第 3 周：图像预处理与增强

试卷拍照模糊怎么办？

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

预处理是计算机视觉的”门卫”

现实问题

- 拍摄角度不正
- 光照不均匀
- 纸张有折痕
- 背景有杂物
- **手机拍照 vs 扫描仪**

预处理目标

1. 去除噪声干扰
2. 增强目标特征
3. 规范图像格式
4. **提升识别准确率**

预处理质量直接决定后续识别效果！

真实场景案例分析

低光照拍摄

光照不均

拍摄角度

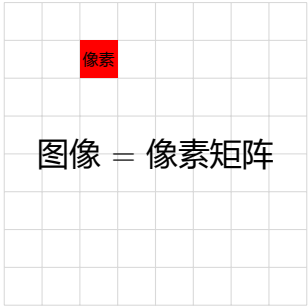
核心结论

没有好的预处理，再先进的算法也无法发挥威力！

图像的基本概念

核心概念：

- **像素**：图像的最小单位
- **分辨率**：宽度 × 高度（如 1920×1080）
- **位深**：每个像素的比特数（8bit = 256 级灰度）



OpenCV 中的图像表示：

```
读取图像 img = cv2.imread('exam.jpg')
numpy 是数组 print(type(img))  <class 'numpy.ndarray'>
print(img.shape)  (height, width, channels)
```

色彩空间与图像格式

常见色彩空间：

色彩空间	特点	应用
RGB	红、绿、蓝三通道	显示、存储
GRAY	单通道灰度	图像处理、OCR
HSV	色调、饱和度、亮度	颜色分割
YCrCb	亮度与色度分离	视频压缩

图像格式对比：

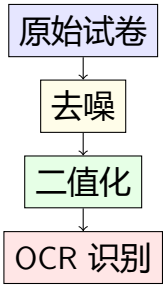
- **PNG**：无损压缩，适合处理中间结果
- **JPEG**：有损压缩，不适合后续处理
- **BMP**：无压缩，文件较大

```
# 色彩空间转换
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

OCR 文字识别预处理

试卷识别流程：

1. 去噪（去除拍摄噪点）
2. 二值化（黑白分明）
3. 透视矫正（展平试卷）
4. 区域分割（定位答题区）
5. 文字识别（OCR）



关键点

试卷识别的成功率，80% 取决于预处理质量！

其他应用场景

医学影像

- X 光片增强
- CT 图像去噪
- 病灶区域增强

工业质检

- 缺陷检测预处理
- 尺寸测量图像优化
- 表面纹理增强

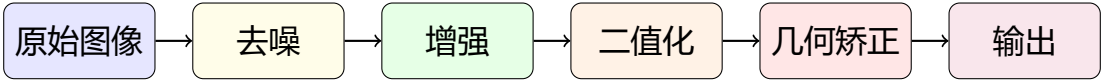
自动驾驶

- 雨天图像增强
- 夜视图像处理
- 车道线清晰化

人脸识别

- 光照归一化
- 人脸对齐
- 直方图均衡化

标准预处理流水线



各步骤作用：

步骤	作用
去噪	消除拍摄噪声、传感器噪声
增强	提升对比度、恢复细节
二值化	简化数据、突出目标
几何矫正	消除透视变形、统一尺寸

流水线顺序的重要性

错误顺序示例

先二值化 $\rightarrow$ 再去噪 $\times$

问题：二值化后的图像只有 0 和 255，滤波效果极差！

正确顺序

去噪 $\rightarrow$ 增强 $\rightarrow$ 二值化 $\rightarrow$ 矫正 ✓

本课程学习顺序：

1. 图像去噪（消除噪声）
2. 图像增强（提升质量）
3. 图像二值化（简化数据）
4. 几何变换（矫正变形）

课程概览

1. 为什么需要预处理?
2. 数字图像基础
3. 预处理应用场景
4. 预处理流水线
5. 图像去噪
6. 图像增强
7. 图像二值化
8. 几何变换
9. 案例分析
10. 课堂互动
11. 知识点总结
12. 课后作业
13. 下节预告

噪声类型与特征

噪声类型	特征	典型场景
高斯噪声	随机分布的亮度变化	传感器噪声、低光拍摄
椒盐噪声	随机的黑点或白点	传输错误、老化传感器
泊松噪声	与信号强度相关	低光摄影、X 射线
周期噪声	规则的干扰条纹	电气干扰、扫描缺陷

高斯噪声

服从正态分布 $N(\mu, \sigma^2)$
最常见，容易处理

椒盐噪声

随机像素变为 0 或 255
需要非线性滤波

试卷扫描常见噪声

实际问题

- 手机拍照：ISO 噪声（高斯）
- 压缩传输：块效应、JPEG 伪影
- 扫描仪：灰尘颗粒（椒盐）
- 纸张：纹理干扰

噪声对识别的影响：

- 字符边缘模糊 → OCR 识别率下降
- 背景噪声干扰 → 边缘检测失败
- 压缩伪影 → 细节丢失

结论：试卷预处理首选中值滤波！

空间域滤波原理

卷积操作：

- 滤波核（Kernel）在图像上滑动
- 每个位置计算加权求和
- 输出新的像素值

3×3 均值滤波核

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3×3 高斯滤波核

$$K \approx \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

边界处理

- **零填充**：边界外补 0
- **复制边界**：复制边缘像素

OpenCV 卷积实现

自定义滤波核:

