机器学习工程师纳米学位毕业项目
基于深度学习的算式字符识别

徐勇皇

2018 年 12 月 1 日

目录

1 定义

1.1 项目概述

项目将训练一个从自然图像中识别算式图像及其字符符号的模型，模型可以应用到手机 APP ，网页应用中，实时显示出数字序列。

在现实生活的场景中，有许多使用到字符场景的地方。比如店铺的店名，品牌标志的文字 logo，以及书本的印刷等等。因为文字符号作为语意表达的载体而出现，所以这些都渗透到人们生活的方方面面。因此自然场景中的文本检测识别才会显得尤其重要，即 OCR (Optical Character Recognition，光绪字符识别)，传统上指对输入扫描文档图像进行分析处理，识别出图像中文字信息。场景文字识别 (Scence Text Recognition, STR) 指识别自然场景图片中的文字信息。自然场景图像中的文字识别，其难度远大于扫描文档图像中的文字识别，因为它的文字展现形式及其丰富：

1. 允许多种语言文本混合，字符可以有不同的大小、字体、颜色、对比度等。
2. 文本行可能有横向、竖向、弯曲、旋转、扭曲等样式。
3. 图像中的文字区域还可能会产生变形（透视、仿射变换）、残缺、模糊等现象。
4. 自然场景图像的背景及其多样。如文字可以出现在平面、曲面或这周面上；文字区域附近有复杂的干扰纹理、或者非文字区域有近似文字的纹理，比如沙地、草丛、栅栏、砖墙等。

也有人使用 OCR 技术泛指所有图像文字检测和识别技术，包括传统 OCR 技术与场景文字识别技术。这是因为，场景文字识别技术可以被看成是传统 OCR 技术的自然演进与升级换代。

图像文字检测和识别技术有着广泛的应用场景。已经被互联网公司落地的相关应用涉及了识别名片、识别菜单、识别快递单、识别身份证、识别营业证、识别车牌、路牌等等。

已经有不少服务商在提供图像文字检测和识别服务，这些服务商包括了腾讯、百度、阿里、微软、谷歌等大型云服务企业，也包括了一些活跃在物流、教育、安防、视频直播等垂直细分行业的服务企业。这些企业既可以使用提前训练好的模型直接提供场景图文识别、卡证识别、扫描文档识别等云

服务，也可以使用客户提供的数据集训练定制化模型，以及提供定制化 AI 服务系统集成等。

1.2 问题陈述

项目选择的数据集包含有 100000 张算式的图片，图片上是类似于验证码的算式字符，数据集给出了相应的算式字符串，需要解决的问题是读入一张宽为 300 像素，高为 64 像素的图片，并识别图片中的算式，其中算式仅包含 +，-，* 和 =，这四个运算符，还可能包含一对括号，最长的长度为 11 个字符。需要依据读取的图片，识别出图片中的算式字符串，给出准确的预测，而只用当识别预测的字符串与原有标签完全一致，才认为是一致的。

1.3 评价指标

对于模型的预测，认为只有预测的标签算式字符串，与数据集给定的标签算式字符串是完全相同时，才认为预测是准确的。这是因为算式本身是一个整体，如果是其中仅仅有个别字符预测出现偏差，仍认为是错误的，因为错误的预测并不会有任何残留的利用价值。基于以上原因依此准确率来评判模型最终的准确率。

2 分析

2.1 数据可视化

给定数据集是一个验证码样式的算式图，尽管数据集中已经包含 10 万张图片，但仍认为对于所有 11 个字符内的算式的可能性是远远不够的，基于以上原因，我们利用验证码生成工具，自己生成类似地训练数据集，而给定的数据集仅仅作为验证集来验证模型的准确率。这样训练得到的模型将会有很好的泛化性，因为所有的训练数据都是随机生成而不是给定的。下图显示了随机生成的图片和数据集中的图片，左边一列为数据集中的图片，右边一列为随机生成的图片。

而对于原始数据作分析，可得到接下来的可视化结果。首先对于不同长度算式的分布如下图 2所示，其中长度为 9,10 的长度较多占了所有算是的，而长度为 9,8 和 11 的算式较少，下面对于各个字符给出算式中的个数统计如下，其中-，= 号最多，其次是 +，*，（和）都有超过 6000 个，而 0 到 9，

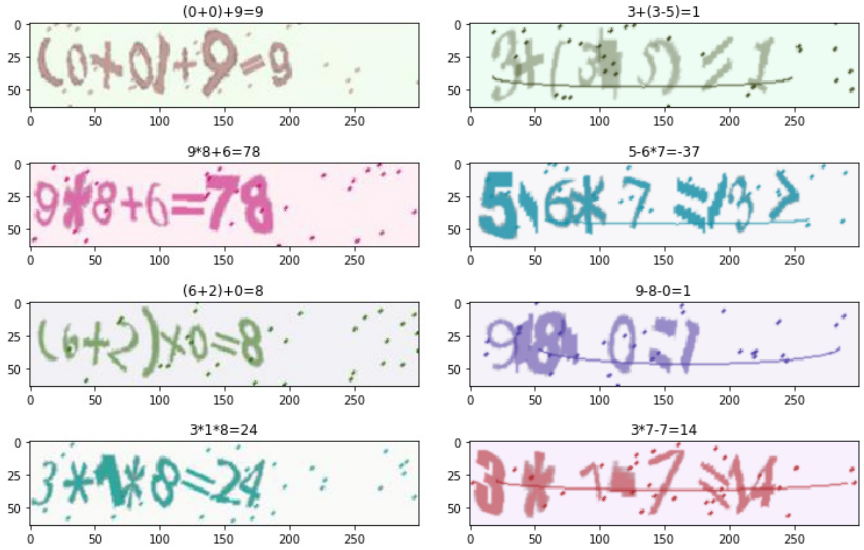

图 1: 数据集中图片以及随机生成验证码图片

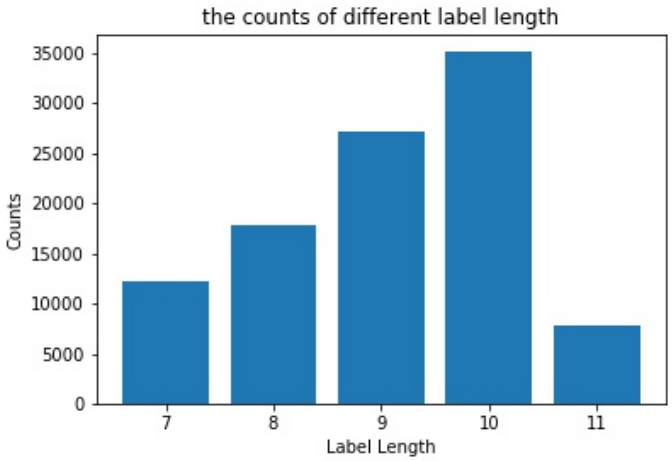

图 2: 算式长度的分布

十个数字中 1 相较于其他各个数字要更多为 6600 多，而其余数字则大都在 4000 左右。对于等式中是否含有括号，也作业了相应的统计，结果如下，其中包含括号为 7000 多，而不包含括号的算式个数仅 3000 个。

