人工智能 2204 王子旭 U202215150

一、 傅里叶变换与傅里叶系数的物理含义

1.1 傅里叶变换

傅里叶变换是一种线性变换，通常定义为一种积分变换。其基本思想是一个函数可以用无穷多个正弦函数的线性组合来逼近，从而将一个信号从时域转换为频域，通过频谱可以看出一个信号由哪些频率的成分构成，以及每个频率成分的强弱（即振幅）和相位。

1.2 傅里叶系数

由傅里叶级数得到的展开式如下，其中a_0是常数项，代表了信号的直流分量或在一个周期内的平均值，a_n表示信号某个余弦分量的振幅，振幅较大意味着原信号在该频率上有较大的能量，b_n表示不同频率的正弦信号的强度大小

$$f(x) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right)$$

二、 图片旋转和放大

2.1 灰度图转换

基本思想是根据人眼对不同颜色的敏感程度对 RGB 三个通道的值进行加权平均得到灰度值

2.1.1 代码

```
function grayImage = rgbToGray(colorImage)
    % 获取图像的尺寸
    [rows, cols, ~] = size(colorImage);

    % 初始化灰度图像矩阵
    grayImage = zeros(rows, cols);

    % 使用加权平均计算每个像素的灰度值
    for row = 1:rows
        for col = 1:cols
            % 获取像素的 R、G、B 通道值
            pixel = colorImage(row, col, :);
            R = double(pixel(1));
            G = double(pixel(2));
            B = double(pixel(3));
```

```
            % 计算灰度值
            grayValue = 0.2989 * R + 0.5870 * G + 0.1140 * B;
            % 将灰度值写入灰度图像矩阵
            grayImage(row, col) = grayValue;
        end
    end
    imwrite(grayImage,'E:\华中科技大学\大三\Output\gray1.jpg');
end
```

2.1.2 实验结果

使用 matlab 自带库中的转换函数得到的结果与利用上述方法得到的结果几乎没有差别

2.2 旋转

2.2.1 代码

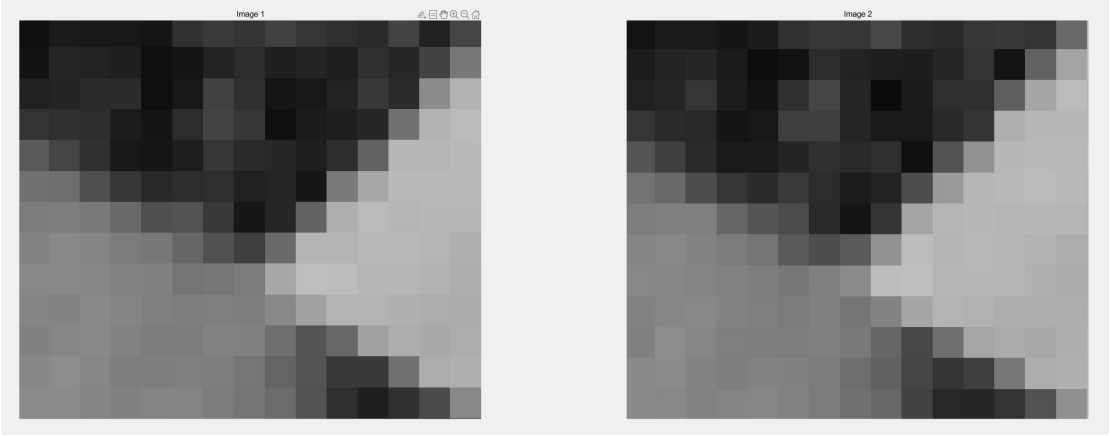
```
function rotatedImage = myRotateImage(inputImage, angleDegrees)
    % 读取输入图像
    originalImage = imread(inputImage);
    % 将角度转换为弧度
    angleRadians = deg2rad(angleDegrees);

    % 计算旋转矩阵
    rotationMatrix = [cos(angleRadians) -sin(angleRadians); sin(angleRadians)
cos(angleRadians)];

    % 计算输出图像的大小
    [rows, cols, ~] = size(originalImage);
```

```
    newRows = ceil(rows * abs(cos(angleRadians)) + cols *
abs(sin(angleRadians)));
    newCols = ceil(rows * abs(sin(angleRadians)) + cols *
abs(cos(angleRadians)));

    % 创建一个新图像，用于存储旋转后的图像
    rotatedImage = uint8(zeros(newRows, newCols, 3));
    % 扩充原始图像，防止旋转后超出 boundary
    disp_col = 0;
    disp_row = 0;
    if angleRadians > 0 % 向下
        disp_col = rows * abs(sin(angleRadians));
        disp_row = 0;

    elseif angleRadians < 0 % 向上
        disp_col = 0;
        disp_row = cols * abs(sin(angleRadians));
    end
    % 计算旋转后的图像
    for row = 1:rows
        for col = 1:cols
            % 计算原始图像中的对应像素位置
            newRow = round(row * cos(angleRadians) + col * sin(angleRadians));
            newCol = round(-row * sin(angleRadians) + col * cos(angleRadians));

            % 检查像素是否在原始图像范围内
            % 复制像素值到旋转后的图像
            if angleDegrees == 90 || angleDegrees == -90
                 rotatedImage(newRow, newCol, :) = originalImage(row, col, :);
            else
                rotatedImage(newRow+ceil(disp_row), newCol+ceil(disp_col), :)
= originalImage(row, col, :);

            end
        end
    end

    % 显示旋转后的图像
    imshow(rotatedImage);

    % 保存旋转后的图像
    imwrite(rotatedImage, 'E:\华中科技大学\大三\Output\rotated_image0.jpg');
end
```

