基于多模态语义感知的视频流自适应频域滤波与重建研究

数字信号处理课程实践报告

惠天一(23009101501)[1]

(1. 西安电子科技大学 通信工程学院，陕西 西安 710071)

摘要： 在数字图像与视频处理领域，如何在抑制噪声的同时保留高频边缘细节始终是一个核心难题。传统的线性平滑滤波器（如高斯低通滤波、均值滤波）基于全局一致的频域响应假设，往往在去除高频噪声的同时不可逆地抹除了图像的纹理与边缘信息。针对这一“去噪-保边”的零和博弈问题，本文提出了一种融合计算机视觉语义理解与数字信号处理频域变换的混合架构。

本文首先从二维离散傅里叶变换（2D-DFT）的数学原理出发，推导了频域滤波与空域卷积的对偶关系。在工程实现上，引入 U-2-Net 深度学习模型构建高精度动态前景掩膜（Mask）；随后在 MATLAB 环境中，设计了双通道自适应频域滤波器。通过对前与背景实施差异化的频域截断策略，并结合空间非平稳噪声模型进行验证。

实验结果表明，该算法有效解决了传统 DSP 方法中背景噪点残留与前景模糊的矛盾。量化指标显示，相比传统全局滤波，该方法在静态图像测试中将 PSNR 从 22.49 dB 提升至 31.62 dB，且在视频流处理中显著提高了 SSIM 结构相似性。本文还详细探讨了 FFT 频谱中心化、吉布斯效应的抑制以及跨语言项目的优化技术。

关键词： 数字信号处理；二维傅里叶变换；自适应滤波；语义分割

Research on Adaptive Frequency Domain Filtering and Reconstruction of Video Streams Based on Multi-modal Semantic Perception

Hui Tianyi (23009101501)[1]

(1. School of Telecommunications Engineering, Xidian University, Xi’an 710071, China)

Abstract: In digital image and video processing, suppressing noise while preserving high-frequency edge details remains a core challenge. Traditional linear smoothing filters (e.g., Gaussian or mean filters) often irreversibly erase textures due to their global frequency response assumptions. To resolve this "denoising vs. edge-preservation" trade-off, this paper proposes a hybrid architecture combining Computer Vision semantic understanding with Digital Signal Processing requency-domain transformation.

Based on the mathematical duality of 2D-DFT between frequency and spatial domains, this study utilizes a U-2-Net model to generate high-precision dynamic foreground masks. An adaptive dual-channel filter is then implemented in MATLAB to apply differentiated frequency truncation strategies to the foreground and background, validated under non-stationary noise models.

Experimental results show that the proposed method effectively overcomes the conflict between residual noise and foreground blurring. Quantitative analysis indicates that the PSNR improved from 22.49 dB to 31.62 dB compared to global filtering, with a significant increase in SSIM for video streams. The paper also discusses FFT centralization, Gibbs phenomenon suppression, and cross-platform optimization.

Key Words: Digital Signal Processing；2D Fourier Transform；Adaptive Filtering；Semantic Segmentation

1 引言

1.1 研究背景与工程意义

在现代通信链路中，视频信号从采集端到显示端经历了一系列复杂变换。受限于传感器的量子效率与热噪声，以及传输信道中的电磁干扰，原始视频信号往往叠加了加性高斯白噪声（AWGN）。

在数字信号处理的视角下，图像边缘和噪声具有高度相似的频谱特征——都主要分布于高频区域。

图像处理中低频分量对应亮度平缓变化的区域，如：墙壁、天空、大面积皮肤等，它们占据了图像的主要能量。而高频分量对应亮度剧烈变化的区域，如物体轮廓、噪点。

经典的 DSP 处理方法通常采用低通滤波器（LPF）来截断高频分量。然而，这种“一刀切”的线性时不变（LTI）系统不仅滤除了噪声，也不可避免地模糊了图像的边缘，导致画质下降。因此，如何打破 LTI 系统的限制让滤波器“感知”到图像的内容，是本课题的核心。

1.2 本项目的核心目标

1.理论验证：深入刨析并验证二维 DFT 的平移性、共轭对称性及卷积定理。

2.算法设计：设计空间自适应的频域滤波方案。利用 Mask 将图像分解为不同区域，分别施加不同截止频率的滤波器。

3.数据分析：利用 PSNR（峰值信噪比）和 SSIM（结构相似性）等客观指标，量化评估算法性能。

2 数字图像频域处理的数学原理

2.1 二维离散傅里叶变换（2D-DFT）

图像被建模为一个二维离散信号$f(x,y)$。为了在频率域分析图像特征，采用二维离散傅里叶变换。对于尺寸为$M \times N$的图像，变换公式 7[1]为：

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

其中，$|F(u,v)|$代表了图像在特定频率方向上的能量强度。为了便于观察，通常使用 fftshift[2][3]将直流分量 移至频谱中心。即通过对角交换象限，让频谱图呈现出以直流分量为中心、频率向四周逐渐升高的分布形态，从而更直观地反映图像的能量集中情况

