基于 openCV 的图片校正和处理

小组成员：王思远（信科 2000013180），王小翔（信科 2000013146）

重要文件说明：

1、server.py 是服务器后端程序，运行此程序，只需使用命令行指令 python ./server.py

2、cam_filter.py 用于给图片施加滤镜

3、baiduocr.py 用于调用百度 ocr 来实现文字识别

4、scan.py 用于识别图片边框并校正图片

5、template 文件夹下保存了两个网页页面的代码

6、static 文件夹下保存着上传的图片和处理好的图片，以及各种 js 和 css 文件

需求分析

在平常的学习生活中，很多时候课程作业的提交方式是需要提交电子版的。像《程序设计思维》这样的编程课，提交电子版的作业很方便，只需要将代码文件等打包成一个压缩包即可。而像《高等数学》这样的非编程课，提交电子版作业，无外乎使用 latex 等软件进行编写，或是手写在纸上然后拍照，提交照片上去。前者会比较美观，但是需要学生额外去学习和掌握 latex 等软件。后者比较容易，但是效果上会不好看，且使得助教批改不方便。因此，我们做出了这个程序，可以自动校正手写文档的图片，以及添加上合适的滤镜让其更加清晰，让作业变得美观，让助教批改作业变得更加轻松。

另外，我们日常学习生活中也有另一个需求：想把某本书上的文字引用到自己的文章中。一种做法是直接对着书一个字一个字地敲到文章中，但是这样费时费力。因此，我们的程序解决了这一困难，我们的程序可以进行文字识别，即拍下书的内容的照片，传进我们的程序里进行文字识别，即可获取书中的文字，此时再将其方便地拷贝到文章中即可。

项目概览

我们小组实现了基于 openCV 的图片校正和处理。用户上传一张或多张图片后，可以进行图片的校正和文字识别，也可以给图片加上多种滤镜。

图 1 一张图片在不同滤镜下的效果

我们的工作分为前后端两部分：①前端负责绘制网页，提供用户上传图片的途径，以及展示图片处理的结果，并且允许用户在网页上进行编辑 ②后端负责接受前端送来的图片和一系列参数，进行图片的处理，并在处理好后返回结果。

重难点分析及解决方案

重难点分析可以分成前端和后端两个部分。

一、后端部分

（1）后端部分的重难点在于：

①如何校正图片，其包括图片重要区域的识别，识别到重要区域后如何将其合理有效第校正。②如何给照片添加上合适的效果，使其更加清晰明了美观。③如何进行文字识别。

（2）解决方案（具体细节见后文）：

①我们假定重要区域的形状为四边形，设计了一个算法进行该四边形的查找。之后，我们利用透视投影的原理将其校正。②我们设定了几个滤镜，分别有不同的效果，供用户选择。③我们发现百度 OCR 文字识别可以很好地解决这一问题。

二、前端部分

（1）前端部分的重难点在于：

①网页的绘制和设计，如何设计的更美观，布局合理，且对用户友好。②图片的上传，进一步地说，如何实现图片的拖拽上传，给用户以方便。③结果的展示，如何合理清晰地展示多张图片处理好的结果，即如何布局，以及如何设计网页使得用户可以方便的选择处理图片的参数（滤镜，锐化程度）。

（2）解决方案（具体细节见后文）：

①我们设计了两个页面，一个用于上传图片，一个用于结果展示，我们直接使用北大树洞的背景作为它们的背景，以及添加了一系列的 css。②我们发现了一个很方便的开源库 dropzone 实现拖拽上传③我们使用 HTML 的 table 标签进行布局，添加了一系列的 css 进行美化。另外，我们设计了一个页面内弹窗，当用户点击某张结果的图片时，该弹窗就会跳出，进行选择参数等操作。

简要操作方式

在根目录下输入指令 python ./server.py，然后进入网页上传图片，待所有图片都上传好（会有上传信息的展示），且所有图片都处理好后（会有“完成”两个字），点击确认进入结果的展示界面，根据需求点击合适的按钮即可。

下面介绍具体细节，先介绍后端部分，再介绍前端部分。

第一部分：后端

（负责成员：王思远）

一、整体流程

1. 加载原图：用户上传图片后，从相应路径获取原图。

2. 识别文档四边形：在原图中寻找文档四边形，也可由用户在网页上自行选定四边形。

3. 裁剪并恢复透视投影：即校正图片。

4. 锐化处理以及加滤镜。

5. OCR 文字识别（可选）。

第 1 点加载原图内容较少，下面主要介绍后四点。

二、图片的校正

1. 识别文档四边形

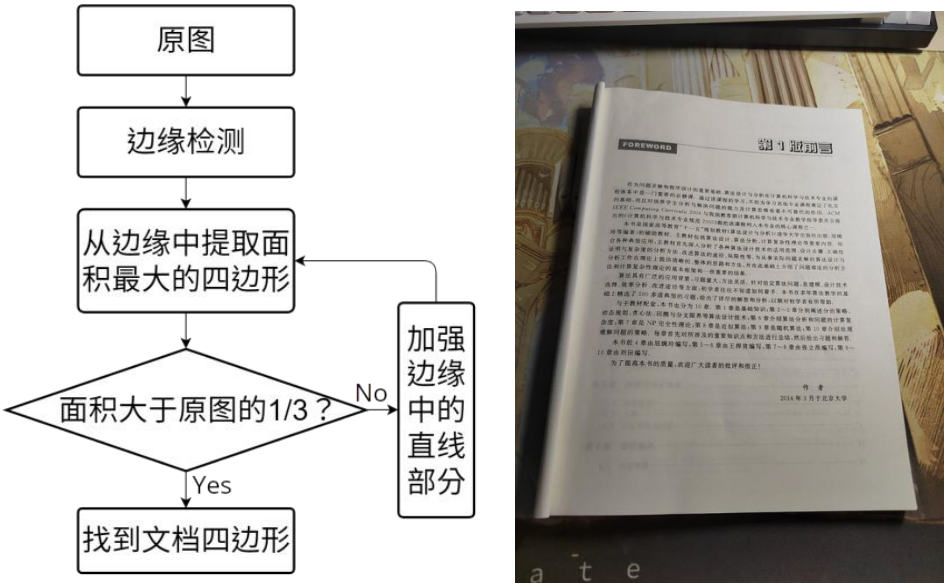


图 2 文档四边形寻找的整体流程（左）和原图（右）

为了使得边缘检测效果更好，我们先将原图的 RGB 三个颜色通道都分别提取出来。之后进行高斯模糊减少文字对边缘检测的影响。之后再利用 openCV 的 Canny 算法，对三个通道的图片进行边缘检测。该流程在图 3 中展示。