2.2.2 实验结果

使用自己的旋转函数和 matlab 自带旋转函数得到的图像如下，其中 Image1 为自带库函数，Image2 为手写旋转函数

对比边缘细节可以发现，matlab 使用的函数在边缘的过渡处理上更加细节



2.3 放大

2.3.1 代码

```
nearest_img_2x = imresize(gray_img1, 2, 'nearest');
bilinear_img_2x = imresize(gray_img1, 2, 'bilinear');
nearest_img_4x = imresize(gray_img1, 4, 'nearest');
bilinear_img_4x = imresize(gray_img1, 4, 'bilinear');
% 显示放大后的图像
subplot(2,2,1), imshow(nearest_img_2x), title('Nearest 2x');
subplot(2,2,2), imshow(bilinear_img_2x), title('Bilinear 2x');
subplot(2,2,3), imshow(nearest_img_4x), title('Nearest 4x');
subplot(2,2,4), imshow(bilinear_img_4x), title('Bilinear 4x');

input = 'E:\华中科技大学\大三\Output\gray1.jpg';
```

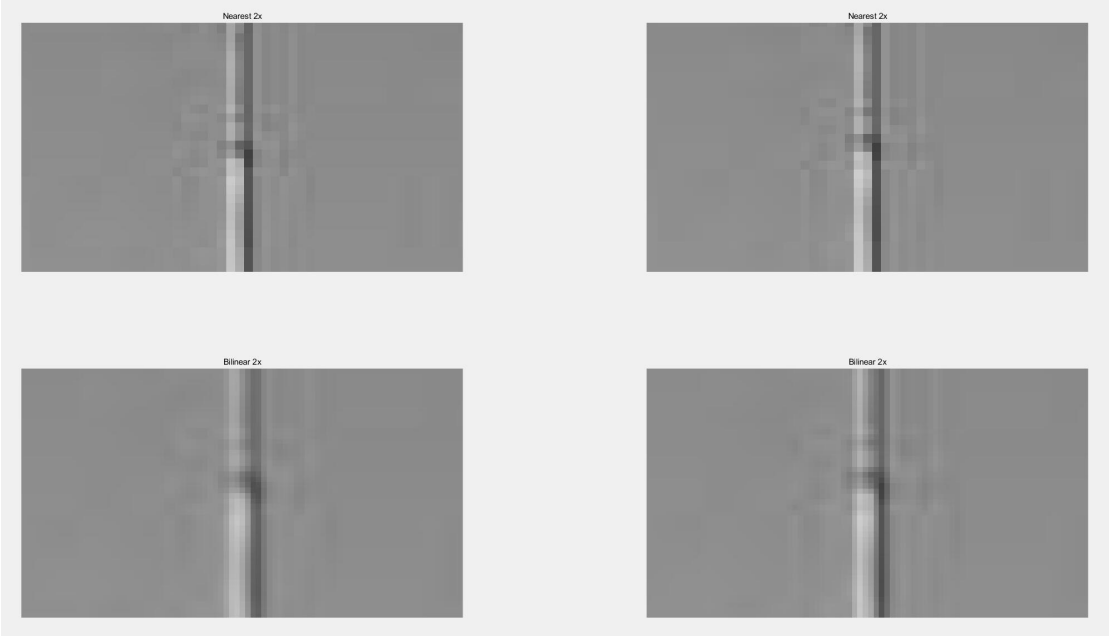
```
% 双线性插值和最近邻插值法
enlargeImage_NN = enlarge_NN(input, 4);
enlargeImage_BI = enlarge_BI(input, 4);
function enlargeImage = enlarge_NN(inputImage, zoom)
   originalImage = imread(inputImage);
   [rows, cols, ~] = size(originalImage);
    newRows = rows * zoom;
    newCols = cols * zoom;
    % 创建一个新图像
    enlargedImage = uint8(zeros(newRows, newCols, 3));
    % 进行最近邻插值
    for newRow = 1:newRows
        for newCol = 1:newCols
            % 计算在原始图像上的坐标
            originalRow = round(newRow / zoom);
            originalCol = round(newCol / zoom);

            % 确保坐标在有效范围内
            originalRow = max(1, min(originalRow, rows));
            originalCol = max(1, min(originalCol, cols));

            % 设置放大后的像素值
            enlargedImage(newRow, newCol, :) = originalImage(originalRow,
originalCol, :);
        end
    end
    imwrite(enlargedImage, 'E:\华中科技大学\大三
\Output\enlarged_NN_image.jpg');  % 可选：保存放大后的图像
    % 设置函数返回值
    enlargeImage = enlargedImage;
end
```