```
import cv2
import numpy as np

# 自定义锐化核
kernel = np.array([
    [-1, -1, -1],
    [-1,  9, -1],
    [-1, -1, -1]
])

# 应用卷积
result = cv2.filter2D(img, -1, kernel)
```

常用滤波函数:

基础滤波器对比

均值滤波

```
blur = cv2.blur(
    img, (5, 5)
)
```

- 简单平均
- 速度快
- 边缘模糊

高斯滤波

```
blur = cv2.GaussianBlur(
    img, (5, 5), 0
)
```

- 加权平均
- 自然模糊
- 适合高斯噪声

中值滤波

```
blur = cv2.medianBlur(
    img, 5
)
```

- 中值替代
- 保边缘
- 适合椒盐噪声

滤波效果对比

滤波器	高斯噪声	椒盐噪声	边缘保留
均值滤波	良好	一般	差
高斯滤波	**优秀**	差	一般
中值滤波	一般	**优秀**	良好

选择建议

- **试卷扫描**：中值滤波（去除点状噪声）
- **照片美化**：高斯滤波（自然模糊）
- **边缘检测前**：双边滤波（保边去噪）

双边滤波 - 保边去噪

原理：同时考虑空间距离和像素值差异

```
# 双边滤波参数说明
bilateral = cv2.bilateralFilter(
    img,            # 输入图像
    9,              # 邻域直径
    75,             # 颜色空间标准差
    75              # 坐标空间标准差
)
```

特点：

- 保持边缘清晰
- 去除平滑区域噪声
- 计算量较大

应用场景

- 人像磨皮（卡通效果）
- 边缘检测前的预处理
- 需要保留边缘的场景

非局部均值去噪（NLM）

原理：利用图像的自相似性

```
# 快速非局部均值去噪
denoised = cv2.fastNlMeansDenoisingColored(
    img,                # 输入图像
    None,               # 输出
    10,                 # 滤波强度 (h)
    10,                 # 滤波强度 (hColor)
    7,                  # 模板窗口大小
    21                  # 搜索窗口大小
)
```

参数说明：

- **h**：滤波强度，值越大去噪越强（但会丢失细节）
- **templateWindowSize**：通常取 7（奇数）
- **searchWindowSize**：通常取 21（奇数，大于 template）

优势

去噪实战：完整代码

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

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

# 方法1：高斯滤波（适合高斯噪声）
gaussian = cv2.GaussianBlur(gray, (5, 5), 0)

# 方法2：中值滤波（适合椒盐噪声）- 推荐用于试卷
median = cv2.medianBlur(gray, 5)

# 方法3：双边滤波（保边去噪）
bilateral = cv2.bilateralFilter(gray, 9, 75, 75)

# 方法4：非局部均值（效果最好，速度较慢）
nlm = cv2.fastNlMeansDenoising(gray, None, 10, 7, 21)

# 显示对比
titles = ['原图', '高斯', '中值', '双边', 'NLM']
images = [gray, gaussian, median, bilateral, nlm]

plt.figure(figsize=(15, 3))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
```

PSNR 评估去噪质量

PSNR（峰值信噪比）：评估图像质量的指标

```
def psnr(img1, img2):
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return float('inf')
    return 20 * np.log10(255.0 / np.sqrt(mse))

# 对比各方法PSNR
print(f"高斯滤波: {psnr(gray, gaussian):.2f} dB")
print(f"中值滤波: {psnr(gray, median):.2f} dB")
print(f"双边滤波: {psnr(gray, bilateral):.2f} dB")
print(f"NLM去噪: {psnr(gray, nlm):.2f} dB")
```