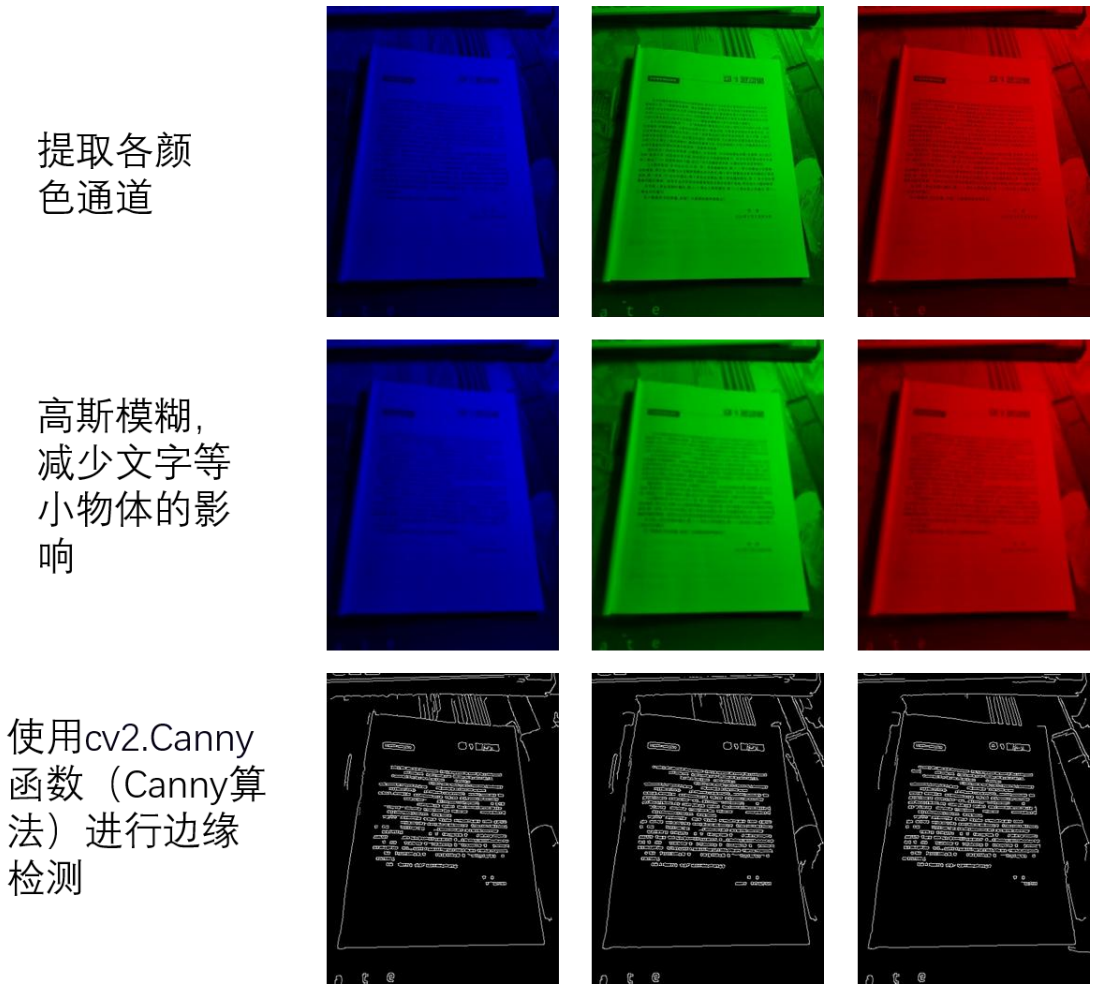


图 3 边缘检测

在此之后，可以将三通道提取的边缘合并（图 4 左），并对边缘进行膨胀处理（图 4 右）。

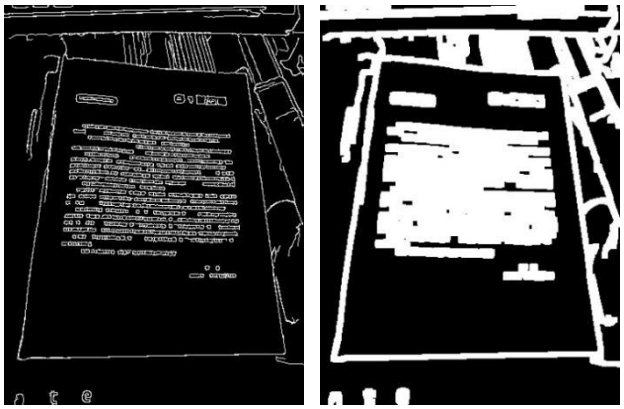


图 4 边缘检测结果

对上面得到的结果，使用 cv2.findContours 函数，在边缘中提取闭合的轮廓，并按面积从大到小排序，此处展示面积前三大的。

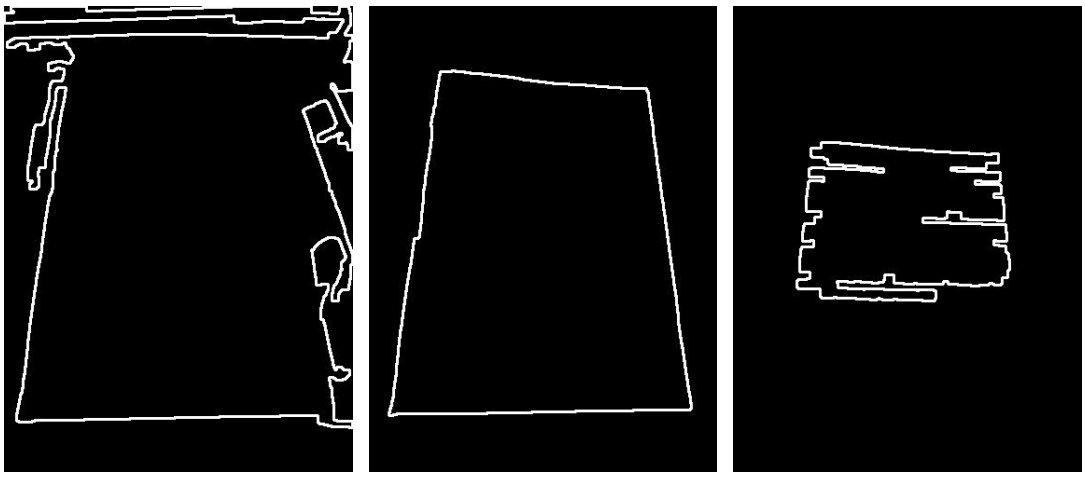


图 5 闭合轮廓

之后，按面积从大到小的顺序遍历各轮廓，并使用 cv2.approxPolyDP 函数，用多边形拟合轮廓。若找到一个四边形（此处是第二个），则返回。

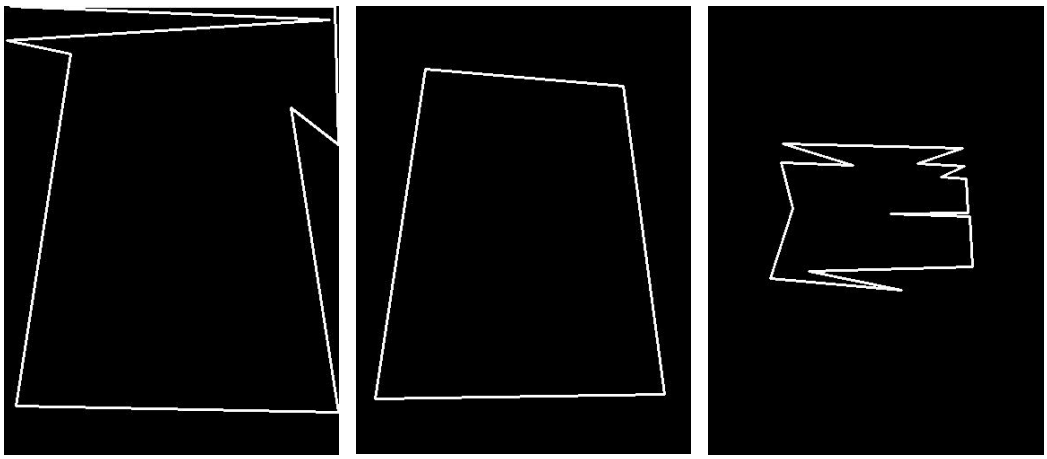


图 6 多边形拟合轮廓

如果我们提取出来的四边形足够大（大于原图的 1/3），那么我们就认为已经是不错的结果了，上面的例子就是这样；否则，我们会加强边缘中的直线部分，然后再次提取四边形，下面以另一个具体的例子来说明。

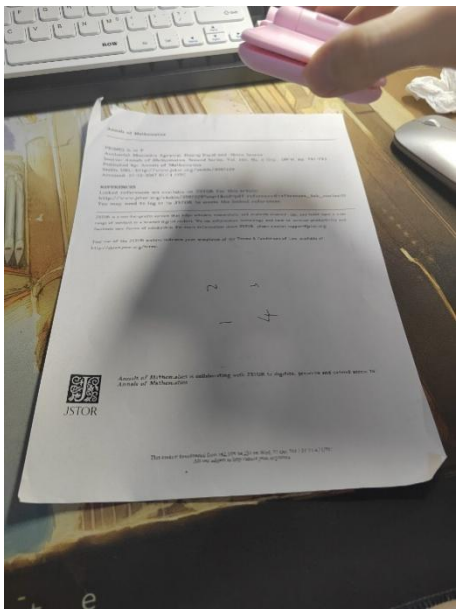


