

# 第 4 周：试卷版面分析

## 怎么知道选择题、简答题在哪里？

北京石油化工学院\人工智能研究院\王文通

通选课

2025-2026 学年

# 课程概览

- 1 版面分析概述
- 2 边缘检测
- 3 轮廓检测
- 4 形状特征
- 5 区域定位
- 6 思考题
- 7 课后作业
- 8 下节预告

# 什么是版面分析？

## 定义

从文档图像中识别和定位不同区域（标题、正文、表格、图片等）

## 在阅卷系统中的作用：

- ① 找到试卷边界（定位试卷）
- ② 定位选择题区域（OMR 识别）
- ③ 定位判断题区域（符号匹配）
- ④ 定位简答题区域（手写识别）
- ⑤ 定位填空题区域（内容提取）

# 为什么需要版面分析?

## 没有版面分析:

- 不知道答题卡在哪
- 无法区分题型
- OCR 识别范围过大
- 处理效率低下

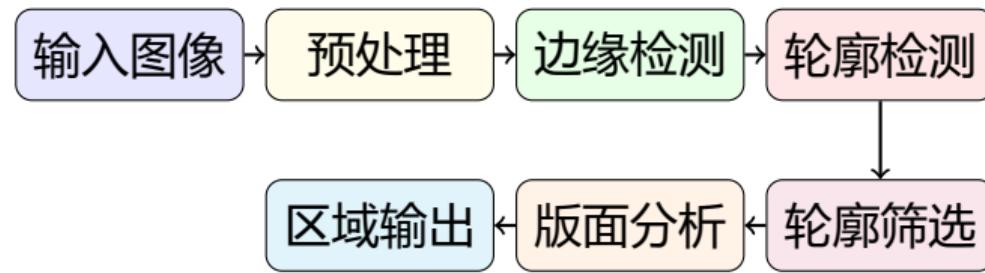
## 有了版面分析:

- 精确定位各区域
- 分类处理不同题型
- 提高识别准确率
- 加快处理速度

## 核心价值

版面分析是自动阅卷的“地图导航”!

# 版面分析完整流程



# 版面分析方法分类

## 传统方法：

- **边缘检测**: Canny、Sobel
- **轮廓检测**: findContours
- **投影法**: 水平/垂直投影
- **连通域分析**: blob 检测

## 深度学习方法：

- **目标检测**: YOLO、Faster R-CNN
- **语义分割**: U-Net、DeepLab
- **版面分析模型**: LayoutLM、DocFormer

## 本课程重点

传统方法 (简单、高效、可控)

# 应用场景

## 教育阅卷

- 答题卡识别
- 试卷自动批改
- 成绩统计

## 文档处理

- 发票识别
- 表单提取
- 合同分析

## 档案数字化

- 版面重建
- 区域提取
- 内容索引

# 边缘检测基础

## 什么是边缘？

- 图像中像素值发生剧烈变化的位置
- 反映了物体边界、纹理变化等信息
- 是图像的重要特征

## 边缘类型：



阶跃边缘



斜坡边缘

# 梯度与边缘

梯度原理：

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

梯度幅值：

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

梯度方向：

$$\theta = \arctan\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

核心思想

边缘处梯度幅值最大！

# Sobel 算子

水平方向卷积核 (检测垂直边缘):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

垂直方向卷积核 (检测水平边缘):

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

```
# Sobel 边缘检测
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
sobel = np.sqrt(sobel_x**2 + sobel_y**2)
```

# Canny 边缘检测

## 为什么选择 Canny?

- 最优边缘检测算法 (1986 年提出)
- 检测准确率高 (低误检率)
- 定位精确 (边缘位置准确)
- 单边缘响应 (每个边缘只有一条响应线)