PSNR 参考值：

- > 40 dB：优秀

去噪方法总结对比

方法	速度	去噪效果	边缘保留	适用场景
均值滤波				快速预览
高斯滤波				高斯噪声
中值滤波				**椒盐噪声**
双边滤波				保边去噪
NLM 去噪				**高质量去噪**

试卷预处理推荐方案

首选：中值滤波（快速、有效、保边）
备选： NLM 去噪（质量要求高时）
不推荐：均值滤波（边缘模糊严重）

去噪参数调优建议

核大小选择：

- 3×3：轻微去噪，保留细节
- 5×5：**平衡选择**（推荐）
- 7×7：强去噪，可能模糊细节

组合策略：

1. 先用中值滤波去除椒盐噪声
2. 再用高斯滤波平滑处理
3. 最后用 NLM 精细化处理（可选）

注意

过度去噪会导致字符边缘模糊，反而降低 OCR 识别率！

为什么要增强图像?

低质量图像问题:

- 对比度低（灰蒙蒙）
- 亮部过曝或暗部欠曝
- 细节不清晰

增强目标

- 提升对比度
- 恢复细节
- 改善视觉效果
- **方便后续处理**

图像增强 $\neq$ 去噪

去噪是”清理”，增强是”优化”

图像直方图

直方图：显示各像素值级别的分布

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

img = cv2.imread('exam.jpg', 0)

# 计算直方图
hist = cv2.calcHist(
    [img], [0], None,
    [256], [0, 256]
)

# 绘制直方图
plt.plot(hist, color='black')
plt.xlabel('像素值')
plt.ylabel('像素数量')
plt.show()
```

直方图解读：

- **左偏**：图像偏暗
- **右偏**：图像偏亮
- **集中**：对比度低
- **分散**：对比度高

直方图均衡化

原理：重新分布像素值，使直方图更均匀

```
# 全局直方图均衡化
equalized = cv2.equalizeHist(gray)

# 显示对比
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# 原图与直方图
axes[0,0].imshow(gray, cmap='gray')
axes[0,0].set_title('原图')
axes[0,1].plot(cv2.calcHist([gray], [0], None, [256], [0, 256]))
axes[0,1].set_title('原图直方图')

# 均衡化后与直方图
axes[1,0].imshow(equalized, cmap='gray')
axes[1,0].set_title('均衡化后')
axes[1,1].plot(cv2.calcHist([equalized], [0], None, [256], [0, 256]))
axes[1,1].set_title('均衡化直方图')

plt.tight_layout()
plt.show()
```

CLAHE - 自适应直方图均衡化

问题：全局均衡化可能导致局部过度增强

CLAHE（Contrast Limited Adaptive Histogram Equalization）：

- 分块处理，保留局部对比度
- 限制对比度，避免噪声放大

```
# 创建CLAHE对象
clahe = cv2.createCLAHE(
    clipLimit=2.0,      # 对比度限制
    tileGridSize=(8, 8) # 网格大小
)

# 应用CLAHE
enhanced = clahe.apply(gray)
```

参数说明：

- **clipLimit**：对比度限制（1-3），值越大对比度越高
- **tileGridSize**：网格大小（4×4 到 16×16），越小越局部

CLAHE 在试卷增强中的应用

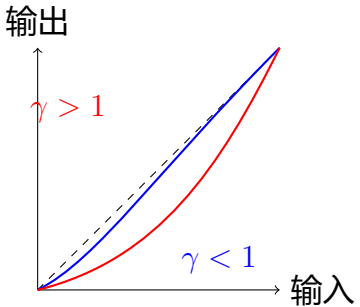
```python
# 试卷增强最佳实践
def enhance_exam(img_path):
    # 读取图像
    img = cv2.imread(img_path, 0)

    # 去噪
    denoised = cv2.medianBlur(img, 3)

    # CLAHE增强
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    enhanced = clahe.apply(denoised)

    return enhanced

# 使用
result = enhance_exam('exam.jpg')

# 保存结果
cv2.imwrite('exam_enhanced.jpg', result)
```

试卷增强推荐参数

- clipLimit = 2.0（平衡对比度与噪声）
- tileGridSize = (8, 8)（适合文字区域）

伽马校正原理

原理：幂律变换 $O = (I/255)^{\gamma} \times 255$



伽马值的影响：

- $\gamma < 1$：提亮暗部（曲线向上）
- $\gamma = 1$：无变化
- $\gamma > 1$：压暗亮部（曲线向下）

应用场景：

- 暗部细节恢复（$\gamma < 1$）
- 过曝图像修正（$\gamma > 1$）
- 显示设备校正

伽马校正代码实现

