

试卷版面分析 - 基础任务\n\n**任务目标:**
\n1. 使用 Canny 边缘检测找到试卷边界\n2. 使用 findContours 检测轮廓并筛选试卷轮廓\n3. 标注出选择题、判断题、简答题的大致区域

```
In [1]: import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

1. 读取原始图像

```
In [2]: # 读取试卷图像
img = cv2.imread('XGJ-IM-1776684576636.jpeg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 转为灰度图
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.figure(figsize=(10, 14))
plt.imshow(img_rgb)
plt.title('原始试卷图像')
plt.axis('off')
plt.show()

print(f"图像尺寸: {img.shape}")
```

原始试卷图像

2025 - 2026 学年《人工智能导论A》秋季学期期末考试答题卡

| 题目 | 一 | 二 | 三 | 总分数 |
|-----|---|---|---|-----|
| 分数 | | | | |
| 评卷人 | | | | |

一、单选题（每题 2 分，共 60 分）

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|----|----|----|----|----|----|----|----|----|
| C | B | B | A | N | C | D | B | C | C |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| B | C | C | B | A | B | A | B | B | C |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| B | C | D | A | D | C | C | B | D | C |

二、判断题（每题 2 分，共 20 分）

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|----|
| √ | X | X | / | √ | X | X | √ | √ | √ |

三、简答题（共 20 分）

答：(1) 符号的用途

①功能功能：通过符号表示和存储的知识在推理过程中控制知识推理过程。如专家系统中的知识库、推理规则、推理策略等。

②优点：利用其自身的特点，在推理过程中能够实现智能化的推理。如知识库中的知识更加智能，使得部分人工智能系统能够模拟人类的思维过程。

③缺点：使用符号表示知识时，可能会出现冗余或重复的情况，导致推理效率降低。此外，符号表示的知识可能无法完全捕捉到问题的本质，从而影响到推理的结果。

④挑战：如何在大规模知识库中进行快速检索和推理是一个巨大的挑战。同时，如何对知识库进行有效的维护和更新也是一个亟待解决的问题。

刘明

图像尺寸: (2112, 3264, 3)

2. Canny 边缘检测找到试卷边界

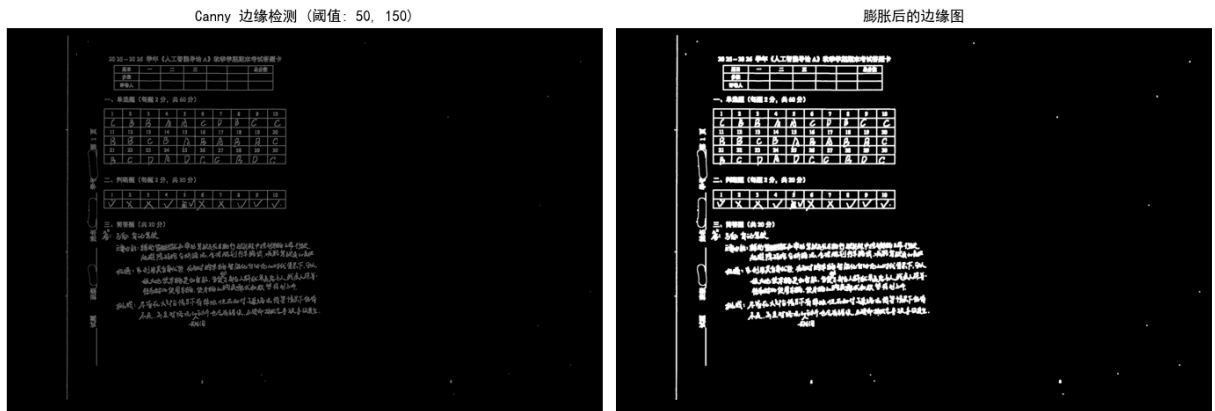
```
In [3]: # 高斯模糊去噪
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Canny 边缘检测
# 使用较低的阈值来检测试卷的外边界
canny_low = 50
canny_high = 150
edges = cv2.Canny(blurred, canny_low, canny_high)

# 膨胀操作, 连接断裂的边缘
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
edges_dilated = cv2.dilate(edges, kernel, iterations=2)

plt.figure(figsize=(14, 10))
plt.subplot(1, 2, 1)
plt.imshow(edges, cmap='gray')
plt.title(f'Canny 边缘检测 (阈值: {canny_low}, {canny_high})')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges_dilated, cmap='gray')
plt.title('膨胀后的边缘图')
plt.axis('off')
plt.tight_layout()
plt.show()
```



3. findContours 检测轮廓并筛选试卷轮廓