## Canny 算法步骤:

- ① 高斯滤波降噪
- ② 计算梯度幅值和方向
- ③ 非极大值抑制
- ④ 双阈值检测和边缘连接

# Canny 算法详解 (1/4)

## 步骤 1：高斯滤波

- 去除图像噪声
- 防止噪声被误认为边缘
- 通常使用  $5 \times 5$  高斯核

## 步骤 2：计算梯度

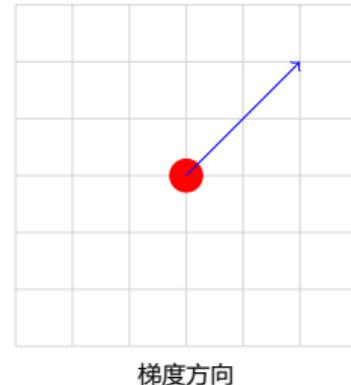
- 使用 Sobel 算子计算梯度
- 得到梯度幅值和方向
- 梯度方向垂直于边缘方向

## 步骤 3：非极大值抑制

**目的：**将粗边缘变成细边缘（单像素宽）

**方法：**

- ① 在梯度方向上比较当前像素
- ② 如果不是局部最大值，则抑制（置 0）
- ③ 保留梯度方向上的最大值点



# Canny 算法详解 (3/4)

## 步骤 4：双阈值检测

两个阈值：

- **高阈值** (threshold2)：强边缘
- **低阈值** (threshold1)：弱边缘

分类：

- 梯度  $>$  高阈值  $\rightarrow$  强边缘
- 低阈值  $<$  梯度  $<$  高阈值  $\rightarrow$  弱边缘
- 梯度  $<$  低阈值  $\rightarrow$  非边缘

边缘连接：

- 强边缘直接保留
- 弱边缘如果与强边缘相连则保留
- 孤立的弱边缘被抑制

推荐比例

$$\text{高阈值} = \text{低阈值} \times 2.3$$

# Canny 代码实现

```
import cv2
import numpy as np

# 预处理
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)

# Canny 边缘检测
edges = cv2.Canny(
    blur,
    50,           # 输入图像
    150           # 低阈值 threshold1
    )             # 高阈值 threshold2

# 可视化
cv2.imshow('Edges', edges)
cv2.waitKey(0)
```

# Canny 参数调优

参数影响：

参数	调整方向	效果
低阈值	提高	减少噪声，但可能丢失弱边缘
低阈值	降低	检测更多边缘，但噪声增多
高阈值	提高	只保留强边缘，边缘更少
高阈值	降低	保留更多边缘，可能引入噪声

试卷处理推荐值：

- 低阈值：50-100
- 高阈值：150-200
- 可根据图像质量调整

# 自动阈值选择

问题：手动调参太繁琐

解决方案：基于图像中值自动计算

```
def auto_canny(gray, sigma=0.33):
    """
    自动计算 Canny 阈值
    sigma: 控制阈值范围, 通常 0.33
    """
    # 计算图像中值
    v = np.median(gray)

    # 计算阈值
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))

    # 应用 Canny
    return cv2.Canny(gray, lower, upper)

# 使用
edges = auto_canny(blur)
```

# 轮廓 vs 边缘

## 边缘 (Edges)

- 不连续的像素集合
- 梯度变化的位置
- 原始数据
- 来自 Canny/Sobel

## 轮廓 (Contours)

- 连续的边界曲线
- 封闭的、有序的
- 可用于形状分析
- 来自 findContours

关系：边缘  $\xrightarrow{\text{连接}}$  轮廓

边缘图像  $\rightarrow$  轮廓检测  $\rightarrow$  轮廓列表

# 轮廓检测流程

- ① **输入**: 二值图像 (通常来自 Canny 或 threshold)
- ② **扫描**: 遍历图像, 寻找边界
- ③ **连接**: 将相邻边缘点连接成轮廓
- ④ **层级**: 建立轮廓之间的层级关系
- ⑤ **输出**: 轮廓点集 + 层级结构

