从文件总读出来，都成相应的list方便操作。**

|  |
| --- |
| # 根据路径得到文本的内容  def getStrContent(path):  return open(path, 'r', encoding="utf-8").read()  # 返回 训练样本路径的list 和 对应的标签用来以后训练  def get\_image\_path\_labels(IMAGE\_PATH=IMAGE\_PATH, LABEL\_PATH=LABEL\_PATH, IMAGE\_MUMBER=IMAGE\_MUMBER):  image\_path = IMAGE\_PATH  label\_path = LABEL\_PATH  image\_paths = []  for each in range(IMAGE\_MUMBER):  image\_paths.append(image\_path + str(each) + ".jpg")  string = getStrContent(label\_path)  labels = string.split("#")  return image\_paths, labels  def main():  # 得到训练样本路径list和标签的list  image\_paths, labels = get\_image\_path\_labels()  train\_code\_cnn(image\_paths, labels) |

##### **2. 数据被放在了对应的list中那么就可对应的取数据，训练数据**

我们训练样本生成了5000个测试样本生成了1000个  
下面代码中的batch设置为了100 每次把100张图片存成一个tensor.  
所以我们需要对5000个数据进行迭代读取。  
****生成一个batch****

|  |
| --- |
| # 生成一个训练batch 把batch个image和lebel拼成两个tensor  # 这里的each是一个迭代器 因为有5000个图片每次100个，所以是循环传入的0到49  def get\_next\_batch(batch\_size, each, images, labels):  # image的tensor  batch\_x = np.zeros([batch\_size, IMAGE\_HEIGHT \* IMAGE\_WIDTH])  # label的tensor  batch\_y = np.zeros([batch\_size, 10])  def get\_text\_and\_image(i, each):  image\_num = each \* batch\_size + i  label = labels[image\_num]  image\_path = images[image\_num]  captcha\_image = Image.open(image\_path) #按照路径打开图片  captcha\_image = np.array(captcha\_image)  return label, captcha\_image  # 按照 batch\_size迭代  for i in range(batch\_size):  text, image = get\_text\_and\_image(i, each)  image = convert2gray(image)#转为灰度  batch\_x[i, :] = image.flatten() / 255 # (image.flatten()-128)/128 mean为0  batch\_y[i, :] = text2vec(text)  return batch\_x, batch\_y |

****训练数据****  
****在训练的时候会进行测试，分别用测试数据集和训练数据集合进行测试****  
****一般来说训练数据要比测试数据好很多，不过我训练了一会发现两个差不多****

|  |
| --- |
| **for epoch in range(EPOCH):**  **# 每个epoch**  **for each in range(int(IMAGE\_MUMBER / BATCH\_SIZE)):**  **batch\_x, batch\_y = get\_next\_batch(BATCH\_SIZE, each, image\_paths, labels)**  **\_, loss\_ = sess.run([train\_step, cross\_entropy]**  **, feed\_dict={xs: batch\_x, ys: batch\_y, keep\_prob: 0.5})**  **print("epoch: %d iter: %d/%d loss: %f"**  **% (epoch + 1, BATCH\_SIZE \* each, IMAGE\_MUMBER, loss\_))**  **# 测试样本计算准确率**  **# 这里还是按照训练的时候的方法，把image好label分别都成list**  **# 然后统一转为tensor再通过测试函数进行测试。**  **test\_iamge\_path = "data/image/test/"**  **test\_labels\_path = "data/code\_test\_text.txt"**  **test\_image\_paths, test\_labels = \**  **get\_image\_path\_labels(test\_iamge\_path, test\_labels\_path, 200)**  **batch\_x\_test, batch\_y\_test = \**  **get\_random\_batch(BATCH\_SIZE, test\_image\_paths, test\_labels,200)**  **accuracy\_test = compute\_accuracy(batch\_x\_test, batch\_y\_test, sess)**  **print("测试样本测试 epoch: %d acc: %f" % (epoch + 1, accuracy\_test))**  **# 训练样本计算准确率**  **batch\_x\_test, batch\_y\_test = get\_random\_batch(BATCH\_SIZE, image\_paths, labels)**  **accuracy = compute\_accuracy(batch\_x\_test, batch\_y\_test, sess)**  **print("训练样本测试 epoch: %d acc: %f" % (epoch + 1, accuracy))** |

因为图片比较小，训练起来很快 再28个epoch的时候训练数据能达到100%，训练数据能达到99%

|  |
| --- |
| epoch: 28 iter: 4300/5000 loss: 0.070238  epoch: 28 iter: 4400/5000 loss: 0.039228  epoch: 28 iter: 4500/5000 loss: 0.039181  epoch: 28 iter: 4600/5000 loss: 0.048799  epoch: 28 iter: 4700/5000 loss: 0.165373  epoch: 28 iter: 4800/5000 loss: 0.040288  epoch: 28 iter: 4900/5000 loss: 0.061203  测试样本 epoch: 28 acc: 0.990000  训练样本 epoch: 28 acc: 1.000000 |