从以上分析中可以看出一些数据集中算式标签的统计特征，这将可以有

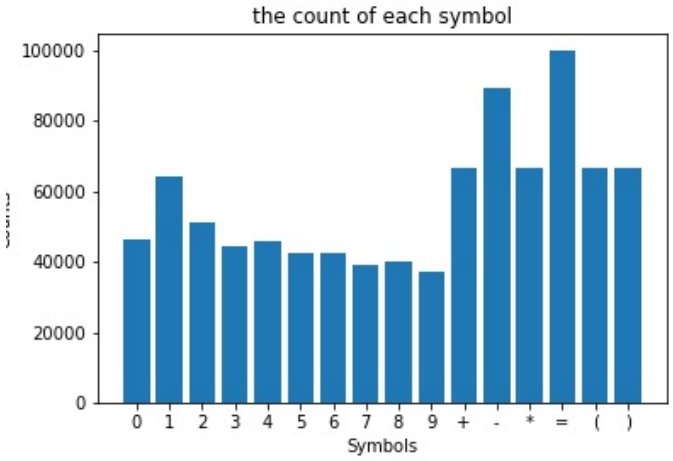

图 3: 各个算式字符数统计

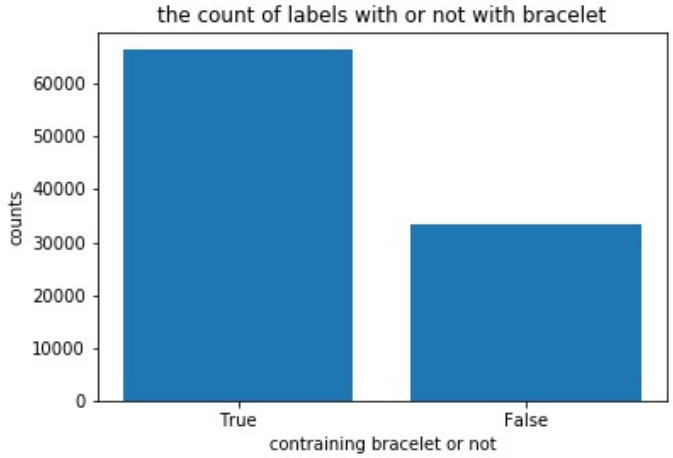

图 4: 算是中是否包含有括号的个数统计

利于我们更好的理解数据并且在随机生成算式图片的时候能得到更为近似的图片，从而提高模型训练的准确性。

2.2 算法和技术

对于模型的构建主要分为两部分，第一部分是特征提取，另一部分是模型的训练。其中特征提取主要根据文章 [5] 的模型结构，先用 CNN 网络提取特征，再用双向 GRU 的 RNN 网络递归特征从而得到整体的整体，最后根据 CTC Loss 和 SGD 优化器训练模型。经过训练集训练，采取其中模型在验证集上准确率最高的模型权重，作为最终的模型。下面就一些用到的技术作介绍。

2.2.1 RNN 递归循环网络

RNN(Recurrent Neural Network) 是一类用于处理序列数据的神经网络。它不同于以往基础的神经网络只在层与层之间建立了链接，RNN 最大的不同之处就是层之间的神经元之间也建立了链接。如上图所示就是一个

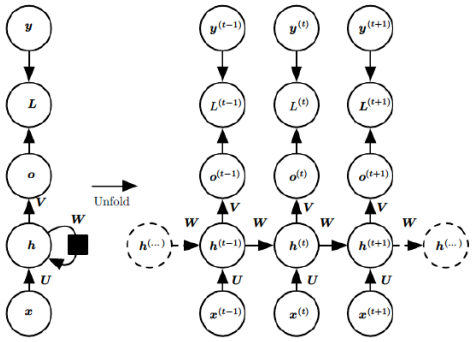

图 5: RNN 连接结构示意图

标准的 RNN 结构图，途中每个箭头代表做一次变换，也就是说箭头链接带有权值，左侧是折叠起来的样子，右侧是展开的样子，左侧中 h 旁边的箭头代表此结构中的“循环”隐层。RNN 的数学表达式如下：

$$\begin{cases} h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t = \sigma_y(W_y h_t + b_y) \end{cases} \tag{1}$$

在展开结构中我们可以观察到，在标准 RNN 接哦股中，隐层的神经元之

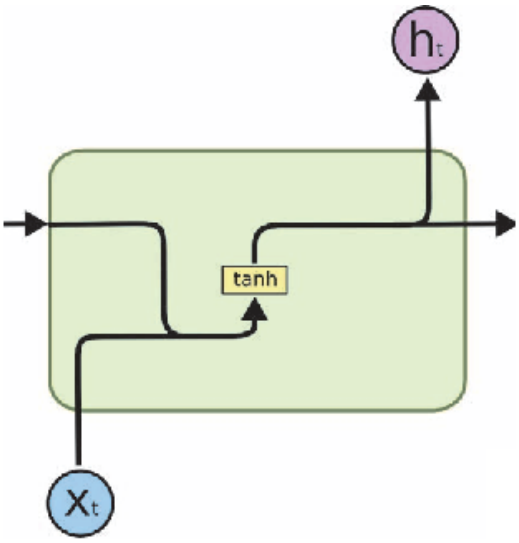

图 6: RNN 单元结构图

间也是带有权值的。也就是说，随着序列的不断推进，前面的隐层将会影响后面的隐层。图中 O 代表输出，y 代表样本给出的确定值，L 代表损失函数，可以看到“损失”也是随着序列的推进而不断累积的。

