Sepackageamsmath Sepackageamssymb

第 6 周教案：判断题识别（符号匹配）

计算机视觉课程组

1 基本信息

周次	第 6 周
主题	判断题识别（符号匹配）
学时	3 学时（160 分钟）
故事问题	怎么看到是 还是 ×?
OBE 目标	A3-特征识别：能识别判断题符号（/×）
项目贡献	实现判断题自动识别模块

2 教学目标

1. **知识目标：**
   - 理解形状匹配算法原理
   - 掌握轮廓特征提取方法
   - 了解模板匹配技术
2. **能力目标：**
   - 能够提取符号的轮廓特征
   - 能够使用轮廓特征区分 和 ×
   - 能够实现判断题答案识别
3. **素养目标：**
   - 理解形状特征在识别中的应用
   - 培养抽象思维能力

3 教学重点与难点

教学重点

- 轮廓特征提取（圆度、凸性、长宽比）
- 形状匹配算法
- 判断题识别流程

教学难点

- 形状特征的几何意义
- 手写符号的形变处理

4 教学过程设计

4.1 环节一：判断题识别概述（15 分钟）

4.1.1 1.1 判断题的特点（5 分钟）

常见符号：

- （正确/对）
- ×（错误/错）
- √（正确）
- ◯（正确）

与选择题的区别：

- 选择题：关注填涂密度
- 判断题：关注符号形状

4.1.2 1.2 识别方案（10 分钟）

方案 1：轮廓特征法

- 提取符号轮廓
- 计算形状特征
- 根据特征判断符号类型

方案 2：模板匹配法

- 准备标准符号模板
- 与模板进行匹配
- 选择最佳匹配结果

4.2 环节二：轮廓特征提取（45 分钟）

4.2.1 2.1 基础轮廓特征（20 分钟）

```
import cv2
import numpy as np

def extract_basic_features(contour):
    """
    提取基础轮廓特征

    参数:
        contour: 轮廓

    返回:
        特征字典
    """
    features = {}

    # 1. 面积
    features['area'] = cv2.contourArea(contour)

    # 2. 周长
    features['perimeter'] = cv2.arcLength(contour, True)

    # 3. 边界矩形
    x, y, w, h = cv2.boundingRect(contour)
    features['bbox'] = (x, y, w, h)

    # 4. 长宽比
    features['aspect_ratio'] = float(w) / h if h > 0 else 0

    # 5. 占空比（面积/边界矩形面积）
    bbox_area = w * h
```

```
    features['extent'] = features['area'] / bbox_area if bbox_area >
        0 else 0

    return features
```

4.2.2 2.2 高级形状特征（25 分钟）

1. 圆度（Circularity）

```
def calculate_circularity(contour):
    """
    计算圆度

    公式: 4 * pi * area / perimeter^2
    完美圆形的圆度接近1
    """
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)

    if perimeter == 0:
        return 0

    circularity = 4 * np.pi * area / (perimeter ** 2)
    return circularity
```

圆度特征：

- 圆形：接近 1
- 正方形：约 0.785
- ：较低（开口形状）
- ×：更低（两线交叉）

2. 凸性（Convexity）

```
def calculate_convexity(contour):
    """
    计算凸性

    凸性 = 轮廓面积 / 凸包面积
    """
```

```
    area = cv2.contourArea(contour)
    hull = cv2.convexHull(contour)
    hull_area = cv2.contourArea(hull)

    if hull_area == 0:
        return 0

    convexity = area / hull_area
    return convexity
```

凸性特征：

- 凸图形：接近 1（如 ◯）
- 凹图形：小于 1（如 ）

3. 固性（Solidity）

```
def calculate_solidity(contour):
    """
    计算固性（与凸性类似）

    固性 = 轮廓面积 / 凸包面积
    """
    area = cv2.contourArea(contour)
    hull = cv2.convexHull(contour)
    hull_area = cv2.contourArea(hull)

    if hull_area == 0:
        return 0

    solidity = float(area) / hull_area
    return solidity
```

4. 偏心率（Eccentricity）

```
def calculate_eccentricity(contour):
    """
    计算偏心率

    使用椭圆拟合，偏心率 = sqrt(1 - (b/a)^2)
    a: 长半轴，b: 短半轴
    """
```

```
    if len(contour) < 5:
        return 0

    # 拟合椭圆
    ellipse = cv2.fitEllipse(contour)

    # 获取长短轴
    center, axes, angle = ellipse
    major_axis = max(axes) / 2
    minor_axis = min(axes) / 2

    if major_axis == 0:
        return 0

    eccentricity = np.sqrt(1 - (minor_axis / major_axis) ** 2)
    return eccentricity
```

