第 1 周：计算机视觉导论与图像基础

让机器“看懂”试卷的第一步

计算机视觉课程组

2024-2025 学年

视觉：人类获取信息的最主要渠道

人类视觉：

- 人脑约 50% 的神经元参与视觉处理。
- **语义理解**：我们看到的不是像素，是“人”、“车”、“试卷”。

计算机视觉 (CV)：

- 目标：给机器安装“眼睛”和“大脑”。
- 挑战：图像在计算机眼中只是一组**数字**。

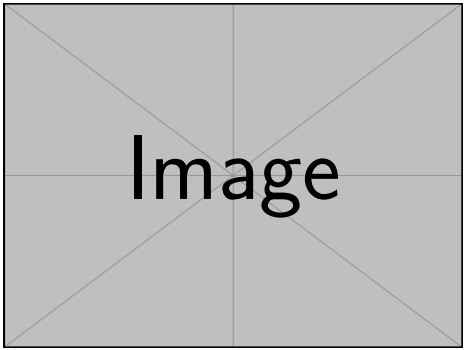


图: 语义 vs 矩阵

CV 的历史与现状

- **1960s**：Larry Roberts (CV 之父) 尝试让机器识别积木世界。
- **1970s-1980s**：提出边缘检测、Marr 视觉计算理论。
- **2012-至今**：**深度学习爆发**，AlexNet 在 ImageNet 竞赛中夺冠。

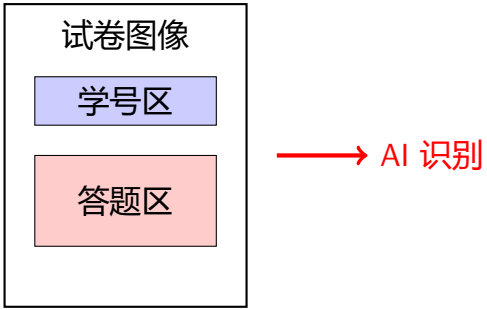
核心任务演变

分类 (是什么?) $\rightarrow$ 检测 (在哪儿?) $\rightarrow$ 分割 (形状如何?) $\rightarrow$ 生成 (画一个出来)

贯穿本学期的项目：AI 阅卷助手

任务分解：

1. **图像采集**：拍照、扫描。
2. **预处理**：纠偏、增强（本周内容）。
3. **定位**：找到答题卡、填空区。
4. **识别**：OCR (光学字符识别)。
5. **评分**：逻辑比对。



项目深度：答题卡的” 定位锚点” (Timing Marks)

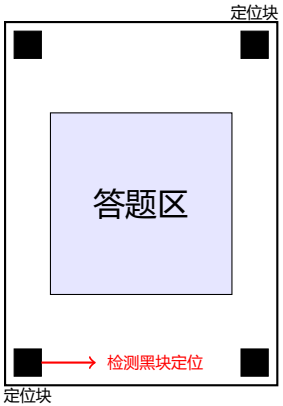
问题：如果试卷皱了或者拍摄角度极度倾斜，如何准确定位答题卡？

Timing Marks 的作用：

- 答题卡边缘的黑色小方块
- 用于精确定位答题卡区域
- 类似二维码的定位图案

设计规范：

- 位置：四个角或边缘
- 大小：固定的几何尺寸
- 对比度：黑色 vs 白纸
- 排列：特定模式（如 L 形）



工程价值

定位锚点检测：从轮廓到坐标

核心思路：检测黑色方块 → 计算中心 → 透视变换

```python
import cv2
import numpy as np

# 1. 读取答题卡
img = cv2.imread('answer_sheet.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 2. 二值化
_, binary = cv2.threshold(gray, 127, 255,
                          cv2.THRESH_BINARY)

# 3. 查找黑色方块轮廓
contours, _ = cv2.findContours(
    binary, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE
)

# 4. 筛选定位块（小方块）
marks = []
for cnt in contours:
    area = cv2.contourArea(cnt)
    # 定位块面积通常在特定范围
    if 100 < area < 1000:
        # 计算中心点
        M = cv2.moments(cnt)
        if M['m00'] != 0:
            cx = int(M['m10'] / M['m00'])
```

检测流程：

1. **二值化**：
   - 分离黑色定位块和白色背景
2. **轮廓查找**：
   - 找到所有黑色区域
3. **面积筛选**：
   - 定位块面积固定
   - 排除过大或过小的轮廓
4. **中心计算**：
   - 使用质心矩计算中心点
   - 得到精确定位坐标