图 7 例二原图

我们以上面这张图作为原图，进行上述的文档四边形的识别。

首先进行三通道提取，三通道边缘检测，然后合并三通道的边缘后，进行膨胀处理，得到下面的结果（图 8 左）。

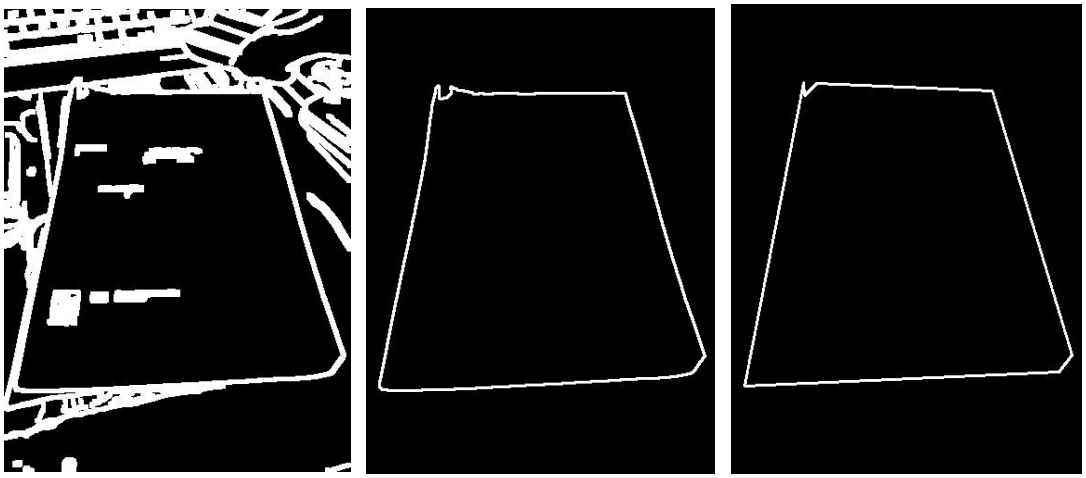


图 8 边缘检测结果（左）、闭合轮廓（中）和多边形拟合（右）

我们提取出边缘中的闭合轮廓。其中，最大的轮廓图形是上面这个（图 8 中）。虽然它是闭合轮廓，且显然面积大于原图的 1/3，可惜它无法用一个四边

形来拟合（图 8 右，左上角缺角）。事实上，最大面积的闭合的四边形为图 9 中展示的四边形。但是，很显然它的面积不大于原图的 1/3，故接下来需要加强边缘中的直线部分，从而有机会找到更大的闭合的四边形。

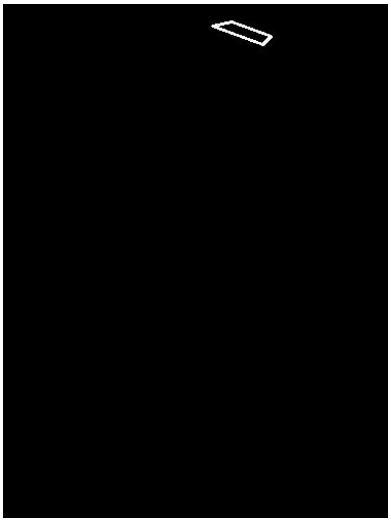


图 9 最大面积的闭合的四边形

我们使用 Hough 变换算法提取边缘中足够长的直线段，加粗并绘制在边缘图中，以加强直线部分。下面展示的分别为：Hough 变换提取的直线段（图 10 左）、原始边缘（图 10 中）、加强后边缘（图 10 右）。此处使用红色仅为了便于展示，实际算法中绘制直线段用的依然是白色。

