论文图片中定位行内公式位置

张浩然 1500010684
北京大学 数学科学院 信息科学系

Contents

Abstract

1 背景介绍

现在的论文多以 pdf 格式或图片格式进行传播，在使用这些论文时，其中的公式部分往往是我们关心的地方，而快速找到并获取其中的公式就成了一项有意义的工作。我们在这里试图利用神经网络来解决论文图片中公式定位这个问题。

在处理论文图片中定位公式位置这个问题上，我们准备了两个方向上的方法。一是将论文图片切割为段落图片后使用目标检测方法来定位公式的位置，二是在预处理上更进一步将论文图片切割为单词图片，再将单词图片分类。

在目标检测这个问题上有许多的经典算法。基于卷积神经网络的目标检测开始于 2013 年 RBG 的论文 [2] 提出的 RCNN。RCNN 的算法过程大致为生成候选区域后使用 CNN 进行特征提取，将提取的特征通过 SVM 分类，最后通过边框回归 (bounding-box regression) 得到精确的目标区域。此算法的主要问题在于候选区域过多，大量的区域重复和无效造成了巨大的计算浪费。另一个问题在于使用 CNN 需要输入固定尺寸的图片，而图片的截取和拉伸等操作造成了输入信息的丢失。之后有许多算法以此为基础进行了改进。

首先是空间金字塔池化 SPP-Net [3]，在全连接层前加入了一层将输入的特征图池化为特定尺寸的输出的特殊池化层，通过输入的尺寸与需要的输出尺寸计算出所需的池化核和步长从而实现了输出固定尺寸至全连接层。而之前的卷积层并不依赖于输入图片的尺寸，从而实现了任意尺寸的输入。实际上是将原图片多尺度采样输入带 SPP 层的 CNN 进行训练，也是被称为金字塔的原因。

之后 RBG 又提出了新的 Fast-RCNN [1]，借鉴了 SPP 的思路提出了 ROI 池化层，以及将 SVM 分类改为使用 softmax 进行分类，并将分类和边框回归整合，不再独立进行训练。这个算法将除了候选框提取的所有步骤整合在一起进行训练，并引入类似 SPP 的池化层解决了不同尺寸的输入问题，使得训练过程大大提高了。

之后 Faster-RCNN [8] 又更进一步，提出了 RPN 解决了候选框提取的问题。RPN 的特点在于不是在原图上进行候选框提取，而是在特征图上进行。原图通过 CNN 后首先在特征图上进行候选框提取，并将候选框进行分类，只将感兴趣的区域输入到 ROI 池化并进行下一步的分类学习。这样做可以让网络自己学习生成候选区域，大大减少选取候选区域的冗余，提高了预测时间，使得预测可以做到实时。至此候选框选取，CNN，ROI 池化，分类与边框回归都整合到一起训练。

YOLO [7] 则使用了另外一个思路，直接将整个图像进行训练，不预先进行候选框提取。将整个图像分为 $S \times S$ 的网格，物体的中心所在的网格负责该物体的检测，直接经过神经网络得到输出，输出包含物体位置、类别和置信度信息。YOLO 全称为 You Only Look Once，

体现了该算法的简介和迅速。该算法相对于 RCNN 系列的算法拥有检测速度快和背景误检率低等优势，但在准确率和物体位置精度上较差。而且 YOLO 只在一个网格尺度上进行回归，缺乏多尺度信息，容易丢失小目标。

SSD [6] 在 YOLO 之上做了许多改进，采用了多尺度特征图的检测来适应不同大小的物体，最后的输出不是使用全连接层而是用卷积来取得检测结果，同时引入了 Faster R-CNN 中 anchor 的概念，设置不同长宽比和尺寸的先验框。这些改进使得 SSD 同时获得了较高的准确率和速度。

除了使用目标检测的方法，另一方面从单词切割入手，提前获得单词的位置，再将单词图片利用 CNN 分类。我在这个方向上自己写了具体的代码实现，下面对这个方法进行详细的叙述。