下一步（下周预告）：

引出下周：透视变换的力量

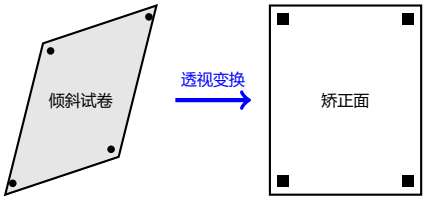
场景：检测到定位点后，如何矫正倾斜的试卷？

当前问题：

- 手机拍照难免倾斜
- 试卷可能有透视变形
- 直接处理会降低识别率

下周解决方案：

1. **透视变换**：
   - 将任意四边形矫正为矩形
   - 需要 4 个角点坐标
2. **特征匹配**：
   - 自动找到定位点
   - 无需人工标注



关键技术

`cv2.getPerspectiveTransform()` + `cv2.warpPerspective()`

图像的底层本质：矩阵 (Matrix)

- 一张灰度图 = 一个 **二维矩阵**。
- 矩阵中的每个元素称为 **像素 (Pixel)**。
- 常用数据类型：`uint8` (0-255)。

$$\begin{bmatrix} 255 & 255 & 254 \\ 120 & 0 & 118 \\ 255 & 253 & 255 \end{bmatrix}$$

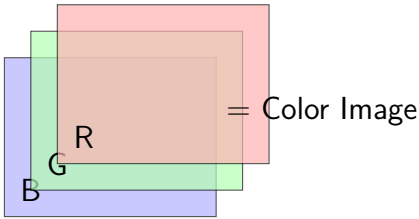
(矩阵数值 $\rightarrow$ 图像亮度)

注意坐标系！

计算机图像坐标系：**左上角为原点 (0,0)**，X 轴向右，Y 轴向 **下**。

彩色图像：RGB 三通道

- 彩色图像 = 三个二维矩阵堆叠 (**三维张量**)。
- 每个通道代表一种颜色光的强度。



OpenCV 的特殊性：默认读取顺序是 **BGR**，而非 RGB。

互动练习：调色盘

如果一个像素的 RGB 值为以下数值，它是什么颜色？

R	G	B	预测颜色
255	255	0	
0	255	255	
128	128	128	
0	0	0	

互动练习：调色盘

如果一个像素的 RGB 值为以下数值，它是什么颜色？

R	G	B	预测颜色
255	255	0	黄色
0	255	255	
128	128	128	
0	0	0	

互动练习：调色盘

如果一个像素的 RGB 值为以下数值，它是什么颜色？

R	G	B	预测颜色
255	255	0	黄色
0	255	255	青色/浅蓝
128	128	128	
0	0	0	

互动练习：调色盘

如果一个像素的 RGB 值为以下数值，它是什么颜色？

R	G	B	预测颜色
255	255	0	黄色
0	255	255	青色/浅蓝
128	128	128	灰色
0	0	0	

互动练习：调色盘

如果一个像素的 RGB 值为以下数值，它是什么颜色？

R	G	B	预测颜色
255	255	0	黄色
0	255	255	青色/浅蓝
128	128	128	灰色
0	0	0	**黑色**

图像的统计特性：直方图 (Histogram)

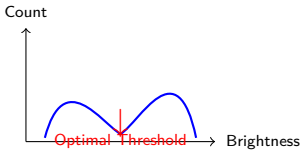
什么是直方图?

- 横坐标：亮度级别 (0-255)
- 纵坐标：该亮度像素出现的频次

在阅卷中的意义：

- **曝光检查**：判断照片是否太暗或过曝
- **二值化参考**：寻找波谷作为分割阈值

```python
import cv2
import matplotlib.pyplot as plt

# 计算直方图
img = cv2.imread('exam.jpg',
                 cv2.IMREAD_GRAYSCALE)
hist = cv2.calcHist([img], [0], None,
                    [256], [0, 256])

plt.plot(hist)
plt.title('Pixel Intensity Distribution')
plt.show()
```