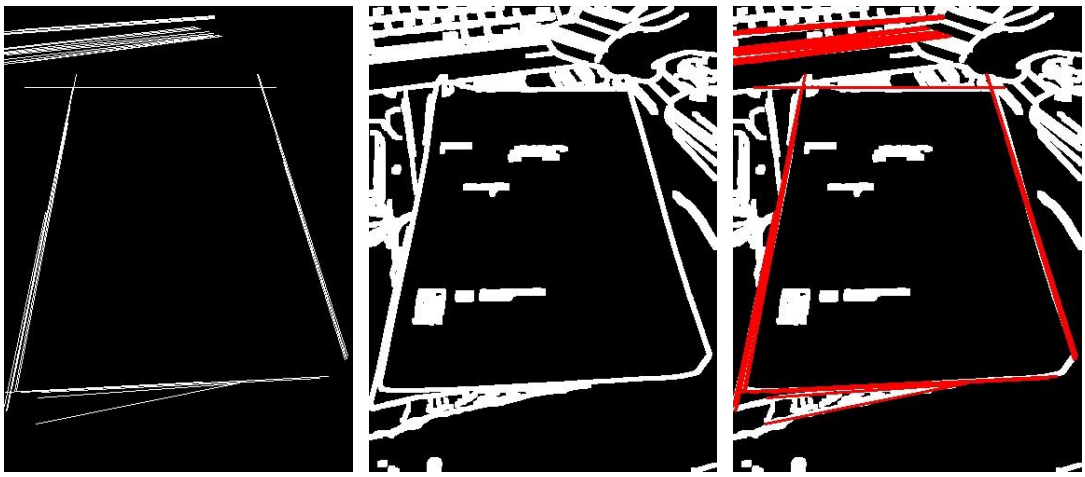


图 10 Hough 变换提取的直线段（左）、原始边缘（中）和加强一次后边缘（右）

数次加强后：

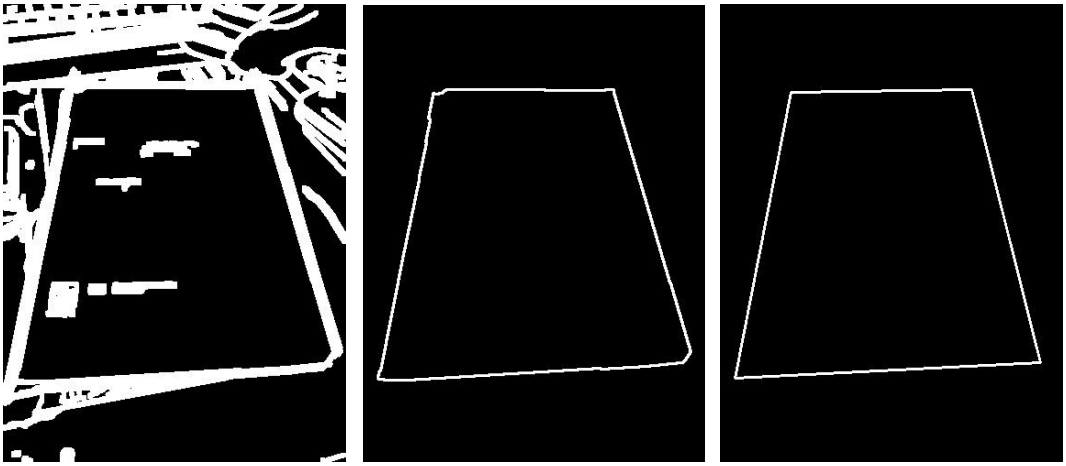


图 11 加强后边缘（左）、闭合轮廓（中）和文档四边形（右）

这样，对于任意一张合理的图片，我们就可以提取出来合适的文档四边形。当然，并不是每张图片都可以提取出来文档四边形（比如风景图片，其没有明确的四边形区域），这时我们会将整张图片视为文档四边形。

提取出来文档四边形，是为之后的图片校正和处理做准备。

2. 裁剪并恢复透视投影

有了上述提取出来的四边形，我们可以认为，四边形中的内容就是该图片的主要内容，四边形外的内容是不重要的内容，我们会将四边形外的部分裁剪删去，剩余的部分经过透视投影计算，将其校正。

我们认为，书页的拍摄环境可以简化成这样的一个模型：空间中存在一个矩形，摄像机为空间中与该矩形所在平面相离的一个点。在摄像机的拍摄下，该矩形投影成一个不规则四边形。要尽量完美地恢复原书页的形状，我们需要根据这个不规则四边形的形状得到原来矩形的长宽比。我们发现，开源代码、开源项目的实现大多在计算长宽比时可分为两种策略。策略一：无论原矩形长宽比如何，都放缩成某一固定比例，例如简单认定所有拍摄的书页都为 A4 纸比例。策略二：选取不规则四边形相对的两条边的加权平均值为原矩形的长，另两条边的加权平

均值为原矩形的宽，依此计算原矩形长宽比。然而，这两种策略都有明显的问题。对于策略一，若原矩形的长宽比与预设的长宽比相差较多，扫描出的书页会产生严重的变形。对于策略二，若拍摄角度较为倾斜（例如图 12 左），也无法得到准确的原始长宽比。与这两种常见的较简单的策略不同，我们选用了 Whiteboard Scanning and Image Enhancement[1]中的技术，根据不规则四边形的形状，计算出拍摄环境中相机的焦距，进而可以准确地求解原矩形长宽比。根据该长宽比，我们可以较好地恢复透视投影，得到不被变形的原始书页图片。