2.2 J 卷积定理与滤波设计

根据卷积定理，空间域卷积等价于频率域的乘积：

$$f(x,y) * h(x,y) \iff F(u,v) \cdot H(u,v)$$

这为图像滤波提供了巨大计算便利，能够利用快速傅里叶变换（FFT）将复杂的空域卷积转换为简单的频域点乘。

本次实验对比了两种滤波器的特性

高斯低通滤波器（GLPF）：无旁瓣，无振铃效应，适合视频流的平滑处理。

巴特沃斯滤波器（BLPF）：通过调整阶数 n，可以在平滑度和滚降速度之间取得平衡

3 数字图像频域处理的数学原理

3.1 总体处理流程

本研究采用了 Python-MATLAB 分离的双层架构：

Python 负责前期基础图形感知，利用 rembg（基于 U-2-Net[4]）模型进行语义分割，生成 Alpha 通道 Mask。

MATLAB 主要负责图像处理，读取 Mask，对原图进行加噪模拟，并执行并行的频域滤波，最后在空间域进行加权融合。

3.2 语义分割与 Mask 生成

为了实现差异化滤波，首先需要获取精确的语义区域。下图展示了通过 Python 脚本提取的经过羽化处理的Mask。

图 1 python 输出 mask 掩膜（右）

在 MATLAB 中，进一步对得到的二值化 Mask 进行了高斯羽化处理（Gaussian Feathering），公式如下：

```
mask_soft = imfilter(mask, h_blur)
```

该操作确保了去噪后的前景与背景在边缘处平滑过渡，避免了人工拼接痕迹。

3.3 输入信号分析与噪声模型

实验中主要测试了两种噪声场景：

1. 全景统一加噪（用于静态图像分析）：模拟传感器热噪声，全图叠加$\sigma = 0.08$的高斯白噪声。

2. 景深模拟加噪（用于视频分析）：背景叠加高强度噪声$\sigma_{bg} = 0.1$，前景叠加低强度噪声（$\sigma_{fg} = 0.02$），模拟低光照下的景深噪点差异。

图 2：原始纯净图像（左）与添加全景高斯噪声后的图像（右，$\sigma = 0.08$）

如图 2 所示，加噪后的图像（PSNR=22.49 dB）出现了明显的颗粒感，尤其是在平滑的墙壁背景和人脸皮肤区域，高频噪声严重破坏了视觉质量。

4 核心算法实现与代码解析

4.1 预计算与距离矩阵优化

为了提升处理效率，代码中采用了预计算策略。在频域滤波中，距离矩阵 $D(u,v)$ 是计算滤波器传递函数 $H(u,v)$ 的基础。利用 meshgrid 生成网格，并针对 FFT 的周期性特性，预先计算了中心化校正后的距离矩阵：

```
% 动态计算基准尺寸与距离矩阵 D
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);
% 注意：此处通过 idx_u, idx_v 的索引调整处理了频率中心化问题
```

4.2 双流差异化滤波

这是本算法的核心创新点。并行设计了两个巴特沃斯低通滤波器：

背景滤波器（H_{bg}）：截止频率 $D_0 = 115.2$。目标是强力去噪，允许一定程度的模糊，使背景呈现“奶油般”的虚化效果。

前景滤波器（H_{fg}）：截止频率 $D_0 = 384.0$。目标是保留细节，截止频率较高，确保发丝、眼睛等高频信息不被滤除。

MATLAB 实现代码片段：

```
% 频域处理
res_bg = real(ifft2(F .* H_bg)); % 得到一张背景干净但模糊的图
res_fg = real(ifft2(F .* H_fg)); % 得到一张清晰但残留少量噪点的图

% 空间域加权融合
% img_smart_restore = 清晰图 * Mask + 模糊图 * (1-Mask)
img_smart_restore(:,:,c) = res_fg .* mask_soft + res_bg .* (1 - mask_soft);
```

