**Chenrui’s Summary of Image Processing Project**

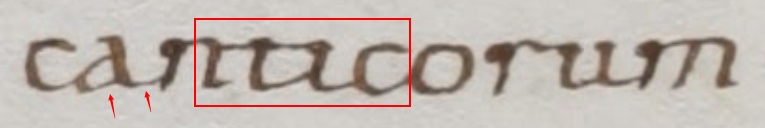


Figure 1. 箭头部分表示字体内部的墨迹粗细和练笔部分的墨迹粗细是一样的；方框内的字符难以辨认：它可以是ntlc,也可以是nuc。

# new

生成具有手写文字或者是特殊字体风格的书本对我来说非常有趣的，不仅因为它具有很多可选择的创造性的实现细节，也因为这个项目可以被其它有需求的人复用——谁都可以上传自己的手稿，然后通过本项目生成对应的个人风格书本。我是朝着这个目标努力的，可惜时间不够，没有在更多数据集上进行尝试（我妹妹有这个需求——将notability上的手写笔记整理到一个文档里）。

我对本项目的理解是这样的：这个项目希望从已有的书页中分离出书页背景以及文字，分别对它们进行处理使其更加多变、充满细节。再利用原本的文字在书页中的相对位置关系，将新生成的文字放入新生成的书页中，从而批量实现输入文字生成对应书本图片的功能。这些生成好的图片可以继续被用来训练下游的神经网络，比如OCR等任务。

任务一是分离出背景图片，对其进行去噪、形变、添加污渍等细节，使其更加真实。能大量提供真实清晰的背景图片是任务一的目标。小组同学问过我如何平衡去掉文字后，颜色如何填充的问题，其实选择文字附近的空白区域颜色来进行填充，效果就会很好。关于色差的问题我也想过，我觉得如果真要考虑到色差，可以先把整张图的背景颜色都平衡一下，然后再假定一个光照方向，增加一个逐渐变化的阴影，这样效果会比较好。

任务二是我和Qinglin一起完成的，细节会在下文进行描述。

任务三是关于如何给文字添加墨迹。其实我觉得先生成文字的灰度图片再给它添加墨迹这个任务划分有些问题——为何不在对文字采样的时候就保持透明背景的、具有墨迹的RGB文字图片呢？我这样想是因为其实每个文字的落笔都是有规律的，比如字母n，左上角开始落笔的时候会重一些，中间轻一点。而任务二中会对很多字体都进行采样，也同时保证了墨迹的多样性。在生成中也可以在对文字进行形变的同时，对墨迹的颜色、深色和浅色的区域大小等等因素进行小范围的随机化，这样做的话效果会很好。

对于任务四生成完整图片，我觉得还存在行对齐和形变上的问题。如果我来做的话，会先将页面尽量展平，然后再记录每行的位置，添加完文字后就可以形变回去，或者变成其他样子。关于如何“尽量展平”，我觉得可以人工标注页面的四个边（因为我们采用了这个方法处理文本，效果非常好）。

以上就是我对此次任务的理解和一些想法。接下来我会接着Qinglin的工作继续讲我的任务部分。

# new

经过Qinglin的预处理工作，我们已经得到了质量非常好的、清晰可用的页面图片。但是关于怎么对其进行采样以得到字母图片字典却是一个需要确定的问题。

我们在本科阶段做过一些中文或者日语的文字分割任务，但这次的拉丁文不一样——其包括了大量的连写字体。我们没有办法像之前那样直接对其进行分割。

我们考虑了是否要使用腐蚀来对连写字体进行处理，使其连笔部分“断掉”，但是失败了——因为连笔部分的墨迹宽度和字母本身的宽度是一样的，练笔部分断掉的时候，字母也会断掉，如图1所示。不能用腐蚀还不仅仅是这个问题——我们并不懂拉丁文，就算能成功将连写字体分隔开，我们也不知道哪些图形对应哪个字母，如图2所示。所以我们必须换一个方法。