直方图形态分析：试卷照片的曝光诊断

场景：自动判断试卷照片的质量

1. 正常曝光（双峰分布）

- 白纸（高亮度）+ 黑字（低亮度）
- 波谷在中间，适合二值化
- **阅卷理想状态**

2. 欠曝（左偏分布）

- 大部分像素集中在暗部
- 可能是光照不足
- 需要亮度增强

3. 过曝（右偏分布）

- 大部分像素集中在亮部
- 可能是闪光灯太强

```python
def check_exposure(img):
    """检查图像曝光情况"""
    gray = cv2.cvtColor(img,
                        cv2.COLOR_BGR2GRAY)

    # 计算平均亮度
    mean_brightness = np.mean(gray)

    # 判断曝光状态
    if mean_brightness < 80:
        return "欠曝，建议增强"
    elif mean_brightness > 200:
        return "过曝，建议降低"
    else:
        return "曝光正常"

# 使用
status = check_exposure(img)
print(f"曝光状态：{status}")
```

图像缩放：插值算法对比

问题：cv2.resize 时，像素是如何" 凭空产生" 或" 消失" 的?

```python
import cv2
import numpy as np

img = cv2.imread('exam.jpg')
h, w = img.shape[:2]

# 1. 最近邻插值（最快）
# 取最近的像素值，会有马赛克
near = cv2.resize(img, (w*2, h*2),
                  interpolation=cv2.INTER_NEAREST)

# 2. 双线性插值（默认）
# 周围4个像素加权平均，较平滑
linear = cv2.resize(img, (w*2, h*2),
                    interpolation=cv2.INTER_LINEAR)

# 3. 双三次插值（最慢但最好）
# 周围16个像素加权平均
cubic = cv2.resize(img, (w*2, h*2),
                  interpolation=cv2.INTER_CUBIC)
```

插值效果对比：

- **最近邻**：
  - 速度：最快
  - 质量：有锯齿
  - 应用：像素风游戏
- **双线性**：
  - 速度：中等
  - 质量：较平滑
  - 应用：日常缩放
- **双三次**：
  - 速度：最慢
  - 质量：最平滑
  - 应用：高质量缩放

OpenCV 核心函数详解

图像读取的隐患

```
import cv2

# 路径千万不能有中文（新手常见错误）
img = cv2.imread('paper.jpg')

# 检查是否读取成功
if img is None:
    print("错误：文件不存在或路径含有中文！")

# 打印维度 (H, W, C)
print(img.shape)
```

常用 Flag

工程实战：处理中文路径的”终极方案”

真实场景

阅卷系统中，学生姓名经常是中文，文件名如：张三 _ 试卷.jpg
cv2.imread() 直接读取会失败（返回 None）

问题原因：

- OpenCV 的 C++ 库不支持 Unicode 路径
- 中文路径会乱码
- 这不是 Python 的问题，是 OpenCV 的限制

错误做法：

```
# 直接读取，返回 None
img = cv2.imread('张三_试卷.jpg')
if img is None:
```

正确做法：

```
import cv2
import numpy as np

# 使用 np.fromfile + cv2.imdecode
def imread_chinese(path):
    """读取中文路径的图片"""
    # 从文件读取字节数据
    img_array = np.fromfile(path, dtype=np.uint8)

    # 解码为图像
    img = cv2.imdecode(img_array, -1)

    return img

# 使用
img = imread_chinese('张三_试卷.jpg')
cv2.imshow('成功！', img)
```

工程实战：保存中文路径图片

配套技巧：保存时也支持中文路径

```python
import cv2
import numpy as np

def imwrite_chinese(path, img):
    """保存中文路径的图片"""
    # 编码为字节数据
    is_success, img_buf = cv2.imencode(
        ".jpg", img
    )

    if is_success:
        # 保存到文件
        img_buf.tofile(path)
        return True
    return False

# 使用
img = cv2.imread('exam.jpg')
imwrite_chinese('处理结果_张三.jpg', img)
print("保存成功！")
```

原理说明：

1. **imdecode**：
   - 从内存中的字节数组解码
   - 绕过了文件系统的 Unicode 问题
2. **imencode + tofile**：
   - 编码为字节数组
   - 用 NumPy 的 tofile 写入
   - 同样绕过了 Unicode 问题

阅卷系统应用：

- 批量处理学生试卷
- 生成带学生姓名的结果文件
- 完全支持中文路径

Matplotlib 显示与 BGR 转换

为什么用 `plt.imshow` 显示出来的人脸是青蓝色的?

```
import matplotlib.pyplot as plt

# OpenCV 是 BGR, Matplotlib 是 RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img_rgb)
plt.show()
```

思考: 灰度图显示时需要设置 `cmap='gray'`，否则会变成“原油色”。

滤镜 1：灰度化 (Grayscale)

为什么要灰度化?

