从扫描文档到结构化 Markdown

杨子凡

Jun 16, 2025

纸质文档数字化面临诸多挑战，包括文本不可搜索、编辑困难以及格式混乱等问题。OCR 技术结合 Markdown 转换，能生成可搜索、轻量级且版本可控的结构化文本。本文将通过手把手实战指南，实现扫描件到高精度 Markdown 的自动化流水线，覆盖从理论到实践的完整流程。

1 核心工具与技术栈

OCR 引擎选型需综合考虑精度、速度和多语言支持。本地化方案中，Tesseract 以其开源特性和成熟生态著称，而 PaddleOCR 在中文识别和速度优化上表现优异；云服务方案如 Google Vision 和 Amazon Textract 在复杂表格处理方面具有明显优势。Markdown 生成工具方面，Pandoc 作为格式转换神器，支持多种文档格式互转；自定义 Python 脚本则通过正则表达式和文本处理库实现灵活控制。辅助工具链包括图像预处理的 OpenCV，用于去噪、倾斜校正和二值化等操作；工作流自动化可通过 Python 或 Bash 脚本实现，提升处理效率。

2 四步核心实战流程

2.1 Step 1：文档扫描与预处理

文档扫描是 OCR 精度的基石。扫描时需设置分辨率不低于 300dpi，并控制光照均匀性以避免阴影干扰。图像预处理使用 OpenCV 实现四步法：灰度化降低计算复杂度，二值化增强文本对比度，倾斜校正确保文本对齐。以下 Python 代码展示了核心流程：

```
import cv2
img = cv2.imread("scan.jpg") # 读取扫描图像文件
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 将彩色图像转换为灰度图像，减少通道数
img_thresh = cv2.threshold(img_gray, 0, 255, cv2.THRESH_OTSU)[1] # 应用大津算法自动阈值二值化
img_deskew = deskew(img_thresh) # 调用自定义函数校正图像倾斜角度
```

代码解读：`cv2.imread` 加载图像；`cv2.cvtColor` 的 `COLOR_BGR2GRAY` 参数指定灰度转换；`cv2.threshold` 中 `THRESH_OTSU` 实现自适应二值化；`deskew` 是需自定义的函数，用于旋转图像至水平。常见问题如阴影消除可通过直方图均衡化处理，手指遮挡需重扫，曲面变形则用透视变换校正。

2.2 Step 2：OCR 文本提取进阶

Tesseract 5.0 提供高效文本提取。以下 Bash 命令配置多语言混合识别：

```
tesseract scan.jpg output -l eng+chi_sim --psm 1 --oem 3 pdf
```

代码解读：`tesseract` 调用引擎；`scan.jpg` 是输入文件；`output` 指定输出前缀；`-l eng+chi_sim` 启用英文和简体中文识别；`--psm 1` 设置页面分割模式为自动分析；`--oem 3` 选择基于 LSTM 的 OCR 引擎；`pdf` 生成 PDF 格式结果。表格提取需专项处理：Amazon Textract 解析坐标输出结构化表格；PaddleOCR 可直接生成 HTML 表格，通过坐标映射保留布局。

2.3 Step 3：从纯文本到结构化 Markdown

标题层级识别基于规则引擎分析字体大小、位置和加粗特征。机器学习方案如 BERT 文本分类需标注数据训练。列表与段落处理使用正则表达式，例如识别有序列表：

```
re.sub(r'(\d+)\.\s+(.*)', r'\1.␣\2', text)
```

代码解读：`re.sub` 执行正则替换；模式 `r'(\d+)\.\s+(.*)'` 匹配数字加点号的列表项；`r'\1. \2'` 重组格式确保空格规范。复杂元素转换中，Pandoc 处理表格：`pandoc -s output.html -t markdown -o table.md` 将 HTML 转为 Markdown；公式保留使用 LaTeX 片段，如行内公式 $E = mc^2$，块公式：

$$\int_a^b f(x)dx$$

Katex 集成方案确保渲染兼容性。

2.4 Step 4：Markdown 增强与校验

语法标准化统一标题符号（如用 # 替代 ===）并自动添加代码块语言标识符。视觉还原度提升涉及 Mermaid 图表生成，将文本描述转为流程图；图片嵌入优化语法 `![alt-text](image.jpg){width=80%}` 控制显示比例。质量验证使用 diffchecker.com 对比原始 PDF，markdownlint 检查语法规范，确保输出无误。

3 高级技巧与自动化

批量处理脚本通过 Python 实现全流程自动化。以下示例遍历扫描目录：

```
for img_path in scan_dir:
    preprocess(img_path) # 调用预处理函数
    text = ocr(img_path) # 执行 OCR 提取文本
    md = convert_to_md(text) # 转换为 Markdown
    postprocess(md) # 后处理如添加 YAML 头元数据
```

代码解读：循环处理 `scan_dir` 中每个图像；`preprocess` 封装 OpenCV 操作；`ocr` 调用 Tesseract 或 PaddleOCR；`convert_to_md` 实现正则转换；`postprocess` 添加文档元数据。Docker 化部署预构建镜像包

含 Tesseract、PaddleOCR 和 Python 环境；Git 集成版本化文档变更，可视化 Markdown diff 跟踪修改历史。

4 典型场景应用案例

学术论文转换需保留公式，Mathpix API 可识别 LaTeX 片段；参考文献编号通过正则表达式处理。企业合同处理中，关键条款用 **高亮** 标记，签署位置添加自定义标签如 [signature_block]。古籍数字化项目支持竖排文本识别，配置异体字映射表处理历史字符变体。

5 效能评估与优化方向

精度指标 CER（字符错误率）控制在 3% 内，通过调整 OCR 参数和预处理优化实现；表格结构还原率基于 IoU（交并比）评估。速度优化利用 GPU 加速，如 CUDA 版 Tesseract；分布式 OCR 集群处理海量文档。成本对比显示自建方案每页成本低于云服务，但需权衡硬件投入。

当前技术边界限制手写体和复杂排版识别，但 LLM 在文档理解中展现新应用潜力。推荐资源包括开源项目 OCRopy 和 Unstructured.io，数据集如 SROIE 票据识别数据集。持续优化将推动文档数字化进入新阶段。