验证码是根据随机字符生成一幅图片，然后在图片中加入干扰象素，用户必须手动填入，防止有人利用机器人自动批量注册、灌水、发垃圾广告等等 。

*验证码的作用是验证用户是真人还是机器人；设计理念是对人友好，对机器难。*

TensorFlow练习20: 使用深度学习破解验证码

上图是常见的字符验证码，还有一些验证码使用提问的方式。

**我们先来看看破解验证码的几种方式：**

1. 人力打码（基本上，打码任务都是大型网站的验证码，用于自动化注册等等）
2. 找到能过验证码的漏洞
3. 最后一种是字符识别，这是本帖的关注点

我上网查了查，用Tesseract OCR、**[OpenCV](http://lib.csdn.net/base/opencv" \o "OpenCV知识库" \t "_blank)**等等其它方法都需把验证码分割为单个字符，然后识别单个字符。分割验证码可是人的强项，如果字符之间相互重叠，那机器就不容易分割了。

本帖实现的方法不需要分割验证码，而是把验证码做为一个整体进行识别。

**相关论文**：

* [Multi-digit Number Recognition from Street View Imagery using Deep CNN](https://arxiv.org/pdf/1312.6082.pdf)
* [CAPTCHA Recognition with Active Deep Learning](https://vision.in.tum.de/_media/spezial/bib/stark-gcpr15.pdf)
* <http://matthewearl.github.io/2016/05/06/cnn-anpr/>

使用**[深度学习](http://lib.csdn.net/base/deeplearning" \o "深度学习知识库" \t "_blank)**+训练数据+大量计算力，我们可以在几天内训练一个可以破解验证码的模型，当然前提是获得大量训练数据。

获得训练数据方法：

1. 手动（累死人系列）
2. 破解验证码生成机制，自动生成无限多的训练数据
3. 打入敌人内部（卧底+不要脸+不要命+多大仇系列）

我自己做一个验证码生成器，然后训练CNN模型破解自己做的验证码生成器。

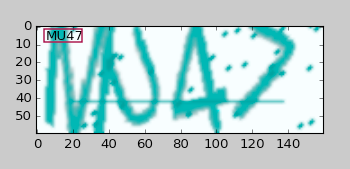
我觉的验证码机制可以废了，单纯的增加验证码难度只会让人更难识别，使用[CNN](http://blog.topspeedsnail.com/archives/10833)+[RNN](http://blog.topspeedsnail.com/archives/10542)，机器的识别准确率不比人差。Google已经意识到了这一点，他们现在使用**[机器学习](http://lib.csdn.net/base/machinelearning" \o "机器学习知识库" \t "_blank)**技术检测异常流量。

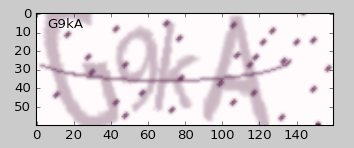
**验证码生成器**

**[python]** [view plain](http://blog.csdn.net/u014365862/article/details/53869816) [copy](http://blog.csdn.net/u014365862/article/details/53869816)

[在CODE上查看代码片](https://code.csdn.net/snippets/2075286)

1. **from** captcha.image **import** ImageCaptcha  # pip install captcha
2. **import** numpy as np
3. **import** matplotlib.pyplot as plt
4. **from** PIL **import** Image
5. **import** random
7. # 验证码中的字符, 就不用汉字了
8. number = ['0','1','2','3','4','5','6','7','8','9']
9. alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
10. ALPHABET = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
11. # 验证码一般都无视大小写；验证码长度4个字符
12. **def** random\_captcha\_text(char\_set=number+alphabet+ALPHABET, captcha\_size=4):
13. captcha\_text = []
14. **for** i **in** range(captcha\_size):
15. c = random.choice(char\_set)
16. captcha\_text.append(c)
17. **return** captcha\_text
19. # 生成字符对应的验证码
20. **def** gen\_captcha\_text\_and\_image():
21. image = ImageCaptcha()
23. captcha\_text = random\_captcha\_text()
24. captcha\_text = ''.join(captcha\_text)
26. captcha = image.generate(captcha\_text)
27. #image.write(captcha\_text, captcha\_text + '.jpg')  # 写到文件
29. captcha\_image = Image.open(captcha)
30. captcha\_image = np.array(captcha\_image)
31. **return** captcha\_text, captcha\_image
33. **if** \_\_name\_\_ == '\_\_main\_\_':
34. # 测试
35. text, image = gen\_captcha\_text\_and\_image()
37. f = plt.figure()
38. ax = f.add\_subplot(111)
39. ax.text(0.1, 0.9,text, ha='center', va='center', transform=ax.transAxes)
40. plt.imshow(image)
42. plt.show()

