任务一流程介绍　图像定位与信息输出

陈冠韬　陈昭羽　付慧妮

2024 年 10 月 23 日

一、任务描述

现在给定一含有多页封泥的 pdf 文档，要求对于其中每一页文档，分别提取出其中的图像与文本信息。每一页文档形如：

图 1: 文档样例一

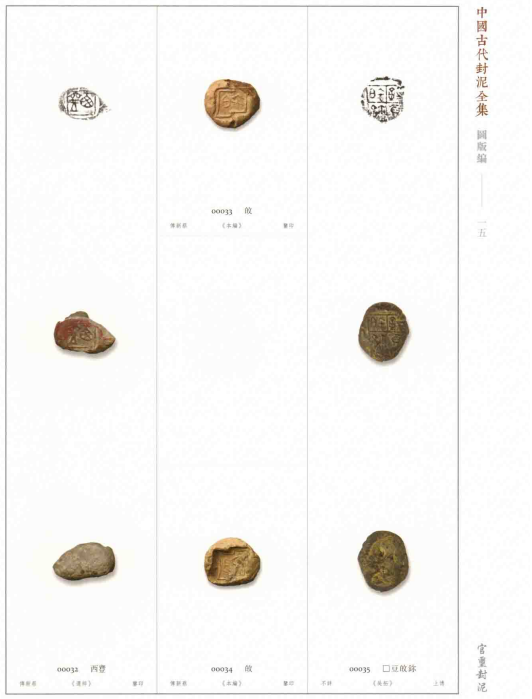

图 2: 文档样例二

二、问题分析

任务可以拆分为以下几个步骤：

1. 从 PDF 文档中提取出来每一页封泥。

2. 对于每一页封泥，进行**大分割**。即根据封泥的序号提取出来不同封泥的所有基本信息。
3. 对于每一个封泥，再进行小分割。即分离出每个封泥的拓本图片、正面照片和其它照片（如果有）、释文及著录信息。
4. 将获取到的全部信息按顺序输出到 Excel 表格中。