- 减少计算量（数据量降至 $1/3$）。
- 识别试卷上的文字，颜色信息通常是不必要的。

原理： $Gray = R \times 0.299 + G \times 0.587 + B \times 0.114$
(为什么绿色权重最高？因为人眼对绿色最敏感。)

滤镜 2：反色 (Inversion)

原理： $NewValue = 255 - OldValue$

- 黑色 (0) $\rightarrow$ 白色 (255)
- 白色 (255) $\rightarrow$ 黑色 (0)

应用：增强暗背景下的试卷特征，或者扫描负片。

滤镜 3：亮度调整与“溢出”陷阱

错误做法： `img + 50`
如果像素值是 220，加 50 变成 270。而在 uint8 类型下，270 会变成 **14** (截断/绕回)，导致图像出现难看的噪点。

安全写法

```
# 使用 numpy 的 clip 函数限制范围
bright_img = np.clip(img.astype(np.int32) + 50, 0, 255).astype(np
    .uint8)

# 或者使用 OpenCV 内置函数（推荐，速度更快）
bright_img = cv2.add(img, np.array([50.0]))
```

代码实战（1/5）：图像翻转与旋转

场景：阅卷时试卷可能被倒置，需要自动旋转

```python
import cv2
import numpy as np

img = cv2.imread('exam.jpg')

# 方法1: NumPy 数组切片
# 垂直翻转（上下颠倒）
flip_v = img[::-1, :, :]

# 水平翻转（左右颠倒）
flip_h = img[:, ::-1, :]

# 水平+垂直翻转（旋转180度）
flip_both = img[::-1, ::-1, :]
```

```python
# 方法2: OpenCV 函数（推荐）
flip_v = cv2.flip(img, 0)       # 垂直
flip_h = cv2.flip(img, 1)       # 水平
flip_both = cv2.flip(img, -1)  # 两者

# 显示对比
cv2.imshow('Original', img)
cv2.imshow('Flip V', flip_v)
cv2.imshow('Flip H', flip_h)
cv2.waitKey(0)
```

性能对比： NumPy 切片比 cv2.flip 快约 20%，但 cv2.flip 更易读

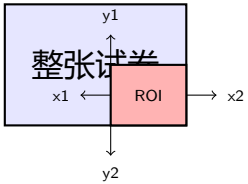
代码实战（2/5）：提取答题卡区域（ROI）

场景：从整张试卷中提取答题卡区域

```python
import cv2
import numpy as np

exam = cv2.imread('exam.jpg')
h, w = exam.shape[:2]

# 假设答题卡在右下角
# 坐标：从宽度的60%到末尾，高度的50%到末尾
x1, x2 = int(w * 0.6), w
y1, y2 = int(h * 0.5), h

# 提取 ROI
roi = exam[y1:y2, x1:x2]

# 保存 ROI
cv2.imwrite('answer_sheet.jpg', roi)

print("原图大小:", exam.shape)
print("ROI 大小:", roi.shape)
```

坐标系统回顾：

- 原点在左上角 (0, 0)
- `img[y1:y2, x1:x2]`
- y 是行（高度），x 是列（宽度）



代码实战（3/5）：通道分离与合并

场景：提取特定颜色通道

```python
import cv2

img = cv2.imread('exam.jpg')

# 方法1：使用 split 函数
b, g, r = cv2.split(img)

# 只保留红色通道，其他设为0
zeros = np.zeros_like(b)
img_r = cv2.merge([zeros, zeros, r])
```

```python
# 方法2：直接索引（更快）
img_r = img.copy()
img_r[:, :, 0] = 0  # B通道
img_r[:, :, 1] = 0  # G通道
# R通道保持不变

cv2.imshow('Red Only', img_r)
cv2.waitKey(0)
```