左上角文本对应验证码图像



**训练**

**[python]** [view plain](http://blog.csdn.net/u014365862/article/details/53869816) [copy](http://blog.csdn.net/u014365862/article/details/53869816)

[在CODE上查看代码片](https://code.csdn.net/snippets/2075286)

1. **from** gen\_captcha **import** gen\_captcha\_text\_and\_image
2. **from** gen\_captcha **import** number
3. **from** gen\_captcha **import** alphabet
4. **from** gen\_captcha **import** ALPHABET
6. **import** numpy as np
7. **import** tensorflow as tf
9. text, image = gen\_captcha\_text\_and\_image()
10. **print**("验证码图像channel:", image.shape)  # (60, 160, 3)
11. # 图像大小
12. IMAGE\_HEIGHT = 60
13. IMAGE\_WIDTH = 160
14. MAX\_CAPTCHA = len(text)
15. **print**("验证码文本最长字符数", MAX\_CAPTCHA)   # 验证码最长4字符; 我全部固定为4,可以不固定. 如果验证码长度小于4，用'\_'补齐
17. # 把彩色图像转为灰度图像（色彩对识别验证码没有什么用）
18. **def** convert2gray(img):
19. **if** len(img.shape) > 2:
20. gray = np.mean(img, -1)
21. # 上面的转法较快，正规转法如下
22. # r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]
23. # gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b
24. **return** gray
25. **else**:
26. **return** img
28. """
29. cnn在图像大小是2的倍数时性能最高, 如果你用的图像大小不是2的倍数，可以在图像边缘补无用像素。
30. np.pad(image,((2,3),(2,2)), 'constant', constant\_values=(255,))  # 在图像上补2行，下补3行，左补2行，右补2行
31. """
33. # 文本转向量
34. char\_set = number + alphabet + ALPHABET + ['\_']  # 如果验证码长度小于4, '\_'用来补齐
35. CHAR\_SET\_LEN = len(char\_set)
36. **def** text2vec(text):
37. text\_len = len(text)
38. **if** text\_len > MAX\_CAPTCHA:
39. **raise** ValueError('验证码最长4个字符')
41. vector = np.zeros(MAX\_CAPTCHA\*CHAR\_SET\_LEN)
42. **def** char2pos(c):
43. **if** c =='\_':
44. k = 62
45. **return** k
46. k = ord(c)-48
47. **if** k > 9:
48. k = ord(c) - 55
49. **if** k > 35:
50. k = ord(c) - 61
51. **if** k > 61:
52. **raise** ValueError('No Map')
53. **return** k
54. **for** i, c **in** enumerate(text):
55. idx = i \* CHAR\_SET\_LEN + char2pos(c)
56. vector[idx] = 1
57. **return** vector
58. # 向量转回文本
59. **def** vec2text(vec):
60. char\_pos = vec.nonzero()[0]
61. text=[]
62. **for** i, c **in** enumerate(char\_pos):
63. char\_at\_pos = i #c/63
64. char\_idx = c % CHAR\_SET\_LEN
65. **if** char\_idx < 10:
66. char\_code = char\_idx + ord('0')
67. **elif** char\_idx <36:
68. char\_code = char\_idx - 10 + ord('A')
69. **elif** char\_idx < 62:
70. char\_code = char\_idx-  36 + ord('a')
71. **elif** char\_idx == 62:
72. char\_code = ord('\_')
73. **else**:
74. **raise** ValueError('error')
75. text.append(chr(char\_code))
76. **return** "".join(text)
78. """
79. #向量（大小MAX\_CAPTCHA\*CHAR\_SET\_LEN）用0,1编码 每63个编码一个字符，这样顺利有，字符也有
80. vec = text2vec("F5Sd")
81. text = vec2text(vec)
82. print(text)  # F5Sd
83. vec = text2vec("SFd5")
84. text = vec2text(vec)
85. print(text)  # SFd5
86. """
88. # 生成一个训练batch
89. **def** get\_next\_batch(batch\_size=128):
90. batch\_x = np.zeros([batch\_size, IMAGE\_HEIGHT\*IMAGE\_WIDTH])
91. batch\_y = np.zeros([batch\_size, MAX\_CAPTCHA\*CHAR\_SET\_LEN])
93. # 有时生成图像大小不是(60, 160, 3)
94. **def** wrap\_gen\_captcha\_text\_and\_image():
95. **while** True:
96. text, image = gen\_captcha\_text\_and\_image()
97. **if** image.shape == (60, 160, 3):
98. **return** text, image
100. **for** i **in** range(batch\_size):
101. text, image = wrap\_gen\_captcha\_text\_and\_image()
102. image = convert2gray(image)
104. batch\_x[i,:] = image.flatten() / 255 # (image.flatten()-128)/128  mean为0
105. batch\_y[i,:] = text2vec(text)
107. **return** batch\_x, batch\_y
109. ####################################################################
111. X = tf.placeholder(tf.float32, [None, IMAGE\_HEIGHT\*IMAGE\_WIDTH])
112. Y = tf.placeholder(tf.float32, [None, MAX\_CAPTCHA\*CHAR\_SET\_LEN])