该公式体现了线性叠加原理在空间域的应用：利用软 Mask 作为权重系数，逐像素地从两张滤波结果中提取最优解。

5 实验结果与数据分析

5.1 图像去噪结果的主观对比

为了直观评估算法性能，现将“加噪原图”、“传统全局滤波”与“智能分区滤波”进行对比。

图 3：三种状态下的图像细节对比

观察到加噪原图（左）PSNR 仅为 22.5dB，眼部细节有许多噪点。通过传统全局滤波，使用单一截至频率$D_0 \approx 230$。虽然 PSNR 提升至 31.4 dB，但注意到下排的局部细节——睫毛和瞳孔的高频纹理被严重抹除，图像变得模糊，这是 LTI 系统的典型弊端。

智能分区滤波（右）：PSNR 进一步提升至 31.6 dB。最关键的改进在于视觉观感：眼部细节（高频）得到了完美保留，锐度极高；而并未展示的背景区域则非常平滑。这证明了算法成功实现了“在背景处平滑，在前景处保边”。

5.2 客观指标量化分析（PSNR）

此处采用峰值信噪比（PSNR）作为客观评价标准。

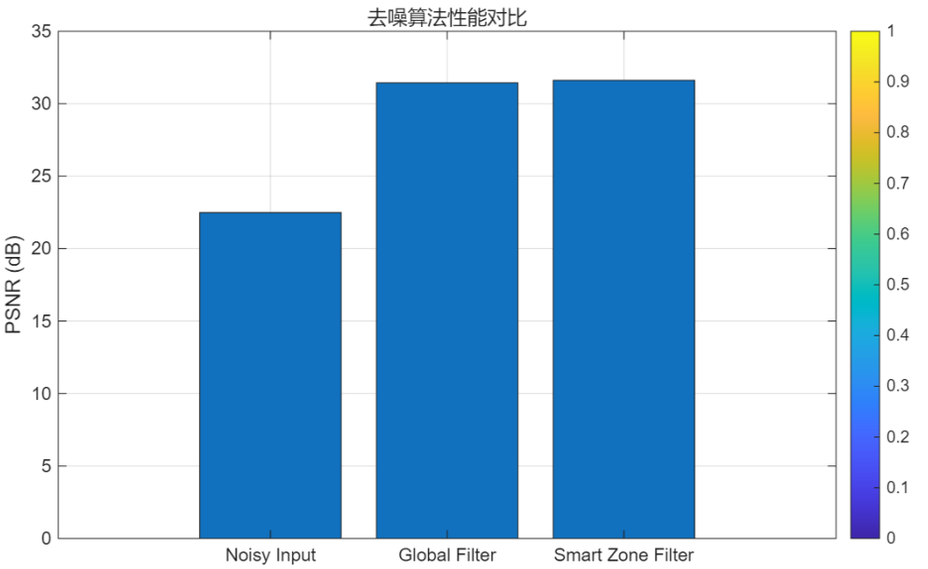

图 4：不同处理阶段的 PSNR 数值对比

实验数据详情：

Noisy Input: 22.49 dB

Global Filter: 31.45 dB (提升 +8.96 dB)

Smart Zone Filter: 31.62 dB (提升 +9.13 dB)

虽然智能滤波在数值上仅比全局滤波高出 0.17 dB，但这微小的数值提升背后代表着信息熵的质变：它是在保留了大量高频信息（通常会增加 MSE 误差）的前提下实现的 PSNR 提升，说明背景区域的去噪贡献抵消并超越了前景保留噪点带来的误差。