出上述特点之外，标准 RNN 还具有以下特点：

1. 权值共享，图中的 W 全是相同的，U 和 V 也是一样。
2. 每一个输入值都只与它本身的那条路线建立权连接，不会和别的神经元连接。

2.2.2 RNN 梯度消失的原因

以如下简单的深层网络为例，图中是一个四层的全连接网络，假设每一层网络激活后的输出为 $f_i(x)$，其中 i 为第 i 层，x 代表第 i 层的输入，也就是第 $i-1$ 层的输出，f 是激活函数，那么 $f_{i+1} = f(f_i * w_{i+1} + b_{i+1})$，简单记为 $f_{i+1} = f(f_i * w_{i+1})$。

BP 算法基于梯度下降策略，以目标的负梯度方向对参数进行调整，参数的更新为 $w \leftarrow w + \delta w$，给定学习率 α，得出 $w = -\alpha \frac{\partial Loss}{\partial w}$。如果要更新

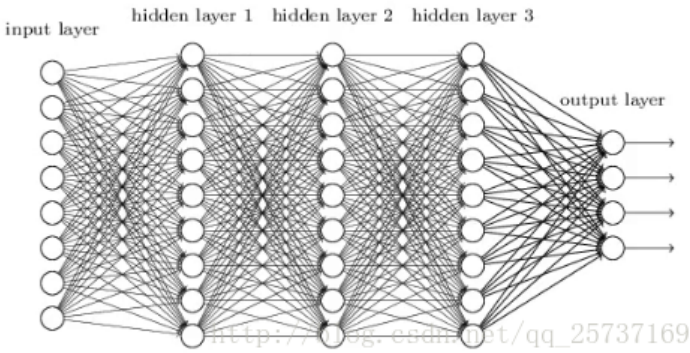

图 7: 5 层简易全连接层神经网络

第二隐藏层的权值信息，根据链式求导法则，更新梯度信息：

$$\begin{aligned}\triangle w_2 &= \frac{\partial Loss}{\partial w_2} \\ &= \frac{\partial Loss}{\partial f_4}\frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial w_2}\end{aligned}$$

很容易看出来 $\frac{\partial f_2}{\partial w_2} = \frac{\partial f}{\partial(f_1 * w_2)} f_1$，即第二隐藏层的输入。所以说，$\frac{\partial f_4}{\partial f_3}$ 就是对激活函数进行求导，如果此部分大于 1，那么层数增多的时候，最终求出的梯度更新将以指数形式增加，即发生梯度爆炸；如果此部分小于 1，那么随着层数增多，求出的梯度更新信息将会以指数形式衰减，即发生梯度消失。正是由于 RNN 是一个深层的神经网络，因而其极易梯度消失。为解决这一问题，下面将着重介绍一下 LSTM 和 GRU 两个 RNN 的变体网络。[1,3]

2.2.3 LSTM，GRU 以及解决梯度消失的原因

LSTM(long short-term memory) 长短期记忆网络是 RNN 的一个变体，RNN 由于梯度消失的原因只能有短期记忆，LSTM 网络通过精妙的门控制将短期记忆与长期记忆想结合，并且在一定程度上解决了梯度消失的问题。

所有 RNN 都具有一种重复神经网络模块的链式的形式。在标准的 RNN 中，这个重复的模块只有非常简单的结构，例如一个 tanh 层。LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单一神经网络层，整体上除了 h 在随时间流动，细胞状态 c 也在随时间流动，细胞状态 c 就代表着长期记忆。

LSTM 的核心思想在于给予了细胞的状态值，细胞状态类似于传送带。

直接在真个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。LSTM 通过精心设计的称作“门”的结构来去除或者增加信息到细胞状态的能力。而 LSTM 的数学表达式如下：

$$\begin{cases} f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f), & \text{（输入门）} \\ i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i), & \text{（遗忘门）} \\ o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o), & \text{（候选记忆单元）} \\ c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c), & \text{（当前记忆单元）} \\ h_t = o_t \circ \sigma_h(c_t), & \text{（输出门）} \end{cases} \tag{2}$$

由上小节知道 RNN 产生梯度消失或者爆炸的原因就在于多次循环的 RNN

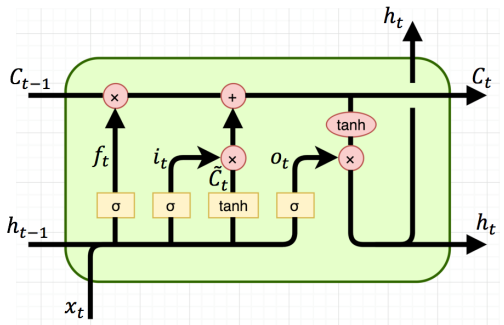

图 8: LSTM 单元结构图

单元，使得链式法则求导收到其中各项连乘的影响。而从 LSTM 的数学表达式中，可以看到三个门的激活函数都是 sigmoid，这也就意味着这三个门的输出要么接近于 0，要么接近于 1。对于 $\frac{\partial c_t}{\partial c_{t-1}} = f_t$，而 f_t 是门的输出，当 $f_t = 1$ 时，梯度能够很好的传导到上一时刻，此时即使层数较深也不会发生梯度消失问题；当 $f_t = 0$ 时，即上一时刻的信号对当前时刻没有影响，梯度也没必要传回去。所以，这就是为什么通过门机制能够解决梯度问题的原因。

同样的，GRU(Gated Recurrent Unit) 可以看作是 LSTM 的变种，GRU 把 LSTM 的遗忘门和输入门用更新门来替代。把当前记忆单元和候选记忆单元进行合并，并提供了不同的更新信息的方法具体的 GRU 的数学表达式

如下：

$$\begin{cases} z_t = \sigma(W_z \cdot [h_{t-1}, x_t]), & \text{（重置门）} \\ r_t = \sigma(W_r \cdot [h_{t-1}, x_t]), & \text{（更新门）} \\ \tilde{h_t} = \tanh(W \cdot [r_t * h_{t-1}, x_t]), & \text{（候选记忆单元）} \\ h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}, & \text{（当前记忆单元）} \end{cases} \tag{3}$$