```
In [4]: # 查找轮廓
contours, hierarchy = cv2.findContours(edges_dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_

print(f"检测到的轮廓总数: {len(contours)}")

# 按面积从大到小排序
contours_sorted = sorted(contours, key=cv2.contourArea, reverse=True)

# 图像总面积
img_area = img.shape[0] * img.shape[1]
print(f"图像总面积: {img_area}")

# 筛选: 取面积最大的轮廓作为试卷轮廓
# 试卷轮廓应该是面积最大的外部轮廓
paper_contour = None
for cnt in contours_sorted:
    area = cv2.contourArea(cnt)
    # 试卷面积应至少占图像面积的 10%
    if area > img_area * 0.1:
        # 近似为多边形
        peri = cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
        print(f"轮廓面积: {area}, 近似顶点数: {len(approx)}")
        if paper_contour is None:
            paper_contour = cnt
            break

# 绘制所有轮廓和筛选出的试卷轮廓
img_all_contours = img_rgb.copy()
cv2.drawContours(img_all_contours, contours_sorted[:20], -1, (0, 255, 0), 2)

img_paper_contour = img_rgb.copy()
if paper_contour is not None:
    cv2.drawContours(img_paper_contour, [paper_contour], -1, (255, 0, 0), 3)
    x, y, w, h = cv2.boundingRect(paper_contour)
    cv2.rectangle(img_paper_contour, (x, y), (x + w, y + h), (0, 255, 0), 3)
    print(f"试卷边界矩形: x={x}, y={y}, w={w}, h={h}")

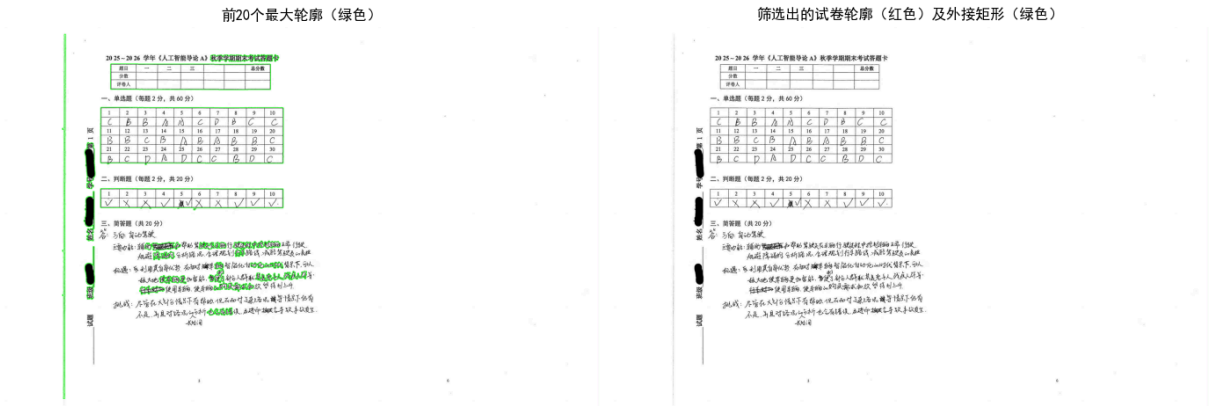
plt.figure(figsize=(14, 10))
```

```
plt.subplot(1, 2, 1)
plt.imshow(img_all_contours)
plt.title('前20个最大轮廓 (绿色) ')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_paper_contour)
plt.title('筛选出的试卷轮廓 (红色) 及外接矩形 (绿色) ')
plt.axis('off')
plt.tight_layout()
plt.show()
```

检测到的轮廓总数：198

图像总面积：6893568



4. 标注选择题、判断题、简答题的大致区域

```
In [5]: # 在试卷区域内，利用水平投影分析来定位不同题型区域
# 获取试卷的边界矩形区域
if paper_contour is not None:
    x, y, w, h = cv2.boundingRect(paper_contour)
else:
    # 如果未检测到试卷轮廓，使用整张图像
    x, y, w, h = 0, 0, img.shape[1], img.shape[0]