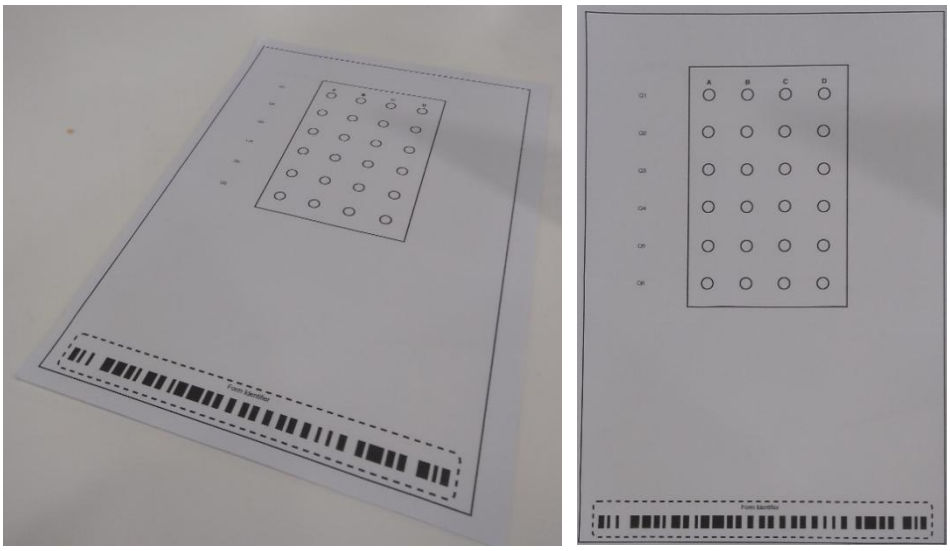


图 12 校正前（左）校正后（右）

三、图片处理和文字识别

1. 图片处理

我们提供了 6 种滤镜（“原图”是将原图校正后得到的图片，这里我们将其算做滤镜）和图片锐化程度（0-10）的选项。这些选项由用户在网页来选择。

图 13 图片处理的选项

在进行锐化处理时，我们使用了反锐化遮罩技术。假设原始图片为 X，高斯模糊处理后的图片为 Y，我们可以认为 X-Y 就是图像中的清晰部分。给原始图片加上α(X-Y)，就可以实现对图片的锐化处理。控制α可以控制锐化程度。

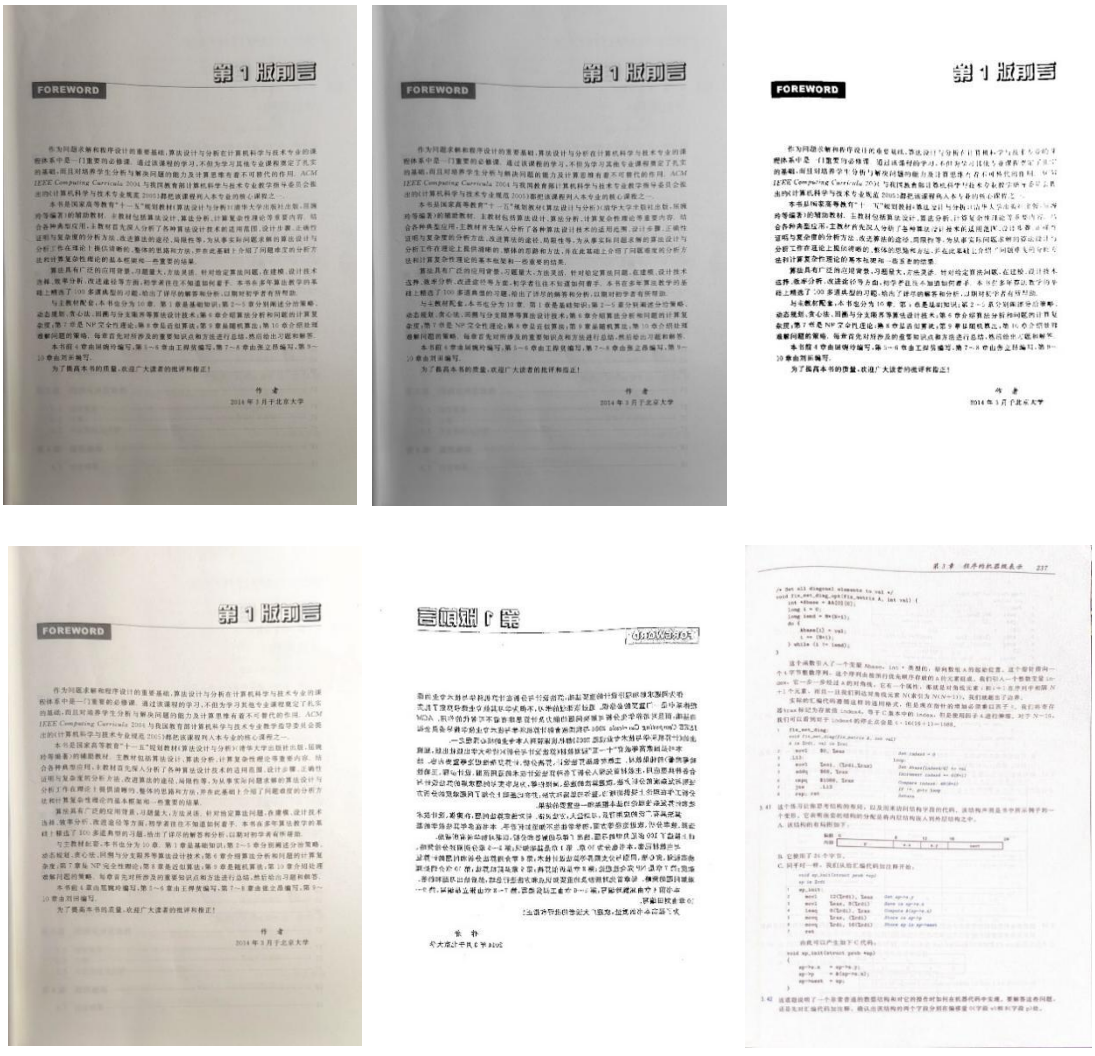


图 14 各个滤镜的效果

图 14 展示了各个滤镜的效果，从左到右、从上到下依次为：原图、灰度、黑白、增亮、推荐滤镜（黑白）、推荐滤镜（彩色）。因为原图中的字都是黑色

的，所以推荐滤镜（彩色）的示例，我们用的是另一张图片。

② 原图滤镜：原图校正后的图片。以下各个滤镜处理的都是校正后的图片。

②灰度滤镜：将图片转换为灰度图。

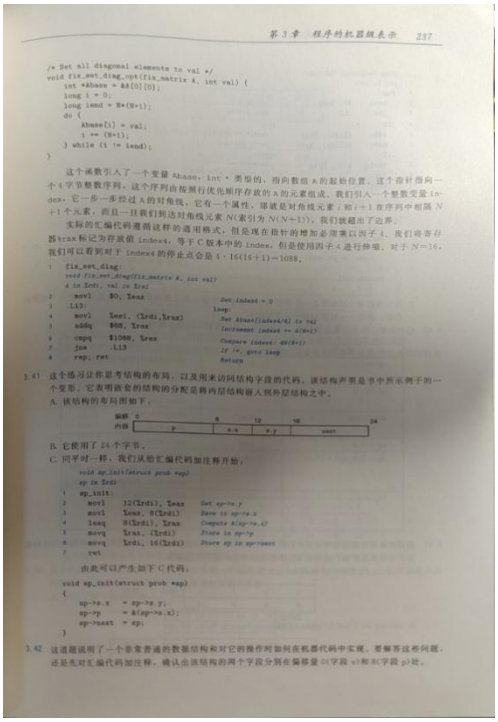
③黑白滤镜：这里我们采用二值化技术，即将原图转为灰度图后，再将亮度大于阈值的像素设为白色，小于阈值的像素设为黑色。