最终的分割效果应如下图所示：

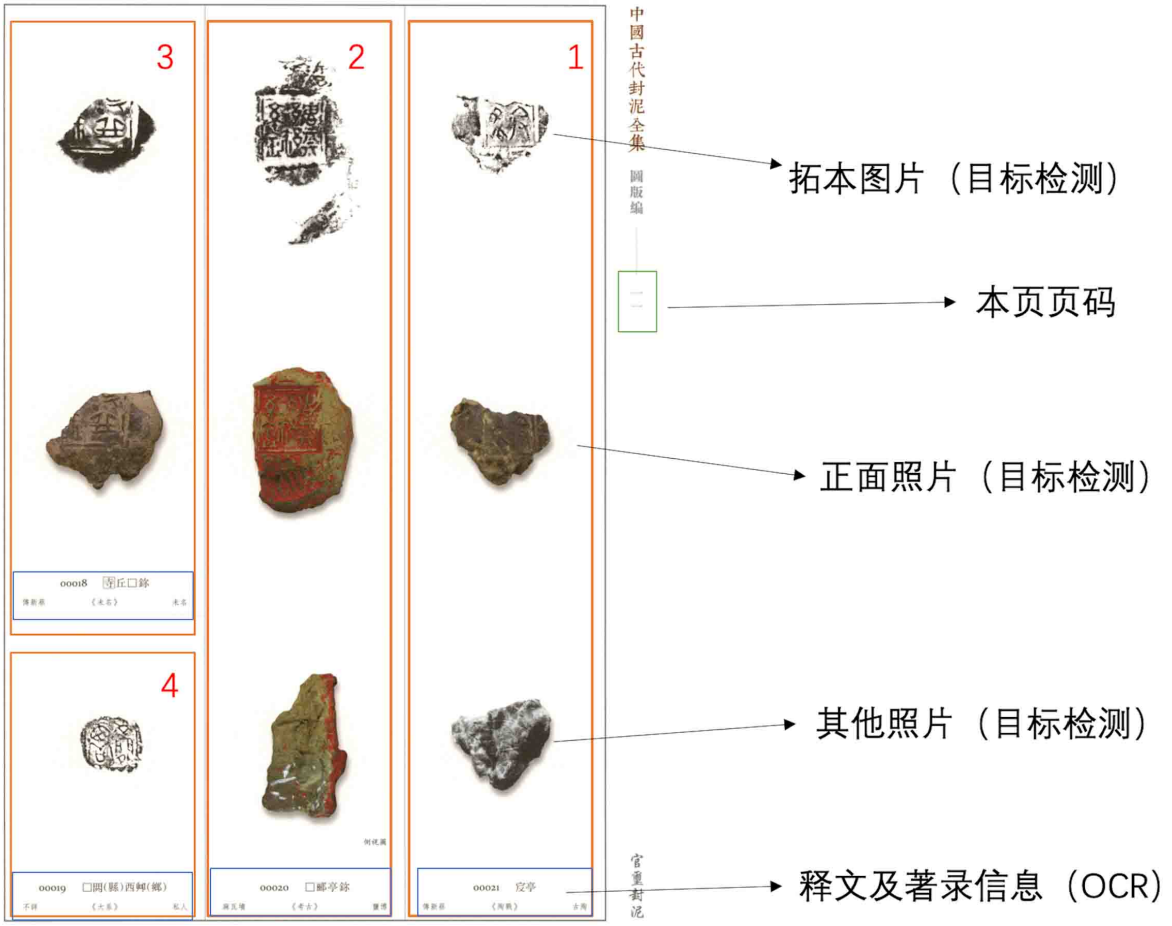

图 3: 分割样例

三、设计过程

写在前面：本次任务用到了许多 python 扩展库，如 pdf2image、matplotlib、numpy、cv2 等等，如果电脑尚未安装，可以在终端或者 cmd 中使用如下命令安装：

```
pip3 install pdf2image matplotlib numpy opencv-python
```

下面详细介绍每一个步骤的实现过程。

（一） 从 PDF 文档中提取图像

我们使用 python 的 pdf2image 库中的 convert_from_path 方法来实现图像提取、os 库来将输出图像保存在电脑中。

```python
# 导入如上所述的库
from pdf2image import convert_from_path
import os

# 需要提取的 PDF 的文件路径。
# 在自己电脑上运行时如果以下的相对路径不好用，请更改为绝对路径。
base_path = os.path.dirname(__file__)
pdf_path = os.path.join(base_path,'../01. 中国古代封泥全集 · 图版编.pdf')

# 绝对路径形如下面这一行（要删除开头的 #）
#pdf_path = '/Users/tony/Desktop/Tony/大学/实验室/数字人文研究院任务/任务 1/备份/01. 中国古代封泥全集 · 图版编.pdf'

# 输出图像的目录（注意事项同上）
output_dir = '/Users/tony/Desktop/Tony/大学/实验室/数字人文研究院任务/任务 1/导出图片'

# 确保输出目录存在
os.makedirs(output_dir, exist_ok=True)

# 将 PDF 文件的每一页转换为图像
images = convert_from_path(pdf_path)

# 保存每一页的图像
for i, image in enumerate(images):
    image_path = os.path.join(output_dir, f'page_{i + 1}.png')
    image.save(image_path, 'PNG')
    print(f'文件第{i+1}页已经保存在目录中')

print(f'PDF 文件的每一页已成功转换为图像并保存在 {output_dir} 目录中。')
```

除去输入输出路径需要自己更改外，其他的代码直接运行即可。

开始运行后稍等一两分钟，总共有大概 260 张图片。注意整个文档的每一页都生成了图像，生成完之后可以手动把非封泥的页面删掉（保留 164-259 页）。当然也可以直接在代码中更改。