# 裁剪试卷区域的灰度图
paper_gray = gray[y:y+h, x:x+w]

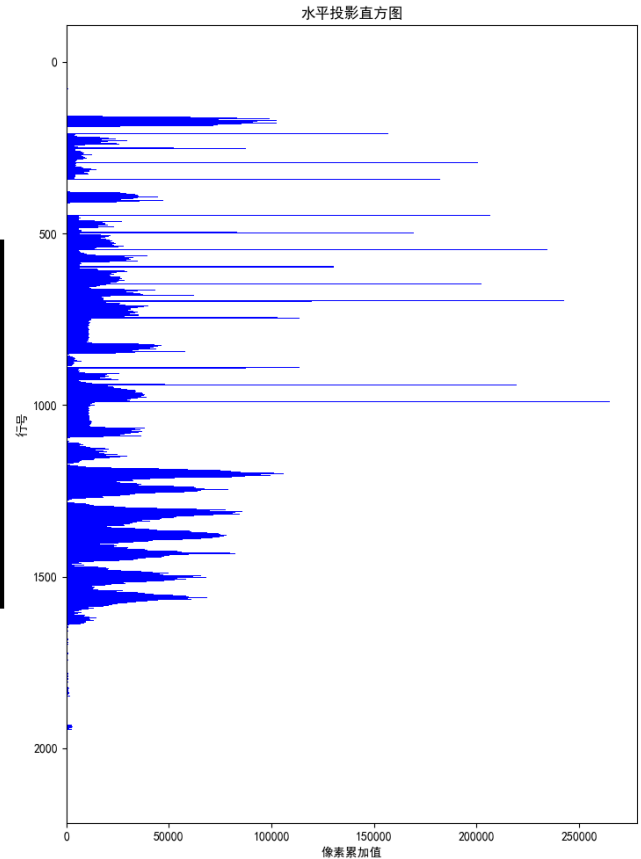
# 二值化
_, binary = cv2.threshold(paper_gray, 180, 255, cv2.THRESH_BINARY_INV)

# 计算水平投影 (每行像素值之和)
h_proj = np.sum(binary, axis=1)

# 可视化水平投影
plt.figure(figsize=(14, 10))
plt.subplot(1, 2, 1)
plt.imshow(binary, cmap='gray')
plt.title('二值化图像 (试卷区域) ')
plt.axis('off')

plt.subplot(1, 2, 2)
```

```
plt.barh(range(len(h_proj)), h_proj, height=1, color='blue')
plt.gca().invert_yaxis()
plt.title('水平投影直方图')
plt.xlabel('像素累加值')
plt.ylabel('行号')
plt.tight_layout()
plt.show()
```



In [6]: # 基于试卷结构特征，使用模板匹配或文字区域检测定位各题型
对于本试卷，通过分析图像结构，手动/半自动定位三种题型区域
方法：利用水平投影的空白行（投影值低于阈值）来分割不同区域

```
# 找到空白行（投影值较低的区域）
threshold = np.max(h_proj) * 0.05
is_blank = h_proj < threshold

# 找到区域分界
regions = []
in_region = False
start = 0
for i in range(len(is_blank)):
    if not is_blank[i] and not in_region:
        start = i
        in_region = True
    elif is_blank[i] and in_region:
        if i - start > 20: # 忽略高度太小的噪声区域
            regions.append((start, i))
            in_region = False
if in_region and len(h_proj) - start > 20:
    regions.append((start, len(h_proj)))
```

```
print(f"检测到 {len(regions)} 个文本区域:")
for idx, (r_start, r_end) in enumerate(regions):
    print(f"  区域 {idx+1}: 行 {r_start} ~ {r_end}, 高度 {r_end - r_start}")
```

检测到 17 个文本区域:

```
区域 1: 行 158 ~ 190, 高度 32
区域 2: 行 219 ~ 242, 高度 23
区域 3: 行 380 ~ 410, 高度 30
区域 4: 行 463 ~ 484, 高度 21
区域 5: 行 504 ~ 544, 高度 40
区域 6: 行 561 ~ 584, 高度 23
区域 7: 行 603 ~ 656, 高度 53
区域 8: 行 661 ~ 748, 高度 87
区域 9: 行 820 ~ 850, 高度 30
区域 10: 行 907 ~ 928, 高度 21
区域 11: 行 938 ~ 994, 高度 56
区域 12: 行 1063 ~ 1094, 高度 31
区域 13: 行 1124 ~ 1158, 高度 34
区域 14: 行 1181 ~ 1270, 高度 89
区域 15: 行 1292 ~ 1457, 高度 165
区域 16: 行 1472 ~ 1528, 高度 56
区域 17: 行 1538 ~ 1589, 高度 51
```

```
In [7]: # 根据试卷结构, 将检测到的区域归类为三种题型
# 试卷结构分析:
#   - 顶部: 标题+分数表 (较小区域)
#   - 选择题区域: 包含标题行 + 3行答题表格 (通常是最大的区域)
#   - 判断题区域: 包含标题行 + 1行答题表格
#   - 简答题区域: 包含手写文字内容