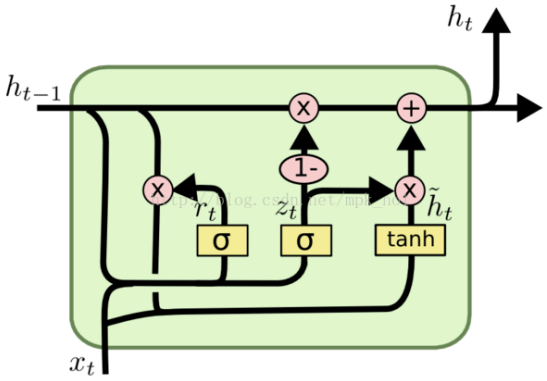

图 9: GRU 单元结构图

2.2.4 CTC Loss 介绍

CTC 的全称是 Connectionist Temporal Classification。[4] 这个方法主要是解决神经网络 label 和 output 不对齐的问题 (alignment problem)。这种问题经常出现在场景文本识别 (Scene Text Recognition)，语音识别 (Speech Recognition)，手写识别 (Hand Writting Recognition) 这样的应用里。比如图10中的语音识别，就会识别出很多 ww，很多个 r，如果不做处理，就会变成 wworrrld（这里忽略了空白 (blank)）。有一种简单粗暴的方法就是把重复的字母都当做一个字母。但是这样就会出现不识别单词 happy 这样的单词。这个时候，空白的作用就非常大了，在这里，他把同一个字母给隔开，比如 happy，就会变成和 hhh aaaa ppp ppp yyy–>happy。后面的推导与计算都在 [2] 中都有提及。

用数学的表达式就是，一个映射 $F : X \to Y$，同时存在多个 $x \in X$ 使得 $F(x) = y \in Y$。

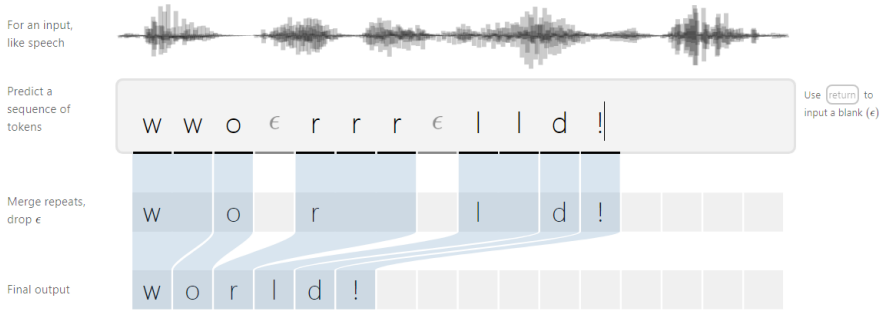

图 10: CTC 工作原理

但是这又会存在另一个问题，就是输入 x 的与输出 y 之间长度的关系，这里仅仅假定输出 y 的长度必须小于输入 x 的长度，一旦长度大于等于时，此时的 $loss$ 记为 ∞。

而在传统的方法 Framewise 中，它将每个音素进行标记，再依据标记的长度给出相应的预测。由图11中可以看出 CTC 给出了不同的处理方法，它将音素出现的峰值分别进行标记，再引入 blank 插入其中，最后依据不同音素组合出现的概率大小给出预测。这就避免了 Framewise 方法可能不能准确识别相邻紧挨的两个音素。

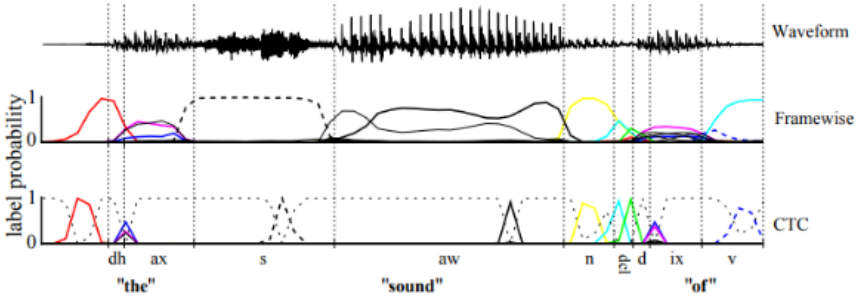

图 11: CTC 与 Framewise 的区别

2.2.5 CTC Loss 的计算

CTC Loss 的计算分为前向和后向。对于前向传播而言，通过一个序列 lt 通常可以有多条路径经过映射后可以得到，而随着序列 l 长度的增加，相

对应的路径的数目是呈指数增加的，因此我们需要一种高效的算法来计算它。

有一种类似于 HMM 的前向传播的算法可以解决这个问题。关键在于把路径 L 所有对应的路径概率都可以通过迭代来计算得到。但是在开始计算时，需要现在序列的开头或是结尾分别加上空格，并且在字母与字母之间也都添加空格。如果原来序列 l 的长度为 C，那么经过处理之后，序列 l' 的长度为 $2C+1$。

对于一个特定的序列 l，可以定义前向变量 $\alpha(t,u)$ 为输出所有长度 t，且经过 F 映射之后为序列 l 的路径的概率之和，用公式表达如下：

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \sum_{i=1}^{t} y_{\pi_i}^i \tag{4}$$

其中 $V(t,u) = \{\pi \in A'^t : F(\pi) = l_{1:u/2}, \pi_t = l'u\}$ 代表所有满足经过 F 映射之后为序列 l，长度为 t 的路径集合，且在第 t 时间步的输出为 label：$l'u$。

所有正确路径的开头必须是空格或者 labell_1，因此存在着初始化的约束条件：

$$\begin{aligned}
&\alpha(1,1) = y_b^1 \\
&\alpha(1,2) = y_{l_1}^1 \\
&\alpha(1,u) = 0, \quad \forall u > 2
\end{aligned}$$

也就是当路径长度为 1 时，它只可能对应到空格或者序列 l 的第一个 label，不可能对应到序列 l 第一个之后的 label 中。

因此，$p(l|x)$ 可以由前向变量来表示，因为

$$p(l|x) = \alpha(T,U') + \alpha(T,U'-1) \tag{5}$$

其中 $\alpha(T,U')$ 可以理解为所有路径长度为 T，经过 F 映射之后为序列 l，且第 T 时刻的输出 label 为 l'_U 或者 l'_{U-1}。也就是路径的最后一个是否包括了空格。

可以由上图12进行理解。上图中，白色的点表示一个 label，黑色的点表示空格，纵向每一列表示的是路径的长度 T，箭头符号代表路径在下一个时刻可以输出都哪个 label 去。如果在时刻 1 的 label 为空格，那么路径在下一时刻只有两个选择，第一个输出空格，第二个就是输出序列 l 中对应的空格的下一个 label，C；如果时刻 2 的 label 为 C，那么在时刻 3，它可以