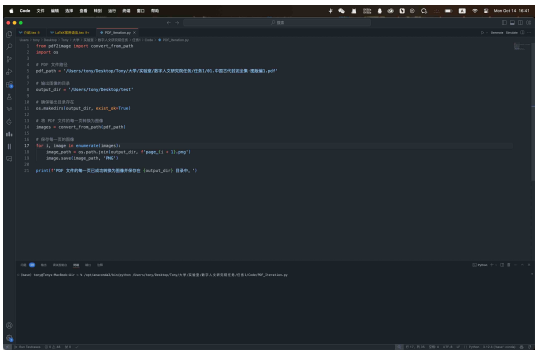

图 4: 运行图

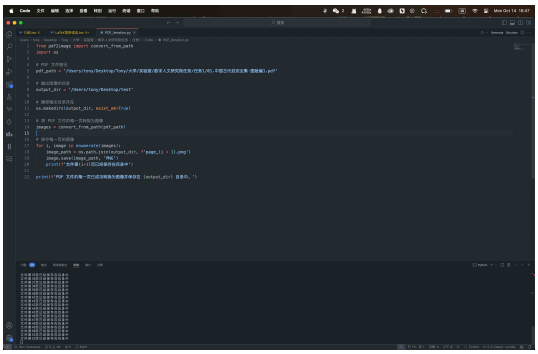

图 5: 生成图

（二） 处理图片

写在前面：我们注意到中国古代封泥全集·图版编中的每个封泥间都已经用淡颜色的线分割开了。因此在做大分割步骤时，我们可以直接使用这些线作为分割的参考线。（对于扩展文件陶文目录，它的每一个封泥印仅有图像和文字两部分，提取图像时可以直接使用下文所述的小分割进行处理，跳过大分割这一步骤）。

从 pdf 中提取出来了每一页图像之后我们就可以开始进行图像识别了。为了使识别结果更加精准，我们可以先对原始图像进行处理，并将结果另存为一个副本。对副本进行假分割操作获得参数后，再对原图像进行分割。

我们的目标就是识别出淡色的分割线。我最开始尝试使用 *cv2* 模块的 *cvtColor* 和 *threshold* 方法将图像转化为灰度图像、进而转化为二值图像使得分割线更为清晰，但是识别后发现效果不好。

```
# 读取图片
image = cv2.imread('path_to_image.png')

# 将图片转换为灰度图像
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 将灰度图像转换为二值图像
_, binary_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)
```

然后我尝试使用 Adobe Photoshop 先对单张图像进行处理：

1. 适当降低曝光，使得黑线更加突出。
2. 提高对比度。
3. 适当拉高高光，否则背景会出现灰色色块影响识别。
4. 拉低阴影。
5. 适当提高白色色阶。

6. 降低黑色色阶。

7. 曲线方面，我们可以简单拉出一个 S 型，使得黑色部分更黑，白色部分更亮。

我的参数放在这里：

- 曝光 -1.55
- 对比度 +100
- 高光 +57
- 阴影 -70
- 白色色阶 +15
- 黑色色阶 -100
- 纹理 +3
- 清晰度 +26

最后发现识别效果很好。我也尝试用了 python 模拟 photoshop 的调色过程，但是收效甚微：

```python
def adjust_exposure(image, exposure):
    return cv2.convertScaleAbs(image, alpha=exposure, beta=0)

def adjust_contrast(image, contrast):
    return cv2.convertScaleAbs(image, alpha=contrast, beta=0)

def adjust_whites_blacks(image, whites, blacks):
    image = np.clip(image + whites, 0, 255)
    image = np.clip(image - blacks, 0, 255)
    return image
```

抛弃 python，我们将所有图片导入 Adobe Lightroom 进行批量编辑，将一张图片的参数黏贴到每一张图片（全选即可，不用一张张黏贴），最后导出。这样导出的图片就可以直接进行识别了。