2.3.2 实验结果

使用 matlab 自带函数得到的结果如下

使用自己编写的函数得到的结果如下

直接对比发现两组图大体上差异不大，但放大之后横向比较来看，matlab 的处理结果在细节上要优于我们编写的结果，而且在实际运行时速度明显更快；纵向上比较，使用双线性插值放大的结果要比最近邻插值的边缘部分处理更好



三、 图片傅里叶变换

3.1 方法

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

根据二维离散傅里叶变换公式对图像进行处理，但如果直接使用两层 for 循环，将导致计算复杂度和运行时间大大提高，因此根据傅里叶变换的可分离下，分别对行和列进行两次 1-D 的变换。为了将频率原点移动到图像中心，可以使用 $(-1)^{x+y}$ 乘以$f(x,y)$,将其的傅里叶变换的原点移到相应$N\times N$频率方阵的中心。

3.2 代码

```
function magnitude_spectrum = perform_dft(gray_img)
[M,N]=size(gray_img);
Wm = exp(-1i*2*pi/M);
Wn = exp(-1i*2*pi/N); % 不同 G 中用不同的 W
Em = zeros(M);
En = zeros(N); % E 是辅助计算矩阵
Gm = zeros(M)+Wm;
Gn = zeros(N)+Wn; % G 是计算时要用的矩阵
F = zeros(M,N); % F 是转换到频域的结果
E = zeros(M,N);
% 对 Gm 的计算：循环长度为 M
for row = 0:M-1
```

```
    for col = 0:M-1
        Em(row+1,col+1) = row * col;
        Gm(row+1,col+1) = Gm(row+1,col+1)^Em(row+1,col+1);
    end
end
% 对 Gn 的计算：循环长度为 N
for row = 0:N-1
    for col = 0:N-1
        En(row+1,col+1) = row * col;
        Gn(row+1,col+1) = Gn(row+1,col+1)^En(row+1,col+1);
    end
end
for row =1:M
    %变换到图像中点
    for col =1:N
        E(row,col)=double(gray_img(row,col))*((-1)^(row+col));
    end
end
F = real(Gm*E*Gn);
% 计算幅度谱并进行对数变换以增强可视化效果
magnitude_spectrum = log(1 + abs(F));
% 显示
figure;
imshow(magnitude_spectrum, []);
end
```

3.3 结果分析

得到的频率图结果如下，观察到频率图中心亮度较高，由于我们进行了频率原点的平移操作，所以这代表着原图像的低频成分较丰富，对应原图像的天空、草坪等较为平滑的场景。同时在频率图中也有一部分高频成分，这是由南一楼的窗户、两侧的树木等边缘丰富的高频成分产生的