5. 拓扑特征：端点数

```
def count_endpoints(binary_image):
    """
    计算端点数量

    用于区分符号类型：
    - √: 2个端点
    - ×: 4个端点
    """
    # 定义端点模式（黑色像素周围只有一个白色像素）
    kernel = np.array([
        [0, 0, 0],
        [0, 1, 0],
        [0, 0, 0]
    ])

    # 膨胀操作
    dilated = cv2.dilate(binary_image, kernel, iterations=1)

    # 端点 = 原图 - 膨胀后的图
    endpoints = cv2.absdiff(binary_image, dilated)

    # 统计端点
```

```
    endpoint_count = cv2.countNonZero(endpoints)

    return endpoint_count
```

4.3 环节三：基于特征的分类（35 分钟）

4.3.1 3.1 特征提取完整函数（15 分钟）

```
def extract_symbol_features(roi):
    """
    提取符号的完整特征集

    参数:
        roi: 符号区域图像（灰度图）

    返回:
        特征字典
    """
    # 二值化
    _, binary = cv2.threshold(roi, 127, 255, cv2.THRESH_BINARY_INV)

    # 查找轮廓
    contours, _ = cv2.findContours(
        binary,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )

    if len(contours) == 0:
        return None

    # 使用最大轮廓（假设符号是最大的对象）
    contour = max(contours, key=cv2.contourArea)

    features = {}

    # 基础特征
    features['area'] = cv2.contourArea(contour)
    features['perimeter'] = cv2.arcLength(contour, True)
```

```
    # 形状特征
    features['circularity'] = calculate_circularity(contour)
    features['convexity'] = calculate_convexity(contour)
    features['solidity'] = calculate_solidity(contour)

    # 边界矩形
    x, y, w, h = cv2.boundingRect(contour)
    features['aspect_ratio'] = float(w) / h if h > 0 else 0
    features['extent'] = features['area'] / (w * h) if w * h > 0 else
        0

    # 端点数量
    features['endpoints'] = count_endpoints(binary)

    return features
```

4.3.2 3.2 基于规则的分类器（20 分钟）

```
def classify_by_rules(features):
    """
    基于规则判断符号类型

    参数:
        features: 特征字典

    返回:
        符号类型 ('check', 'cross', 'circle', 'unknown')
    """
    if features is None:
        return 'unknown'

    # 规则1: 根据圆度判断
    # 圆形（ ）圆度最高
    if features['circularity'] > 0.8:
        return 'circle'

    # 规则2: 根据凸性判断
    # √ 是凹图形，凸性较低
    # × 的凸性介于 √ 和    之间
```

```
    if features['convexity'] < 0.8:
        return 'check'  # 可能是 √

    # 规则3：根据端点数量
    # × 有4个端点，√ 有2个端点
    if features['endpoints'] >= 3:
        return 'cross'
    elif features['endpoints'] >= 1:
        return 'check'

    # 规则4：根据长宽比
    # √ 通常是斜向的，长宽比较大
    if features['aspect_ratio'] > 1.5:
        return 'check'

    return 'unknown'
```

训练集特征统计：

符号	圆度	凸性	固性	长宽比
	0.3-0.6	0.6-0.8	0.6-0.8	1.2-2.0
×	0.2-0.5	0.8-0.95	0.6-0.8	0.8-1.5
◯	0.7-1.0	0.95-1.0	0.9-1.0	0.8-1.2

4.4 环节四：模板匹配法（35 分钟）

4.4.1 4.1 模板匹配原理（10 分钟）

原理：** 在图像中滑动模板，计算相似度

相似度度量方法：

方法	说明
TM_SQDIFF	平方差匹配，值越小越相似
TM_SQDIFF_NORMED	归一化平方差匹配，0 表示完美匹配
TM_CCORR	相关匹配，值越大越相似
TM_CCOEFF	系数匹配，值越大越相似
TM_CCOEFF_NORMED	归一化系数匹配，1 表示完美匹配，-1 完全不相似

4.4.2 4.2 模板匹配实现（15 分钟）

```
def match_template(roi, templates, threshold=0.7):
    """
    模板匹配

    参数:
        roi: 待匹配区域
        templates: 模板字典 {'check': img_check, 'cross': img_cross,
            ...}
        threshold: 匹配阈值

    返回:
        最佳匹配的符号类型
    """
    best_match = None
    best_value = -float('inf')

    for symbol_type, template in templates.items():
        # 调整模板大小以匹配roi
        if template.shape[:2] != roi.shape[:2]:
            template = cv2.resize(template, (roi.shape[1], roi.shape
                [0]))

        # 转灰度
        if len(roi.shape) == 3:
            roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        else:
            roi_gray = roi

        if len(template.shape) == 3:
            template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY
                )
        else:
            template_gray = template

        # 模板匹配
        result = cv2.matchTemplate(roi_gray, template_gray, cv2.
            TM_CCOEFF_NORMED)
```

```
        # 获取匹配值
        match_value = result[0, 0]

        if match_value > best_value:
            best_value = match_value
            best_match = symbol_type

    # 阈值判断
    if best_value >= threshold:
        return best_match, best_value
    else:
        return 'unknown', best_value
```