## 注意

`findContours` 会修改输入图像! 如果需要保留, 请先复制。

# findContours API

## 函数原型：

```
contours, hierarchy = cv2.findContours(  
    image,           # 输入（二值图，最好是白底黑字）  
    mode,            # 轮廓检索模式  
    method          # 轮廓逼近方法  
)
```

## 轮廓检索模式 (mode)：

模式	说明
cv2.RETR_EXTERNAL	只检测最外层轮廓
cv2.RETR_LIST	检测所有轮廓，不建立层级
cv2.RETR_CCOMP	检测所有轮廓，建立两级层级
cv2.RETR_TREE	检测所有轮廓，建立完整层级树

# 轮廓逼近方法 (method)

存储方式:

方法	说明
cv2.CHAIN_APPROX_NONE	存储所有边界点
cv2.CHAIN_APPROX_SIMPLE	压缩水平、垂直、对角线段
cv2.CHAIN_APPROX_TC89_L1	Teh-Chin 链逼近算法

```
# 常用配置
contours, hierarchy = cv2.findContours(
    binary,
    cv2.RETR_EXTERNAL,      # 只要外层轮廓
    cv2.CHAIN_APPROX_SIMPLE # 压缩存储，节省内存
)
```

**推荐:** RETR\_EXTERNAL + CHAIN\_APPROX\_SIMPLE

# 层级结构 (hierarchy)

## 什么是层级?

- 轮廓之间的包含关系
- 外轮廓 vs 内轮廓 (孔洞)
- 用树形结构表示

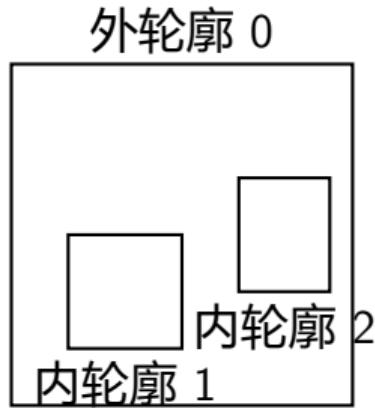
## hierarchy 数组格式:

$[next, previous, first\_child, parent]$

- **next**: 同级下一个轮廓
- **previous**: 同级前一个轮廓
- **first\_child**: 第一个子轮廓
- **parent**: 父轮廓

# 层级结构示例

hierarchy 示例:



next	prev	child	parent
1	-1	2	-1
2	0	-1	0
-1	1	-1	0

- 轮廓 0: 外轮廓 (无父级)
- 轮廓 1: 内轮廓 (父级 =0)
- 轮廓 2: 内轮廓 (父级 =0)

# 绘制轮廓

## drawContours 函数：

```
# 绘制所有轮廓
output = cv2.drawContours(
    image.copy(),      # 目标图像（会修改原图）
    contours,          # 轮廓列表
    -1,                # -1=绘制所有, 0=绘制第一个
    (0, 255, 0),       # 颜色 BGR
    2                  # 线宽
)

# 只绘制特定轮廓
output = cv2.drawContours(
    image.copy(), contours, 0, (0, 0, 255), 3
)
```

## 填充轮廓：

```
# 线宽设为 -1 表示填充
output = cv2.drawContours(
    image.copy(), contours, -1, (0, 255, 0), -1
)
```

# 轮廓检测完整示例

```
import cv2
import numpy as np

# 读取图像
img = cv2.imread('exam.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 预处理
blur = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(blur, 50, 150)

# 查找轮廓
contours, hierarchy = cv2.findContours(
    edges,
    cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE
)

# 绘制轮廓
output = img.copy()
cv2.drawContours(output, contours, -1, (0, 255, 0), 2)

print(f"检测到 {len(contours)} 个轮廓")
cv2.imshow('Contours', output)
cv2.waitKey(0)
```