④增亮滤镜：我们将原来 RGB 三通道的图片，转至 HSV 三通道。HSV 三通道的图片，每个像素都有三个值：色调（H）、饱和度（S）和明度（V）。增大 V 通道的值，即可实现对整张图片的增亮。最后再转回 RGB 格式即可。

⑤推荐滤镜（黑白）：我们最初选择使用自适应二值化技术：每个像素的阈值依赖于其邻近的像素。取以该像素为中心的一个正方形（例如 9*9 正方形），这个像素的阈值就是正方形中所有像素的亮度平均值。如果该像素亮度大于阈值则设为白色，小于阈值则设为黑色。

我们的目标是处理成“白纸黑字”的效果，而上面的做法还有优化空间。上述直接使用均值作为阈值，在背景较多的部分，总会有一些背景像素的亮度小于阈值，被设成黑色。将阈值按比例稍稍降低，例如乘以 0.9，可大大减少亮度小于阈值的背景像素数目。

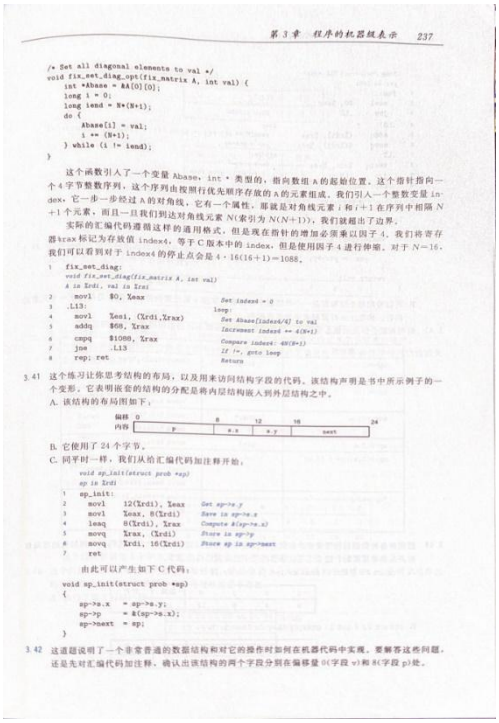
⑥推荐滤镜（彩色）：主要作用是去除图片上的阴影和色块，并保留有色线条。我们的做法的主要思想是：假设原始图片为 X，对 X 进行膨胀和高斯滤波来模糊文本部分，得到一个只含原图中色块的背景图像 Y。计算 X-Y，并将其反相即可（见图 15）。



X

Y

X-Y



最终结果

图 15 推荐滤镜（彩色）的效果

2. OCR 文字识别

这里我们是直接调用百度文字识别 API 来实现。

```
def recognize_img(self, img_path):
    """
    OCR识别图片
    img_path: 待识别图像存储路径
    return:   识别结果
    """
    img = None
    with open(img_path, 'rb') as fp:
        img = fp.read()
    options = dict()
    options['language_type'] = 'CHN_ENG'
    options['detect_direction'] = 'true'
    options['detect_language'] = 'true'
    res = self.__client.basicAccurate(img, options)
    ret = []
    for result in res['words_result']:
        ret.append(result['words'])
    return '\n'.join(ret)
```

图 16 OCR 文字识别

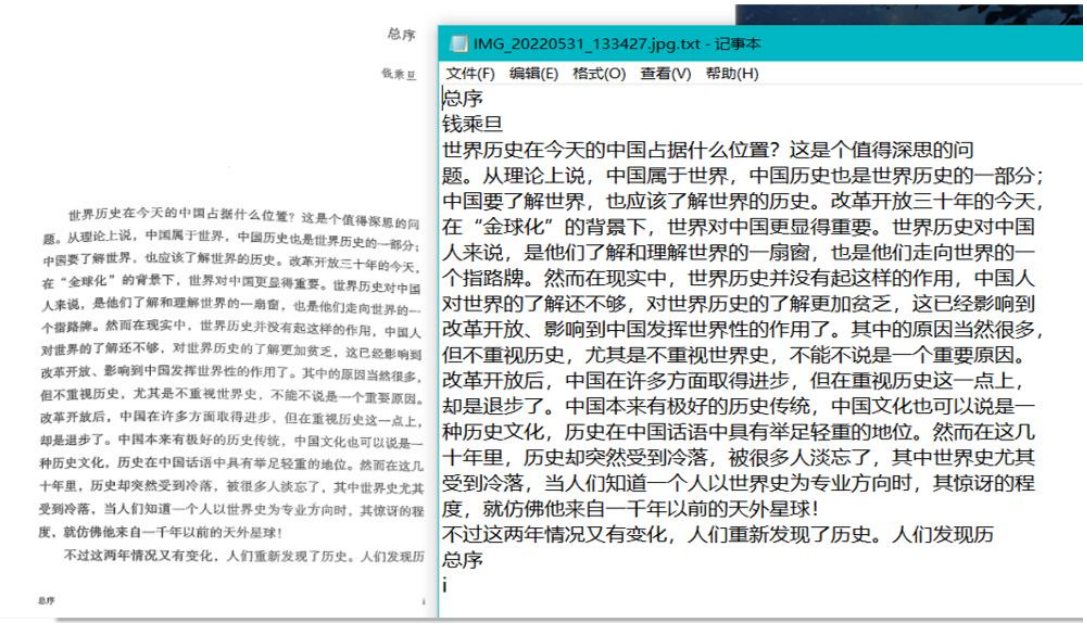


图 17 文字识别结果

第二部分：前端

（负责成员：王小翔）

一、整体组成

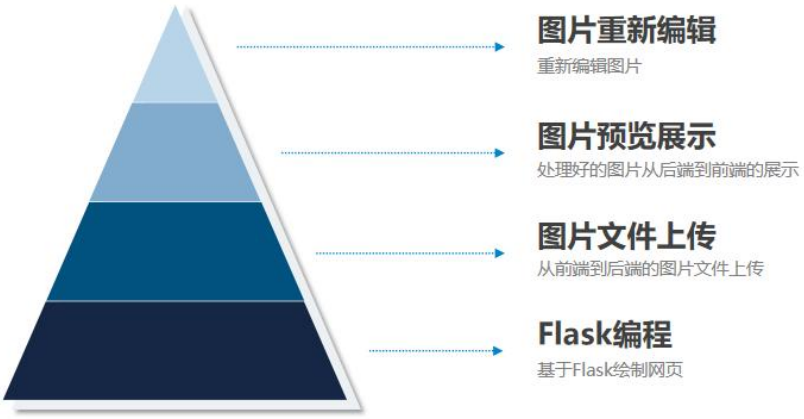


图 18 前端组成

二、网页绘制和拖拽上传

网页绘制包含两个页面，一个页面是上传图片的页面，另一个页面是展示图片处理结果的页面，分别对应 template 文件夹下的 index.html 和 show.html。网页绘制主要使用的是 HTML，CSS，jquery 以及许多网上找到的 js 和 css。