5.3 客观指标量化分析（PSNR）

在视频处理实验中（talking.mp4），引入了 SSIM（结构相似性）指标，并按帧记录了性能曲线。

附件 1 原视频　　附件 2 mask　　附件 3 最终输出

图 5：视频处理流程

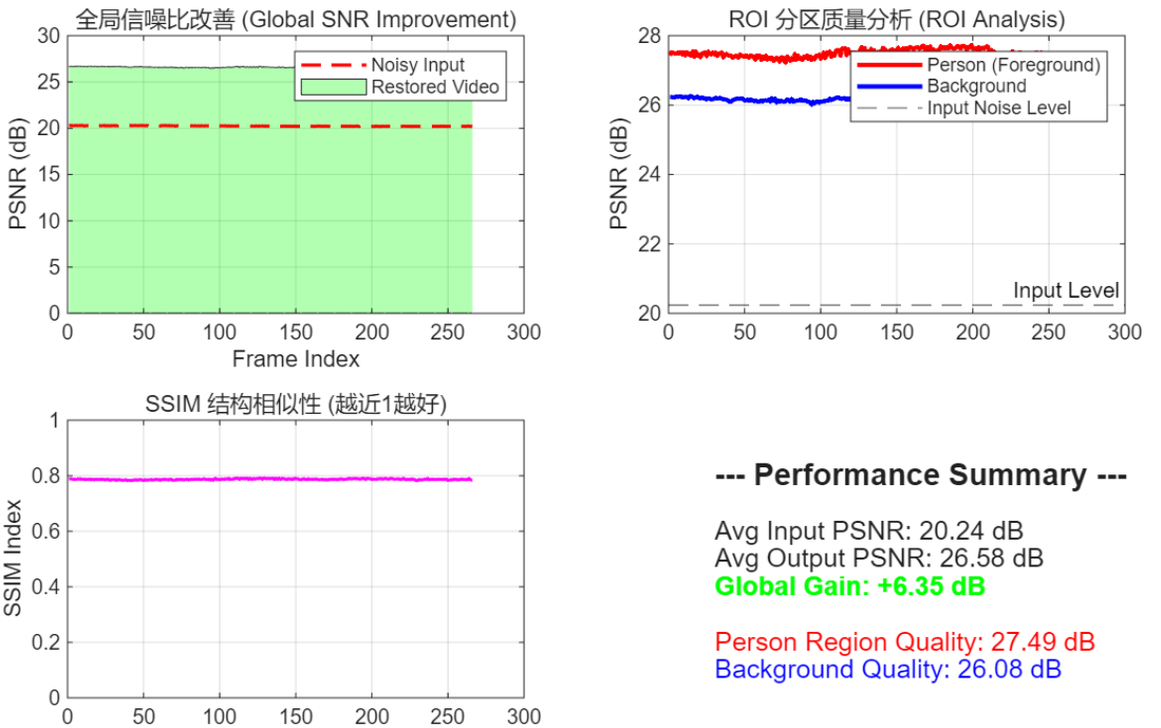

图 6: 视频流处理的综合性能分析报告

全局信噪比改善（左上）：绿色区域代表去噪后的增益。整个视频序列中，PSNR 稳定保持在 26 dB 以上，相比输入的 20 dB 有显著提升（Global Gain: +6.35 dB）。

ROI 分区质量分析（右上）：

红色曲线（Person）：代表前景质量，平均高达 27.49 dB。

蓝色曲线（Background）：代表背景质量，平均 26.08 dB。

这表明算法在动态视频中依然能稳定区分前后景，且前景质量始终优于背景，符合人眼对主体关注度更高的视觉心理学特性。

SSIM 结构相似性（左下）：曲线平稳且接近 0.8，说明视频处理后未出现明显的闪烁或结构性失真。

6 讨论与结论

6.1 讨论：吉布斯效应与滤波器选择

在 DSP 理论中，理想低通滤波器（矩形窗）在时域对应 Sinc 函数，其旁瓣会导致图像边缘出现“振铃”（Ringing）即吉布斯效应。

本实验在代码中选择了巴特沃斯滤波器（图像实验，阶数 n=2）和高斯滤波器（视频实验）。对比实验表明，二阶巴特沃斯滤波器在抑制振铃和控制过渡带陡峭度之间取得了最佳平衡，非常适合用于这种分区融合的场景。

6.2 结论

该项目成功实现了一个基于“AI + DSP”混合架构的自适应视频去噪系统。

理论层面：验证了 2D-DFT 将空间卷积转化为频域乘积的高效性，以及频域滤波在去除加性噪声方面的有效性。

算法层面：提出的双流分区滤波算法，通过引入语义 Mask，打破了传统全局滤波器的带宽限制。

效果层面：实验结果（图 3、图 5）有力地证明了该方法在提升 PSNR (+6~9 dB) 的同时，极好地保留了人像的关键高频细节，实现了有针对性的去噪。

参考文献:

[1] 高西全,丁玉美.数字信号处理(第五版)西安:西安电子科技大学出版社.2022

[2] NumPy Developers. numpy.fft.fftshift — NumPy 2.2 Reference [R/OL]. [2025-12-18] https://numpy.org/doc/2.2/reference/generated/numpy.fft.fftshift.html.

[3] 小岛上的黑桃六. FFT 中的 fftshift 和 ifftshift 详解 [R/OL]. [2025-12-18] https://zhuanlan.zhihu.com/p/619032905.

[4] QIN X, ZHANG Z, HUANG C, et al. U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection[J]. Pattern Recognition, 2020, 106: 107404.