# 轮廓筛选技巧

## 按面积筛选：

- 排除过小的轮廓（噪声）
- 排除过大的轮廓（背景）

## 按位置筛选：

- 只保留图像中心区域
- 排除边缘区域的轮廓

## 按形状筛选：

- 按长宽比筛选
- 按纵横比筛选
- 按凸包面积比筛选

# 轮廓特征

## 常用几何特征：

特征	说明	OpenCV 函数
面积	轮廓所围区域大小	cv2.contourArea()
周长	轮廓长度	cv2.arcLength()
边界矩形	外接矩形	cv2.boundingRect()
最小外接矩形	旋转矩形	cv2.minAreaRect()
最小外接圆	外接圆	cv2.minEnclosingCircle()
凸包	最小凸多边形	cv2.convexHull()
凸缺陷	凹陷部分	cv2.convexityDefects()

# 面积与周长

```
# 计算面积
area = cv2.contourArea(contour)

# 计算周长
perimeter = cv2.arcLength(contour, True) # True=封闭

# 面积比（用于筛选）
image_area = img.shape[0] * img.shape[1]
area_ratio = area / image_area

# 筛选：保留面积占图像 10%-80% 的轮廓
if 0.1 < area_ratio < 0.8:
    # 保留该轮廓
    pass
```

## 应用：根据面积筛选目标轮廓

# 边界矩形

## 直立边界矩形（不考虑旋转）：

```
x, y, w, h = cv2.boundingRect(contour)

# 绘制矩形
cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

## 最小外接矩形（考虑旋转）：

```
rect = cv2.minAreaRect(contour)
box = cv2.boxPoints(rect)
box = np.int0(box)

# 绘制旋转矩形
cv2.drawContours(img, [box], 0, (0, 0, 255), 2)
```

# 多边形逼近

**原理：**用较少的点逼近轮廓形状

**approxPolyDP 函数：**

$$\epsilon = \text{precision} \times \text{perimeter}$$

- **precision**: 逼近精度 (0.01-0.05)
- 值越小，逼近越精确（点越多）
- 值越大，逼近越粗糙（点越少）

**应用：**判断轮廓形状

- 4 个点 → 四边形（试卷）
- 3 个点 → 三角形
- 点很多 → 圆形

# 多边形逼近代码

```
# 计算周长
peri = cv2.arcLength(contour, True)

# 多边形逼近
approx = cv2.approxPolyDP(
    contour,          # 输入轮廓
    0.02 * peri,     # 精度参数
    True              # 轮廓是否封闭
)

# 获取顶点数
num_vertices = len(approx)

print(f"顶点数: {num_vertices}")

# 绘制逼近结果
cv2.drawContours(img, [approx], 0, (0, 255, 0), 2)
```

# 凸包与凸缺陷

## 凸包 (Convex Hull):

- 包含轮廓的最小凸多边形
- 类似于“橡皮筋”包住轮廓

## 凸缺陷 (Convexity Defects):

- 轮廓与凸包之间的凹陷
- 用于手势识别、形状分析

```
计算凸缺 hull_idx = cv2.convexHull(contour, returnPoints = False)  
defects = cv2.convexityDefects(contour, hull_idx) 陷
```

# 找到试卷轮廓

```
def find_paper_contour(contours, image_area):
    """
    从轮廓中找到试卷
    """
    for contour in contours:
        area = cv2.contourArea(contour)

        # 面积筛选：应该是图像的一定比例
        if area > image_area * 0.5:
            # 计算周长
            peri = cv2.arcLength(contour, True)

            # 多边形逼近
            approx = cv2.approxPolyDP(contour, 0.02 * peri, True)

            # 如果是四边形
            if len(approx) == 4:
                return approx

    return None

# 使用
image_area = gray.shape[0] * gray.shape[1]
paper_contour = find_paper_contour(contours, image_area)
```

# 形状匹配

## Hu 矩:

- 具有旋转、缩放、平移不变性
- 用于形状相似度比较