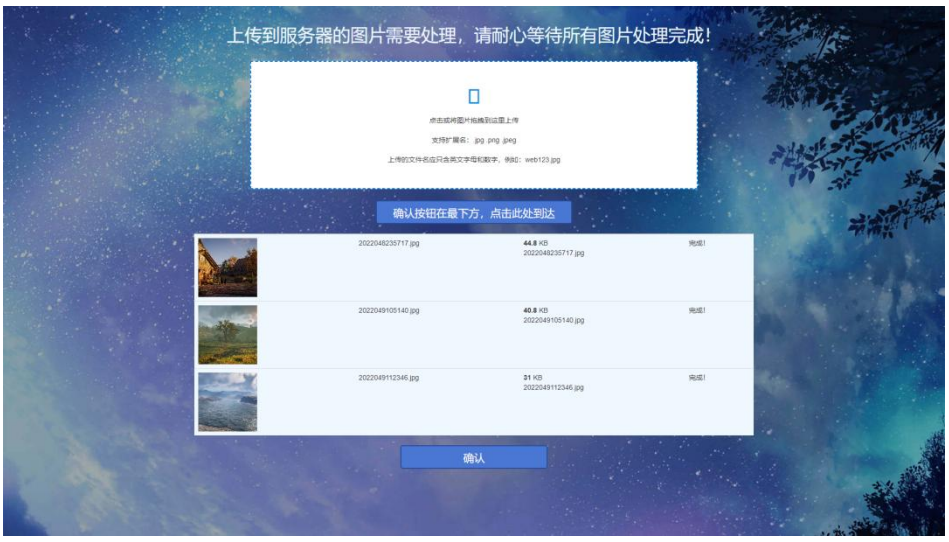


图 19 index.html

图 20 show.html

拖拽上传，我们是直接调用一个现有的前端的开源库 dropzone[2]。实现了以下功能：

①文件拖动或点击上传

②文件上传限制和提示

③文件上传事件函数自定义

③ 上传信息展示和图片预览

图 21 拖拽上传

三、图片处理结果的展示与编辑

图片处理结果的展示见图 20，我们利用 HTML 的 table 标签进行布局，利用 CSS 进行美化。（比如将 table 设置为半透明，给图片上方的编号添加荧光效果等）

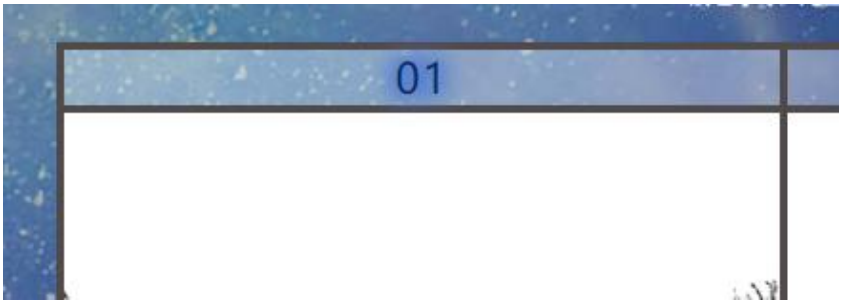


图 22 美化效果

该页面中有三个按钮：返回，生成 PDF 和文字识别（见图 23）。返回就是返回到上传图片的页面，即 index.html；生成 PDF 会将所有图片按顺序合并成一个 PDF，然后返回给用户下载，这里使用的是 response 实现的；文字识别便是对所有图片按顺序进行文字识别，此时文字识别的结果除了文字的内容，还会有其对应的图片的编号，最后合并在一个 txt 文件中，并返回给用户下载。

图 23 三个按钮

在该页面，还可以进行图片的编辑。用户左键单击任意一张图片，会跳出来一个弹窗，该弹窗展示的是放大后的结果图以及一系列按钮（见图 24），按钮的名字就是其功能。左键点击”编辑图片“的按钮，弹窗会发生改变（见图 25），弹窗内的图片会变为原图（未经过任何处理的原始上传的图片），用户可以选择