在思考的过程中，我想起来我这学期趁着闲暇时间实现了一个游戏的辅助代码（https://github.com/Sosekie/PocketDowntown）。在这个项目中，我利用cv2.matchTemplate和指定的图片实现了自动寻找出租车标识和商品进货标识，从而实现了自动驾驶功能和自动进货功能。“对啊，为什么我不能先人工分离出单个的拉丁文字符，然后对其进行缩放、旋转、变形等变换后用来当作模板进行使用呢？”

事实证明，我的想法是可行的。我设计了以下判断逻辑：如果模板匹配到了相似的图形，并且相似度大于threshold，那么我将匹配到的图形按照模板的大小复制下来，添加到matches数组，并将原来的图形区域置为空（这里是(0,0,0)，即为黑色）。持续这个过程直到这个模板不再能找到任何相似度大于threshold的图形。这样，我们就可以通过置空操作来分割连写的字符。这里有一个小技巧：我们可以先匹配m再匹配n，以及在除了i以外的所有字符都匹配完后再进行i的匹配。这样做是因为m的形状就像是包括了n，而拉丁文的i是没有上方那个点的，所以基本很多字符都包括了i的形状。

而且使用模板匹配还有一个优点：如果我们想要继续增加字符写法的多样性，我们可以继续从页面图片中没有匹配上的字符里选择一些字符作为新的模板，来继续进行匹配查找，直到我们满意为止。在此过程中我们会得到大量的字符，需要对其进行清洗和编号，以便我们在生成的时候供选择。筛选字符部分的任务是由Qinglin完成的。

当然，给定的图片中还缺少一些字符，比如几乎所有大写字符以及数字。我使用了carolus.ttf中的一些字体作为替代，并在句子生成过程中对其进行了缩放、变形等操作，保证其多样性。

接下来是句子生成部分的任务。这部分包括的内容较多，我将其分为以下几个部分：

1. 读取用户输入的句子并查找对应的字符图片；

2. 计算该句子的行高和行宽，以及字母之间、单词之间的间距；

3. 挨个处理字符图片，对其进行形变，以及水平和垂直方向的偏移；

4. 生成背景图片，将处理好的图片写入背景图片中。

在第一步中，我们除了会遇到小写字母、大写字母、数字、标点符号，还会有特殊符号表示的特殊字符（比如christi，prae等等）。遇到这些字符时，不能采用单个读入的方式，而是应该将连续的一批字符保存下来，以进行字符检索。这里，我采用了用"("和")"将特殊字符包裹起来的方式。当读到了"("时，新建空字符串并添加当前字符；之后的循环中，如果没有读到")"，就继续添加当前字符，直到读到了")"。然后我们就可以用这个字符串来进行字符图片的检索。

在第二步中，行的长度是每个字母的长度、字母之间的间隔以及单词之间的间隔的总和。而关于字母之间的间隔是需要仔细考虑的一点——并不是所有字母之间都是固定的间隔，比如在字母f之后的几乎任何字母都会更靠近f。我和Qinglin统计了两两字母之间的大致距离，为一些特殊的字符设置了特殊的水平偏移量。比如，当读到f时，其宽度为f的宽度减去偏移量，而当前位置为：current\_width += image.size[0] + right\_offset + letter\_spaces[letter\_space\_index]，也就是说下一个字母写入时会向左偏移。另外，字母之间的间隔以及单词之间的间隔是在一定范围内随机生成的，我使用数组记录了随机生成的间隔长度，并在生成句子图片时挨个读取并更新当前位置，从而调整在哪里写入字符。

第三步和第四步主要是继续调用了calculate\_bottom\_offset(char)和calculate\_right\_offset(char)来获得字母的垂直和水平偏移量，记录当前位置，并将字母图片写入到纯黑的背景图片上。我在这里给出了行高、偏移量、当前写入位置等变量，方便做任务三的同学得到baseline和word position等信息。

总的来说，这次的项目对我的能力有所提升，特别是针对特殊问题进行特殊处理的能力——很多方案口头说起来非常可行，实际上还是得自己学会灵活变通。另外，我希望自己实现的代码能成为一个可以被复用的功能，给更多人一种实现的思路，而不仅仅是为了应付作业——不能被他人复用的工作对我来说是没有意义的。所以我尽量考虑了一种简单的、易于实现的方式，以方便有需要的人没有阻碍地跑通我们的代码，并实现他们的需求。