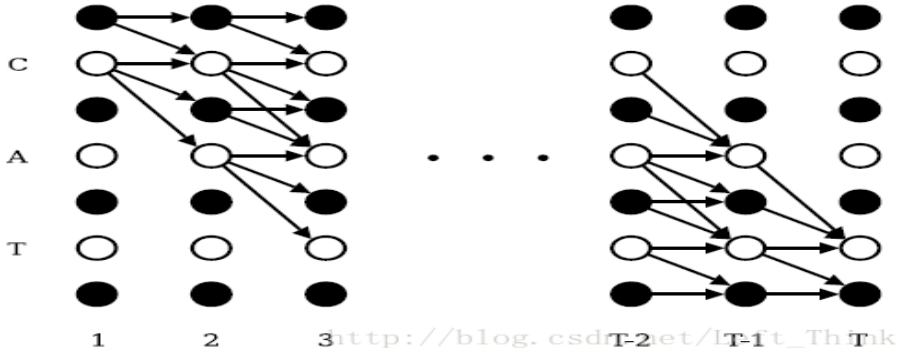

图 12: 前向概率计算图

有三种选择：第一种就是输出还是 C，第二种是输出为空格，第三种是直接输出 A。

从上图可以看出长度为 T 的输出路径映射到序列 l:cat，可以由第 T 步为 label: T 的所有路径和第 T 步为空格的所有概率之和来表示。现在就可以引出递推公式如下，

$$\alpha(t,u) = y_{l'}^{t} \sum_{i=f(u)}^{u} \alpha(t-1,i) \tag{6}$$

其中

$$f(u) = \begin{cases} u-1, & \text{if } l'_u = blank \text{ or } l'_{u-2} = l'_u \\ u-2, & \text{otherwise.} \end{cases} \tag{7}$$

同样的，也可以定义一个反向变量 $\beta(t,u)$ 为从 $t+1$ 时刻开始，在前向变量 $\alpha(t,u)$ 上添加路径 π'，使得最后通过 F 映射之后为序列 l 的概率之和，用公式表示为：

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \sum_{i=1}^{T-t} y_{\pi_i}^{t+i} \tag{8}$$

其中 $W(t,u) = \{\pi \in A'^{T-t} : F(\pi' + \pi) = l, \forall \pi' \in V(t,u)\}$。以及反向传播的处置化条件：

$$\beta(T,U') = \beta(T,U'-1) = 1$$
$$\beta(T,u') = 0, \quad \forall u' \in U'-1$$

以及递推公式如下

$$\beta(t,u)=\sum_{i=u}^{g(u)}\beta(t+1,i)y_{l'_i}^{t+1} \tag{9}$$

其中

$$g(u)=\begin{cases}u-1, & \text{if } l'_u=\text{blank or } l'_{u-2}=l'_u\\ u-2, & \text{otherwise.}\end{cases} \tag{10}$$

于是最后，就可以给出 CTC 的损失函数定义如下

$$L(S)=-\ln\prod_{(x,z)\in S}p(z|x)=-\sum_{(x,z)\in S}\ln p(z|x) \tag{11}$$

其中 $p(z|x)$ 代表给定输入 x，输出序列 z 的概率，S 为训练集。损失函数可以解释为：给定样本后输出正确 label 的概率的乘积，再取负对数就是损失函数了，其中去负对数的作用就是可以通过极小化损失函数，从而使输出正确 label 的概率得到最大。

下面就把前面定义个前向变量与反向变量用到损失函数中去，令序列 $l=z$，定义一个新的集合 $X(t,u)=\{\pi\in A'^T:F(\pi)=z,\pi_t=Z'_u\}$，$X(t,u)$ 代表了在时刻 t 经过 label：l'_u 的所有路径的集合，这样有之前对前向变量与反向变量的定义，它俩的乘积就可以写成：

$$\alpha(t,u)\beta(t,u)=\sum_{\pi\in X(t,u)}\prod_{t=1}^{T}y_{\pi_t}^t \tag{12}$$

而 $p(\pi|x)=\prod_{t=1}^{T}(y_k^t)$，因此进一步转化可以得到

$$\alpha(t,u)\beta(t,u)=\sum_{\pi\in X(t,u)}p(\pi|x) \tag{13}$$

因此，对于任意时刻 t，给定输入 x，输出序列 z 的概率就可以表示成

$$p(z|x)=\sum_{u=1}^{|z'|}\alpha(t,u)\beta(t,u) \tag{14}$$

也就是在任意一个时刻 t 分开，前向变量与反向变量的乘积为在该时刻经过 label：l'_u 的所有概率之和，然后再遍历了序列 l'_u 的每一个 label，因此就得到了所有输出为序列 l'_u 的概率之和。

于是损失函数就可以进一步转化为

$$L(x,z)=-\ln\sum_{u=1}^{|z'|}\alpha(t,u)\beta(t,u). \tag{15}$$

2.3 基准阈值

对于该项目模型给定训练后模型准确预测全数据集不低于 99%，作为此次项目模型的基准阈值。

3 方法

3.1 数据预处理

对于图片数据，由于图片的宽和高均固定为 300 像素和 64 像素，所有在预处理的过程中并没有对图片的尺寸作改变，而仅仅只是做了灰度处理。但是如果在模型预测的时候，则需要对不相同尺寸的图片进行尺寸的缩放使得为宽和高为 300 像素和 64 像素的图片。

3.2 执行过程

而对于模型的构建，采用 CNN 加 RNN 的处理方式提取特征，CNN 用于提取图片的区域特征，而 RNN 则采用双向的 GRU，将所得的图片特征进行序列化得到语意特征，最终根据 CTC Loss 和 SGD 优化器来训练模型的权重，采取对于训练集的 50 次迭代，并根据验证集得出模型的预测准确率。

首先，先具体讨论 CNN 的部分，先把卷积层，批标准化层和最大值池化层进行组合封装,利用封装好的卷积层,分别建立 16,32,64,128,128,256,256,256 个特征；而卷积大小为 3*3，除了最后两个是 2*2；步长均为 1 具体模型如下图所示，