```python
import numpy as np

def gamma_correction(img, gamma=1.0):
    """
    伽马校正
    gamma < 1: 提亮暗部
    gamma > 1: 压暗亮部
    """
    # 构建查找表
    inv_gamma = 1.0 / gamma
    table = np.array([
        ((i / 255.0) ** inv_gamma) * 255
        for i in np.arange(0, 256)
    ]).astype("uint8")

    # 应用查找表
    return cv2.LUT(img, table)

# 使用示例
# 提亮暗部（适合暗光拍摄）
gamma_bright = gamma_correction(gray, gamma=0.6)
```

图像锐化原理

目的：增强边缘，使图像更清晰

常用锐化方法：

1. **拉普拉斯锐化**：基于二阶导数
2. **USM 锐化**：Unsharp Mask，反锐化掩模
3. **高反差保留**：保留高频细节

注意

锐化会放大噪声！建议在去噪后进行。

拉普拉斯锐化

拉普拉斯核：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

```
# 拉普拉斯锐化核
kernel = np.array([
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]
])

# 应用锐化
sharpened = cv2.filter2D(img, -1, kernel)

# 更强的锐化核
kernel_strong = np.array([
    [-1, -1, -1],
    [-1, 9, -1],
    [-1, -1, -1]
])
```

USM 锐化（推荐）

原理： 原图 + (原图 - 模糊图) × 强度

```python
def usm_sharpen(img, sigma=1.0, strength=1.5):
    """
    Unsharp Mask 锐化
    sigma: 高斯模糊半径
    strength: 锐化强度
    """
    # 高斯模糊
    blurred = cv2.GaussianBlur(img, (0, 0), sigma)

    # 计算差值（高频成分）
    high_freq = cv2.subtract(img, blurred)

    # 叠加高频成分
    sharpened = cv2.addWeighted(
        img, 1,
        high_freq, strength,
        0
    )

    return sharpened
```

图像增强总结

方法	作用	适用场景
直方图均衡化	提升整体对比度	低对比度图像
CLAHE	局部自适应增强	**试卷增强**
伽马校正	调整亮度分布	暗部/亮部修正
锐化	增强边缘细节	模糊图像

试卷增强推荐流程

去噪 $\rightarrow$ CLAHE $\rightarrow$ 轻度锐化

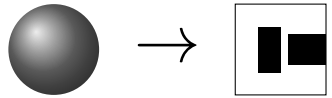
什么是二值化?

定义

将灰度图像转换为只有黑白两种颜色的图像（0 或 255）

转换规则：

$$f(x) = \begin{cases} 255 & \text{if } I(x,y) > T \\ 0 & \text{if } I(x,y) \leq T \end{cases}$$



为什么要二值化?

- 简化数据（减少信息量）
- 突出目标（文字边缘清晰）
- 便于后续处理（OCR 输入）

二值化的应用场景

OCR 文字识别

- 试卷答题卡识别
- 身份证识别
- 发票处理
- 文档数字化

其他应用

- 边缘检测预处理
- 形态学操作输入
- 条形码识别
- 图像压缩

关键点

二值化质量直接影响 OCR 识别率!

全局阈值法

原理：整个图像使用同一个阈值

```
# 固定阈值二值化
ret, binary = cv2.threshold(
    gray,                  # 输入图像
    127,                   # 阈值
    255,                   # 最大值
    cv2.THRESH_BINARY      # 类型
)

# 反色二值化（试卷常用）
ret, binary_inv = cv2.threshold(
    gray,
    127,
    255,
    cv2.THRESH_BINARY_INV  # 黑白反转
)
```

阈值类型：

类型	说明

全局阈值的局限

问题

光照不均时，单一阈值无法适应所有区域！

阈值过高：

- 亮部细节丢失
- 背景变为黑色

阈值过低：

- 暗部噪声保留
- 背景变为白色

解决方案：自适应阈值 / Otsu 算法

Otsu 算法原理

核心思想：最大化类间方差

算法步骤：

1. 遍历所有可能的阈值 T
2. 计算每个阈值下的类间方差 σ^2
3. 选择使 σ^2 最大的阈值作为最优阈值

$$\sigma^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2$$

其中：

- ω_0, ω_1：两类像素的比例
- μ_0, μ_1：两类像素的平均灰度

适用条件

直方图呈**双峰分布**（前景与背景明显分离）

Otsu 算法代码实现

