论文图片中定位行内公式位置

张浩然 1500010684

Contents

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 [?] 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 [?]，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 [?]，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 [?] 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 [?] 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 [?] 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

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

3.2

relu 激活 conv1, pool1, conv2, pool2, spp, fc 阶梯衰减学习率过拟合，正则化滑动平均 spp batch_normalization 批标准化

sigmoid cross entropy loss

[?] [?] [?] [?] [?] [?] [?] [?]

4 结果分析

5 改进