**一、实验目的和要求**

1.了解图像压缩的必要性和可行性；

2.了解图像中的冗余、图像的保真度、通用压缩模型；

3.了解DCT频率矩阵、基图像；

4.熟悉DCT变换，变换编码流程；

5.熟悉DFT变换，变换编码流程。

**二、实验主要仪器和设备**

1.计算机；

2.Python与Opencv库；

3.实验所需数据。

**三、实验方法和步骤**

1.打开计算机，启动Python运行环境；

2.采集实验所需图像；

3.完成DCT变换处理，并显示处理结果；

4.完成DFT变换处理，并显示处理结果；

5.完成傅里叶变换处理，并显示运算结果；

6.完成逆傅里叶变换处理，并显示处理结果；

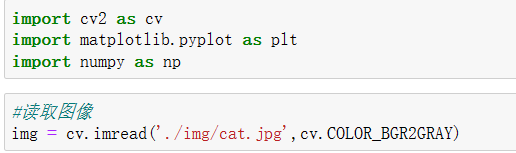
7.记录和整理实验报告。

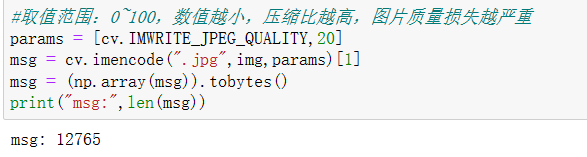
**四、实验结果分析**

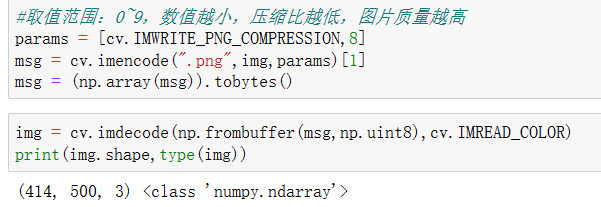
**（实验源代码、实验结果图、实验分析比较、结论）**

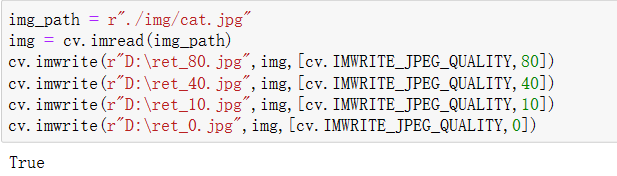
**1. 采集图像：**

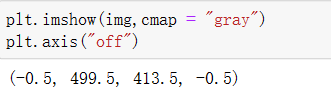
实验源代码：

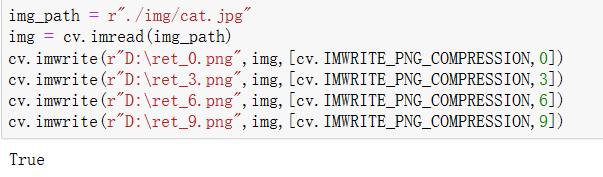






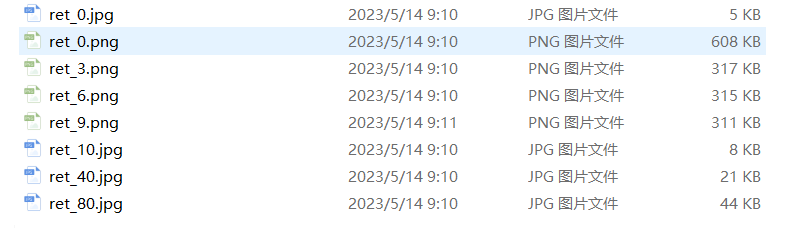






实验结果图：



、



**图1：ret\_80.jpg**



**图2：ret\_40.jpg**



**图3：ret\_10.jpg**



**图4：ret\_0.jpg**



**图5：ret\_9.png**

****

**图6：ret\_6.png**

****

**图7：ret\_3.png**

****

**图8：ret\_0.png**

实验分析比较、结论：

这段代码使用了OpenCV（cv2）和Matplotlib库来读取、处理和保存图像。下面是对代码的分析：

（1）JPEG压缩：

params = [cv.IMWRITE\_JPEG\_QUALITY, 20]

msg = cv.imencode(".jpg", img, params)[1]

msg = np.array(msg).tobytes()

print("msg:", len(msg))