```
# Otsu 自动阈值
ret, otsu = cv2.threshold(
    gray,
    0,                              # 自动计算
    255,
    cv2.THRESH_BINARY + cv2.THRESH_OTSU
)

print(f"Otsu 最优阈值: {ret}")

# 反色 Otsu（试卷常用）
ret, otsu_inv = cv2.threshold(
    gray,
    0,
    255,
    cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU
)
```

Otsu 优势

Otsu 算法的局限

问题场景

- 光照不均（单峰直方图）
- 噪声较多
- 前景背景比例悬殊

适用：

- 扫描仪扫描
- 光照均匀拍摄
- 双峰直方图

不适用：

- 手机拍照（光照不均）
- 阴影遮挡
- 复杂背景

解决方案：使用自适应阈值

自适应阈值原理

问题：光照不均时，全局阈值无法适应

解决方案：为每个像素点计算局部阈值

均值自适应：

$$T(x, y) = \text{mean}(\text{邻域}) - C$$

特点：

- 计算简单
- 对噪声敏感

高斯自适应：

$$T(x, y) = \text{加权均值}(\text{邻域}) - C$$

特点：

- 加权平均
- 效果更平滑
- **推荐使用**

自适应阈值代码实现

```
# 自适应阈值 - 均值法
adaptive_mean = cv2.adaptiveThreshold(
    gray,
    255,
    cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY,
    11,      # 邻域大小（奇数）
    2        # 常数 C
)

# 自适应阈值 - 高斯法（推荐）
adaptive_gaussian = cv2.adaptiveThreshold(
    gray,
    255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV,  # 反色，试卷常用
    11,      # 邻域大小（奇数）
    2        # 常数 C
)
```

参数说明：

自适应阈值参数调优

blockSize 选择：

- 太小（如 5）：对噪声敏感，产生细碎斑点
- **适中（11-21）：推荐范围**
- 太大（如 31）：接近全局阈值，失去自适应效果

C 值选择：

- 正值：使阈值降低，更多像素变为黑色
- 负值：使阈值升高，更多像素变为白色
- **通常取 2-10**

试卷二值化推荐配置

```
blockSize=15, C=5, THRESH_BINARY_INV
```

直方图指导阈值选择

```
import cv2
import matplotlib.pyplot as plt

# 计算直方图
hist = cv2.calcHist([gray], [0], None, [256], [0, 256])

# 绘制直方图，标记 Otsu 阈值
plt.figure(figsize=(10, 4))
plt.plot(hist, color='black')
plt.axvline(x=ret, color='red', linestyle='--',
            label=f'Otsu阈值={ret:.1f}')
plt.xlabel('像素值')
plt.ylabel('像素数量')
plt.legend()
plt.title('灰度直方图与最优阈值')
plt.grid(alpha=0.3)
plt.show()
```

直方图分析：

- **双峰明显**：适合 Otsu

二值化质量评估

```python
def evaluate_binary(binary_img):
    """
    评估二值化质量
    """
    # 黑白像素比例
    black_ratio = np.sum(binary_img == 0) / binary_img.size
    white_ratio = np.sum(binary_img == 255) / binary_img.size

    # 噪声评估（孤立白点）
    contours, _ = cv2.findContours(
        255 - binary_img,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )
    small_noise = sum(1 for c in contours if cv2.contourArea(c) < 10)

    return {
        'black_ratio': black_ratio,
        'white_ratio': white_ratio,
        'noise_count': small_noise
    }
```

二值化完整流程

```python
import cv2
import numpy as np

def binarize_exam(img_path, method='adaptive'):
    """
    试卷二值化处理
    method: 'global', 'otsu', 'adaptive'
    """
    # 读取图像
    img = cv2.imread(img_path, 0)

    if method == 'global':
        _, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)

    elif method == 'otsu':
        _, binary = cv2.threshold(img, 0, 255,
            cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    elif method == 'adaptive':
        # 推荐用于试卷
        binary = cv2.adaptiveThreshold(
            img, 255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY_INV,
            15, 2
        )

    return binary

# 使用
```

三种方法对比展示

```
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('exam.jpg', 0)

# 三种二值化方法
_, global_bin = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
_, otsu_bin = cv2.threshold(img, 0, 255,
    cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
adaptive_bin = cv2.adaptiveThreshold(img, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV, 15, 2)