不同的滤镜和锐化程度，以及可以自行选取四边形区域。

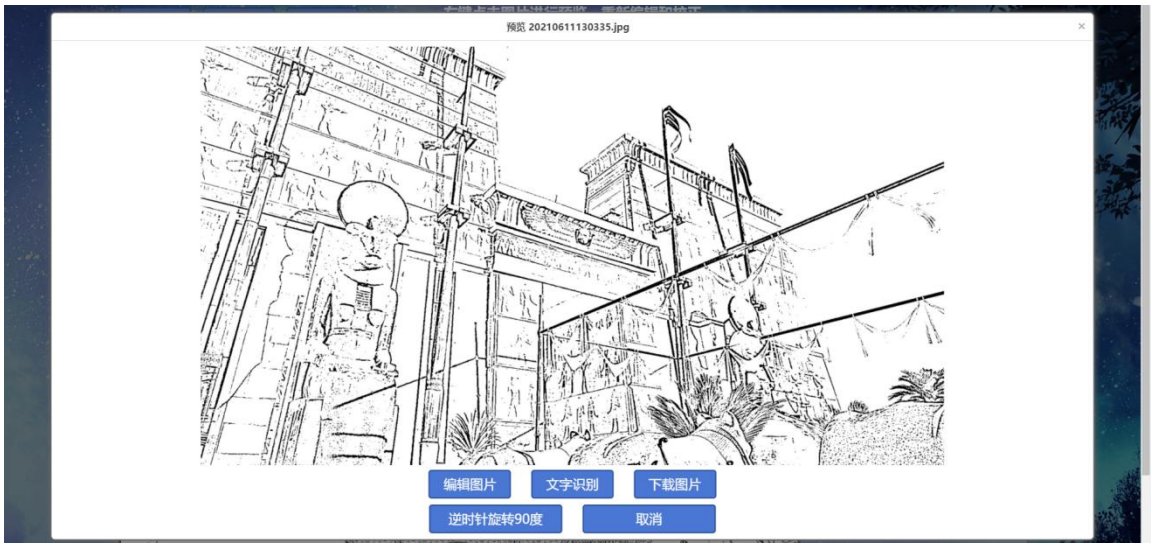


图 24 弹窗 1

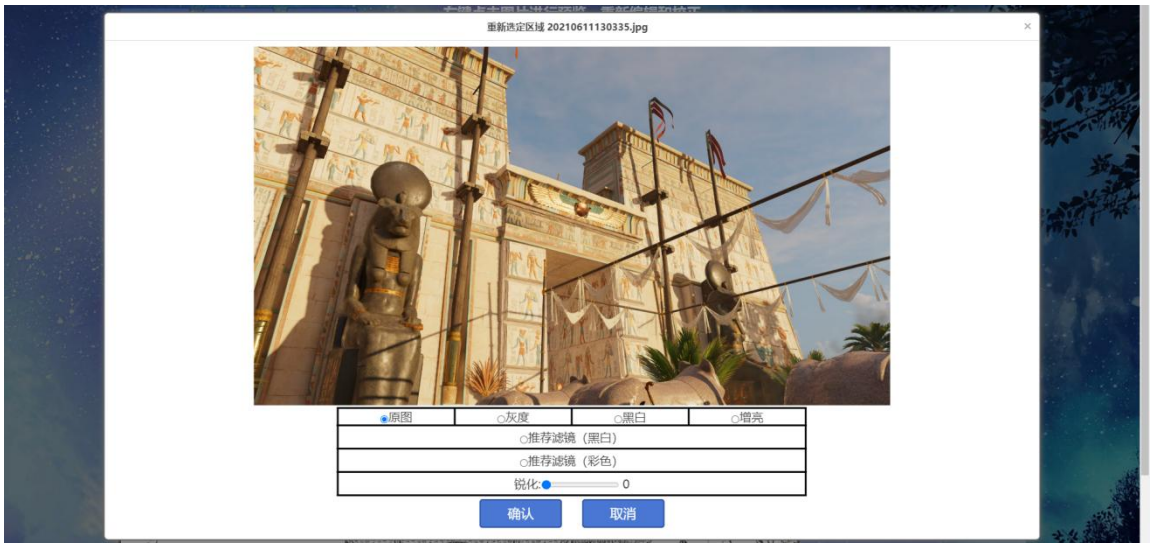


图 25 弹窗 2

在弹窗 2 中，用户可以自行选定四边形区域，选定方法是按左上、右上、右下、左下的顺序（也就是从左上开始的顺时针顺序），左键点击选择图片的四点。自行选定区域的过程中，边框会实时显示，而不是在点完四个点之后才出现。选定好后，若又想重新选定，只需重复点击四下即可，上次的边框会自动消失。边框的样式见图 26。

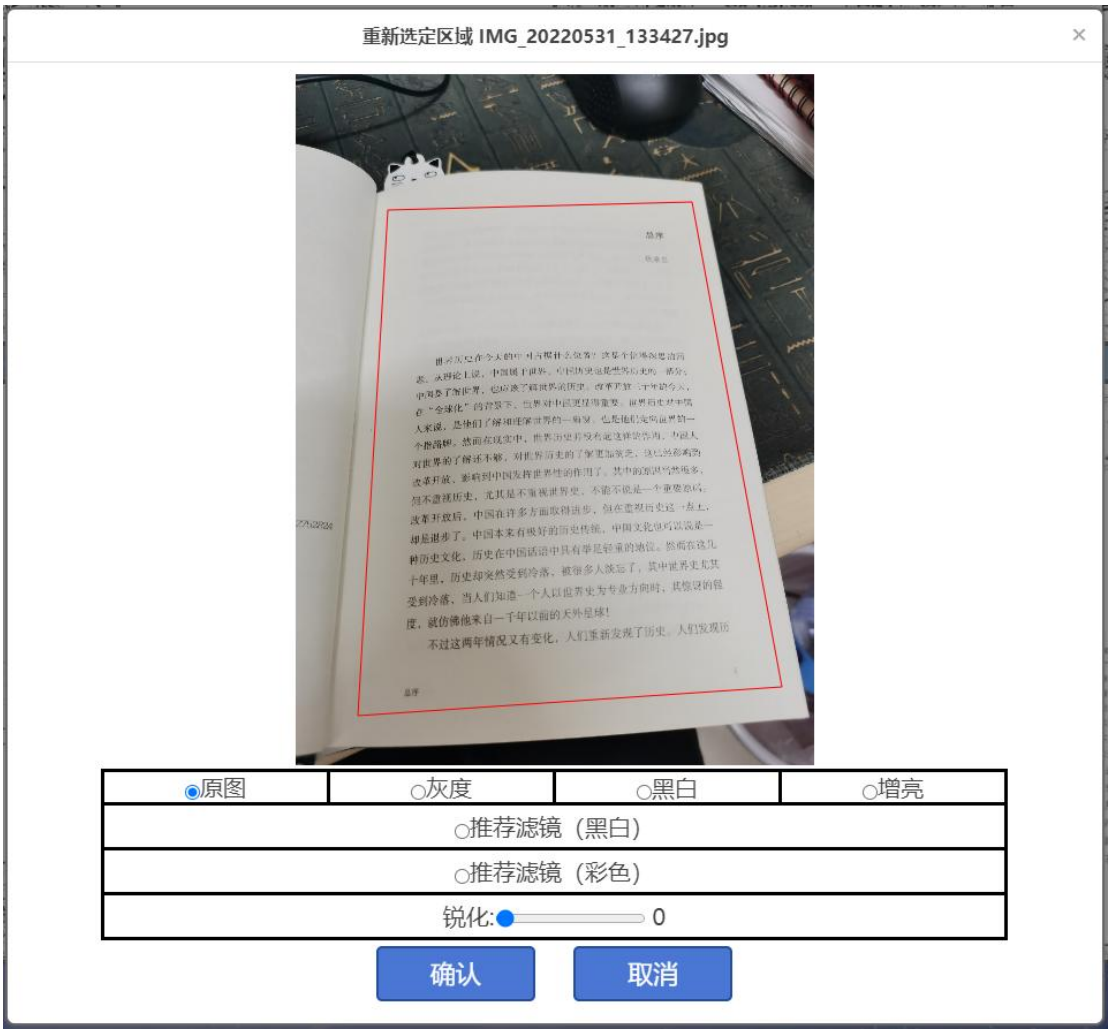


图 26 自行选定区域的边框样式（注意图片中的红色框线）

总结

这次大作业让我们更加了解了图片处理的技术，更熟悉了前端代码的编写，对前后端的交互的设计有了我们自己的见解。在编写过程中，我们遇到并解决了诸多困难（比如图片缓存问题，使得我们不能实时更新图片），从中收获了许多新的知识。最后感谢老师的教导，以及众多线上代码交流平台、各种开源代码和相关论文为我们提供的帮助！

参考资料

[1] Zhang, Zhengyou and He, Li-wei, Whiteboard Scanning and Image Enhancement, Digital Signal Processing, 2007 April, 414-432

[2] Dropzone.js, https://www.dropzone.dev/js/