2023 年秋季学期 图像处理 编程作业 02

姓名：梁乐彬 学号：[redacted]

问题 1-通过计算一维傅里叶变换实现图像二维快速傅里叶变换（task1_2.py）

实现一个函数 F = dft2D(f)，其中 f 是一个灰度源图像，F 是其对应的二维快速傅里叶变换(FFT)图像. 具体实现要求按照课上的介绍通过两轮一维傅里叶变换实现。也就是首先计算源图像每一行的一维傅里叶变换，然后对于得到的结果计算其每一列的一维傅里叶变换。如果实现采用 MATLAB，可以直接调用函数 fft 计算一维傅里叶变换。如果采用 Python，请选择并直接调用相应的例如 NumPy 或者 SciPy 软件包中的一维傅里叶变换函数。

dft2D(f)实现流程：

1.对输入图像 f 的每一行执行一维傅里叶变换（一维 FFT），这将产生一个复数数组，其中包含了每一行的频域表示。

2.对上一步得到的每一行频域表示执行一维傅里叶变换，这将产生包含每一列的频域表示的复数数组。

3.返回最终的二维傅里叶变换结果 F，它代表了图像的频域图。

```python
def dft2D(f):
    # 对每一行进行一维FFT
    f_row = np.fft.fft(f, axis=1)

    # 对每一列进行一维FFT
    F = np.fft.fft(f_row, axis=0)

    return F
```

问题 2 图像二维快速傅里叶逆变换（task1_2.py）

实现一个函数 f = idft2D(F)，其中 F 是一个灰度图像的傅里叶变换，f 是其对应的二维快速傅里叶逆变换 (IFFT)图像，也就是灰度源图像. 具体实现要求按照课上的介绍通过类似正向变换的方式实现。

idft2D(F)实现流程：

1.获取输入频域图像 F 的形状，其中 M 和 N 分别表示图像的高度和宽度。

2.对 F 的每一列执行一维逆傅里叶变换（一维 IFFT），这将产生一个复数数组，其中包含了每一列的空间域表示。

3.对上一步得到的每一列的空间域表示执行一维逆傅里叶变换，这将产生包含每一行的空间域表示的复数数组。

4.从复数结果中提取实部部分，因为逆傅里叶变换的结果可能包含复数，但通常图像是实数值。

5.返回最终的二维逆傅里叶变换结果 f，它包含了图像在空间域中的表示。

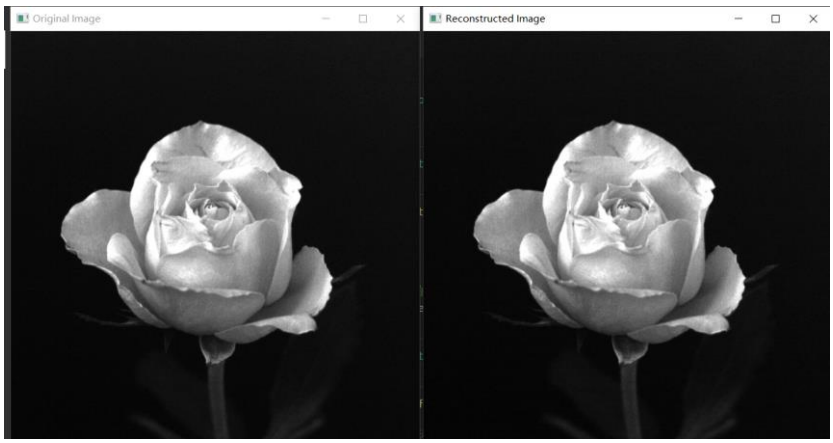
```python
def idft2D(F):
    # 获取频域图像的形状
    M, N = F.shape

    # 对每一列进行一维逆FFT
    f_row = np.fft.ifft(F, axis=0)

    # 对每一行进行一维逆FFT
    f = np.fft.ifft(f_row, axis=1)

    # 取实部部分，因为逆FFT的结果可能包含复数
    f = np.real(f)

    return f
```