# 对比显示
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes[0,0].imshow(img, cmap='gray')
axes[0,0].set_title('原图')
axes[0,1].imshow(global_bin, cmap='gray')
axes[0,1].set_title('全局阈值')
axes[1,0].imshow(otsu_bin, cmap='gray')
axes[1,0].set_title('Otsu')
axes[1,1].imshow(adaptive_bin, cmap='gray')
axes[1,1].set_title('自适应阈值（推荐）')
plt.show()
```

二值化方法总结

方法	光照均匀	光照不均	速度
全局阈值	良好	差	快
Otsu	**优秀**	差	快
自适应阈值	良好	**优秀**	较慢

试卷二值化建议

- **扫描件**：Otsu（速度快，效果好）
- **手机拍照**：自适应阈值（应对光照不均）
- **参数推荐**：blockSize=15, C=2-5

仿射变换基础

仿射变换：保持" 平直线" 和" 平行性" 的变换

包含的操作：

- 平移（Translation）
- 旋转（Rotation）
- 缩放（Scaling）
- 倾斜/剪切（Shear）

变换矩阵（2×3）：

$$M = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

新坐标：$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$

仿射变换代码实现

```python
import cv2
import numpy as np

# 图像平移
def translate(img, x, y):
    M = np.float32([[1, 0, x], [0, 1, y]])
    return cv2.warpAffine(img, M, (img.shape[1], img.shape[0]))

# 图像旋转
def rotate(img, angle, center=None):
    h, w = img.shape[:2]
    if center is None:
        center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    return cv2.warpAffine(img, M, (w, h))

# 使用
translated = translate(img, 50, 30)   # 向右50，向下30
rotated = rotate(img, 15)             # 旋转15度
```

仿射变换应用场景

文档矫正

- 轻微倾斜修正
- 水平对齐

图像增强

- 数据增强
- 随机变换

局限

仿射变换不能处理透视变形（近大远小）

透视变换原理

问题：拍照时相机与试卷不平行，产生透视变形

透视变换矩阵（3×3）：

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

需要 **4 个点对**来确定变换矩阵

透视变换实现

```
import numpy as np
import cv2

# 定义四个角点（原图像）
pts1 = np.float32([
    [100, 150],   # 左上
    [450, 120],   # 右上
    [480, 380],   # 右下
    [80, 400]     # 左下
])

# 定义目标矩形
width, height = 400, 300
pts2 = np.float32([
    [0, 0],                   # 左上
    [width - 1, 0],           # 右上
    [width - 1, height - 1],  # 右下
    [0, height - 1]           # 左下
])

# 计算透视变换矩阵
M = cv2.getPerspectiveTransform(pts1, pts2)
```

输出尺寸计算

如何确定输出尺寸?

```python
import numpy as np

def calculate_output_size(pts):
    """
    根据四个角点计算输出尺寸
    """
    # 计算宽度（取上下边长的最大值）
    w1 = np.linalg.norm(pts[1] - pts[0])
    w2 = np.linalg.norm(pts[2] - pts[3])
    width = max(int(w1), int(w2))

    # 计算高度（取左右边长的最大值）
    h1 = np.linalg.norm(pts[3] - pts[0])
    h2 = np.linalg.norm(pts[2] - pts[1])
    height = max(int(h1), int(h2))

    return width, height

# 使用
width, height = calculate_output_size(pts1)
```

自动文档矫正流程

核心步骤：

1. 边缘检测（Canny）
2. 轮廓查找（findContours）
3. 四边形近似（approxPolyDP）
4. 透视变换

关键点

如何从图像中找到四个角点?

- 假设最大轮廓是文档边缘
- 将轮廓近似为四边形
- 按顺序排列四个角点

自动矫正代码框架

```python
def auto_correct_document(img):
    # 1. 预处理
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(blur, 50, 150)

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

    # 3. 找到最大轮廓（假设是文档）
    doc_contour = max(contours, key=cv2.contourArea)

    # 4. 近似为四边形
    peri = cv2.arcLength(doc_contour, True)
    approx = cv2.approxPolyDP(doc_contour, 0.02 * peri, True)

    # 5. 透视变换
    if len(approx) == 4:
        # 获取四个角点并排序
        pts = order_points(approx.reshape(4, 2))
        # ... 执行透视变换

    return result
```

角点排序

问题：轮廓检测返回的四个点顺序是随机的

解决方案：统一排序为 [左上, 右上, 右下, 左下]