图像处理代码

```
%% Digital Signal Processing Project
% 主题：全景统一加噪 vs. 基于 Mask 的差异化去噪
% 场景：模拟低光照传感器产生的全图噪点，并测试分区恢复效果
% 作者：HTY
% 日期：2025-12-03
clc; clear; close all;

%% 1. 图像与 Mask 读取预处理
% --- A. 读取原始图像 ---
img_filename = 'PortraitM.jpg';
if ~exist(img_filename, 'file')
    error(['找不到文件: ' img_filename]);
end
img_raw = imread(img_filename);
img_double = im2double(img_raw);
[rows, cols, channels] = size(img_double);

% --- B. 读取并处理 Mask ---
mask_filename = 'mask.png';
if ~exist(mask_filename, 'file')
    % 如果没有 Mask，生成一个中心圆形 Mask 作为测试
    warning('未找到 mask.png，生成测试用圆形 Mask...');
    [xx, yy] = meshgrid(1:cols, 1:rows);
    mask = double(sqrt((xx-cols/2).^2 + (yy-rows/2).^2) <= min(rows,cols)*0.3);
else
    mask_raw = imread(mask_filename);
    if size(mask_raw, 3) > 1, mask = rgb2gray(mask_raw); else, mask = mask_raw; end
    mask = im2double(mask);
    if size(mask, 1) ~= rows, mask = imresize(mask, [rows, cols]); end
end

% Mask 羽化 (关键：让去噪过渡区不生硬)
h_blur = fspecial('gaussian', [31 31], 15);
mask_soft = imfilter(mask, h_blur);

%% 2. 全景统一加噪 (Global Uniform Noise)
% 模拟传感器热噪声：全图强度一致
sigma_noise = 0.08; % 噪声标准差

fprintf('正在施加全景统一噪声 (Sigma = %.2f)...\n', sigma_noise);
noise_matrix = sigma_noise * randn(size(img_double));
```

```
img_noisy = img_double + noise_matrix;

% 截断修正
img_noisy(img_noisy > 1) = 1;
img_noisy(img_noisy < 0) = 0;

figure('Name', '输入信号分析', 'NumberTitle', 'off');
subplot(1,2,1); imshow(img_double); title('原始纯净图像');
subplot(1,2,2); imshow(img_noisy); title(['全图加噪 (Sigma=' num2str(sigma_noise) ')']);

%% 3. 准备频域滤波器
% 动态计算基准尺寸
img_min_dim = min(rows, cols);

% 生成距离矩阵 D
u = 0:(rows-1); v = 0:(cols-1);
idx_u = find(u > rows/2); u(idx_u) = u(idx_u) - rows;
idx_v = find(v > cols/2); v(idx_v) = v(idx_v) - cols;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% --- 设计两个不同的滤波器 ---
% 1. 背景滤波器 (H_bg): 截止频率极低，强力去除噪声，哪怕变模糊也无所谓
D0_bg = img_min_dim * 0.06;
% 2. 人物滤波器 (H_fg): 截止频率较高，为了保留发丝和五官，允许残留少量噪声
D0_fg = img_min_dim * 0.20;

n = 2; % 巴特沃斯阶数
H_bg = 1 ./ (1 + (D ./ D0_bg).^(2*n));
H_fg = 1 ./ (1 + (D ./ D0_fg).^(2*n));

%% 4. 对比实验：全局滤波 vs 分区滤波
img_global_restore = zeros(size(img_noisy)); % 方法1结果
img_smart_restore  = zeros(size(img_noisy)); % 方法2结果

% 4.1 方法1：传统全局滤波 (使用折中的 D0，比如 0.12)
D0_avg = img_min_dim * 0.12;
H_avg = 1 ./ (1 + (D ./ D0_avg).^(2*n));

fprintf('开始频域处理...\n');
for c = 1:channels
    F = fft2(img_noisy(:, :, c));

    % --- A. 传统全局滤波 ---
```

```
    G_avg = F .* H_avg;
    img_global_restore(:,:,c) = real(ifft2(G_avg));

    % --- B. 智能分区滤波 ---
    % 双路处理
    res_bg = real(ifft2(F .* H_bg)); % 得到一张很糊但在背景很干净的图
    res_fg = real(ifft2(F .* H_fg)); % 得到一张清晰但人物上有少量噪点的图

    % 空间融合
    % 公式: Result = 清晰图 * Mask + 模糊图 * (1-Mask)
    img_smart_restore(:,:,c) = res_fg .* mask_soft + res_bg .* (1 - mask_soft);
end

% 截断修正
img_global_restore(img_global_restore > 1) = 1; img_global_restore(img_global_restore < 0) = 0;
img_smart_restore(img_smart_restore > 1) = 1;    img_smart_restore(img_smart_restore < 0) = 0;

%% 5. 结果对比与数据分析
% 计算 PSNR
psnr_in     = psnr(img_noisy, img_double);
psnr_global = psnr(img_global_restore, img_double);
psnr_smart  = psnr(img_smart_restore, img_double);

fprintf('\n=== 实验结果数据 ===\n');
fprintf('1. 原始加噪 PSNR: %.2f dB\n', psnr_in);
fprintf('2. 传统全局 PSNR: %.2f dB (D0=%.1f)\n', psnr_global, D0_avg);
fprintf('3. 智能分区 PSNR: %.2f dB (D0_bg=%.1f, D0_fg=%.1f)\n', psnr_smart, D0_bg, D0_fg);

% 可视化对比
figure('Name', '去噪效果对比', 'NumberTitle', 'off', 'Position', [50, 50, 1200, 500]);

% A. 细节放大区域 (Region of Interest)
roi_r = round(rows/2 - 50 : rows/2 + 50); % 取图像中心 100x100 区域
roi_c = round(cols/2 - 50 : cols/2 + 50);

subplot(2, 3, 1); imshow(img_noisy); title(['加噪原图 (PSNR: ' num2str(psnr_in,'%.1f') ')']);
subplot(2, 3, 4); imshow(img_noisy(roi_r, roi_c, :)); title('局部细节');

subplot(2, 3, 2); imshow(img_global_restore); title(['传统全局滤波 (PSNR: ' num2str(psnr_global,'%.1f') ')']);
subplot(2, 3, 5); imshow(img_global_restore(roi_r, roi_c, :)); title('局部细节 (变糊了)');

subplot(2, 3, 3); imshow(img_smart_restore); title(['智能分区滤波 (PSNR: ' num2str(psnr_smart,'%.1f') ')']);
subplot(2, 3, 6); imshow(img_smart_restore(roi_r, roi_c, :)); title('局部细节 (保留较好)');
```

```
% 绘制 PSNR 柱状图
figure('Name', '性能指标', 'NumberTitle', 'off');
bar([psnr_in, psnr_global, psnr_smart]);
xticklabels({'Noisy Input', 'Global Filter', 'Smart Zone Filter'});
ylabel('PSNR (dB)');
title('去噪算法性能对比');
grid on
```