2 数据处理

2.1 tex 文件到图片

我们首先从网上获得了大量的论文 tex 文件，通过正则表达式找到其中被 $...$ 框住的公式部分，由于我们的关注点只在于行内公式，故被 $$...$$ 框住的行间公式部分需要排除，找到后在公式外加上可以框住公式的 LaTeX 命令，并分为使用红框和使用白框两个版本。使用白框的是我们进行训练的主要数据，红框版本是获得标记使用的。为了支持我们新增的 LaTeX 命令，仍需要使用正则表达式检测是否含有我们需要的宏包，若没有则在开头加上。接着将处理完毕的 tex 文件编译为 pdf 文件，在编译过程中发现大量的编译失败，主要原因一是使用的 tex 文件较为久远，主要为 2001-2003 年的数据，编译格式和使用的宏包各种各样，缺少相应的宏包支持，二是文件的编码格式不是 utf-8，而编译时统一以 utf-8 为标准，故导致了读取失败。成功编译的 pdf 中也有少量缺失正文，只有公式存在。而且由于为了迅速编译，故所有文件只进行了一次编译，这样所有的引用都不会生效，参考文献的编号都变成了?，认为不影响本工作，所以忽略不需解决。

然后是将 pdf 文件转化为 png 图片。由于预计使用工具 magick 来进行转化，而为了全程使用 python 编程，故使用了 magick 的 python 包 PythonMagick，而此包缺少文档说明，故一开始转化为 png 时遇到了困难，故一开始使用的是 jpg 格式。在查阅了许多解决方法后才终于得到了 png 文件，发现相同尺寸下 png 格式的图片只有 jpg 格式的几分之一，大大减小了硬盘占用，加快了数据传输。

以上方法都写在了文件 texf_topng.py 中。宏包使用了 re, os, pdflatex, PythonMagick, PyPDF2, 并导入了自己写的图片处理工具文件。re 为使用正则表达式的宏包，pdflatex 是将 tex 编译为 pdf 的

宏包，PythonMagick 是将 pdf 转化为 png 的宏包，PyPDF2 是辅助 pdf 分页生成图片的宏包。在生成图片的同时裁去了图片的空白边框并通过生成的图片是否有红框来删去了不带公式的图片。单个 tex 文件处理使用文件 ttp.py，批量文件处理使用 ttpb.py。通过以上方法生成了红框版本和白框版本共计 16 万张图片，每张图片的生成速度在一秒以内，实际生成时使用并行处理。本方法由于还要将论文图片分割为单词，故实际使用的图片数没有这么多。

2.2 单词分割及数据预处理

单词分割主要分为两个部分，文行分割与行内单词分割。以下处理均使用灰度图像。行和是指图像矩阵一行中非空白元素的比例，由于提前做了图像反转，白色为 0，黑色非零，故只需要将一行二值化后求和除以列数，故称为行和。文行的中心行和指文行中间一行的行和，同理，开始行和和结尾行和指文行开始行和结尾行的行和。

在处理之前首先判断图片方向，认为一般只有正向和逆时针 90 度方向。由于灰度图像白色为 255，黑色为 0，故先将图片矩阵反转为白色为 0，黑色为 255，再分别求得图片矩阵的行和和列和。如果图象是正向的，空白边框也已经被截去，故认为行和中 0 的比例应大于列和中 0 的比例，因文行和文行之间有固定的空白，而单词与单词之间的空白位置每行不一。以此作为是否要将图片旋转的依据。此判断只对整页论文有效果，若进行单行或单个单词测试则无效，故设置为可以关闭。

文行分割这个问题上，文行与文行之间有明显的空行，以空行作为文行的边界即可。由于只关心行内公式，故文行分割还有更多的要求，需要在分割时排除行间公式和一些特殊的文行，如一条直线、图表、页码、特殊符号等。故以空行作为边界分割后，又以中心行和，前四分之一行和，开始行和，结尾行和和行高度作为标准来去掉不符合需求的文行。行间公式和一条直线这种情况，一般高度与正常的文行有差别，行间公式大部分高度比较大，一条直线、特殊符号等则是高度比较小，故筛选出高度在平均高度一定范围内的文行。而页码则是中心行和极小，实际上行间公式的中心行和也小于正常文行的行和。同时行间公式的前面通常是一片空白，故同时使用前四分之一中心行和来同时作为辅助标准。为开始行和和结尾行和则是为了去掉图表。

文行分割后，就是对每一文行进行单词分割了，我们使用的都是英文文档，故这里只考虑英文的单词分割。同样预先进行图像反转，使得白色为 0，黑色非零。单词分割与文行分割有相似之处，同样是利用单词与单词之间的空白。但容易注意到一行内的空白有三种情况，单词与单词间的空白、字母与字母间的空白和另外一些比较大的空白，如一行结尾后还有大片空白，每段开始文行的开头空白等。这些空白