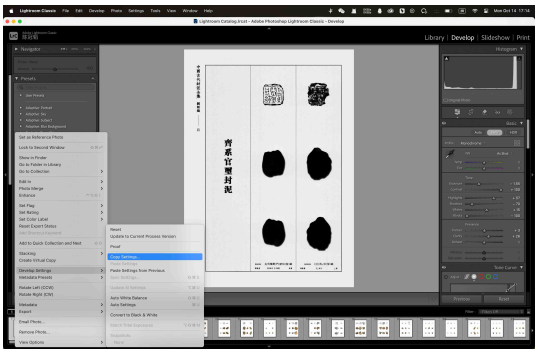

图 6: lr 复制参数

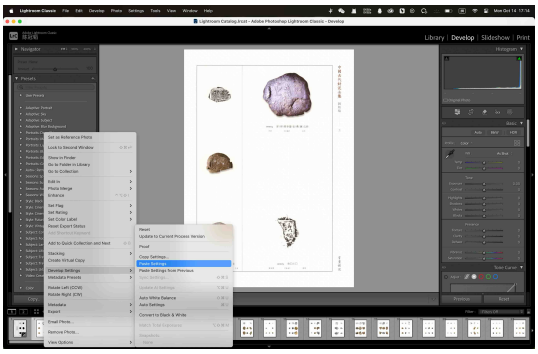

图 7: lr 黏贴参数

（三） 大分割

我们先来看主程序：

```python
# 引入所需文件库，其中 func 是我自己编写的存放所有函数实现的文件
import cv2
import numpy as np
import matplotlib.pyplot as plt
from func import *

# 读取图像
image = cv2.imread('/Users/tony/Desktop/page_242.jpg')
# vertical_horizontal_lines 方法，进行图像分割并返回所有水平线、竖直线，以及所有线的汇总
vertical,horizontal,new_lines = vertical_horizontal_lines(image)

# 显示原图
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Picture')
plt.axis('off')
plt.show()

# 如果没有检测到分割线，直接显示原始图像
if(len(new_lines) == 0):
    cv2.imshow('img',image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    #vertical 表示所有竖直的分割线、horizontal 表示水平的
    for i in range(len(vertical)-1):
        # 先对每张图像进行竖分割
        img = image[:,vertical[i]:vertical[i+1]]
        height,width,_ = img.shape
        top_left = (vertical[i],0)
        bottom_right = (vertical[i+1],height)
        # 如果分割出来的图像中有水平线，接着对水平线进行横分割
        if(horizontal_line_in_area(new_lines,top_left,bottom_right)):
            for j in range(len(horizontal)-1):
                new_img = img[horizontal[j]:horizontal[j+1],:]
                cv2.imshow('img',new_img)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
        # 如果没有，显示进行了竖分割的图像
        else:
            cv2.imshow('img',img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
```

了解大概思路后，我们来看每个重要的小方法是如何实现的：

1、 vertical_horizontal_lines 方法

分割线识别 常规操作，参数设置参看形式参数列表。

```
def vertical_horizontal_lines(
    image,apertureSize=3,minLineLength=300,maxLineGap=10):
    # 读取图像并转换为灰度图像
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 使用 Canny 边缘检测，调整阈值
    edges = cv2.Canny(gray, 50, 150, apertureSize=apertureSize)

    # 使用概率霍夫变换检测直线
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, 100,
    ↪  minLineLength=minLineLength, maxLineGap=maxLineGap)
```