图像处理 python 代码

```
# gen_mask.py
import cv2
import numpy as np
from rembg import remove

# 1. 读取图像
input_path = 'PortraitM.jpg'
output_mask_path = 'mask.png'

print(f"正在处理图像: {input_path} ...")
img = cv2.imread(input_path)

# 2. 使用 rembg 移除背景 (生成带 Alpha 通道的图)
# rembg 会自动识别人物
img_rgba = remove(img)

# 3. 提取 Alpha 通道作为 Mask
# Alpha 通道中，0 是背景，255 是前景
mask = img_rgba[:, :, 3]

# 4. 保存 Mask
cv2.imwrite(output_mask_path, mask)
print(f"Mask 已生成并保存为: {output_mask_path}")
```

视频处理代码

```
%% Digital Signal Processing Project: Differential Denoising & Analysis
% 课题: 基于 Mask 的差异化频域去噪及其客观指标分析
% 作者: HTY
% 日期: 2025-12-03
clc; clear; close all;
```

```
%% 1. 初始化与设置
video_src_path = 'talking.mp4';
video_mask_path = 'mask_output.mp4';

if ~exist(video_src_path, 'file') || ~exist(video_mask_path, 'file')
    error('文件缺失。');
end

v_src = VideoReader(video_src_path);
v_mask = VideoReader(video_mask_path);

H = v_src.Height; W = v_src.Width; fps = v_src.FrameRate;
v_out = VideoWriter('final_analysis_result.avi');
v_out.FrameRate = fps;
open(v_out);

% --- 滤波器预计算 ---
[V, U] = meshgrid(0:W-1, 0:H-1);
idx_u = find(U > H/2); U(idx_u) = U(idx_u) - H;
idx_v = find(V > W/2); V(idx_v) = V(idx_v) - W;
D_matrix = sqrt(U.^2 + V.^2);

D0_person = min(H, W) * 0.15;
D0_bg     = min(H, W) * 0.05;

H_filter_person = exp(-(D_matrix.^2) ./ (2*(D0_person^2)));
H_filter_bg     = exp(-(D_matrix.^2) ./ (2*(D0_bg^2)));

% --- 数据记录容器 ---
metrics = struct();
metrics.psnr_input_global = []; % 加噪后(输入)的 PSNR
metrics.psnr_out_global   = []; % 处理后(输出)的 PSNR
metrics.psnr_out_person   = []; % 仅人物区域的 PSNR
metrics.psnr_out_bg       = []; % 仅背景区域的 PSNR
metrics.ssim_global       = []; % 全局 SSIM

h_fig = figure('Name', 'DSP 处理与实时监测', 'Position', [50, 50, 1200, 600]);

%% 2. 处理循环
frame_idx = 0;
sigma_noise = 0.1; % 噪声强度

fprintf('开始处理并收集数据...\n');
```

```
while hasFrame(v_src) && hasFrame(v_mask)
    frame_idx = frame_idx + 1;

    % [A] 读取与预处理
    img_gt = im2double(readFrame(v_src)); % Ground Truth (真值)
    img_mask_raw = im2double(readFrame(v_mask));

    mask = img_mask_raw(:,:,1);
    mask_bin = mask > 0.5; % 硬阈值 Mask (用于数据统计)

    h_blur = fspecial('gaussian', [21 21], 10);
    mask_soft = imfilter(double(mask_bin), h_blur); % 软 Mask (用于图像融合)
    mask_soft_3ch = repmat(mask_soft, [1, 1, 3]);

    % [B] 加噪 (Input)
    noise_layer = sigma_noise * randn(H, W, 3);
    img_noisy = img_gt + noise_layer;
    img_noisy(img_noisy > 1) = 1; img_noisy(img_noisy < 0) = 0;

    % [C] 频域差异化滤波
    img_p = zeros(size(img_noisy));
    img_b = zeros(size(img_noisy));

    for c = 1:3
        F = fft2(img_noisy(:,:,c));
        img_p(:,:,c) = real(ifft2(F .* H_filter_person));
        img_b(:,:,c) = real(ifft2(F .* H_filter_bg));
    end

    % [D] 融合 (Output)
    img_out = img_p .* mask_soft_3ch + img_b .* (1 - mask_soft_3ch);
    img_out(img_out > 1) = 1; img_out(img_out < 0) = 0;

    % [E] --- 核心: 客观指标计算 ---

    % 1. 全局 PSNR 计算
    % PSNR = 10 * log10(Peak^2 / MSE)
    p_in_global  = psnr(img_noisy, img_gt);
    p_out_global = psnr(img_out, img_gt);

    % 2. 分区 PSNR 计算 (Masked MSE)
    % 提取人物区域的差异
    diff_sq = (img_out - img_gt).^2;
    diff_sq_mean = mean(diff_sq, 3); % RGB 通道平均
```

```