实现流程：

1.读取名为'rose512.tif'的图像并将其转换为灰度图像。
2.将像素值范围归一化到[0, 1]，确保像素值都在这个范围内。
3.使用 dft2D 函数将归一化的图像转换为频域图像 F。
4.idft2D 函数将频域图像 F 转换回空域，并存储在 reconstructed_image 中。
5.提取 reconstructed_image 的实部，逆傅里叶变换的结果包含复数部分。

```python
# 读取.tif图像
image = cv2.imread('rose512.tif', cv2.IMREAD_GRAYSCALE)

# 将像素值范围归一化到[0, 1]
normalized_image = image.astype(float) / 255.0

# 使用dft2D将图像转换为频域图像
F = dft2D(normalized_image)

# 使用idft2D将频域图像转换回空域
reconstructed_image = idft2D(F)

# 转换后的图像可能包含复数部分，取实部
reconstructed_image = np.real(reconstructed_image)

# 显示原始图像和转换后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Reconstructed Image', (reconstructed_image * 255).astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

运行效果：



问题 3 测试图像二维快速傅里叶变换与逆变换（task3.py）

对于给定的输入图像 rose512.tif，首先将其灰度范围通过归一化调整到[0,1]. 将此归一化的图像记为 f. 首先调用问题 1 下实现的函数 dft2D 计算其傅里叶变换，记为 F。然后调用问题 2 下的函数 idft2D 计算 F 的傅里叶逆变换，记为 g. 计算并显示误差图像 d = f - g.

实现流程：

1.读取名为'rose512.tif'的图像并将其转换为灰度图像。
2.将像素值范围归一化到[0, 1]，确保所有像素值都在这个范围内。
3.使用 dft2D 函数将归一化的图像转换为频域图像 F。
4.使用 idft2D 函数将频域图像 F 转换回空域，并存储在 g 中。
5.提取 g 的实部，因为逆傅里叶变换的结果可能包含复数部分。
6.计算原始图像 f 和重建图像 g 之间的误差图像 d，以了解还原的准确性。
7.将还原的图像范围从[0, 1]缩放到[0, 255]，以便于显示。

```python
# 读取.tif图像
image = cv2.imread('rose512.tif', cv2.IMREAD_GRAYSCALE)

# 将像素值范围归一化到[0, 1]
f = image.astype(float) / 255.0

# 使用dft2D将图像转换为频域图像
F = dft2D(f)

# 使用idft2D将频域图像转换回空域
g = idft2D(F)

# 转换后的图像可能包含复数部分，取实部
g = np.real(g)

# 计算误差图像
d = f - g

# 将还原的图像范围缩放到[0, 255]
g_scaled = (g * 255).astype(np.uint8)

# 创建一个画板并显示原始图像、还原图像和误差图像
plt.figure(figsize=(12, 4))
plt.subplot(131)
plt.imshow((f * 255).astype(np.uint8), cmap='gray')
plt.title('Original Image (f)')
plt.axis('off')

plt.subplot(132)
plt.imshow(g_scaled, cmap='gray')
plt.title('Reconstructed Image (g)')
plt.axis('off')

plt.subplot(133)
plt.imshow((d * 255).astype(np.uint8), cmap='gray')
plt.title('Error Image (d)')
```

运行效果：

问题 4 计算图像的中心化二维快速傅里叶变换与谱图像 （task4.py）

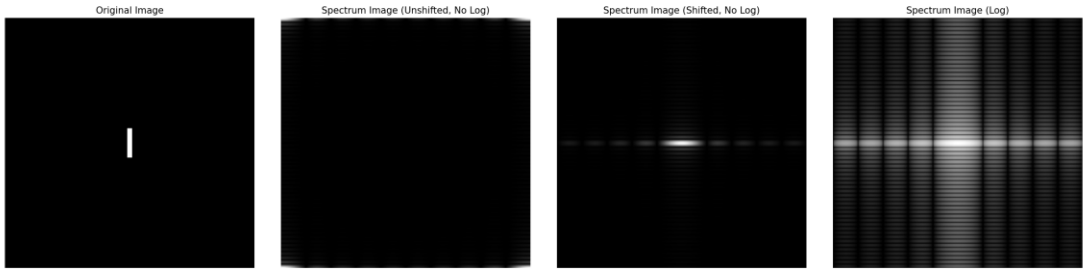
我们的目标是复现下图中的结果。首先合成矩形物体图像，建议图像尺寸为 512×512，矩形位于图像中心，建议尺寸为 60 像素长，10 像素宽，灰度假设已归一化设为 1. 对于输入 图像 f 计算其中心化二维傅里叶变换 F。然后计算对应的谱图像 S = log(1+abs(F)). 显示该谱图像。

实现步骤：

1. 创建一个大小为 512x512 的全黑图像，图像数据类型为浮点数，初始值为 0。这个图像将用作后续操作的基础。
2. 定义了一个矩形的参数，包括矩形的宽度（rect_width）、高度（rect_height）以及位于图像中心的位置（rect_center_x 和 rect_center_y）。
3. 使用矩形的参数，在图像上绘制一个白色矩形。矩形的位置是根据中心坐标和宽度/高度来计算的，因此它位于图像中心。
4. 使用 dft2D 的函数来计算图像的二维傅里叶变换。这个函数将图像从空域转换到频域，产生一个复数数组 F，其中包含了频域表示。
5. 计算频谱的幅度，即取 F 的绝对值，以获得每个频率成分的强度信息。这产生了一个幅度图像 magnitude，其中每个像素表示了相应频率成分的幅度。
6. 将频域图像 F 进行反变换，将其移动回非中心化的形式。这一步骤不改变频域的信息，但会将频域原点移到图像的左上角，以便于显示和分析。
7. 最后，对频谱的幅度进行对数缩放，以便于可视化。这是通过计算 1 + abs(F_unshifted) 的对数来实现的。对数缩放后的频谱图被存储在 S_log 中。