113. keep\_prob = tf.placeholder(tf.float32) # dropout
115. # 定义CNN
116. **def** crack\_captcha\_cnn(w\_alpha=0.01, b\_alpha=0.1):
117. x = tf.reshape(X, shape=[-1, IMAGE\_HEIGHT, IMAGE\_WIDTH, 1])
119. #w\_c1\_alpha = np.sqrt(2.0/(IMAGE\_HEIGHT\*IMAGE\_WIDTH)) #
120. #w\_c2\_alpha = np.sqrt(2.0/(3\*3\*32))
121. #w\_c3\_alpha = np.sqrt(2.0/(3\*3\*64))
122. #w\_d1\_alpha = np.sqrt(2.0/(8\*32\*64))
123. #out\_alpha = np.sqrt(2.0/1024)
125. # 3 conv layer
126. w\_c1 = tf.Variable(w\_alpha\*tf.random\_normal([3, 3, 1, 32]))
127. b\_c1 = tf.Variable(b\_alpha\*tf.random\_normal([32]))
128. conv1 = tf.nn.relu(tf.nn.bias\_add(tf.nn.conv2d(x, w\_c1, strides=[1, 1, 1, 1], padding='SAME'), b\_c1))
129. conv1 = tf.nn.max\_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
130. conv1 = tf.nn.dropout(conv1, keep\_prob)
132. w\_c2 = tf.Variable(w\_alpha\*tf.random\_normal([3, 3, 32, 64]))
133. b\_c2 = tf.Variable(b\_alpha\*tf.random\_normal([64]))
134. conv2 = tf.nn.relu(tf.nn.bias\_add(tf.nn.conv2d(conv1, w\_c2, strides=[1, 1, 1, 1], padding='SAME'), b\_c2))
135. conv2 = tf.nn.max\_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
136. conv2 = tf.nn.dropout(conv2, keep\_prob)
138. w\_c3 = tf.Variable(w\_alpha\*tf.random\_normal([3, 3, 64, 64]))
139. b\_c3 = tf.Variable(b\_alpha\*tf.random\_normal([64]))
140. conv3 = tf.nn.relu(tf.nn.bias\_add(tf.nn.conv2d(conv2, w\_c3, strides=[1, 1, 1, 1], padding='SAME'), b\_c3))
141. conv3 = tf.nn.max\_pool(conv3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
142. conv3 = tf.nn.dropout(conv3, keep\_prob)
144. # Fully connected layer
145. w\_d = tf.Variable(w\_alpha\*tf.random\_normal([8\*32\*40, 1024]))
146. b\_d = tf.Variable(b\_alpha\*tf.random\_normal([1024]))
147. dense = tf.reshape(conv3, [-1, w\_d.get\_shape().as\_list()[0]])
148. dense = tf.nn.relu(tf.add(tf.matmul(dense, w\_d), b\_d))
149. dense = tf.nn.dropout(dense, keep\_prob)
151. w\_out = tf.Variable(w\_alpha\*tf.random\_normal([1024, MAX\_CAPTCHA\*CHAR\_SET\_LEN]))
152. b\_out = tf.Variable(b\_alpha\*tf.random\_normal([MAX\_CAPTCHA\*CHAR\_SET\_LEN]))
153. out = tf.add(tf.matmul(dense, w\_out), b\_out)
154. #out = tf.nn.softmax(out)
155. **return** out
157. # 训练
158. **def** train\_crack\_captcha\_cnn():
159. output = crack\_captcha\_cnn()
160. # loss
161. #loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(output, Y))
162. loss = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(output, Y))
163. # 最后一层用来分类的softmax和sigmoid有什么不同？
164. # optimizer 为了加快训练 learning\_rate应该开始大，然后慢慢衰
165. optimizer = tf.train.AdamOptimizer(learning\_rate=0.001).minimize(loss)
167. predict = tf.reshape(output, [-1, MAX\_CAPTCHA, CHAR\_SET\_LEN])
168. max\_idx\_p = tf.argmax(predict, 2)
169. max\_idx\_l = tf.argmax(tf.reshape(Y, [-1, MAX\_CAPTCHA, CHAR\_SET\_LEN]), 2)
170. correct\_pred = tf.equal(max\_idx\_p, max\_idx\_l)
171. accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))
173. saver = tf.train.Saver()
174. with tf.Session() as sess:
175. sess.run(tf.global\_variables\_initializer())
177. step = 0
178. **while** True:
179. batch\_x, batch\_y = get\_next\_batch(64)
180. \_, loss\_ = sess.run([optimizer, loss], feed\_dict={X: batch\_x, Y: batch\_y, keep\_prob: 0.75})
181. **print**(step, loss\_)
183. # 每100 step计算一次准确率
184. **if** step % 100 == 0:
185. batch\_x\_test, batch\_y\_test = get\_next\_batch(100)
186. acc = sess.run(accuracy, feed\_dict={X: batch\_x\_test, Y: batch\_y\_test, keep\_prob: 1.})
187. **print**(step, acc)
188. # 如果准确率大于50%,保存模型,完成训练
189. **if** acc > 0.5:
190. saver.save(sess, "crack\_capcha.model", global\_step=step)
191. **break**
193. step += 1
195. train\_crack\_captcha\_cnn()