```
形状匹 match = cv2.matchShapes( contour1, contour2, cv2.CONTOURS_MATCH_I1, 0.0) 配
```

## 应用：识别标准形状（圆形、方形等）

# 投影法原理

## 什么是投影？

- 统计图像在某个方向上的像素分布
- **水平投影**: 统计每行的白色像素数
- **垂直投影**: 统计每列的白色像素数

## 水平投影应用:

- 检测文本行分隔
- 找到题目之间的空白
- 切割试卷区域

## 垂直投影应用:

- 检测字符分隔
- 找到列边界
- 对齐文本

# 水平投影

```
def horizontal_projection(binary):
    """
    水平投影：统计每行的白色像素数
    binary: 二值图像 (0=黑, 255=白)
    """
    # 将图像归一化到 0-1
    binary = binary // 255

    # 沿水平方向求和 (每行的白色像素数)
    proj = np.sum(binary, axis=1)

    return proj

# 使用
binary = cv2.adaptiveThreshold(gray, 255, ...)
h_proj = horizontal_projection(binary)
```

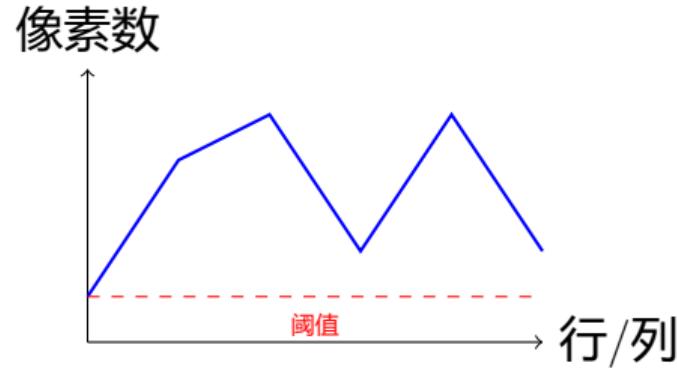
# 垂直投影

```
def vertical_projection(binary):
    """
    垂直投影：统计每列的白色像素数
    """
    binary = binary // 255
    proj = np.sum(binary, axis=0)
    return proj

# 使用
v_proj = vertical_projection(binary)

# 可视化
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 4))
plt.plot(v_proj)
plt.title('Vertical Projection')
plt.show()
```

# 投影波形分析



分隔线检测：找投影值低于阈值的波谷位置

## 波峰 (Peak):

- 像素数多的位置
- 对应文本/图形区域

## 波谷 (Valley):

- 像素数少的位置
- 对应空白/分隔

# 找到分隔线

```
def find_divider_lines(proj, threshold=10, min_length=5):
    """
    找到分隔线（波谷）
    proj: 投影数组
    threshold: 波谷阈值
    min_length: 最小连续长度（过滤噪声）
    """
    dividers = []
    current_start = None

    for i, value in enumerate(proj):
        if value < threshold:
            if current_start is None:
                current_start = i
            else:
                if current_start is not None:
                    length = i - current_start
                    if length >= min_length:
                        dividers.append((current_start, i))
                    current_start = None
    return dividers
```

# 投影法实战

```
def layout_analysis(binary):
    """
    版面分析：找到题目分隔线
    """
    # 水平投影
    h_proj = np.sum(binary // 255, axis=1)

    # 找分隔线
    threshold = np.mean(h_proj) * 0.1  # 自适应阈值
    dividers = find_divider_lines(h_proj, threshold)

    # 计算题目区域
    regions = []
    prev_end = 0
    for start, end in dividers:
        if start > prev_end:
            regions.append((prev_end, start))
        prev_end = end

    # 最后一个区域
    if prev_end < binary.shape[0]:
        regions.append((prev_end, binary.shape[0]))

    return regions

# 使用
regions = layout_analysis(binary)
print(f"检测到 {len(regions)} 个区域")
```

# 连通域分析

## 什么是连通域？

- 相邻的相同像素值构成的区域
- 4 连通：上下左右
- 8 连通：包括对角线

## 应用场景：

- 识别填涂区域（选择题气泡）
- 分离独立字符
- 去除噪声斑点

## 下周预告

下周将用连通域分析来识别选择题填涂！

# 连通域检测