```
# 创建一个空白的图像，大小为512x512，初始值为0（全黑）
image_size = 512
image = np.zeros((image_size, image_size), dtype=float)

# 定义矩形的参数：位于图像中心，60像素长，10像素宽
rect_width = 10
rect_height = 60
rect_center_x = image_size // 2
rect_center_y = image_size // 2

# 在图像上绘制矩形
image[rect_center_y - rect_height // 2: rect_center_y + rect_height // 2,
      rect_center_x - rect_width // 2: rect_center_x + rect_width // 2] = 1.0

# 计算图像的二维傅里叶变换
F = dft2D(image)

# 计算频谱的幅度（绝对值）
magnitude = np.abs(F)

# 不中心化的频谱图
F_unshifted = np.fft.ifftshift(F)

# 对幅度进行对数缩放以便于可视化
S_log = np.log(1 + np.abs(F_unshifted))
```

运行效果：



问题 5 测试更多图像的二维快速傅里叶变换 （task5.py）

计算其他 5 幅图像的二维快速傅里叶变换： house.tif, house02.tif, lena_gray_512.tif, lunar_surface.tif, characters_test_pattern.tif。注意，有些图像的尺寸不是 2 的整数次幂，需要进行相应的像素填补处理。如果图像有多个通道可以选择其中的一个通道进行计算。

实现步骤：

1.指定了一组图像文件的名称。
2.使用 Matplotlib 创建一个包含 5 行和 2 列子图的图形布局。每个子图将包含两个图像：原始图像和对数化的频谱图。
3.使用 enumerate 函数遍历图像文件列表。在循环中，首先读取当前图像文件，将其转换为灰度图像，并存储在 image 变量中。
4.计算填充后的图像尺寸，以确保它们都是 2 的整数次幂。这是通过取图像高度和宽度的对数，然后向上取整，再将结果转换回整数来完成的。这确保了图像在进行傅里叶变换之前的尺寸是 2 的整数次幂。
5.创建一个空白的填充后图像，其尺寸为计算得到的填充后尺寸，数据类型为浮点数。
6.将原始图像制到填充后图像的左上角，确保图像的内容不会变化，但是尺寸会被填充到 2 的。整数次幂。
7.使用您提供的 dft2D 函数计算填充后图像的二维傅里叶变换 F。
8.计算频谱的幅度，即取 F 的绝对值，以获得每个频率成分的强度信息。
9.显示原始图像在子图的第一个位置，使用灰度色彩映射。
10.显示傅里叶变换的幅度谱在子图的第二个位置，同时应用对数变换以便于可视化。
11.最后，通过调用 plt.tight_layout()来确保子图的布局合理，并使用 plt.show()来显示图像。

```python
# 读取图像并进行必要的填充，确保尺寸是2的整数次幂
image_files = ['house.tif', 'house02.tif', 'lena_gray_512.tif', 'lunar_surface.tif', 'characters_test_pattern.tif']

# 创建一个5x2的子图布局
fig, axes = plt.subplots(5, 2, figsize=(10, 15))

for i, image_file in enumerate(image_files):
    image = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)

    # 计算填充后的图像尺寸
    padded_height = 2**int(np.ceil(np.log2(image.shape[0])))
    padded_width = 2**int(np.ceil(np.log2(image.shape[1])))

    # 创建一个空白的填充后图像
    padded_image = np.zeros((padded_height, padded_width), dtype=float)

    # 将原始图像复制到填充后图像的左上角
    padded_image[:image.shape[0], :image.shape[1]] = image

    # 计算图像的二维傅里叶变换
    F = dft2D(padded_image)

    # 计算频谱的幅度（绝对值）
    magnitude = np.abs(F)

    # 显示原始图像
    axes[i, 0].imshow(image, cmap='gray')
    axes[i, 0].set_title('Original Image')
    axes[i, 0].axis('off')

    # 显示傅里叶变换的幅度谱
    axes[i, 1].imshow(np.log(1 + magnitude), cmap='gray')
    axes[i, 1].set_title('Spectrum Image (Log)')
    axes[i, 1].axis('off')
```

运行效果：