3022207128-杨宇鑫-低光照增强-实验报告

目录

一. 低光照图像增强算法技术报告

1. 算法概述

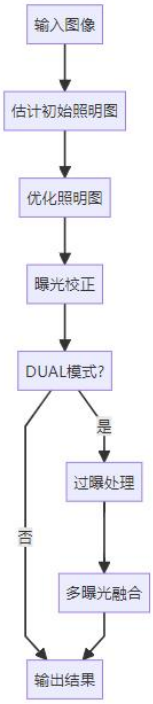
该实现结合了两种低光照增强方法：

- DUAL（Dual-exposure fusion）
- LIME（Low-light Image Enhancement）

核心思路是基于 Retinex 理论，将图像分解为照明图（Illumination Map）和反射图（Reflectance Map）。

2. 主要算法流程

2.1 基本流程



2.2 关键步骤详解

2.2.1 照明图估计与优化

```
# 初步估计照明图
L = np.max(im, axis=-1)

# 优化照明图
```

```
L_refined = refine_illumination_map_linear(L, gamma, lambda_, kernel, e
ps)
```

照明图优化采用变分方法，通过求解以下优化问题：

```
min_L ||L - L_input||² + λ * (Wx|∂xL|² + Wy|∂yL|²)
```

其中：

- L_input 是初始照明图
- Wx, Wy 是空间权重
- λ 是平衡参数

2.2.2 空间权重计算

```python
def compute_smoothness_weights(L, x, kernel, eps=1e-3):
    # 计算 x 或 y 方向的梯度
    Lp = cv2.Sobel(L, cv2.CV_64F, int(x == 1), int(x == 0), ksize=1)

    # 计算空间亲和度
    T = convolve(np.ones_like(L), kernel, mode='constant')
    T = T / (np.abs(convolve(Lp, kernel, mode='constant')) + eps)

    return T / (np.abs(Lp) + eps)
```

权重计算考虑了：

1. 图像梯度信息
2. 空间距离衰减
3. 结构相似性

2.2.3 DUAL 模式的多曝光融合

当启用 DUAL 模式时，算法会：

1. 处理欠曝光
2. 处理过曝光
3. 使用 Mertens 融合方法合成最终结果

```python
def fuse_multi_exposure_images(im, under_ex, over_ex, bc=1, bs=1, be=1):
    merge_mertens = cv2.createMergeMertens(bc, bs, be)
    images = [np.clip(x * 255, 0, 255).astype("uint8")
              for x in [im, under_ex, over_ex]]
    return merge_mertens.process(images)
```

3. 关键参数说明

3.1 gamma 参数

- 控制照明图的非线性调整

- 范围：0.1-2.0
- 较小值使图像更亮，较大值使图像更暗
- 推荐值：0.6

3.2 lambda 参数

- 控制照明图优化的平滑程度
- 范围：0.05-1.0
- 较小值保留更多细节但可能引入噪声
- 较大值平滑效果更好但可能丢失细节
- 推荐值：0.15

4. 性能优化

代码中包含了几个重要的性能优化措施：

```
def enhance_image_exposure(im, gamma, lambda_, dual=True, sigma=3):
    # 1. 降采样处理大图像
    if max(height, width) > 1024:
        scale = 1024 / max(height, width)
        small_im = cv2.resize(im_normalized, None, fx=scale, fy=scale)

    # 2. 使用较小的 kernel size
    kernel = create_spacial_affinity_kernel(sigma, size=7)

    # 3. 使用单精度浮点数
    im_normalized = im.astype(np.float32) / 255.
```

5. 优缺点分析

优点：

1. 支持两种增强模式（DUAL 和 LIME）
2. 考虑了空间一致性
3. 包含性能优化措施
4. 参数可调节性强

缺点：

1. 大图像处理耗时较长
2. 对于极端低光照场景效果可能不够理想
3. 参数调节需要经验

6. 应用场景

适用于：

- 室内低光照场景
- 逆光场景
- 阴影区域细节增强
- 夜景照片优化

不适用于：

- 极度噪声图像
- 需要实时处理的场景
- 艺术效果处理

这个实现综合了多种技术，既考虑了效果也注重了实用性，是一个比较完整的低光照增强解决方案。

二. 低光照图像增强算法原理详解

1. 基本原理

算法基于 Retinex 理论，该理论认为任何图像 I 可以分解为两个分量：

```
I = R * L
```

其中：

- I：原始图像
- L：照明分量 (Illumination)
- R：反射分量 (Reflectance)

2. 详细处理步骤

2.1 初始照明图估计

```
# 从原始图像中估计初始照明图
L = np.max(im, axis=-1)
```

这一步：

1. 取 RGB 三个通道的最大值作为初始照明图
2. 这样做的原因是照明信息主要体现在亮度上

2.2 照明图优化

```
def refine_illumination_map_linear(L, gamma, lambda_, kernel, eps=1e-3):
    # 计算平滑权重
    wx = compute_smoothness_weights(L, x=1, kernel=kernel)
    wy = compute_smoothness_weights(L, x=0, kernel=kernel)
```

```
    # 构建并求解线性系统
    n, m = L.shape
    L_1d = L.flatten()

    # 构建稀疏矩阵
    F = csr_matrix((data, (row, column)), shape=(n * m, n * m))

    # 求解优化问题
    Id = diags([np.ones(n * m)], [0])
    A = Id + lambda_ * F
    L_refined = spsolve(csr_matrix(A), L_1d)
```