由于 GRU 的循环递归网络要比 LSTM 训练效果好，所以使用了双向 GRU 构建 RNN 网络，分两个部分进行 RNN 网络的构造，第一部分是双向 GRU 网络的输出值相加；第二部分是将上述结果，再一次输出到双向 GRU 网络，但这一次将两个 GRU 单元的输出进行并排排列组合，得到用于最终连接全连接层，激发并得到相应的预测值。

而 CTC Loss 部分的模型输入和计算图为，这里 y_pred 为上述 RNN 模型的预测输出，the_label 是序列化后的标签序列，而 input_length 则是预测输出的长度，这里需要注意的是这里值是预测输出长度中有用的长度，主要用于动态长度的预测中根据输入的长度与预测输出的长度使用。最后，

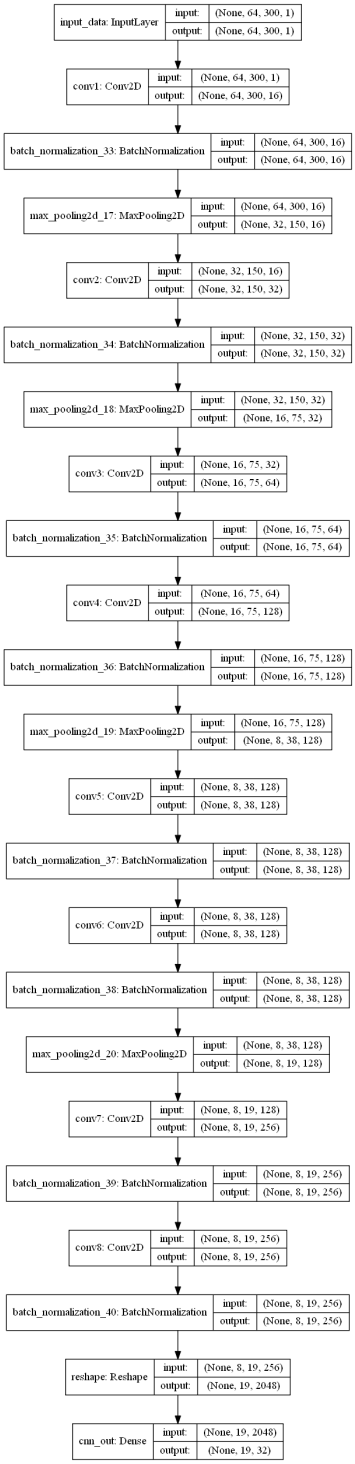
input_data: InputLayer
input: (None, 64, 300, 1)
output: (None, 64, 300, 1)
conv1: Conv2D
input: (None, 64, 300, 1)
output: (None, 64, 300, 16)
batch_normalization_33: BatchNormalization
input: (None, 64, 300, 16)
output: (None, 64, 300, 16)
max_pooling2d_17: MaxPooling2D
input: (None, 64, 300, 16)
output: (None, 32, 150, 16)
conv2: Conv2D
input: (None, 32, 150, 16)
output: (None, 32, 150, 32)
batch_normalization_34: BatchNormalization
input: (None, 32, 150, 32)
output: (None, 32, 150, 32)
max_pooling2d_18: MaxPooling2D
input: (None, 32, 150, 32)
output: (None, 16, 75, 32)
conv3: Conv2D
input: (None, 16, 75, 32)
output: (None, 16, 75, 64)
batch_normalization_35: BatchNormalization
input: (None, 16, 75, 64)
output: (None, 16, 75, 64)
conv4: Conv2D
input: (None, 16, 75, 64)
output: (None, 16, 75, 128)
batch_normalization_36: BatchNormalization
input: (None, 16, 75, 128)
output: (None, 16, 75, 128)
max_pooling2d_19: MaxPooling2D
input: (None, 16, 75, 128)
output: (None, 8, 38, 128)
conv5: Conv2D
input: (None, 8, 38, 128)
output: (None, 8, 38, 128)
batch_normalization_37: BatchNormalization
input: (None, 8, 38, 128)
output: (None, 8, 38, 128)
conv6: Conv2D
input: (None, 8, 38, 128)
output: (None, 8, 38, 128)
batch_normalization_38: BatchNormalization
input: (None, 8, 38, 128)
output: (None, 8, 38, 128)
max_pooling2d_20: MaxPooling2D
input: (None, 8, 38, 128)
output: (None, 8, 19, 128)
conv7: Conv2D
input: (None, 8, 19, 128)
output: (None, 8, 19, 256)
batch_normalization_39: BatchNormalization
input: (None, 8, 19, 256)
output: (None, 8, 19, 256)
conv8: Conv2D
input: (None, 8, 19, 256)
output: (None, 8, 19, 256)
batch_normalization_40: BatchNormalization
input: (None, 8, 19, 256)
output: (None, 8, 19, 256)
reshape: Reshape
input: (None, 8, 19, 256)
output: (None, 19, 2048)
cnn_out: Dense
input: (None, 19, 2048)
output: (None, 19, 32)

图 13: CNN 模型结构

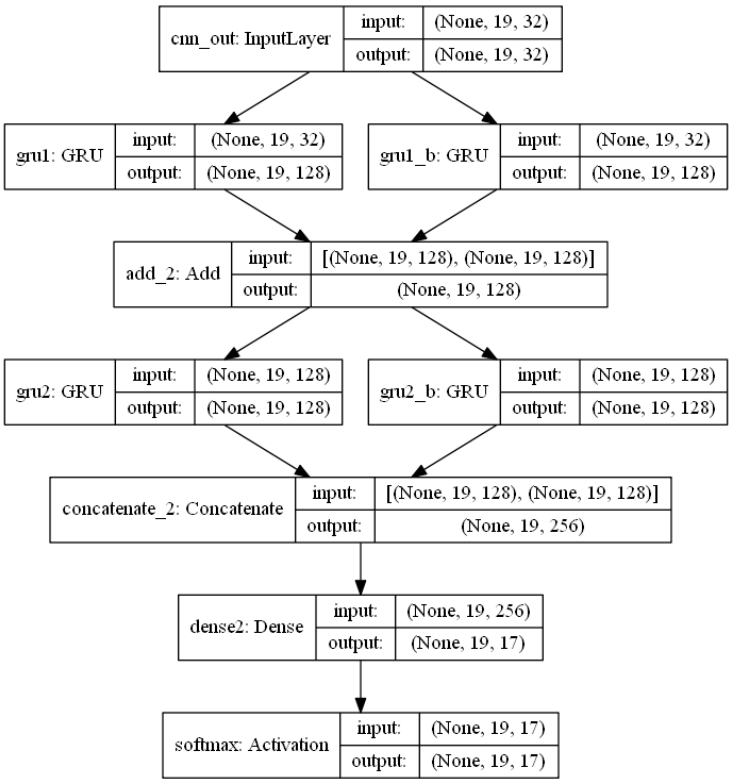
cnn_out: InputLayer
input: (None, 19, 32)
output: (None, 19, 32)
gru1: GRU
input: (None, 19, 32)
output: (None, 19, 128)
gru1_b: GRU
input: (None, 19, 32)
output: (None, 19, 128)
add_2: Add
input: [(None, 19, 128), (None, 19, 128)]
output: (None, 19, 128)
gru2: GRU
input: (None, 19, 128)
output: (None, 19, 128)
gru2_b: GRU
input: (None, 19, 128)
output: (None, 19, 128)
concatenate_2: Concatenate
input: [(None, 19, 128), (None, 19, 128)]
output: (None, 19, 256)
dense2: Dense
input: (None, 19, 256)
output: (None, 19, 17)
softmax: Activation
input: (None, 19, 17)
output: (None, 19, 17)

