图像与视频处理实验报告

姓名：张鹏杰 学号：3220231268

一、直方图均衡化

（一）实验说明

直方图均衡化是图像处理领域中利用图像直方图对对比度进行调整的方法，通过这种方法，亮度可以更均匀地在直方图上分布，这样就可以用于增强局部的对比度而不影响整体的对比度。

（二）实验结果

实验结果如图1所示，左侧为原始图像，右侧为直方图均衡化的结果。可见通过直方图均衡化，过亮和过暗的图像都被转化为了可视化效果更好的结果。



图1 直方图均衡化实验结果

（三）算法源码

import numpy as np

import cv2

def histogramEqualization(grey\_image, K):

    # 0. Initialize

    histogram = np.zeros(K)

    height = grey\_image.shape[0]

    width = grey\_image.shape[1]

    # 1. Histogram

    for i in range(height):

        for j in range(width):

            histogram[grey\_image[i][j]] += 1

    # 2. Normalized Histogram

    histogram\_normalized = histogram/(height\*width)

    # 3. Cumulative Histogram

    histogram\_cumulative = np.zeros(K)

    histogram\_cumulative[0] = histogram\_normalized[0]

    for i in range(1,K):

        histogram\_cumulative[i] = histogram\_cumulative[i-1] + histogram\_normalized[i]

    # 4. FSHS

    map = (K-1)\*(histogram\_cumulative-histogram\_cumulative[0]) / (1-histogram\_cumulative[0])

    map = (map+0.5).astype(np.int32)

    # 5. New Image

    new\_image = np.zeros((height, width))

    for i in range(height):

        for j in range(width):

            new\_image[i][j] = map[grey\_image[i][j]]

    return new\_image.astype(np.uint8)

if \_\_name\_\_ == '\_\_main\_\_':

    # 1. MiniTest

    matrix = [[0,2,2,2],

              [0,3,3,2],

              [1,1,4,6],

              [1,1,5,5]]

    grey\_image = np.array(matrix)

    new\_image = histogramEqualization(grey\_image, 8)

    print(new\_image)

    # 2. RealTest

    img\_path = '.\\Histogram Equalization\\TooBright.jpg'

    image = cv2.imread(img\_path)

    grey\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    new\_image = histogramEqualization(grey\_image, 256)

    cv2.imshow('Initial', grey\_image)

    cv2.imshow('Processed', new\_image)

    cv2.waitKey(0)

cv2.imwrite('.\\Histogram Equalization\\TooBright\_hisEqualization.jpg', new\_image)

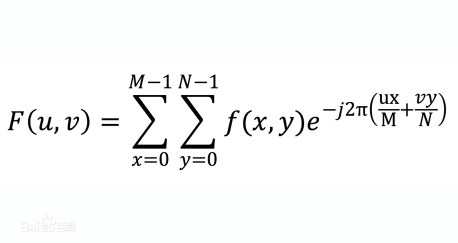
二、频域滤波器

（一）实验说明

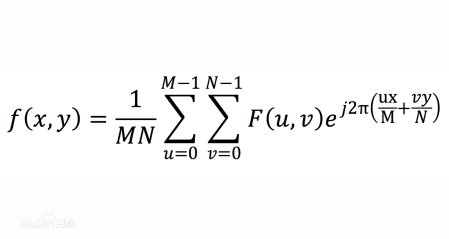
频域滤波是一种基于图像的频率域表示来进行滤波的方法，频域滤波的基本思想是将输入图像的空间信号转化到频率域，然后通过对频率域信号进行滤波来实现图像的增强和去噪。