的位置使用列和就可以轻易找到，接下来就是在这些空白中筛选出单词间空白。由于最后是利用空白的位置来分割出单词，故认为大片空白和单词间空白是同一性质。这里使用的是最小二乘法，找到使得公式最小的空白宽度，然后认为在这宽度以上的都是用来分割单词的空白。这样做的效果还不错，大多数单词都可以分割出来，少数情况下会出现一个单词被分为两个，而常见的情况是一个长公式被分割为多个单词。

以上单词分割不仅可以分割出单词图片，同时可以获得分割出的文行的位置和每个文行中每个单词的位置，结合去除空白边框时获得的左上角的非空白元素的位置，可以将每个单词的位置精确地还原。

将白框版本的图片进行了单词分割，获得了单词图片及其位置，在通过其位置信息在红框版本的图片中检查该单词是否被红框框住，以此获得每个单词的标注。这样我们将原本的图片整理成了单词图片、单词位置信息和单词标注信息，训练神经网络只需要单词图片和标注信息，故将这部分做成 tfrecords 文件以备后续使用。由于一张论文图片中非公式单词比公式单词要多得多，故这里采用了过采样，将公式图片直接复数拷贝，使得公式图片与非公式图片的数量接近。过采样后实际使用的单词图片为 100 万张左右。

3 网络结构与算法

3.1 激活函数与损失函数

神经网络中有两个重要的部分，一个是激活函数，一个是损失函数。激活函数是实现网络非线性化的重要手段，常用的激活函数有 sigmoid 函数、tanh 函数和 ReLu 函数等。其中 sigmoid 函数和 tanh 函数由于当输入比较大时会有梯度接近 0 的问题，即梯度消失，使得非监督训练的效果较差。ReLu 函数如下：

$$relu(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

ReLu 有计算简单迅速而且不会有梯度消失问题的优势。ReLu 仍有些问题，一是若训练发散，会迅速增大或减少到 nan，使得结果报错。故开始训练前应仔细检查网络的设置，确保能够收敛。二是 ReLu 函数将小于零的值直接变为 0，使得该输出都为正值，故可能会造成某些神经元的失活，不管怎样训练都为 0。同时 ReLu 的输出都为正值，使得收敛比较困难。故针对 ReLu 有许多的改进的函数。Leaky ReLu 函数为在 ReLu 的基础上，在 $x < 0$ 时加上一个较小的斜率。PReLu 则是使得这个斜率作为一个可以训练的参数加入网络中，RReLu 的

做法则是将这个斜率根据均匀分布随机抽取。PReLu 的输出更近接近 0，收敛速度比 ReLu 更快，故我使用的是 PReLu。

分类问题使用的最普遍的的损失函数是交叉熵函数

$$-\sum_{x} p(x) \log q(x)$$

要使用交叉熵函数需要输出和目标都满足概率分布，故交叉熵函数一般结合 softmax 函数

$$softmax(y)_i = y'_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}$$

将输出和目标都归一化。在二分类时也可以使用 sigmoid 函数

$$sigmoid(y) = \frac{1}{1 + e^{-y}}$$

将输出转化为 $[0, 1]$ 之间作为概率使用。在二分类时，softmax 的表达式为

$$softmax(y)_1 = \frac{e^{y_1}}{e^{y_1} + e^{y_2}} = \frac{1}{1 + e^{y_2 - y_1}}$$

故神经网络输出的两个值的差与只输出一个值是等价的，差别在于前者的全连接层会有更多的训练参数。我实际使用的是 sigmoid 函数。

3.2 神经网络优化算法

3.2.1 指数衰减学习率

神经网络的学习率代表神经网络参数的更新速度，较大的学习率使得参数每次更新的幅度较大，收敛得更快，但可能会导致无法收敛到最小，每次更新的时候都跳过了能够收敛到最小值的范围；学习率太小又会导致参数更新太慢，网络收敛速度太小。一般而言，开始时希望学习率比较大，使得网络快速收敛，然后学习率逐渐降低，使得收敛更为准确。故使用了如下阶梯状指数衰减学习率，将学习率乘上一个指数衰减率，使得学习率随着训练次数逐渐降低。实际为每学习 1000 轮将学习率乘以 0.9。

3.2.2 批标准化 batch normalization

神经网络中的数据在经过每层的处理后数据分布可能会发生变化，这一过程被称为 Internal Covariate Shift [4]。这种数据分布的变化传递到后层网络中，后层网络也需要不停地去适应这种分布的变化，

这就导致了网络的收敛速度下降。如果采用的是饱和激活函数，如 sigmoid 或 tanh，数据分布变化会导致数据变大进入梯度饱和。而如果是 ReLu 激活函数，则会有数据分布差异大，深层网络收敛困难的问题 [5]。