```
分为左右两 left = xsorted[:2,:]right = xsorted[2:,:]
组左边y按排序（上→ left = left[np.argsort(left[:, 1]), :]  下）右边y按排序（上→ right =
    right[np.argsort(right[:, 1]), :]
下）返回：左上、右上、右下、左 return np.array([ left[0],  下左 right[0],  上右 right[1],  上右
    left[1]  下左 ]) 下
```

完整的文档矫正代码

```python
def four_point_transform(img, pts):
    """
    四点透视变换
    """
    # 排序四个角点
    rect = order_points(pts)
    (tl, tr, br, bl) = rect

    # 计算输出尺寸
    widthA = np.linalg.norm(br - bl)
    widthB = np.linalg.norm(tr - tl)
    maxWidth = max(int(widthA), int(widthB))

    heightA = np.linalg.norm(tr - br)
    heightB = np.linalg.norm(tl - bl)
    maxHeight = max(int(heightA), int(heightB))

    # 目标点
    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]
    ], dtype="float32")

    # 计算变换矩阵并应用
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(img, M, (maxWidth, maxHeight))
```

几何变换总结

变换类型	所需点数	应用场景
平移	1 个位移向量	图像移动
旋转	1 个中心点 + 角度	旋转修正
仿射	3 个点对	倾斜矫正
透视	4 个点对	**文档矫正**

试卷矫正推荐方案

1. 边缘检测（Canny）
2. 找最大轮廓并近似为四边形
3. 排序四个角点
4. 透视变换矫正

案例 1：手机拍摄的模糊试卷

问题描述：

- 拍摄角度不正
- 光照不均匀
- 手持模糊

处理方案：

1. **去噪**：中值滤波去除噪点
2. **增强**：CLAHE 提升对比度
3. **锐化**：轻度 USM 锐化
4. **二值化**：自适应阈值
5. **矫正**：透视变换展平

效果： OCR 识别率从 60% 提升到 95%

案例 2：光照不均的扫描件

问题描述：

- 左侧有阴影
- 中心过曝
- 对比度低

处理方案：

1. **直方图分析**：识别光照分布
2. **CLAHE**：局部自适应增强
3. **伽马校正**：$\gamma = 0.8$ 提亮暗部
4. **自适应二值化**：blockSize=15

关键：避免使用全局阈值和全局均衡化

案例 3：有折痕的老试卷

问题描述：

- 纸张有明显折痕
- 折痕处有阴影
- 字迹被折痕遮挡

处理方案：

1. **去噪**：NLM 去除折痕噪声
2. **方向滤波**：沿文字方向平滑
3. **Inpainting**：修复折痕区域（高级）
4. **增强**：对比度拉伸

注意

严重折痕可能需要手动干预或深度学习修复

扫描全能王的预处理技术

核心技术：

- **自动边缘检测**：智能识别文档边界
- **透视矫正**：自动展平拍摄变形
- **多帧融合**：多张拍摄降噪
- **智能增强**：针对不同场景调优

处理流程：

$$\text{边缘检测} \rightarrow \text{透视矫正} \rightarrow \text{去噪} \rightarrow \text{增强} \rightarrow \text{压缩}$$

银行票据自动处理

应用场景：

- 支票识别
- 发票处理
- 表单录入

预处理要求：

- **高精度**：错误容忍度极低
- **标准化**：统一输入格式
- **鲁棒性**：应对各种拍摄条件
- **可追溯**：保留处理日志

特殊技术：

- 红外去伪（防伪检测）
- 水印去除
- 印章分离

档案数字化预处理

挑战：

- 纸张老化、发黄
- 字迹褪色
- 装订孔遮挡
- 背面透字

解决方案：

- **背景白化**：去除纸张底色
- **字迹增强**：对比度拉伸
- **去网纹**：去除印刷网纹
- **装订修复**：Inpainting 填补

Quiz 1：噪声识别

问题：下面图像中的噪声属于哪种类型？