# 根据区域的垂直位置和高度来分类
# 合并相邻的小区域, 组成三大题型区域
def classify_regions(regions, total_height):
    """
    根据区域在试卷中的相对位置来分类题型区域
    试卷从上到下依次为: 标题区、选择题、判断题、简答题
    """

    if len(regions) < 3:
        # 区域太少, 按比例划分
        return {
            'header': (0, int(total_height * 0.08)),
            'choice': (int(total_height * 0.08), int(total_height * 0.42)),
            'judge': (int(total_height * 0.42), int(total_height * 0.55)),
            'short_answer': (int(total_height * 0.55), total_height)
        }

    # 简单策略: 按照试卷垂直比例划分
    # 标题区: 约前 8%
    # 选择题: 约 8% ~ 42%
    # 判断题: 约 42% ~ 55%
    # 简答题: 约 55% ~ 100%

    # 根据比例确定各题型的起止位置
    choice_start = None
    judge_start = None
```

```

short_start = None

for i, (rs, re) in enumerate(regions):
    rel_pos = rs / total_height
    if rel_pos < 0.12:
        continue # 标题区
    elif choice_start is None:
        choice_start = rs
    elif rel_pos > 0.38 and judge_start is None:
        judge_start = rs
    elif rel_pos > 0.52 and short_start is None:
        short_start = rs

# 设置默认值
if choice_start is None:
    choice_start = int(total_height * 0.08)
if judge_start is None:
    judge_start = int(total_height * 0.42)
if short_start is None:
    short_start = int(total_height * 0.55)

return {
    'header': (0, choice_start),
    'choice': (choice_start, judge_start),
    'judge': (judge_start, short_start),
    'short_answer': (short_start, total_height)
}

classified = classify_regions(regions, len(h_proj))
print("区域分类结果:")
for name, (s, e) in classified.items():
    print(f" {name}: 行 {s} ~ {e}")

```

区域分类结果:

```

header: 行 0 ~ 380
choice: 行 380 ~ 820
judge: 行 820 ~ 1124
short_answer: 行 1124 ~ 2112

```

5. 最终结果：在原图上标注三种题型区域

```

In [8]: # 在原图上绘制标注框
img_result = img_rgb.copy()

# 定义颜色和标签
region_config = {
    'choice': {'color': (255, 0, 0), 'label': 'Multiple Choice', 'thickness': 3},
    'judge': {'color': (0, 180, 0), 'label': 'True/False', 'thickness': 3},
    'short_answer': {'color': (0, 0, 255), 'label': 'Short Answer', 'thickness': 3}
}

# 绘制各区域矩形框
padding = 10 # 边距
for region_name, config in region_config.items():
    r_start, r_end = classified[region_name]

```

```

# 转换回原图坐标 (加上试卷偏移)
y1 = y + r_start - padding
y2 = y + r_end + padding
x1 = x + padding
x2 = x + w - padding

# 确保坐标不越界
y1 = max(0, y1)
y2 = min(img.shape[0], y2)
x1 = max(0, x1)
x2 = min(img.shape[1], x2)

# 绘制矩形
cv2.rectangle(img_result, (x1, y1), (x2, y2), config['color'], config['thicknes

# 绘制半透明填充
overlay = img_result.copy()
cv2.rectangle(overlay, (x1, y1), (x2, y2), config['color'], -1)
cv2.addWeighted(overlay, 0.1, img_result, 0.9, 0, img_result)

# 添加标签文字
font_scale = 1.2
font_thickness = 3
cv2.putText(img_result, config['label'], (x1 + 5, y1 - 10),
            cv2.FONT_HERSHEY_SIMPLEX, font_scale, config['color'], font_thickne

# 显示最终结果
plt.figure(figsize=(12, 16))
plt.imshow(img_result)
plt.title('试卷版面分析结果 - 三种题型区域标注', fontsize=16)
plt.axis('off')