remove_similar 方法 如果没有这个方法，分割出来的图像会有很多相互交错但不重合的分割线，对后续的识别造成影响：可以看到分割线变得清晰。

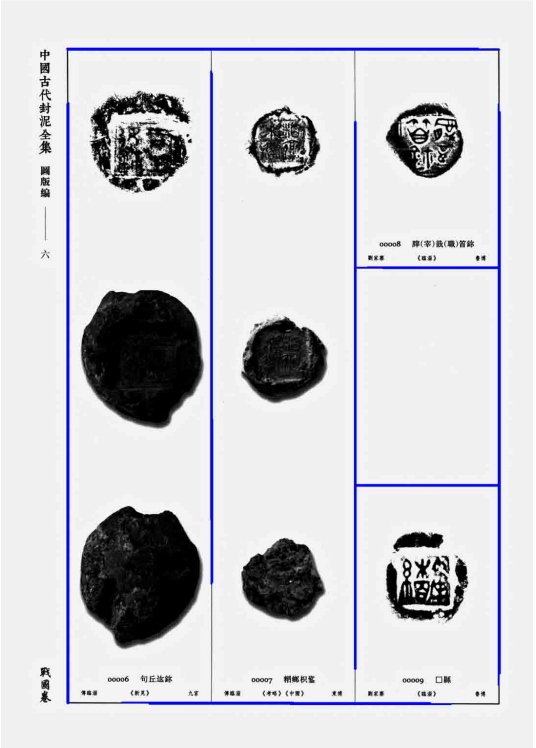

图 8: 处理前

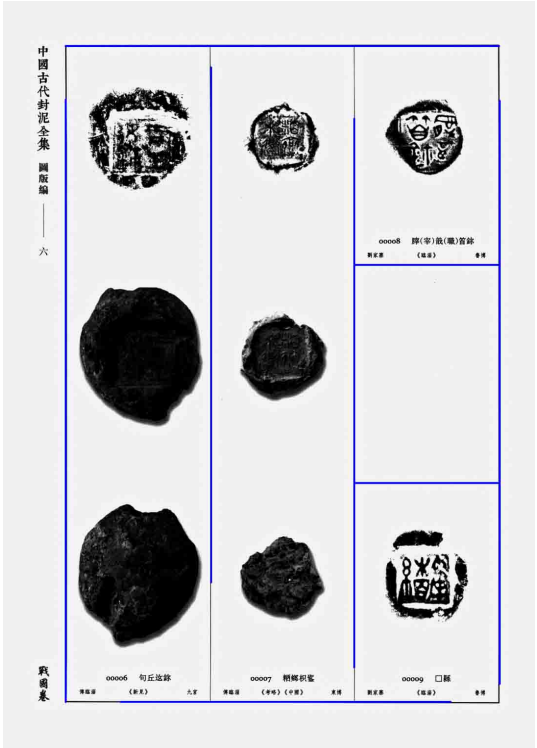

图 9: 处理后

```
def remove_similar(lines):
unique_lines = []
for line in lines:
    line = unify(line)
    if not similar(line, unique_lines):
        unique_lines.append(line)
    else:
        # print("Here expand")
        expand_line(line, unique_lines)
return unique_lines
```

简单来说，就是检测新绘制出的分割线是否与已有的分割线相近。如果不相近，直接绘制出新的分割线；如果相近，对那条相近的线进行拓展（也就是将两条线合并并对齐）。

2、 horizontal_line_in_area 方法

用于检测分割出来的小图像中是否有水平分割线，如果有，则继续进行水平分割。

```
def horizontal_line_in_area(lines,top_left,bottom_right):
print("line is: ",lines)
print("top_left is: ",top_left)
print("bottom_right is: ",bottom_right)
for i in range(len(lines)):
    if(lines[i][1]!=lines[i][3]):
        continue
    elif(length(lines[i]) < bottom_right[0]-top_left[0] and lines[i][0] >
    ↪  top_left[0] and lines[i][0] < bottom_right[0] and lines[i][1] >
    ↪  top_left[1] and lines[i][1] < bottom_right[1] and lines[i][2] >
    ↪  top_left[0] and lines[i][2] < bottom_right[0]):
        return True
return False
```

其中的 length 方法用于计算一条线的长度。当且仅当水平线的长度小于划分出来的小区域的宽度时，这条水平线才有可能成为分割线。

3、 分割结果展示

以上就是主要的函数实现，还有部份实现较为基础就不演示了。下面我们来看识别的结果。我们以图像 *page_166* 为例：

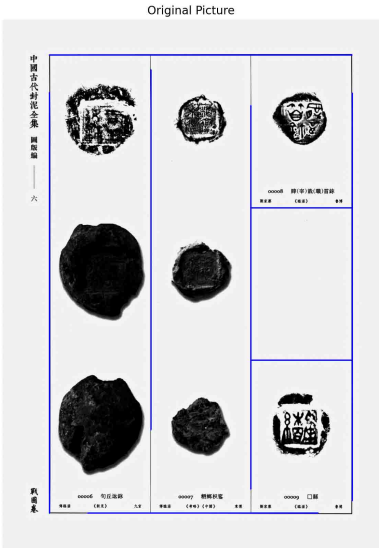

图 10: 原图识别

图 11: 输出一

图 12: 输出二

图 13: 输出三

图 14: 输出四

图 15: 输出五

可以看出识别效果还是相当不错的（周围的蓝线仅为展示方便，可以关闭）。

4、 空白图像去除

我们使用 *cv2* 模块的 *threshold* 与 *countNonZero* 方法来判断图像是否为空白。

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

我们先将图片转化为灰度图像，然后使用 *threshold* 方法将图像转化为二值图像，使用 *countNonZero* 方法来计算非零像素点的个数，最后计算非零像素点占所有像素点的比例。如果比例大于阈值 ϵ，则认为这张图像不是空白图像。

但是仅仅通过一个阈值来判断是否空白是不足够的，因为某些空白图像的边缘是有颜色的，可能会被识别为非空白图像；有些非空白图像的白色背景占据了大部分面积，可能会被识别为空白图像。

我们通过截取图像中央 90% 的部分来判断是否为空白图像，这样可以避免上述问题。

```
# 获取图像的宽度和高度
height, width = image.shape[:2]

# 计算中间 90% 区域的起始和结束坐标
x_start = int(width * 0.05)
y_start = int(height * 0.05)
x_end = int(width * 0.95)
y_end = int(height * 0.95)

# 截取中间 90% 区域
image = image[y_start:y_end, x_start:x_end]
```