这段代码将图像使用JPEG格式进行压缩，并设置压缩质量为20。cv.imencode函数将图像编码为JPEG格式，并返回一个包含压缩数据的元组。通过索引[1]获取压缩数据，并使用np.array将其转换为NumPy数组，然后使用tobytes转换为字节数组。最后，打印出压缩后的数据长度。

（2）PNG压缩：

params = [cv.IMWRITE\_PNG\_COMPRESSION, 8]

msg = cv.imencode(".png", img, params)[1]

msg = np.array(msg).tobytes()

这段代码将图像使用PNG格式进行压缩，并设置压缩级别为8。与JPEG压缩相似，它将图像编码为PNG格式，获取压缩数据，并转换为字节数组。

（3）图像解码：

img = cv.imdecode(np.frombuffer(msg, np.uint8), cv.IMREAD\_COLOR)

print(img.shape, type(img))

这段代码对之前压缩后的数据进行解码，并将其转换回图像。np.frombuffer将字节数组转换为NumPy数组，然后使用cv.imdecode函数对数据进行解码，指定颜色模式为彩色图像（cv.IMREAD\_COLOR）。最后，打印出图像的形状和数据类型。

综上，我们可以看出两种不同压缩格式在显示同一张图片时，不同的压缩比所表现出来的图片质量。

**2.DCT变换处理：**

实验源代码：

**##离散余弦变换（DCT）**

**#整张图DCT变换**

def whole\_img\_dct(img\_f32):

img\_dct = cv.dct(img\_f32)

img\_dct\_log = np.log(abs(img\_dct))

img\_idct = cv.idct(img\_dct)

return img\_dct\_log,img\_idct

**#分块图DCT变换**

def block\_img\_dct(img\_f32):

height,width = img\_f32.shape[:2]

block\_y = height//8

block\_x = width//8

height\_ = block\_y \* 8

width\_ = block\_x \* 8

img\_f32\_cut = img\_f32[:height\_,:width\_]

img\_dct = np.zeros((height\_,width\_),dtype = np.float32)

new\_img = img\_dct.copy()

for h in range(block\_y):

for w in range(block\_x):

img\_block = img\_f32\_cut[8\*h:8\*(h+1),8\*w:8\*(w+1)]

img\_dct[8\*h:8\*(h+1),8\*w:8\*(w+1)] = cv.dct(img\_block)

dct\_block = img\_dct[8\*h:8\*(h+1),8\*w:8\*(w+1)]

img\_block = cv.idct(dct\_block)

new\_img[8\*h:8\*(h+1),8\*w:8\*(w+1)] = img\_block

img\_dct\_log2 = np.log(abs(img\_dct))

return img\_dct\_log2,new\_img

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

img\_u8 = cv.imread("./img/cat.jpg",0)

img\_f32 = img\_u8.astype(np.float)

img\_dct\_log,img\_idct = whole\_img\_dct(img\_f32)

img\_dct\_log2,new\_img = block\_img\_dct(img\_f32.copy())

plt.figure(6,figsize = (12,8))

plt.subplot(231)

plt.imshow(img\_u8,'gray')

plt.title('original image'),plt.xticks([]),plt.yticks([])

plt.subplot(232)

plt.imshow(img\_dct\_log)

plt.title('DCT'),plt.xticks([]),plt.yticks([])

plt.subplot(233)

plt.imshow(img\_idct,'gray')

plt.title('IDCT'),plt.xticks([]),plt.yticks([])

plt.subplot(234)

plt.imshow(img\_dct\_log2)

plt.title('block\_DCT'),plt.xticks([]),plt.yticks([])