CNN需要大量的样本进行训练，由于时间和资源有限，**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "_blank)**时我只使用数字做为验证码字符集。如果使用数字+大小写字母CNN网络有4\*62个输出，只使用数字CNN网络有4\*10个输出。

*TensorBoard是个好东西，既能用来调试也能帮助理解Graph。*

训练完成时的准确率（超过50%我就不训练了）：

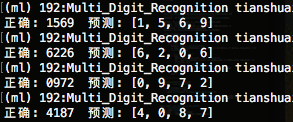
TensorFlow练习20: 使用深度学习破解字符验证码

使用训练的模型识别验证码：

**[python]** [view plain](http://blog.csdn.net/u014365862/article/details/53869816) [copy](http://blog.csdn.net/u014365862/article/details/53869816)

[在CODE上查看代码片](https://code.csdn.net/snippets/2075286)

1. **def** crack\_captcha(captcha\_image):
2. output = crack\_captcha\_cnn()
4. saver = tf.train.Saver()
5. with tf.Session() as sess:
6. saver.restore(sess, tf.train.latest\_checkpoint('.'))
8. predict = tf.argmax(tf.reshape(output, [-1, MAX\_CAPTCHA, CHAR\_SET\_LEN]), 2)
9. text\_list = sess.run(predict, feed\_dict={X: [captcha\_image], keep\_prob: 1})
10. text = text\_list[0].tolist()
11. **return** text
13. text, image = gen\_captcha\_text\_and\_image()
14. image = convert2gray(image)
15. image = image.flatten() / 255
16. predict\_text = crack\_captcha(image)
17. **print**("正确: {}  预测: {}".format(text, predict\_text))



enumerate 函数用于遍历序列中的元素以及它们的下标：

>>> for i,j in enumerate(('a','b','c')):  
 print i,j

0 a  
1 b  
2 c  
>>> for i,j in enumerate([1,2,3]):  
 print i,j

0 1  
1 2  
2 3  
>>> for i,j in enumerate({'a':1,'b':2}):  
 print i,j

0 a  
1 b

>>> for i,j in enumerate('abc'):  
 print i,j

0 a  
1 b  
2 c

ord()函数是chr()函数（对于8位的ASCII字符串）或unichr()函数（对于Unicode对象）的配对函数，它以一个字符（长度为1的字符串）作为参数，返回对应的ASCII数值，或者Unicode数值，如果所给的Unicode字符超出了你的Python定义范围，则会引发一个TypeError的异常。

>>> chr(65)

'A'

>>> ord('a')

97

>>> unichr(12345)

u'\u3039'

>>> chr(12345)