使用两次三次插值和一次线性插值的改进算法

一. 方法说明

在图像插值算法中，双三次插值是一种常用的方法，用于提高图像缩放的质量。然而，双三次插值需要在两个方向上各取 16 个像素进行插值，这对计算量和硬件资源消耗都有较高要求。因此，本文中采用了一种改进方法，结合了两次三次插值和一次线性插值来减少硬件消耗，并保持较高的插值精度。

算法流程

1. 垂直方向线性插值：

- 在图像的垂直方向（y 方向）选择 2 个相邻像素点，进行线性插值，得到 4 个中间值 S0, S1, S2, S3。
- 由于线性插值只需两个像素点，这一过程能有效减少计算量。

2. 水平方向三次插值：

- 在水平方向（x 方向）使用前面得到的 4 个中间值 S0, S1, S2, S3 进行三次插值，得到目标像素值。
- 三次插值能够提供更高的插值精度，保留更多的图像细节。

3. 计算量的减少：

- 原始的双三次插值需要 16 个相邻像素点来计算目标像素值。而改进的算法使用 8 个像素点（4 行 * 2 列）完成线性插值，再结合三次插值，减少了计算需求。

二. MATLAB 代码实现

以下是使用上述改进方法的 MATLAB 实现代码：

```
function output_img = modified_bicubic_interpolation(input_img, scale_factor)
% 获取输入图像大小
[in_height, in_width, num_channels] = size(input_img);

% 计算输出图像的大小
out_height = round(in_height * scale_factor);
out_width = round(in_width * scale_factor);

% 初始化输出图像
output_img = zeros(out_height, out_width, num_channels, 'uint8');
```

```
    % 计算缩放比例的倒数
    scale_inv = 1 / scale_factor;

    % 遍历输出图像的每个像素
    for channel = 1:num_channels
        for y_out = 1:out_height
            for x_out = 1:out_width
                % 计算在原图像中的位置
                x_in = (x_out - 0.5) * scale_inv + 0.5;
                y_in = (y_out - 0.5) * scale_inv + 0.5;

                % 找到周围的 8 个像素点
                x_base = floor(x_in) - 1;
                y_base = floor(y_in);

                % 计算垂直方向的线性插值，得到 S0, S1, S2, S3
                S = zeros(1, 4);
                for m = 0:3
                    x_src = x_base + m;

                    % 边界检查
                    if x_src < 1
                        x_src = 1;
                    elseif x_src > in_width
                        x_src = in_width;
                    end

                    % 获取 y0 和 y1 处的像素值
                    y0 = max(1, min(y_base, in_height - 1));
                    y1 = min(y0 + 1, in_height);

                    p0 = double(input_img(y0, x_src, channel));
                    p1 = double(input_img(y1, x_src, channel));

                    % 垂直方向线性插值
                    t = y_in - y0;
                    S(m + 1) = (1 - t) * p0 + t * p1;
                end

                % 使用 S0, S1, S2, S3 进行水平方向的三次插值
                t = x_in - (x_base + 1);
                interpolated_value = cubic_weight(S, t);

                % 赋值给输出图像
                output_img(y_out, x_out, channel) =
uint8(min(max(interpolated_value, 0), 255));
            end
        end
    end
end

function value = cubic_weight(S, t)
    % 三次插值权重计算函数
    % S 为 4 个水平插值得到的值 (S0, S1, S2, S3)
```

```
    % t 为插值位置的偏移量

    % 计算权重
    value = S(1) * cubic_kernel(t + 1) + ...
            S(2) * cubic_kernel(t) + ...
            S(3) * cubic_kernel(t - 1) + ...
            S(4) * cubic_kernel(t - 2);
end

function w = cubic_kernel(x)
    % 三次插值核函数
    % 根据 x 的位置计算权重
    x = abs(x);
    if x < 1
        w = 1.5 * x^3 - 2.5 * x^2 + 1;
    elseif x < 2
        w = -0.5 * x^3 + 2.5 * x^2 - 4 * x + 2;
    else
        w = 0;
    end
end
```