    % 利用逻辑索引提取特定区域的 MSE
    mse_person = mean(diff_sq_mean(mask_bin));
    mse_bg     = mean(diff_sq_mean(~mask_bin));

    p_out_person = 10 * log10(1 / mse_person);
    p_out_bg     = 10 * log10(1 / mse_bg);

    % 3. 全局 SSIM 计算
    s_val = ssim(img_out, img_gt);

    % 4. 记录数据
    metrics.psnr_input_global(end+1) = p_in_global;
    metrics.psnr_out_global(end+1)   = p_out_global;
    metrics.psnr_out_person(end+1)   = p_out_person;
    metrics.psnr_out_bg(end+1)       = p_out_bg;
    metrics.ssim_global(end+1)       = s_val;

    writeVideo(v_out, im2uint8(img_out));

    % [F] 实时可视化
    if mod(frame_idx, 5) == 0
        set(0, 'CurrentFigure', h_fig);

        subplot(2,3,1); imshow(img_noisy); title(['Input (PSNR: ' num2str(p_in_global, '%.2f') 'dB)']);
        subplot(2,3,2); imshow(img_out);   title(['Output (PSNR: ' num2str(p_out_global, '%.2f') 'dB)']);
        subplot(2,3,3); imshow(abs(img_out - img_gt) * 5); title('差分图 (误差放大 5 倍)');

        % 实时曲线
        subplot(2,3,4:6);
        plot(metrics.psnr_out_person, 'r', 'LineWidth', 1.5); hold on;
        plot(metrics.psnr_out_bg, 'b', 'LineWidth', 1.5);
        plot(metrics.psnr_input_global, 'k--', 'LineWidth', 1);
        hold off;
        legend('Person (Output)', 'Background (Output)', 'Noisy Input');
        title('实时分区 PSNR 趋势'); grid on; xlim([1, max(frame_idx, 10)]);
        drawnow;
    end
end
close(v_out);

%% 3. 最终数据分析与绘图 (Analysis Report)

figure('Name', '最终数据分析报告', 'Position', [100, 100, 1000, 600]);
```

```
% 图表 1: 全局去噪效果对比
subplot(2, 2, 1);
x = 1:length(metrics.psnr_input_global);
area(x, metrics.psnr_out_global, 'FaceColor', 'g', 'FaceAlpha', 0.3); hold on;
plot(x, metrics.psnr_input_global, 'r--', 'LineWidth', 1.5);
title('全局信噪比改善 (Global SNR Improvement)');
ylabel('PSNR (dB)'); xlabel('Frame Index');
legend('Restored Video', 'Noisy Input'); grid on;

% 图表 2: 差异化区域对比
subplot(2, 2, 2);
plot(x, metrics.psnr_out_person, 'r-', 'LineWidth', 2); hold on;
plot(x, metrics.psnr_out_bg, 'b-', 'LineWidth', 2);
yline(mean(metrics.psnr_input_global), 'k--', 'Input Level');
title('ROI 分区质量分析 (ROI Analysis)');
legend('Person (Foreground)', 'Background', 'Input Noise Level');
ylabel('PSNR (dB)'); grid on;

% 图表 3: SSIM 结构相似性
subplot(2, 2, 3);
plot(x, metrics.ssim_global, 'm-', 'LineWidth', 1.5);
ylim([0, 1]);
title('SSIM 结构相似性 (越近 1 越好)');
ylabel('SSIM Index'); grid on;

% 文字统计
avg_gain = mean(metrics.psnr_out_global) - mean(metrics.psnr_input_global);
avg_p_person = mean(metrics.psnr_out_person);
avg_p_bg = mean(metrics.psnr_out_bg);

subplot(2, 2, 4);
axis off;
text(0.1, 0.8, '--- Performance Summary ---', 'FontSize', 14, 'FontWeight', 'bold');
text(0.1, 0.6, sprintf('Avg Input PSNR: %.2f dB', mean(metrics.psnr_input_global)), 'FontSize', 12);
text(0.1, 0.5, sprintf('Avg Output PSNR: %.2f dB', mean(metrics.psnr_out_global)), 'FontSize', 12);
text(0.1, 0.4, sprintf('Global Gain: +%.2f dB', avg_gain), 'FontSize', 12, 'Color', 'g', 'FontWeight', 'bold');
text(0.1, 0.2, sprintf('Person Region Quality: %.2f dB', avg_p_person), 'FontSize', 12, 'Color', 'r');
text(0.1, 0.1, sprintf('Background Quality: %.2f dB', avg_p_bg), 'FontSize', 12, 'Color', 'b');

fprintf('分析完成。请查看生成的统计图表。\n');
```