由于以上问题，我们使用了 batch normalization 方法，即对每层的数据做规范化。对一层中的一批数据的每个通道 $\{x_i : 0 \leq i \leq n\}$，做如下规范化操作

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ 是为了防止方差为 0。这样规范化后每一层网络的数据分布都变得稳定，但数据的表达能力却缺失了。为了获得原有信息，BN 又引进了两个可以在网络中进行学习的参数 γ 和 β，使得规范化后的数据可以通过变换 $\tilde{x}_i = \gamma\hat{x}_i + \beta$ 恢复表达能力。当 $\gamma^2 = \sigma^2, \beta = \mu$ 时，$\tilde{x}_i = x_i$，即是完全恢复为原来的数据。这样我们既使得每层的分布变得稳定，又能够保证数据的表达能力。规范化是在一个 batch 的每个通道上做，而偏置项对于一个通道而言是相同的，故只需要对卷积输出做规范化，然后加上偏置项就可以了。

BN 使得网络的每层数据的分布变得稳定，后层网络不必去适应输入的变化，实现了每层的独立学习，提高了整个网络的学习速度。在使用 ReLu 激活函数时，由于 ReLu 函数的输出都非负，故输出平均值远离 0，网络不容易收敛，若不使用 BN，使用 MSRA 初始化 30 层以上的网络也难以收敛 [5]。同时 BN 处理后，将降低网络中的参数的敏感度，使得调参，如初始化、学习率等的设置更为容易，不用担心参数的变化随着网络层数加深被放大。使用 BN 后，也不用担心数据经过多层网络后落入饱和性激活函数的梯度包和区，从而可以缓解使用 sigmoid，tanh 等激活函数的梯度消失问题。同时 BN 并没有完全保留原始数据的信息，而是通过学习参数 γ, β 来一定程度上保留数据的表达能力，这就相当于给数据加入了随机噪音，可以起到正则化的效果，防止数据的过拟合。在原作者的结果中，BN 可以在没有 dropout 的情况下同样达到很好的泛化效果，而且网络的收敛速度提高了很多。[4] 我在网络中也同样使用了 BN。

3.2.3 滑动平均

滑动平均，或指数加权平均是一个使得神经网络模型在测试数据

上更加健壮 (robust) 的方法，在计算网络输出时，使用的网络模型不是当时的模型，而与一段时间内的历史模型有关。变量 v_t 在更新为 t 时刻的取值 θ_t 时，使用公式

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

上式中 $\beta \in [0,1)$，β 一般取值较大，即变量在更新时使用了上一时刻的取值，做了某种平均，这样不需要保存一段时间内的每个网络就可以获得某种网络的均值。在网络训练的后期，网络会在在一定范围内波动，使用滑动平均后的网络变量来测试数据就能提高测试结果的表现和稳定性。注意到在网络训练的前期我们希望模型可以更新得更快，故希望衰减率较小，来使得变量快速学习更新。Tensorflow 提供了 tf.train.ExponentialmovingAverage 函数来实现滑动平均，还提供了参数 num_updates 来实现动态设置衰减率的大小，使得网络前期衰减率较小，可以快速学习更新，训练到一定程度后则使用较大的衰减率来实现更好的滑动平均。

3.2.4 过拟合优化

大规模的神经网络有一个重要的难题存在，那就是容易过拟合。神经网络在学习中容易过度学习训练数据的特征，将数据中的噪音也一起学习下来，从而在测试集上的表现效果和训练集上差异较大。

处理过拟合有许多的方法，其中上述的 BN 就能在一定程度上取得效果。另外一个常用的方法是正则化。正则化是在损失函数后加上一个刻画模型复杂度的函数。

$$J(\theta) = J_0(\theta) + \lambda R(w)$$

θ 代表神经网络中所有要学习的参数，w 表示网络权重，$J_0(\theta)$ 代表原损失函数，$R(w)$ 使用中对网络复杂程度的刻画，λ 为模型复杂损失在总损失中的比例。再使用优化算法，如梯度下降时，不是直接优化 $J_0(\theta)$，而是同时优化 $R(w)$，为的是减小模型的复杂程度，使得模型没法刻画全部的数据信息。若网络的权重值较小，即使输入增大一些，输出也不会变化太多。而若权重值较大，输入的较小变化也会被放大。

