

数算实习大作业

验证码识别 实验报告

陈志翰，胡家琛，李尚敏，周尚彦

December 21, 2017

Contents

1	算法简介	1
1.1	主要流程	1
1.2	验证码生成	2
1.3	降噪与切割处理	4
1.4	神经网络	6
2	算法与模型性能分析	7
2.1	代码与运行环境	7
2.2	验证码生成与切割算法	7
2.3	神经网络模型	8

1 算法简介

1.1 主要流程

1. 实现一个简单的验证码生成算法，并利用该算法生成训练数据集与测试数据集
2. 将训练集中验证码进行降噪处理
3. 将降噪后的验证码切割为只含有单个字符的统一大小的图片
4. 使用神经网络对训练数据集进行训练
5. 对测试数据集使用相同的算法进行降噪与分割
6. 使用处理后的测试数据集验证模型的准确度

1.2 验证码生成

算法目的 每次以大小写字母与数字作为字符集，生成包含 4 个字符的验证码图片。

利用该算法生成的验证码具有以下特点：

1. 字符相较标准字体有一定角度的偏移与形变
2. 加入了噪点与干扰线以干扰识别
3. 对图片进行了反走样处理与平滑处理，加大了识别难度

算法过程

1. 随机生成一个四字字符串
2. 调用 `PIL.ImageDraw` 的 `text` 方法将 `DroidSansMono` 字体中的字以随机大小打在图片上
3. 生成噪点
 - (a) 随机 n （设置为 30）个坐标
 - (b) 对于每个坐标将其与其左上角坐标以画一条宽度为 $width$ （设置为 3）的连线
4. 随机两个坐标并将其连线作为干扰线
5. 扭曲处理：
 - (a) 随机一个偏移角度，计算出新图片的四个顶点的位置
 - (b) 对四个顶点加入随机的偏移量
 - (c) 调用 `PIL.Image` 的 `QUAD` 方法生成扭曲后的图片
6. 调用 `PIL.ImageFilter` 将图片平滑处理

算法示例 利用该算法生成的验证码示例如下：



Figure 1: 验证码示例



Figure 2: 验证码示例

1.3 降噪与切割处理

算法目的 每次调用是读入给定验证码图片的 `rgba` 矩阵，并将其二值化（被视为背景与噪音的像素点为 0，被视为字符的像素点为 1）。

之后将其切割成仅包含单个字符的统一大小的 01 矩阵作为返回值。

该算法的难点在于

1. 各个字符大小不相同，分割的宽度也不相同
2. 反走样处理使得字符上 `rgb` 值差异较大
3. 平滑处理缩小了背景、噪音、字符的 `rgb` 值的差异

算法流程

1. 读入验证码图片的 `rgb` 矩阵
2. 将 `rgb` 分块处理，即将 `rgb` 值全除以 8
3. 首先在矩阵中找出出现最多的颜色块，作为标准背景颜色块
4. 噪音和平滑处理会使得背景颜色驳杂，故将出现的行数次数之和大于 $width * 3/5$ 的颜色块则设为背景颜色块
5. 在所有不为背景颜色中求出出现次数最多的颜色块，作为标准字符颜色块
6. 寻找与标准颜色块相近的像素点，也视作字符颜色
7. 寻找切割字符像素点较少的垂直线段作为切割线（加入距离筛选），切割图片

算法示例 (降噪效果仅体现在二值化矩阵上，在原图中并未体现)



Figure 3: 字符切割示例



Figure 4: 字符切割示例



Figure 5: 字符切割示例



Figure 6: 字符切割示例

1.4 神经网络

主要思路 采用卷积神经网络进行图像识别。

数据预处理 如上文所述，在降噪步骤中先将图像做 0/1 normalization（二值化），将字符设置为 1，噪音和背景设置为 0（这样的 normalization 是很有必要且很有效果的），得到 30×30 的 01 矩阵。

网络结构 网络结构为：

- two convolutional layers
- two fully connected layers.

两个 conv layer 分别拥有 32 和 64 个 kernel，其中 receptive field 分别为 5×5 和 3×3 ，使用 ReLU 作为 activation。两个 conv layer 之后跟着一个 max pooling layer，其中 stride 是 2，filter size 是 3×3 ，注意到这里我们使用了 filter 之间 overlap 的策略。

为了防止 overfitting，同时因为模型容量不需要太大，fully connected layer 的数目比较少。两个 fully connected layer 均使用 ReLU 作为 activation。Loss 函数使用 prediction 和 label 之间（两个 distribution 之间）的 cross entropy，即：

$$H(L, p) = - \sum_x L(x) \log p(x)$$

其中 p 由输出层（36 个 neuron，分别表示 0-9 和 a-z）做 softmax 得到。softmax 函数为：

$$p(x) = \frac{e^x}{\sum_x e^x}$$

训练方式 使用 SGD 算法进行训练，每个 batch 由 100 张验证码图片进行分割、降噪之后的数组组成在训练过程中忽略分割失败的数据。

改进

1. 经过实验后发现该模型仍然有一点 overfitting，所以在两个 fully connected layer 之间加入了一个 dropout 随机丢弃一些 neurons，dropout rate 设为 0.5。
2. 在改进 1 的基础上，发现 generalization err 仍然有一些大，于是增强了数据以增强网络的泛化性能。

2 算法与模型性能分析

2.1 代码与运行环境

- 硬件配置
 - CPU: AMD Ryzen R5 1600X
 - GPU: Nvidia GeForce GTX 1050TI
 - Memory: DDR4 16G 2400MHZ
- 操作系统: Windows 10 (X64)
- Python 版本: Anaconda 提供的 Python3.5.3
- Tensorflow 相关
 - Tensorflow 1.3
 - CUDA Toolkit 8.0
 - cuDNN v6.1

2.2 验证码生成与切割算法

- 验证码生成与切割一张图片的时间大约为 0.3 秒
- 在没有噪点的情况下, 切割算法的成功率可以达到 98% 以上
- 在加入噪点的情况下, 切割算法的成功率约为 80%

2.3 神经网络模型

无噪点数据集 使用上文所述算法去掉生成噪点和干扰线的步骤, 生成无噪点数据集。

以 30 个 batch 的数据作为训练数据, 在 40000steps (约十分钟) 之后, 对于训练数据集的识别准确率达到 99.92%。

使用另外 30 个 batch 的数据作为测试数据, 对测试数据集的识别准确率约为 92%。

一般数据集 使用上文所述算法生成一般数据集。

将 150 个 batch 的数据分为三部分, 并进行交叉训练。在 280000steps (约三小时) 之后, 对于三组训练数据的识别准确率均在 91% 左右。

使用另外 60 个 batch 的数据作为测试数据, 对测试数据集的识别准确率约为 80%。