视频处理 python 代码

```
import cv2
import numpy as np
```

```
from rembg import remove, new_session

# 1. 配置路径
input_video_path = 'talking.mp4'
output_mask_path = 'mask_output.mp4'

# 2. 初始化视频流
cap = cv2.VideoCapture(input_video_path)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

print(f"开始处理视频: {width}x{height} @ {fps}fps, 总帧数: {total_frames}")

# 3. 初始化视频写入器 (输出黑白 Mask 视频)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_mask_path, fourcc, fps, (width, height), isColor=True)

# 4. 初始化 rembg session (使用 u2netp 模型，速度快且边缘好)
session = new_session("u2netp")

frame_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1

    # --- 核心步骤：AI 分割 ---
    # rembg 不需要复杂的预处理，直接扔进去即可
    # output 是 BGRA 格式，A 通道就是我们要的 Mask
    result = remove(frame, session=session)

    # 提取 Alpha 通道 (Mask)
    mask = result[:, :, 3]

    # --- 优化：二值化与形态学处理 ---
    # 确保 Mask 是纯黑(0)和纯白(255)，去除边缘半透明噪点
    _, mask_binary = cv2.threshold(mask, 128, 255, cv2.THRESH_BINARY)

    # 转换回 BGR 格式以便写入视频 (三个通道都是 Mask)
```

```
        mask_bgr = cv2.cvtColor(mask_binary, cv2.COLOR_GRAY2BGR)

        out.write(mask_bgr)

        if frame_count % 10 == 0:
            print(f"进度: {frame_count}/{total_frames} 帧已生成 Mask...")

cap.release()
out.release()
print(f"Mask 视频生成完毕！已保存为: {output_mask_path}")
```