常用的正则化有 $L1$ 正则化和 $L2$ 正则化。$L1$ 正则化的公式为

$$R(w) = ||w||_1 = \sum_i |w_i|$$

$L2$ 正则化正则化公式为

$$R(w) = ||w||_2^2 = \sum_i |w_i^2|$$

$L1$ 正则化和 $L2$ 正则化都能够缓解过拟合，不同之处在于 $L1$ 正则化会让参数变得稀疏，即许多地方为 0。这是因为 $L1$ 正则化为方形，有许多突出的棱角，这些棱角容易成为优化的结果，而这些棱角上的取值是稀疏的。$L2$ 正则化则是平滑的，不会使得参数变得稀疏，而且 $L2$ 正则化是可导的，优化更简单。实践中常常同时使用 $L1$ 和 $L2$ 正则化。我只使用了 $L2$ 正则化。

除了正则化，另一个常用的防止过拟合的优化方法是 dropout。[9]dropout 形式简单，就是以一定概率使得神经元失活，即输出为 0，dropout 在防止过拟合问题上的效果非常好。为了解决过拟合问题，会想到训练多个模型做组合取平均等，这样就需要大量时间去训练模型。而 dropout 以一定概率使神经元失活，就相当于取了网络的子网络，如果是有 n 个节点的神经网络，每个节点以 50% 的概率失活，则相当于 2^n 个网络的集合，而要训练的参数却是不变的，这样看来 dropout 就取得了很好的抗过拟合的效果。但也可以预见，使用了 dropout 后模型的收敛速度会变慢，需要更长的训练时间。另一种观点认为，dropout 相当于引入了噪声从而增加了样本数量。一般认为卷积层的参数较少，而且是提取数据特征，一般不使用 dropout，故我只在全连接层使用了 dropout。

3.3 网络结构

这个网络要处理的是一个二分类问题，即把公式单词图片和非公式单词图片分类。在分类问题上有许多经典的 CNN 模型，我主要参考了 LeNet、AlexNet 和 VGGNet。

LeNet 是最早用来数字识别的 CNN，是经典的 mnist 数据集分类模型。LeNet 使用了两个卷积层，两个池化层，和两个全连接层。使用的卷积核大小分别为 5×5 和 3×3，输入图片尺寸为 32×32，分为 10 类，最后使用 softmax 交叉熵损失函数。LeNet 使用的激活函数为饱和性激活函数，池化选择的是平均池化。LeNet 在 mnist 数据集上的表现很不错。

AlexNet 比 LeNet 采用了更深的网络结构，使用了 5 个卷积层，3 个池化层和 3 个全连接层。AlexNet 所使用的卷积核尺寸有 $11 \times 11, 5 \times 5, 3 \times 3$，池化核尺寸则都为 3×3，并且池化核步长为 2，使得池化层输出有重叠和覆盖，提高数据特征的丰富性。AlexNet 相比 LeNet 使用了 ReLu 作为激活函数，解决了深层网络梯度弥散问题，验证了 ReLu 的效果，也是从这开始将 ReLu 发扬光大。AlexNet 还在全连接层使用了 dropout 来避免过拟合，实践证实了 dropout 的效果。池化则选择的是最大池化来避免平均池化的模糊化。

VGGNet 则是利用较小尺寸的卷积核和池化核，并不断加深网络来提高网络性能。VGGNet 全部使用了 3×3 的卷积核和 2×2 的池

化核，构筑了 1619 层的深层网络，并随着网络加深不断加大通道数。VGGNet 的网络结构简单，超参数少，几个小尺寸卷积核的连续使用的效果也比一个大尺寸卷积核要好。VGGNet 验证了网络深度对性能的提升，但是网络参数众多，训练速度比较慢。

在参考这些网络模型之后，结合自己的实际情况，测试时间有限，也没有服务器支持，故自行设计了一个网络。网络一共 9 层，三层卷积层，两层池化层，两层全连接层和一层输出层，此外在最后一个卷积层和第一个全连接层之间加入了一层 Spp，前面 Spp-Net 中也提到了 spp 层。spp 层是使用了动态尺寸的池化核将任意尺寸的输入池化到固定尺寸的输出。首先需要一个 $BINS$ 来决定输出的尺寸，通常会选择多个输出尺寸来获得更多的信息。如输入图片尺寸为 $x \times x$，需要的输出尺寸为 $n \times n$，则计算