图像 A	图像 B
有随机分布的黑点和白点	整体有朦胧感，细节模糊

Quiz 1：噪声识别

问题：下面图像中的噪声属于哪种类型？

图像 A

有随机分布的黑点和白点

答案：椒盐噪声

图像 B

整体有朦胧感，细节模糊

答案：高斯噪声

Quiz 1：噪声识别

问题：下面图像中的噪声属于哪种类型？

图像 A

有随机分布的黑点和白点

答案：椒盐噪声

图像 B

整体有朦胧感，细节模糊

答案：高斯噪声

思考：应该分别用什么滤波器处理？

Quiz 2：场景选择

问题：以下场景应该使用哪种二值化方法？

1. 扫描仪扫描的试卷
2. 手机拍照的试卷（有阴影）
3. 光照均匀的发票

Quiz 2：场景选择

问题：以下场景应该使用哪种二值化方法？

1. 扫描仪扫描的试卷 → Otsu 算法
2. 手机拍照的试卷（有阴影）
3. 光照均匀的发票

Quiz 2：场景选择

问题：以下场景应该使用哪种二值化方法？

1. 扫描仪扫描的试卷 → Otsu 算法
2. 手机拍照的试卷（有阴影）→ 自适应阈值
3. 光照均匀的发票

Quiz 2：场景选择

问题：以下场景应该使用哪种二值化方法？

1. 扫描仪扫描的试卷 → Otsu 算法
2. 手机拍照的试卷（有阴影）→ 自适应阈值
3. 光照均匀的发票 → 全局阈值或 Otsu

Quiz 3：参数调优

问题：自适应阈值参数如何选择？

场景 A：大号文字

选择 blockSize：

场景 B：背景噪声多

选择 C 值：

Quiz 3：参数调优

问题：自适应阈值参数如何选择？

场景 A：大号文字

选择 blockSize：

- A. 5
- B. 11
- C. 21 ✓

场景 B：背景噪声多

选择 C 值：

Quiz 3：参数调优

问题：自适应阈值参数如何选择？

场景 A：大号文字

选择 blockSize：

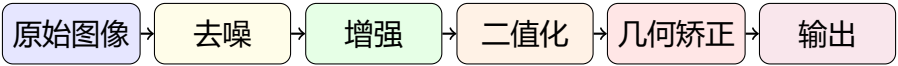
- A. 5
- B. 11
- C. 21 ✓

场景 B：背景噪声多

选择 C 值：

- A. 2
- B. 10 ✓
- C. -5

预处理完整流程回顾



核心方法对比：

类别	基础方法	进阶方法	推荐场景
去噪	高斯/均值滤波	中值/NLM	试卷用中值
增强	直方图均衡化	CLAHE/伽马	试卷用 CLAHE
二值化	全局阈值	Otsu/自适应	拍照用自适应
几何	仿射变换	透视变换	文档用透视

参数速查表

操作	参数	推荐值
中值滤波	kernel	5
CLAHE	clipLimit	2.0
CLAHE	tileGridSize	(8, 8)
自适应阈值	blockSize	15
自适应阈值	C	2-5
伽马校正	gamma	0.7-0.9
USM 锐化	sigma	1.0
USM 锐化	strength	1.5

作业：试卷图像预处理完整实现

任务描述：实现一个完整的试卷图像预处理系统

基本要求（60 分）：

1. 实现去噪功能（至少 2 种方法对比）
2. 实现二值化功能（至少 2 种方法对比）
3. 生成处理前后对比图
4. 代码规范，有注释

进阶要求（40 分）：

1. 实现 CLAHE 增强功能
2. 实现透视矫正功能
3. 编写批量处理脚本
4. 实现 PSNR 等质量评估指标

提交要求

提交内容：

1. Python 代码（.py 文件或 Jupyter Notebook）
2. 测试图像（处理前/后对比）
3. 实验报告（PDF）

实验报告包含：

- 不同方法的对比分析
- 参数调优过程
- 遇到的问题与解决方案
- 处理效果评估

截止时间：下次上课前
提交方式：教学平台上传

评分标准

项目	分值
去噪功能实现	15 分
二值化功能实现	15 分
增强功能实现	10 分
几何矫正实现	10 分
代码规范	15 分
实验报告质量	20 分
创新点（可选）	+15 分
总分	**100 分**

第 4 周：试卷版面分析

故事问题：怎么知道选择题、简答题在哪里？

你将学会：

- 边缘检测（Canny、Sobel）
- 轮廓查找与分析
- 区域定位与分割
- 版面结构理解

延伸学习资源

推荐书籍：

- 《数字图像处理》(冈萨雷斯) - 经典教材
- 《Learning OpenCV》- 实战导向
- 《计算机视觉：算法与应用》- 深入理论

在线资源：

- OpenCV 官方文档：`https://docs.opencv.org/`
- OpenCV 中文网
- GitHub: openencv/opencv

实践项目：

- Tesseract OCR 项目
- PaddleOCR 飞桨文字识别
- 车牌识别项目

谢谢！

有问题随时交流

邮箱： wwtong@bipt.edu.cn