图 14: RNN 模型结构

label_length 则用于给定标签序列中有用的长度，这主要是因为通常标签长不一定相同，通常需要通过填补而达到相同的长度从而方便处理。

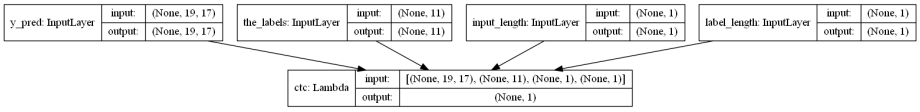

图 15: CTC Loss 计算模型结构

于是这就是最终模型的结构，对应于模型的结构，给出了两种不同的模型训练方式：第一，是通过随机生成的算是图片与其对应的算式标签，共 1e5 万张图片进行训练，再用随机抽样原数据集的 30% 作为验证集由于评估模型的效果；第二是通过每次训练时将数据集分割成 70% 用于训练，另外 30% 用于验证，保证了训练集中不会包含验证集数据，保证了模型的泛化能力。而第一种发法，属于图像训练中的数据增强的方法之一，通过扩样样本数据从而得到更好地训练效果和泛化能力。

而对于训练的具体细节则采用，Adam 作为模型的优化器，并设定初始学习率为 $lr = 1e-3, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-10, \gamma = 0.0$。取 batch_size 为 50，总共训练 400 次 epoch，同时在训练过程中，当 CTC Loss 不再递减时，减少学习率为原来的十分之一，忍耐迭代次数为 10，并采取尽早停机准则在验证集上的 CTC loss 不再递减时，自动停机，忍耐迭代次数为 20 个迭代步。最后在整个训练的过程中仅保存在验证集上效果最好模型的参数作为最终的模型。

4 结果

4.1 模型的评价与验证

从训练的结果上来看，现实采用了第一种训练方法，然而在经过几次迭代之后，训练集上的 CTC Loss 只有 0.012 而验证集上 val_loss 却仍不收敛，仍然有 50 多的 CTC loss 与未训练模型时预测的 CTC Loss 一致，这使得有理由相信，随机生成的算式图片与原有数据集上的图片有较大的误差。而这些误差可能存在与生成随机算式图片的字体，大小，以及生成字符的区域都有关系。由于第一种方法的训练效果不如预期，验证集上的 CTC Loss 并没有明显下降，根本达不到数据增强的目的。只有转而选择第二种训练方

式，而实际的训练效果可从下图看出。

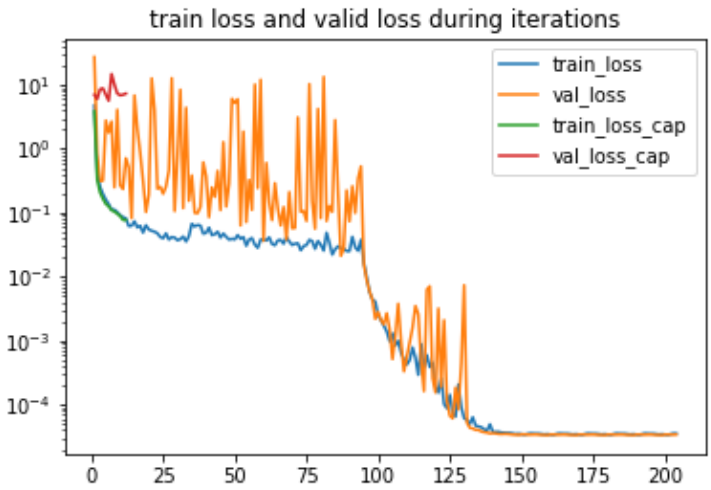

图 16: 模型训练 Loss 收敛情况

在整个训练中先是采用了尽早停止机制，在 80 步左右的迭代自动停止训练，但此时验证集上的 loss 仍比较震荡而没有收敛，于是继续训练，由于学习率一下子变回 0.001 收敛的速度一下子变快而后最终，验证集上的 loss 经过一段震荡后和训练集一起收敛到同一水平，至此完成了整个模型的训练。

接下来，在全数据集上进行预测得到 100% 的预测成功率，这个结果是十分好的，并且达到了设定的 99% 基准阈值。

4.2 合理性分析

在模型设计上，选用了 [5] 一文中的 CNN+RNN 的神经网络模型，从而建立的一个从端到端的模型，利用已经成熟的模型架构使得模型更加的合理。除此之外，相较于原文的模型，加入了 CTC Loss 作为模型的训练依据，而 CTC Loss 也在语意识别中越发越得到广泛的使用，也提高了模型训练的结果。事实上我们的模型也得到了想当不错的预测结果，预测的正确率为 100%。

5 项目分析

5.1 结果可视化

下面就随机在数据集中进行抽取 8 张图片进行预测，结果如下。

表 1: 图片的算式标签和预测结果

图片文件名	算式标签	预测标签
11966.jpg	9-4*9=-27	9-4*9=-27
34212.jpg	(0*2)+1=1	(0*2)+1=1
64264.jpg	9*7-7=56	9*7-7=56
57120.jpg	1+(1+6)=8	1+(1+6)=8
9126.jpg	5*8-5=35	5*8-5=35
92453.jpg	(0-1)+5=4	(0-1)+5=4
26803.jpg	(8*9)-9=63	(8*9)-9=63
46163.jpg	3+(0*9)=3	3+(0*9)=3