$$ksize = \lceil \frac{x}{n} \rceil$$

$$stride = \lfloor \frac{x}{n} \rfloor$$

再利用 $ksize$ 和 $stride$ 做最大池化。对 $BINS$ 中每个输出尺寸都做了最大池化后，把这些数据排成一行输出到全连接层。[3] 最开始是为了实现输入不同尺寸的图片到网络中进行训练所以想使用 spp，但由于 tensorflow 的局限，一是如果输入不同尺寸的图片，就没法使用 batch，只能每次输入一个图片；二是 spp 需要使用图片的动态尺寸，生成动态池化核来进行池化，但 tensor 自带的池化函数只支持静态池化核，需要自己重写池化函数，又遇到了使用 tensor 写循环语句的困难。考虑到图片伸缩对本问题的影响不大，故最后改为将输入单词图片都 resize 到 50×50，但仍然保留 spp 层。尽管 spp 层也需要每次输入的图片尺寸相同，但如果输入图片都变为另外一个尺度，网络也不需要改动，可以直接利用原网络。这样就可以实现多尺度维度的输入来提高效果。

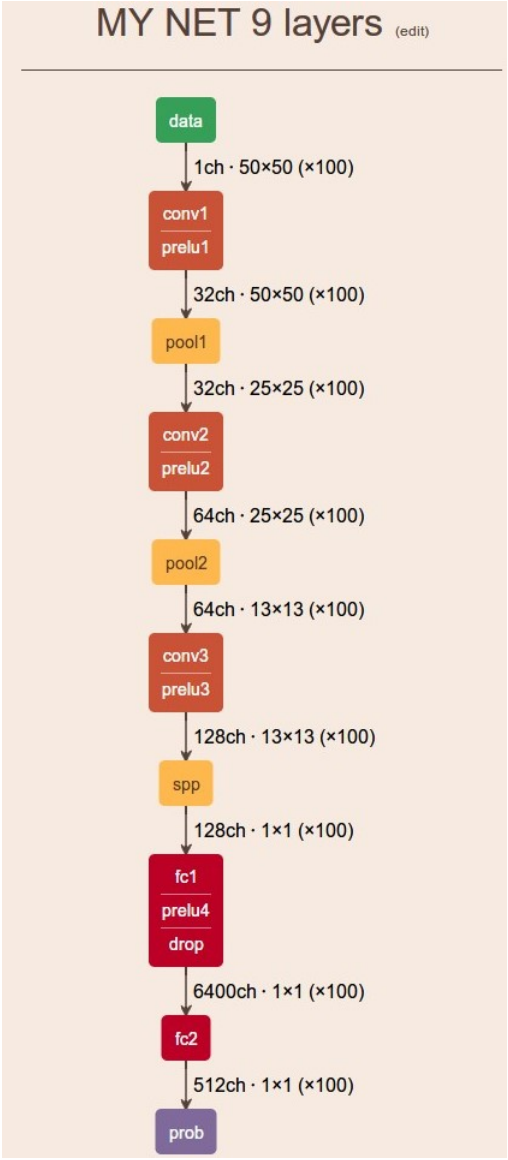

Figure 1: 我的网络结构

4 结果分析与改进

使用了如上网络，以 100 大小的 batch 进行了 50000 次训练。训练集共有含过采样的 100 万张单词图片，测试集为无过采样的 1 万张单词图片。结果在测试集上取得了 93% 的正确率。使用 formula_find.py 进行实际效果查看，错误率也大概在十分之一。错误率较高的在于一些短单词，如 the、and、of 等。对于被分割的公式，在重建时直接将相邻的被预测为公式的单词合并起来，然后用红框标注。

这个结果不尽如人意，对于原因有如下分析。

一是在数据上。数据过于简单，特别是过采样部分，直接使用了复制原图片的方式进行过采样，缺乏变化。而且公式内也有许多字母符号，与某些非公式单词可能难以分别。实际上短单词也容易被误认为是公式，尤其是 and、of。至于单词 the，在测试时发现同一张论文图片中，有的 the 被认为是公式，有的被认为是单词，于是将这些 the 单独提取出来，发现只有图片的高度不同，高度高一点的被认为是公式，高度低一点的被认为是单词，这也是由于数据本身的多样性不足。

二是在参数初始化上，直接使用了截断正态分布来初始化权重，可能会对结果造成影响。

三是网络结构上。这次采用的网络结构主要是根据自身条件进行的舍取，缺乏有效性的验证。

xavier 初始化，MSRA

References

[1] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[2] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[5] Yang Li, Chunxiao Fan, Yong Li, and Qiong Wu. Improving deep neural network with multiple parametric exponential linear units. *CoRR*, abs/1606.00305, 2016.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[7] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.