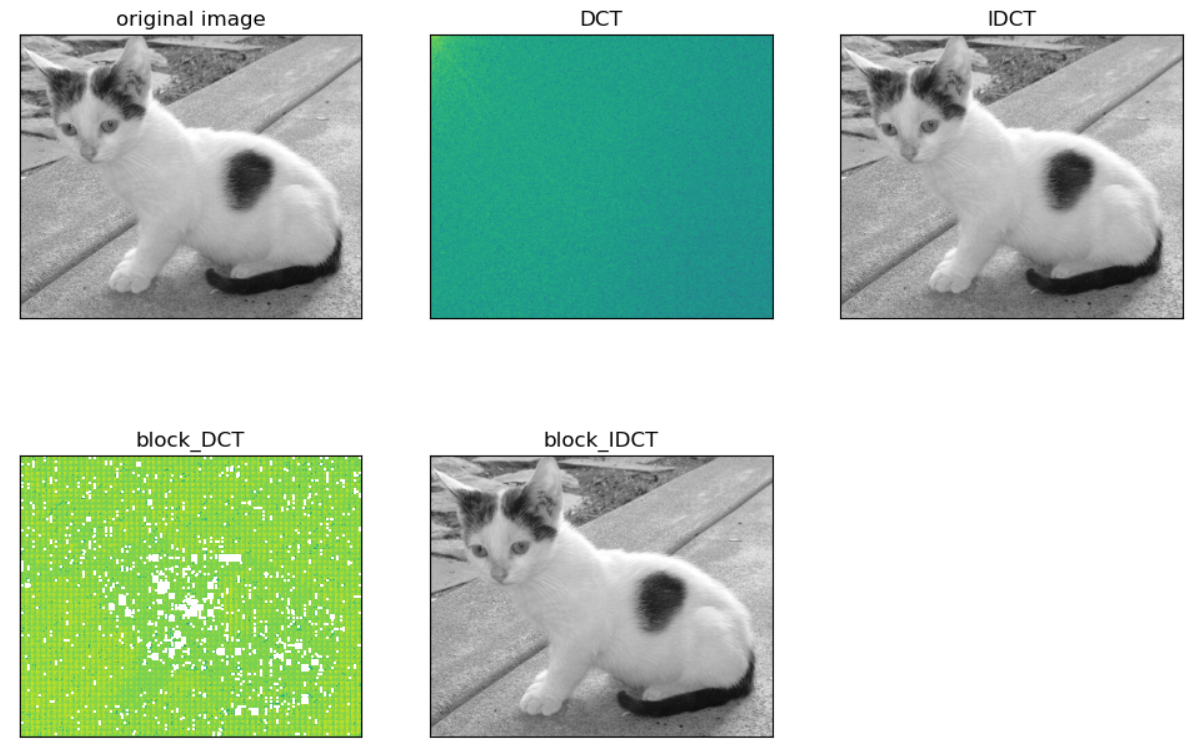
plt.subplot(235)

plt.imshow(new\_img,'gray')

plt.title('block\_IDCT'),plt.xticks([]),plt.yticks([])

plt.show()

实验结果图：



实验分析比较、结论：

这段代码展示了离散余弦变换（DCT）在图像上的应用。下面是对代码的分析：

（1）定义了整张图像进行DCT变换的函数 whole\_img\_dct：

def whole\_img\_dct(img\_f32):

img\_dct = cv.dct(img\_f32)

img\_dct\_log = np.log(abs(img\_dct))

img\_idct = cv.idct(img\_dct)

return img\_dct\_log, img\_idct

该函数将输入的图像 img\_f32 转换为32位浮点数类型，然后使用 cv.dct 进行整张图像的DCT变换。接着，对DCT变换结果取绝对值并取对数，得到 img\_dct\_log。最后，使用 cv.idct 对DCT变换结果进行逆变换，得到恢复的图像 img\_idct。函数返回了DCT变换结果的对数图像和逆变换后的图像。

（2）定义了分块图像进行DCT变换的函数 block\_img\_dct：

def block\_img\_dct(img\_f32):