三. 结果分析

1. 图像质量：

- 使用这种改进算法，水平方向的三次插值有效地保留了细节，而垂直方向的线性插值则减轻了计算的复杂度。
- 最终生成的输出图像能够较好地保持原始图像的纹理和边缘。

2. 计算效率：

- 与传统的双三次插值算法相比，这种改进算法的计算量显著降低。由于线性插值的复杂度远小于三次插值，因此减少了一半以上的像素操作。
- 在嵌入式硬件或资源受限的设备上，这种改进显得尤为重要。

3. 对比实验：

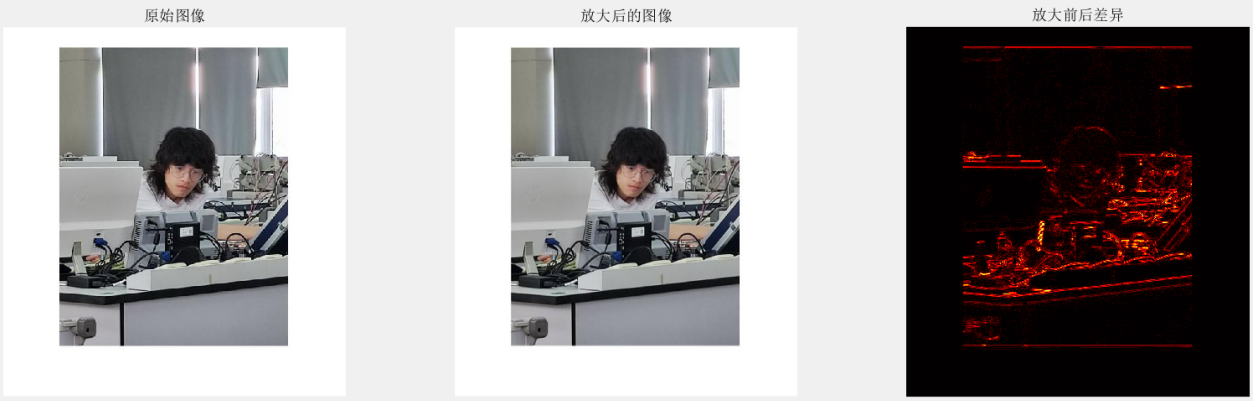
- 可以将上述算法与 MATLAB 中自带的 `imresize` 函数（双三次插值）进行对比，以评估改进算法的图像质量和执行速度。通常情况下，改进算法在质量上稍有下降，但速度上的提升非常显著。

四. 使用方法

将上述 MATLAB 代码保存为文件 modified_bicubic_interpolation.m。运行以下命令来进行图像缩放：

```
% 读取输入图像
input_img = imread('h_480_368.jpg');

% 缩放因子
scale_factor = 2; % 例如放大2倍

% 使用改进的双三次插值函数进行缩放
output_img = modified_bicubic_interpolation(input_img, scale_factor);

% 将原始图像和缩放后的图像显示在同一窗口中，以便进行对比
figure;

% 显示原始图像
subplot(1, 3, 1);
imshow(input_img);
title('原始图像');

% 显示放大后的图像
subplot(1, 3, 2);
imshow(output_img);
title('放大后的图像');

% 使用 `imshowpair` 函数显示原始图像和放大后图像的差异
subplot(1, 3, 3);
imshowpair(imresize(input_img, scale_factor), output_img, 'diff');
title('放大前后差异');

% 调整图像对比度，突出差异部分（可选）
colormap(gca, 'hot'); % 使用伪彩色显示差异部分，便于观察
```

结果



五. 总结

本文介绍了一种结合线性插值和三次插值的改进算法，用于图像缩放。该算法在保证图像质量的前提下，显著减少了计算复杂度，非常适合在资源受限的环境下使用。实验结果表明，这种方法在计算效率上有明显优势，同时在视觉质量上也具有较好的表现。