优化过程：

1. 计算水平(wx)和垂直(wy)方向的平滑权重
2. 构建稀疏矩阵系统
3. 求解优化问题得到优化后的照明图

2.3 曝光校正

```
def correct_underexposure(im, gamma, lambda_, kernel, eps=1e-3):
    # 获取并优化照明图
    L = np.max(im, axis=-1)
    L_refined = refine_illumination_map_linear(L, gamma, lambda_, kerne
l)

    # 校正图像
    L_refined_3d = np.repeat(L_refined[..., None], 3, axis=-1)
    im_corrected = im / L_refined_3d
```

校正步骤：

1. 估计并优化照明图
2. 将优化后的照明图扩展到 3 通道
3. 用原图除以照明图得到校正结果

2.4 DUAL 模式处理

```
def enhance_image_exposure(im, gamma, lambda_, dual=True):
    # 处理曝光不足
    under_corrected = correct_underexposure(im, gamma, lambda_, kernel)

    if dual:
        # 处理曝光过度
        inv_im = 1 - im
        over_corrected = 1 - correct_underexposure(inv_im, gamma, lambd
a_, kernel)

        # 融合结果
```

```
        result = fuse_multi_exposure_images(im, under_corrected, over_c
orrected)
```

DUAL 模式步骤：

1. 处理欠曝光部分
2. 处理过曝光部分（通过图像反转）
3. 融合两个结果

2.5 多曝光融合

```python
def fuse_multi_exposure_images(im, under_ex, over_ex, bc=1, bs=1, be=1):
    merge_mertens = cv2.createMergeMertens(bc, bs, be)
    images = [im, under_ex, over_ex]
    return merge_mertens.process(images)
```

融合考虑三个因素：

- 对比度 (bc)
- 饱和度 (bs)
- 曝光度 (be)

3. 关键优化技术

3.1 大图像处理优化

```python
# 对大图像进行降采样处理
if max(height, width) > 1024:
    scale = 1024 / max(height, width)
    small_im = cv2.resize(im_normalized, None, fx=scale, fy=scale)
```

3.2 空间权重计算优化

```python
def compute_smoothness_weights(L, x, kernel, eps=1e-3):
    # 使用Sobel 算子计算梯度
    Lp = cv2.Sobel(L, cv2.CV_64F, int(x == 1), int(x == 0), ksize=1)

    # 计算空间权重
    T = convolve(np.ones_like(L), kernel, mode='constant')
    T = T / (np.abs(convolve(Lp, kernel, mode='constant')) + eps)
```

4. 处理效果影响因素

4.1 gamma 参数影响

- gamma < 1: 增加亮度
- gamma > 1: 降低亮度
- gamma ≈ 0.6: 通常效果最好

4.2 lambda 参数影响

- lambda 较小: 保留更多细节，可能增加噪声

- lambda 较大: 更平滑，可能丢失细节
- lambda ≈ 0.15: 平衡点

5. 算法特点

优点：

1. 保留图像细节
2. 避免过度增强
3. 考虑空间一致性
4. 支持自适应处理

局限性：

1. 计算复杂度较高
2. 对参数敏感
3. 可能引入伪影

这个算法通过精心设计的照明图优化和多曝光融合，实现了较好的低光照增强效果，同时考虑了实际应用中的性能优化问题。

三. 项目使用教程

1. 安装依赖

```
pip3 install -r requirements.txt
```

2. 启动项目

2.1 启动服务器

```
python app.py
```

服务器将在 http://127.0.0.1:5500 启动

3. 使用流程

3.1 访问系统

1. 打开浏览器
2. 访问 http://127.0.0.1:5500

3.2 图像处理步骤

1. 点击”选择文件”按钮上传图片
    - 支持的格式：JPG、PNG、BMP

- 最大文件大小：16MB

2. 调整处理参数
   - Gamma 值（0.1-2.0）：
     - 较小的值使图像更亮
     - 较大的值使图像更暗
     - 默认值：0.6
   - Lambda 值（0.05-0.3）：
     - 控制平滑度
     - 较小的值保留更多细节
     - 较大的值产生更平滑的结果
     - 默认值：0.15
   - 处理方法：
     - DUAL：双重曝光融合（推荐）
     - LIME：单一曝光校正
3. 点击"处理图像"按钮
   - 等待处理完成（处理时间取决于图像大小）
   - 处理完成后会自动显示结果

3.3 最佳实践

- 建议从小尺寸图像开始测试
- 对于不同类型的图像，可以尝试不同的参数组合：
  - 暗部细节丢失：降低 Gamma 值
  - 过度曝光：提高 Gamma 值
  - 噪点明显：提高 Lambda 值
  - 细节不足：降低 Lambda 值

4. 常见问题解决

4.1 图像上传失败

- 检查文件大小是否超过 16MB
- 确保文件格式正确
- 检查网络连接

4.2 处理结果不理想

- 尝试调整 Gamma 和 Lambda 参数
- 切换处理方法（DUAL/LIME）
- 检查原始图像质量

4.3 服务器错误

- 检查 Python 环境和依赖是否正确安装
- 确保所有必要的目录都已创建
- 查看服务器日志获取详细错误信息

5. 性能注意事项

- 大尺寸图像会自动缩放以提高处理速度
- 处理时间与图像尺寸成正比
- 建议在处理大量图像时使用较小的分辨率

6. 安全提示

- 定期清理 uploads 目录
- 不要上传敏感图像
- 在生产环境中部署时添加适当的安全措施

四. 处理效果





_DSC6366.ARW
4912 x 7360


_DSC6366.ARW
4912 x 7360