# 省略部分代码

img\_dct = np.zeros((height\_, width\_), dtype=np.float32)

new\_img = img\_dct.copy()

for h in range(block\_y):

for w in range(block\_x):

img\_block = img\_f32\_cut[8\*h:8\*(h+1), 8\*w:8\*(w+1)]

img\_dct[8\*h:8\*(h+1), 8\*w:8\*(w+1)] = cv.dct(img\_block)

dct\_block = img\_dct[8\*h:8\*(h+1), 8\*w:8\*(w+1)]

img\_block = cv.idct(dct\_block)

new\_img[8\*h:8\*(h+1), 8\*w:8\*(w+1)] = img\_block

img\_dct\_log2 = np.log(abs(img\_dct))

return img\_dct\_log2, new\_img

该函数将输入的图像 img\_f32 分成8x8的块，然后对每个块进行DCT变换，并将结果存储在 img\_dct 中。接着，对每个块的DCT结果进行逆变换，得到恢复的块图像 img\_block，并将其放置到新的图像 new\_img 中。最后，对整个 img\_dct 进行取绝对值并取对数，得到 img\_dct\_log2。函数返回了分块DCT变换结果的对数图像和恢复的图像。

（3）主程序部分：

在主程序部分，首先使用 cv.imread 读取名为 "cat.jpg" 的图像，并将其以灰度图像的形式读入到 img\_u8 中。然后将 img\_u8 转换为32位浮点数类型，并赋值给 img\_f32。

接着，分别调用 whole\_img\_dct 函数和 block\_img\_dct 函数对图像进行DCT变换。得到的结果分别为整张图像的DCT变换结果 img\_dct\_log 和逆变换结果 img\_idct，以及分块DCT变换结果的对数图像 img\_dct\_log2 和恢复的图像 new\_img。

最后，使用 Matplotlib绘制子图，展示原始图像、整张图像的DCT结果、逆变换结果、分块DCT结果以及分块逆变换结果。

综上所述，这段代码通过整张图像和分块图像的DCT变换，展示了DCT在图像压缩和逆变换中的应用效果。通过对比原始图像、整张图像的DCT结果和逆变换结果，以及分块图像的DCT结果和逆变换结果，可以观察到DCT的压缩和恢复图像的效果。

**3.DFT变换处理：**

实验源代码：

**##离散傅里叶变换（DFT）**

**#以灰度形式读入**

img = cv.imread("./img/cat.jpg",0)

**#使用**cv.dft（）**进行傅里叶变换**

dst = cv.dft(np.float32(img),flags = cv.DFT\_COMPLEX\_OUTPUT)

**#将变换后图像低频部分转移到图像的中心**

dst\_center = np.fft.fftshift(dst)

**#使用**cv.magnitude**将实部和虚部转换为实部，乘以**20**是为了使得结果更大**

result = 20 \* np.log(np.abs(cv.magnitude(dst\_center[:,:,0],dst\_center[:,:,1])))

**#显示图像**

plt.subplot(121)

plt.imshow(img,cmap = "gray")

plt.axis("off")

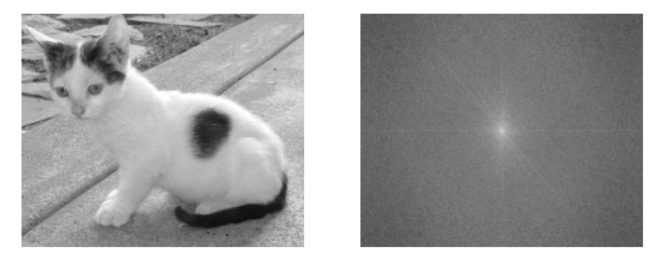
plt.subplot(122)

plt.axis("off")

plt.imshow(result,cmap = "gray")

plt.show()

实验结果图：



实验分析比较、结论：

这段代码展示了离散傅里叶变换（DFT）在图像上的应用。下面是对代码的分析：

（1）读入图像：

img = cv.imread("./img/cat.jpg", 0)

使用 cv.imread 读入名为 "cat.jpg" 的图像，并将其以灰度形式读入到 img 中。

（2）进行傅里叶变换：

dst = cv.dft(np.float32(img), flags=cv.DFT\_COMPLEX\_OUTPUT)

使用cv.dft函数对图像进行傅里叶变换。首先将图像转换为32位浮点数类型（np.float32），然后进行DFT变换，并将结果存储在 dst 中。flags=cv.DFT\_COMPLEX\_OUTPUT 表示输出为复数形式，包括实部和虚部。