```
# 连通域检测
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(
    binary,          # 输入（二值图， 0=背景， >0=前景）
    connectivity=8   # 8连通
)

# stats 格式: [x, y, width, height, area]
# centroids 格式: [cx, cy]

# 遍历所有连通域（跳过背景 label=0）
for i in range(1, num_labels):
    x, y, w, h, area = stats[i]
    cx, cy = centroids[i]

    # 筛选: 面积在一定范围内
    if 100 < area < 10000:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

# 区域定位完整流程

- ① **输入**: 预处理后的二值图像
- ② **轮廓检测**: 找到主要区域边界
- ③ **投影分析**: 定位水平/垂直分隔
- ④ **连通域分析**: 识别填涂区域
- ⑤ **区域标注**: 绘制识别结果

## 组合策略

轮廓检测 + 投影法 + 连通域分析 = 完整版面分析

# 课堂思考题

## 问题 1：边缘检测

- Canny 的双阈值如何选择？
- 如果阈值设置不当会怎样？
- 如何自动确定最优阈值？

## 问题 2：轮廓检测

- 如何从多个轮廓中找到试卷轮廓？
- 如果试卷边缘有破损，怎么办？
- RETR\_EXTERNAL 和 RETR\_TREE 有什么区别？

# 课堂思考题 (续)

## 问题 3：区域定位

- 投影法检测分隔线的原理是什么？
- 如何处理不同版式的试卷？
- 连通域分析在选择题识别中的作用？

## 问题 4：综合应用

- 如何设计一个通用的试卷版面分析系统？
- 如何处理倾斜的试卷？
- 如何处理有多栏的试卷？

# 课后作业

## 题目

实现试卷版面分析，标注三种题型区域

## 要求：

- ① 使用 Canny 边缘检测找到试卷边界
- ② 使用 findContours 检测轮廓
- ③ 使用投影法分析试卷结构
- ④ 标注出选择题、判断题、简答题区域
- ⑤ 提交标注结果的可视化图像

## 评分标准：

- 边缘检测：20 分
- 轮廓检测：20 分
- 区域定位：30 分
- 可视化效果：20 分

# 作业提交要求

## 提交内容：

- ① Python 源代码 (.py 或.ipynb)
- ② 测试图像 (原图 + 标注结果)
- ③ 实验报告 (PDF)

## 实验报告包含：

- 算法流程说明
- 参数选择理由
- 遇到的问题与解决方案
- 处理结果展示

**截止时间：**下次上课前

**提交方式：**教学平台上传

## 第 5 周：选择题识别（填涂检测）

故事问题：怎么知道选了 A 还是 B？

你将学会：

- OMR 光学标记识别原理
- 填涂密度计算方法
- 选择题自动识别流程
- 多选项处理逻辑

# 延伸学习资源

## 推荐阅读：

- OpenCV 官方文档 - Contours 章节
- 《数字图像处理》冈萨雷斯 - 边缘检测章节
- 论文：Canny, J. (1986). "A Computational Approach to Edge Detection"

## 实践项目：

- 答题卡识别系统
- 发票表格提取
- 文档版面重建

## 在线资源：

- OpenCV Python Tutorials
- LearnOpenCV 网站教程

# 知识点总结

## 核心方法回顾：

步骤	方法	关键函数
边缘检测	Canny	cv2.Canny()
轮廓检测	findContours	cv2.findContours()
形状分析	多边形逼近	cv2.approxPolyDP()
区域定位	投影法	np.sum(axis=1/0)
细节识别	连通域	cv2.connectedComponents()

## 关键参数：

- Canny 阈值：(50, 150)
- 逼近精度： $0.02 \times \text{周长}$
- 面积筛选：图像面积的 10

# 谢谢！

有问题随时交流

邮箱: [wwtong@bipt.edu.cn](mailto:wwtong@bipt.edu.cn)