本实验利用离散傅里叶变换将图像从空域转化到频域，利用低通、高通对图像进行滤波，然后利用离散傅里叶反变换将图像转化回空域，实现频域上的图像滤波。

傅里叶变换采用二维离散傅里叶变换，公式如下所示



反变换公式如下所示：



在代码实现过程中，将二维离散傅里叶变换转化为两个一维的离散傅里叶变换，并利用一维的快速傅里叶变换（FFT）实现，保证了代码的运行速度。

（二）实验结果

如图2所示，利用傅里叶变换将原图从空间域转化到频率域，然后利用低通、高通滤波保留图像的低频、高频信息，产生了正确的滤波结果。

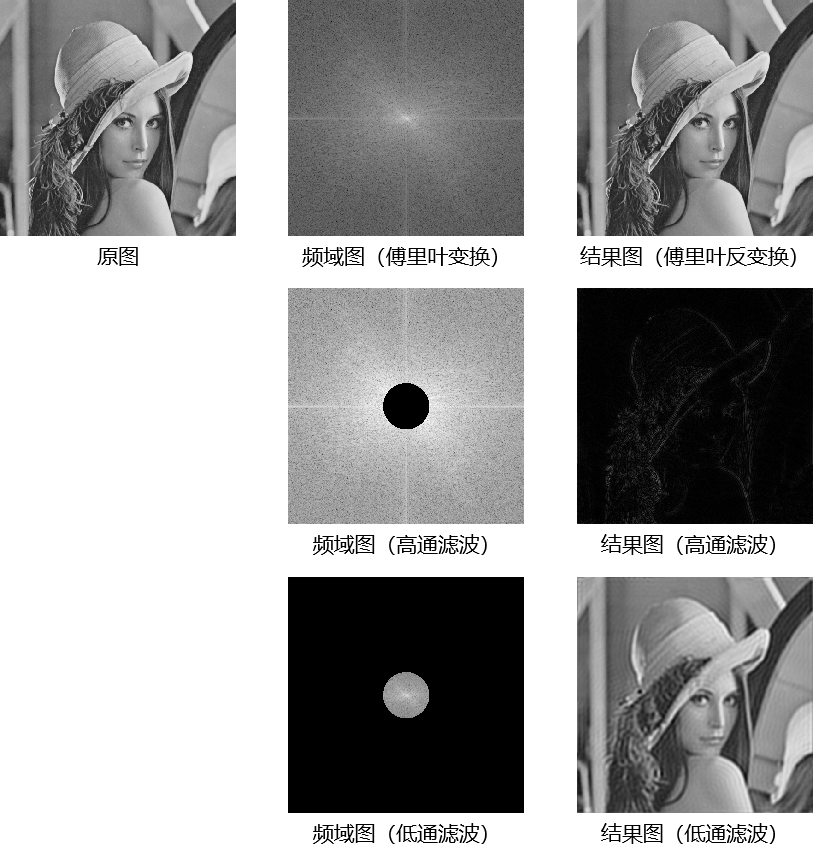


图2 频域滤波实验结果

（三）算法源码

import cv2

import numpy as np

# 快速离散傅里叶变换

def FFT(img):

    img = np.array(img)

    h, w = img.shape

    # 转变为两次一维的FFT

    mid = np.zeros((h, w), dtype=np.complex64)

    for i in range(h):

        mid[i, ...] = FFT\_1D(img[i, ...])

    output = np.zeros((h, w), dtype=np.complex64)

    for j in range(w):

        output[..., j] = FFT\_1D(mid[..., j])

    # 调整中心位置

    output = np.roll(output, h // 2, axis=0)

    output = np.roll(output, w // 2, axis=1)

    return output

# 快速离散傅里叶反变换

def IFFT(img):

    img = np.array(img)

    h, w = img.shape

    # 先将位置调整回来

    img = np.roll(img, w // 2, axis=1)

    img = np.roll(img, h // 2, axis=0)

    # 转变为两次一维的IFFT

    mid = np.zeros((h, w), dtype=np.complex64)

    for i in range(h):

        mid[i, ...] = IFFT\_1D(img[i, ...]) / w

    output = np.zeros((h, w), dtype=np.complex64)

    for j in range(w):

        output[..., j] = IFFT\_1D(mid[..., j]) / h

    return output

# 快速离散傅里叶变换（1维）

def FFT\_1D(f):

    M = len(f)

    K = int(M/2)

    F = np.zeros(M, dtype=np.complex64)

    if M == 1:

        return np.array(f)

    u = np.array([i for i in range(K)])

    F\_even = FFT\_1D([f[index] for index in range(M) if index % 2 == 0])

    F\_odd = FFT\_1D([f[index] for index in range(M) if index % 2 == 1])

    \_F\_odd = F\_odd \* np.exp(-1j \* 2 \* np.pi \* u / M)

    F[0:K] = F\_even + \_F\_odd

    F[K:M] = F\_even - \_F\_odd

    return F

# 快速离散傅里叶反变换（1维）

def IFFT\_1D(f):

    M = len(f)

    K = int(M/2)

    F = np.zeros(M, dtype=np.complex64)

    if M == 1:

        return np.array(f)

    u = np.array([i for i in range(K)])

    F\_even = IFFT\_1D([f[index] for index in range(M) if index % 2 == 0])

    F\_odd = IFFT\_1D([f[index] for index in range(M) if index % 2 == 1])

    \_F\_odd = F\_odd \* np.exp(1j \* 2 \* np.pi \* u / M)

    F[0:K] = F\_even + \_F\_odd

    F[K:M] = F\_even - \_F\_odd

    return F

# 低通滤波器

def low\_filter(img, r=50):

    h, w = img.shape

    hc, wc = h // 2, w // 2

    x, y = np.ogrid[:h, :w]

    # 将距离大于半径的过滤掉

    mask = (x - hc)\*\*2 + (y - wc)\*\*2 <= r \* r

    return img \* mask

# 高通滤波器

def high\_filter(img, r=50):

    h, w = img.shape

    hc, wc = h // 2, w // 2

    x, y = np.ogrid[:h, :w]

    # 将距离小于半径的过滤掉

    mask = (x - hc)\*\*2 + (y - wc)\*\*2 > r \* r

    return img \* mask

def visual\_uv(uv):

    uv\_visual = np.log(1 + abs(uv))

    uv\_visual = (uv\_visual - np.min(uv\_visual))/(np.max(uv\_visual) - np.min(uv\_visual)) \* 255

    return uv\_visual

if \_\_name\_\_ == '\_\_main\_\_':

    # Load Image

    img\_path = '.\\Frequency Filter\\lena.png'

    image = cv2.imread(img\_path)

    grey\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # cv2.imwrite('.\\Frequency Filter\\lena\_grey.png', grey\_image)

    # 计算DFT

    uv = FFT(grey\_image)

    cv2.imwrite('.\\Frequency Filter\\lena\_FFT.png', visual\_uv(uv))

    # 计算IDFT

    iuv = IFFT(uv)

    cv2.imwrite('.\\Frequency Filter\\lena\_IFFT.png', abs(iuv))

    # 低通滤波

    uv\_lowFiltered = low\_filter(uv)

    cv2.imwrite('.\\Frequency Filter\\lena\_lowFiltered\_uv.png', visual\_uv(uv\_lowFiltered))

    xy\_lowFiltered = IFFT(uv\_lowFiltered)

    cv2.imwrite('.\\Frequency Filter\\lena\_lowFiltered\_xy.png', abs(xy\_lowFiltered))

    # 高通滤波

    uv\_highFiltered = high\_filter(uv)

    cv2.imwrite('.\\Frequency Filter\\lena\_highFiltered\_uv.png', visual\_uv(uv\_highFiltered))

    xy\_highFiltered = IFFT(uv\_highFiltered)

    cv2.imwrite('.\\Frequency Filter\\lena\_highFiltered\_xy.png', abs(xy\_highFiltered))