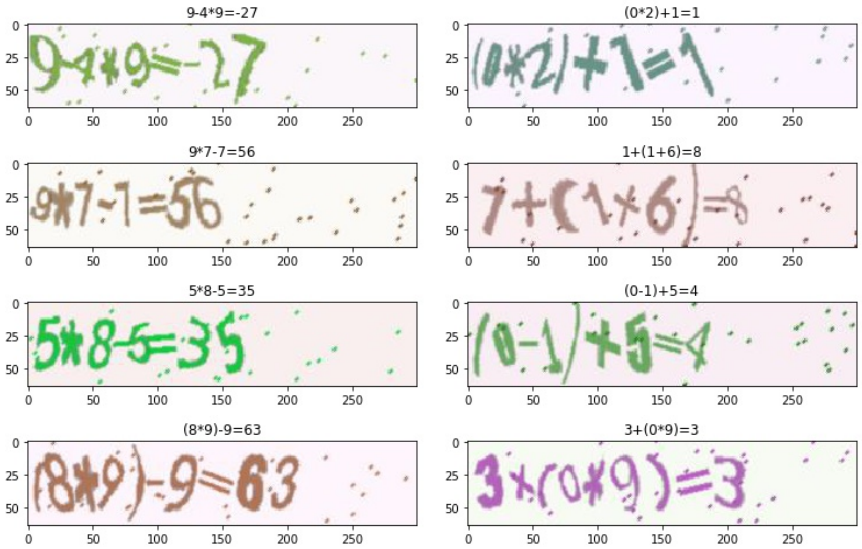

图 17: 模型预测结果

5.2 项目应用

对于该模型，利用 python 的 flask 框架构建了一个简答 OCR 服务的网页，支持提供上传图片，然后根据已经训练好的模型给出预测算式标签，如下图所示。

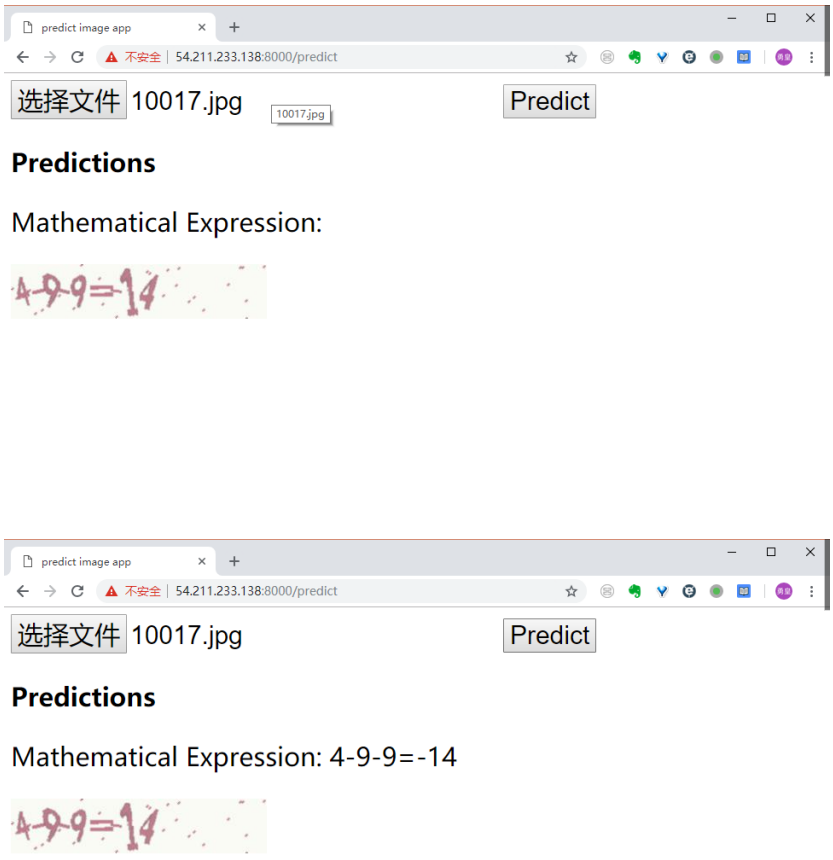

图 18: flask 构建的 OCR 服务网页

5.3 对项目的思考

在这个项目中，主要选择了 keras 作为模型开发的工具，而对用 tensorflow 构建的尝试，也是失败告终，虽然能够使用高级的 API，而对于底层的调用却缺乏解决问题的能力。类似对于 GPU 的利用则交给 keras 自动完成，而对于并行计算则没有任何的设计，使得本身就有一定体量的模型，并不能够更为高效的训练。

5.4 可以作出的改进

在整个项目的过程中，该项目模型对于图像的要求有很大的局限性，这点可以从下图中可见，仅仅是加了边框的图像，预测结果则完全不同，下一步可以在识别模型之前，引入用于检测的模型，将符合识别条件的图像与原图像分离，去除干扰项，用于得到较好的预测结果。

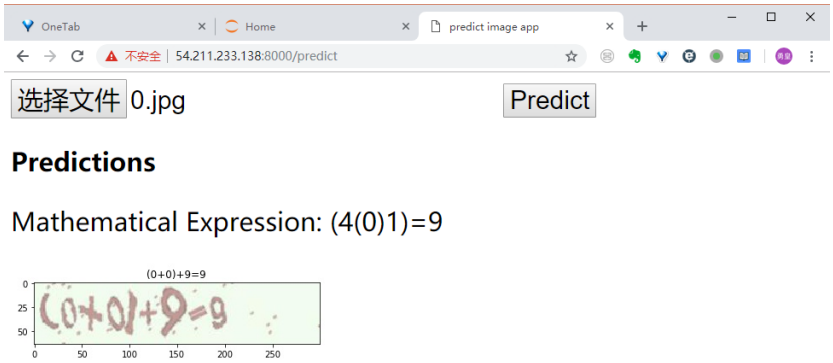

图 19: 存在干扰下的图片算式预测结果

参考文献

[1] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Computer Science*, 2014.

[2] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. 2011.

[3] Alex Graves. *Long Short-Term Memory*. 2012.

[4] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *International Conference on Machine Learning*, pages 369–376, 2006.

[5] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015.