最后我们返回是否大于阈值或者图像的总像素数是否小于 10000（即图像是否过小）。

```
return non_zero_pixels / total_pixels > threshold or total_pixels < 10000
```

可以发现运行的结果是正确的。

5、 批量处理

我们共有两个输入文件夹：原始图像和经过 *Lightroom* 处理后的灰度图像。我们遍历灰度图像文件夹中的每个图像，获得切割数据后在原始图像文件夹中找到对应的原始图像进行切割。对于每一个原始图像，在输出文件夹中生成一个子文件夹，存储该原始图像的所有小分割图像。

```
def process_all_images(parody_file_dir, original_file_dir, base_output_dir):
    # 获取所有 parody 文件
    parody_files = [os.path.join(parody_file_dir, f) for f in
    ↪  os.listdir(parody_file_dir) if not f.startswith('.')]
    # 获取所有 original 文件
    original_files = [os.path.join(original_file_dir, f) for f in
    ↪  os.listdir(original_file_dir) if not f.startswith('.')]

    for parody_file, original_file in zip(parody_files, original_files):
        # 处理图像
        process_image(parody_file, original_file, base_output_dir)
```

（四） 小分割

做完以上步骤我们可以获得如下图所示的文件夹：

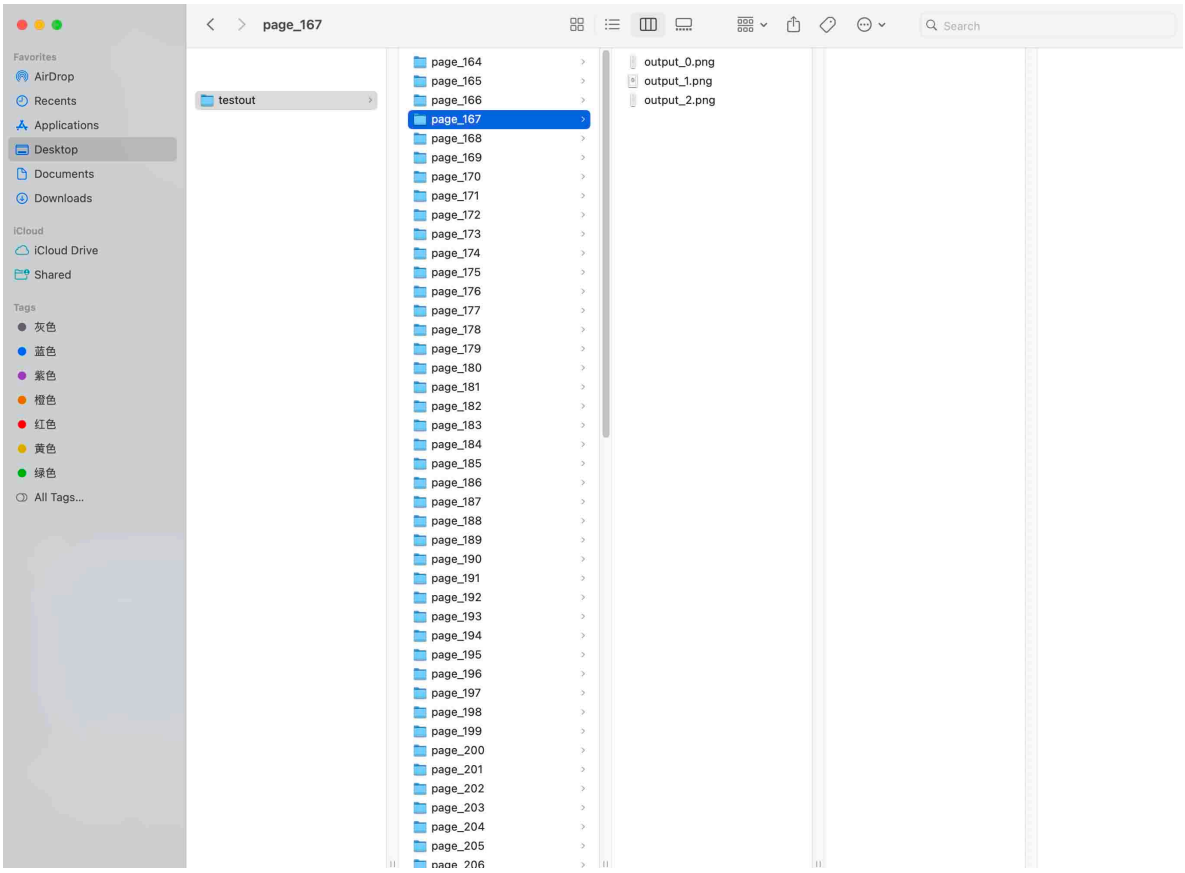

图 16: 大分割处理后文件夹

每一个子文件夹的名称为 *page_xxx*，表示这张图片来自原书的 xxx 页，其中包含了该页所有的大分割图像。我们下面对其中某一个图像进行小分割。

我们先对图像进行预处理：

```
def segmentation_display(input_image_path):

    # os.makedirs(output_dir, exist_ok=True)

    # 读取原始图像
    image = cv2.imread(input_image_path)
    if image is None:
        print("Error: Image not found or path is incorrect.")
        exit()

    # 显示原始图像
    # cv2.imshow('Original Image', image)
    # cv2.waitKey(0)
```

```
    # 进行高斯模糊处理以减少噪声
    blurred = cv2.GaussianBlur(image, (5, 5), 0)  # 增加内核大小