通道顺序：

- OpenCV: **BGR**
- matplotlib: **RGB**
- PIL: **RGB**

常见错误

使用 `plt.imshow(img)` 显示 OpenCV 图像时，颜色会异常！

解决方案：

```python
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
```

代码实战（4/5）：阅卷系统核心代码

场景：检测答题卡填涂位置

```python
import cv2
import numpy as np

# 1. 读取答题卡区域
roi = cv2.imread('answer_sheet.jpg',
                 cv2.IMREAD_GRAYSCALE)

# 2. 二值化
_, binary = cv2.threshold(roi, 127, 255,
                          cv2.THRESH_BINARY)

# 3. 定义选项位置
positions = [
    (100, 100, 120, 120),  # A
    (100, 130, 120, 150),  # B
    (100, 160, 120, 180),  # C
    (100, 190, 120, 200)   # D
]
```

```python
# 4. 检测每个选项是否被填涂
answers = []
for (x1, y1, x2, y2) in positions:
    option = binary[y1:y2, x1:x2]

    # 计算黑色像素比例
    black_pixels = np.sum(option == 0)
    total_pixels = option.size
    ratio = black_pixels / total_pixels

    # 判断是否填涂（阈值30%）
    if ratio > 0.3:
        answers.append('填涂')
    else:
        answers.append('未填')

print(answers)
```

核心思想：填涂区域黑色像素占比显著高于未填涂区域

代码实战（5/5）：图像增强对比

场景：答题卡光照不均，需要增强对比度

```python
import cv2
import numpy as np

img = cv2.imread('exam.jpg')

# 方法1：线性对比度调整
# new = alpha * old + beta
enhanced = cv2.convertScaleAbs(
    img, alpha=1.5, beta=30
)
```

```python
# 方法2：直方图均衡化
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
equalized = cv2.equalizeHist(gray)
```

```python
# 方法3：CLAHE（自适应）
clahe = cv2.createCLAHE(
    clipLimit=2.0,
    tileGridSize=(8,8)
)
enhanced_clahe = clahe.apply(gray)
```

效果对比：

- **线性调整**：简单但效果有限
- **直方图均衡化**：全局优化
- **CLAHE**：局部自适应，效果最好

阅卷推荐： CLAHE 适合光照不均场景

可视化对比（1/5）：不同阈值处理效果

问题：固定阈值 vs 自适应阈值 vs Otsu 阈值

```python
import cv2
import numpy as np

img = cv2.imread('exam.jpg',
                 cv2.IMREAD_GRAYSCALE)

# 1. 固定阈值
_, thresh1 = cv2.threshold(
    img, 127, 255, cv2.THRESH_BINARY
)

# 2. Otsu 自动阈值
_, thresh2 = cv2.threshold(
    img, 0, 255,
    cv2.THRESH_BINARY + cv2.THRESH_OTSU
)
```

```python
# 3. 自适应阈值（均值）
thresh3 = cv2.adaptiveThreshold(
    img, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY, 11, 2
)

# 4. 自适应阈值（高斯）
```

效果对比：

- **固定阈值**：
  - 简单但易受光照影响
- **Otsu**：
  - 全局最优
  - 适合双峰直方图
- **自适应均值**：
  - 局部处理
  - 抗光照变化
- **自适应高斯**：
  - 更平滑的局部阈值

可视化对比（2/5）：RGB vs HSV 色彩空间

场景：根据颜色提取特定对象（如红色印章）

RGB 空间的问题：

- 三个通道耦合
- 亮度变化影响颜色
- 难以定义" 红色" 范围

```
# RGB 方法（不推荐）
lower = [0, 0, 100]  # 难以确定
upper = [100, 100, 255]
mask = cv2.inRange(
    img, np.array(lower),
    np.array(upper)
)
```

HSV 空间的优势：

- H（色调）独立于亮度
- 更符合人类感知

```
# HSV 方法（推荐）
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# 红色在 HSV 中有两个范围
lower1 = np.array([0, 50, 50])
upper1 = np.array([10, 255, 255])
lower2 = np.array([170, 50, 50])
upper2 = np.array([180, 255, 255])

mask1 = cv2.inRange(hsv, lower1, upper1)
mask2 = cv2.inRange(hsv, lower2, upper2)
mask = cv2.bitwise_or(mask1, mask2)
```

问题：不同光照下，RGB 值变化很大

结论：颜色提取任务优先使用 HSV 空间

视觉感知：JPEG 压缩与伪影 (Artifacts)

问题：手机拍照的压缩会影响 OCR 识别率吗?

JPEG 压缩原理：

- 有损压缩：丢弃部分信息
- 将图像分成 8×8 像素块
- 每块独立进行 DCT 变换
- 高频信息被丢弃

压缩伪影（Artifacts）：

1. **块效应**：
   - 8×8 块边界可见
   - 放大 800
2. **振铃效应**：
   - 边缘周围出现波纹
3. **色彩失真**：

阅卷影响：

- 铅笔/钢笔痕迹可能模糊
- OCR 识别率降低
- 细节丢失导致误判