创建模板：

```
# 创建标准符号模板
import cv2
import numpy as np

def create_checkmark_template(size=50):
    """创建✓模板"""
    template = np.zeros((size, size), dtype=np.uint8)

    # 绘制✓
    pts = np.array([
        [size*0.2, size*0.5],
        [size*0.4, size*0.7],
        [size*0.8, size*0.3]
    ], np.int32)

    cv2.polylines(template, [pts], False, 255, 3)

    return template

def create_cross_template(size=50):
    """创建×模板"""
    template = np.zeros((size, size), dtype=np.uint8)

    # 绘制×
    center = (size//2, size//2)
    radius = size//3
```

```
    cv2.line(template, (center[0]-radius, center[1]-radius),
             (center[0]+radius, center[1]+radius), 255, 3)
    cv2.line(template, (center[0]+radius, center[1]-radius),
             (center[0]-radius, center[1]+radius), 255, 3)

    return template

# 创建模板字典
templates = {
    'check': create_checkmark_template(),
    'cross': create_cross_template(),
    # 'circle': create_circle_template()
}
```

4.4.3 4.3 轮廓匹配（10 分钟）

```
def match_contour_shape(contour, template_contours):
    """
    轮廓形状匹配

    参数:
        contour: 待匹配轮廓
        template_contours: 模板轮廓字典

    返回:
        最佳匹配的符号类型
    """
    best_match = None
    best_value = float('inf')

    for symbol_type, template_contour in template_contours.items():
        # Hu矩匹配
        match_value = cv2.matchShapes(contour, template_contour,
                                      cv2.CONTOURS_MATCH_I1, 0.0)

        if match_value < best_value:
            best_value = match_value
            best_match = symbol_type
```

```
    return best_match, best_value
```

4.5 环节五：完整判断题识别模块（20 分钟）

```
class JudgeRecognizer:
    """判断题识别器"""

    def __init__(self, method='feature'):
        """
        初始化

        参数:
            method: 识别方法 ('feature' 或 'template')
        """
        self.method = method
        self.templates = None
        self.template_contours = None

        if method == 'template':
            self._init_templates()

    def _init_templates(self):
        """初始化模板"""
        self.templates = {
            'check': create_checkmark_template(),
            'cross': create_cross_template()
        }

        # 提取模板轮廓
        self.template_contours = {}
        for symbol_type, template in self.templates.items():
            contours, _ = cv2.findContours(
                template, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
            )
            if len(contours) > 0:
                self.template_contours[symbol_type] = contours[0]

    def preprocess(self, image):
```

```
        """预处理"""
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (5, 5), 0)
        _, binary = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
            cv2.THRESH_OTSU)
        return gray, binary

    def recognize(self, roi):
        """
        识别单个判断题

        参数:
            roi: 符号区域图像

        返回:
            (符号类型, 置信度)
        """
        gray, binary = self.preprocess(roi)

        if self.method == 'feature':
            # 特征法
            features = extract_symbol_features(gray)
            symbol = classify_by_rules(features)
            confidence = features.get('circularity', 0) if features
                else 0

        elif self.method == 'template':
            # 模板匹配法
            symbol, confidence = match_template(binary, self.
                templates)

        else:
            symbol = 'unknown'
            confidence = 0

        # 转换为布尔答案
        if symbol == 'check' or symbol == 'circle':
            answer = True
        elif symbol == 'cross':
            answer = False
```

```
        else:
            answer = None

        return answer, (symbol, confidence)

    def recognize_all(self, image, symbol_positions):
        """
        识别所有判断题

        参数:
            image: 完整图像
            symbol_positions: 符号位置列表 [(x,y,w,h), ...]

        返回:
            答案列表
        """
        results = []

        for i, (x, y, w, h) in enumerate(symbol_positions):
            roi = image[y:y+h, x:x+w]
            answer, details = self.recognize(roi)

            results.append({
                'question': i + 1,
                'answer': answer,
                'symbol': details[0],
                'confidence': details[1]
            })

        return results
```

4.6 环节六：实验（10 分钟）

```
# 使用示例
recognizer = JudgeRecognizer(method='feature')

# 读取图像
image = cv2.imread('judge_questions.jpg')
```