    # cv2.imshow('Blurred Image', blurred)  # 显示模糊后的图像
    # cv2.waitKey(0)

    # 转换为灰度图像
    gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
    # cv2.imshow('Gray Image', gray)  # 显示灰度图像
    # cv2.waitKey(0)

    # 使用 Canny 边缘检测
    edges = cv2.Canny(gray, 30, 200)  # 调整阈值以增强边缘检测
    # cv2.imshow('Edges', edges)  # 显示边缘检测结果
    # cv2.waitKey(0)

    # 使用形态学操作（膨胀和闭运算）来连接轮廓
    kernel = np.ones((6, 6), np.uint8)  # 增加结构元素的大小
    dilated = cv2.dilate(edges, kernel, iterations=2)
    morph = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE, kernel)
    # cv2.imshow('Morphological Result', morph)  # 显示形态学处理后的结果
    # cv2.waitKey(0)

    # 进行轮廓检测
    contours, hierarchy = cv2.findContours(morph, cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)
```

轮廓图像都存储在 contours 变量中，我们遍历 contours 并在图像上进行切割即可。

```
for i, contour in enumerate(contours):
    x, y, w, h = cv2.boundingRect(contour)
    area = cv2.contourArea(contour)  # 计算轮廓面积

    print(f"Contour {i}: x={x}, y={y}, w={w}, h={h}, area={area}")
    # 打印轮廓的 (x, y, w, h)

    # 判断轮廓是否有效并保存
    if (h > 100 and w > 30 and 900000 > area > 2000 and abs(h - w) <= 400):
        cut_image = image[y:y + h, x:x + w]
```

实现了对一张特定图像的处之后，我们就可以使用 os 库的 walk 方法遍历所有的图像并进行处理。对于每一个图像，再创建一个文件夹存储所有的小分割图像。（要注意 func 文件中的 *create_dir* 方法会无端地创建许多空文件夹，因此我又写了一个 *remove_empty_dirs* 方法来删除这些空文件夹）。

在 *segmentation_display* 文件中运行玩 10-21 行的小分割代码后，需要单独再运行一次 23 行的 *remove_empty_dirs* 方法，以删除所有无关文件夹。