# 添加图例
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='red', alpha=0.3, label='选择题区域'),
    Patch(facecolor='green', alpha=0.3, label='判断题区域'),
    Patch(facecolor='blue', alpha=0.3, label='简答题区域')
]
plt.legend(handles=legend_elements, loc='lower right', fontsize=14)
plt.tight_layout()
plt.show()

```


20 25 - 20 26 学年《人工智能导论 A》秋季学期期末考试答题卡

| 题目 | 一 | 二 | 三 | | | 总分数 |
|-----|---|---|---|--|--|-----|
| 分数 | | | | | | |
| 评卷人 | | | | | | |

Multiple Choice

一、单选题 (每题 2 分, 共 60 分)

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| C | B | B | A | A | C | D | B | C | C |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| B | B | C | B | A | B | A | B | B | C |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| B | C | D | A | D | C | C | B | D | C |

True/False

二、判断题 (每题 2 分, 共 20 分)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| ✓ | × | × | ✓ | ✓ | × | × | ✓ | ✓ | ✓ |

Short Answer

三、简答题 (共 20 分)

答: 分为: 自由驾驶

主要功能: 辅助驾驶员和自动驾驶车辆在行驶过程中控制车辆正常行驶。
 规划决策: 分析路况, 合理规划行驶路线, 减轻驾驶员的负担。
 协调: 利用其自身优势, 在实时路况智能化辅助的环境下, 可以极大地使车辆更加智能, 方便了部分人群, 尤其是老年人、残疾人等。
 行车时: 使用车辆, 使驾驶员可以减轻负担和欲望得到提升。
 挑战: 尽管在大部分情况下有辅助, 但在对一些路况识别等情况下仍有不足, 且对路况的分析也会有错误, 进而导致交通事故发生。
 相同

```
In [9]: # 保存标注结果图像
result_bgr = cv2.cvtColor(img_result, cv2.COLOR_RGB2BGR)
cv2.imwrite('result_annotated.jpg', result_bgr)
print("标注结果已保存为 result_annotated.jpg")
```

标注结果已保存为 result_annotated.jpg

6. 各步骤汇总展示

```
In [10]: # 汇总展示全流程
fig, axes = plt.subplots(2, 2, figsize=(16, 20))

# 原图
axes[0, 0].imshow(img_rgb)
axes[0, 0].set_title('(a) 原始图像', fontsize=14)
axes[0, 0].axis('off')

# Canny 边缘
axes[0, 1].imshow(edges, cmap='gray')
axes[0, 1].set_title('(b) Canny 边缘检测', fontsize=14)
axes[0, 1].axis('off')

# 轮廓检测
axes[1, 0].imshow(img_paper_contour)
axes[1, 0].set_title('(c) 试卷轮廓检测', fontsize=14)
axes[1, 0].axis('off')

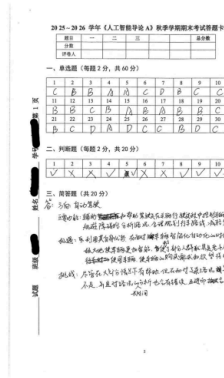
# 最终标注结果
axes[1, 1].imshow(img_result)
axes[1, 1].set_title('(d) 题型区域标注结果', fontsize=14)
```

```
axes[1, 1].axis('off')

plt.suptitle('试卷版面分析全流程', fontsize=18, fontweight='bold')
plt.tight_layout()
plt.savefig('result_summary.jpg', dpi=150, bbox_inches='tight')
plt.show()
print("汇总图已保存为 result_summary.jpg")
```

试卷版面分析全流程

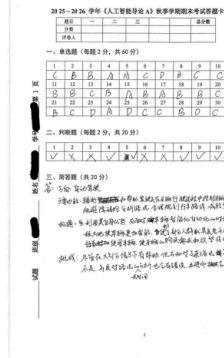
(a) 原始图像



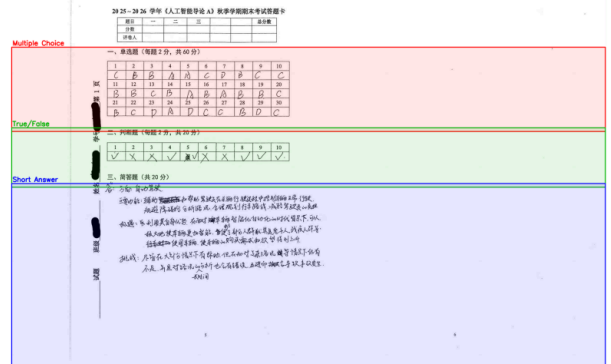
(b) Canny 边缘检测



(c) 试卷轮廓检测



(d) 题型区域标注结果



汇总图已保存为 result_summary.jpg

总结

基础任务完成：

1. **Canny 边缘检测** — 使用高斯模糊去噪后，设定阈值 (50, 150) 进行 Canny 边缘检测，再通过膨胀连接断裂边缘

2. **findContours 轮廓筛选** — 检测外部轮廓，按面积排序，筛选出面积最大的轮廓作为试卷边界
3. **题型区域标注** — 通过水平投影分析和区域分割，定位并标注了选择题、判断题、简答题三种题型的大致区域