在小分割结束后我们可以得到如下的文件夹结构：

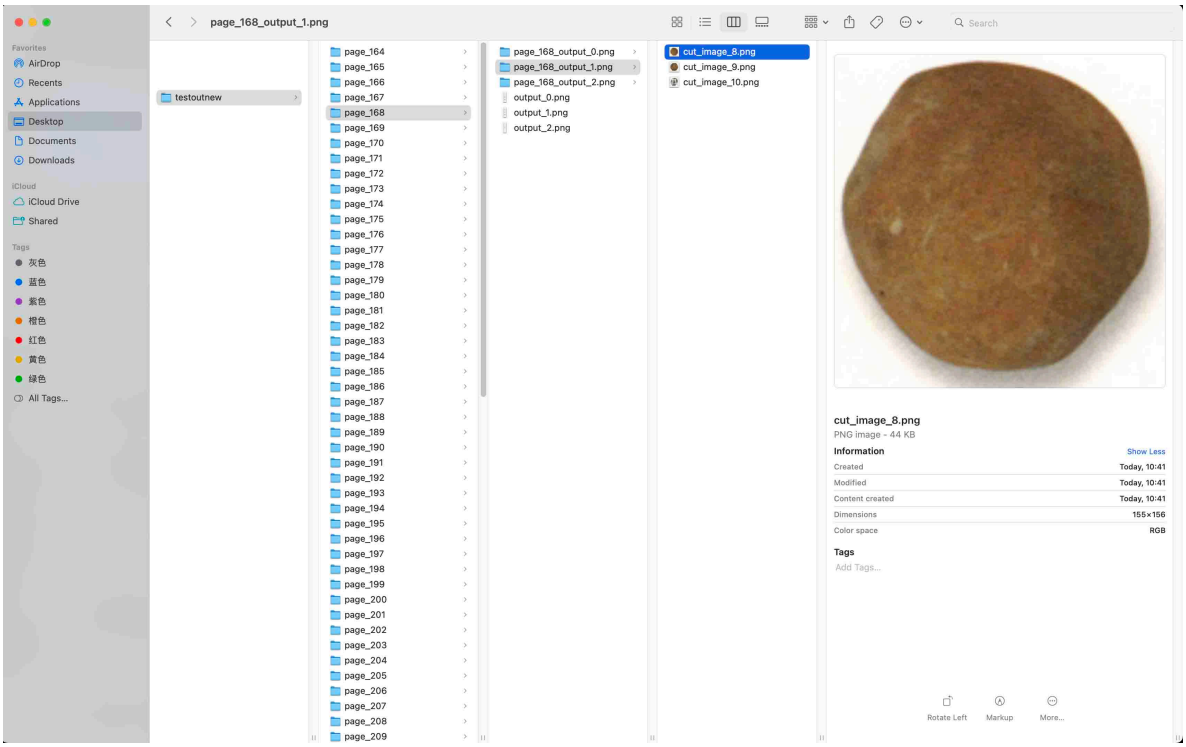

图 17: 小分割处理后文件夹

（五） 文字识别

我们首先尝试了 *pytesseract* 库，但是效果不好。最后我们使用了对中文识别更擅长的 *paddleocr* 库，下面展示几个识别的结果：

图 18: 00007，鄉枳璽，傅淄，《考略》《中國》，東博

图 19: 00019，(縣) 西卿 (鄉)，不详，《大系》，私人

图 20: 00043，成陵市出鉢，傅新蔡，《陶殿》，古陶

图 21: 00031，易口口，傅臨淄，《選粹》，鑿印

图片下方的文字即识别的结果，除少部分拼凑字以及像素不高的字体外，识别效果还是相当不错的。可以注意到，我们已经使用正则表达式去分开了识别的结果，将其分为了数字、地名、作者、书名、出处等等。这样可以更方便地导入到 Excel 表格中。

四、还需解决的问题

1. **小分割**
   - 对于处理图像的参数和阈值还需要优化。现在对于某些图像会出现重复切割或者重复出现的情况；
2. **文字识别**
   - 是否可以先对所有图像增加锐度和清晰度，以提高识别的准确率?
   - 对于拼凑字是否有什么方法识别？或者直接截图?
   - 还没有对文字识别进行批量处理。
3. **导出**
   - 现在使用 csv.writer 导出会出现乱码，是否为编码问题?
   - 需要对前文所述的所有生成文件进行整合，还需提取文件夹中关于页码的信息。
4. **拓展性**
   - 对于陶文目录的处理，处理流程是否还相同？是否可以跳过大分割直接使用小分割?

五、文件结构

```
任务一.zip
├── 1. 大分割
│   ├── main.py                          所有的方法定义与实现
│   ├── func.py                          主程序，在这里运行文件实现大分割
│   ├── PDF_Iteration.py                 用于遍历 pdf 文件，导出图像
│   └── Photoshop_Simulation.py          失败的模仿 Photoshop 调色文件
├── 2. 小分割
│   ├── func.py                          所有的方法定义与实现
│   └── segmentation_display.py          主程序，在这里运行文件实现小分割
└── 3. 文字识别
    └── ocr_test.py                      用于测试文字识别
```