（3）将变换后的图像低频部分转移到图像的中心：

dst\_center = np.fft.fftshift(dst)

使用 np.fft.fftshift 函数将傅里叶变换后的图像的低频部分移动到图像的中心。这一步是为了方便观察和处理图像频谱。

（4）将实部和虚部转换为幅度并进行对数变换：

result = 20 \* np.log(np.abs(cv.magnitude(dst\_center[:,:,0], dst\_center[:,:,1])))

使用 cv.magnitude 函数计算复数的幅度，将 dst\_center 中的实部和虚部作为参数传入。然后，取得到的幅度的绝对值，并进行对数变换，最后乘以20是为了使结果更大。将结果存储在result中。

（5）显示图像：

plt.subplot(121)

plt.imshow(img, cmap="gray")

plt.axis("off")

plt.subplot(122)

plt.axis("off")

plt.imshow(result, cmap="gray")

plt.show()

使用Matplotlib绘制子图，展示原始图像和经过DFT变换后的图像。首先使用 plt.subplot 创建两个子图，第一个子图显示原始图像，第二个子图显示经过DFT变换后的图像。plt.imshow 用于显示图像，cmap="gray" 指定灰度图像的颜色映射。plt.axis("off") 关闭坐标轴，以便更清晰地显示图像。最后使用 plt.show() 显示图像。

综上所述，这段代码通过离散傅里叶变换（DFT）将图像转换到频域，然后将频谱进行幅度和对数变换，最终展示了原始图像和DFT变换后的结果。通过观察DFT变换后的图像，可以分析图像的频域特征。

**4.傅里叶变换处理：**

实验源代码：

**##傅里叶变换**

**#%%**NumPy**实现傅里叶变换**

img = cv.imread('./img/cat.jpg',0)

**#使用**NumPy**进行傅里叶变换**

f = np.fft.fft2(img)

**#无处理（未把零频率分量移到中间）**

mid\_result = 20 \* np.log(np.abs(f))

**#把零频率分量移到中间**

fshift = np.fft.fftshift(f)

result = 20 \* np.log(np.abs(fshift))

**#显示图像**

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.subplot(131)

plt.imshow(img,cmap = "gray")

plt.axis("off")

plt.title('**原图**')

plt.subplot(132)

plt.axis("off")

plt.imshow(mid\_result,cmap = plt.cm.gray)

plt.title('Numpy FFT\n(Zero not moved)')

plt.subplot(133)

plt.axis("off")

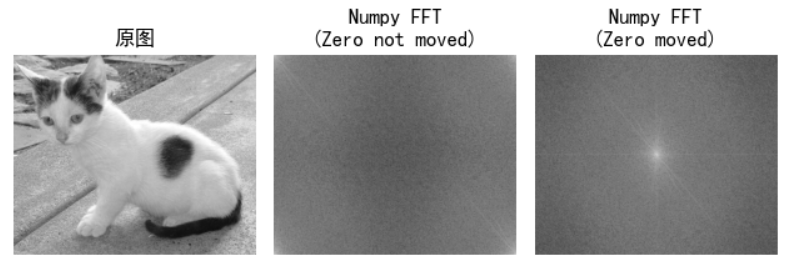
plt.imshow(result,cmap = plt.cm.gray)

plt.title('Numpy FFT\n(Zero moved)')

plt.tight\_layout()

plt.show()

实验结果图：



实验分析比较、结论：

这段代码展示了使用NumPy库进行傅里叶变换的过程。通过傅里叶变换，可以将图像从时域转换到频域，并观察图像在频域中的频谱分布和特征。

具体分析如下：

（1）读入图像：

使用cv.imread函数读取名为"cat.jpg"的图像，并以灰度形式存储在img变量中。

（2）进行傅里叶变换：

使用np.fft.fft2函数对图像进行二维傅里叶变换，将图像从时域转换到频域。变换后的结果存储在变量f中，它是一个复数数组，包含了图像在频域的信息。

（3）未将零频率分量移到中心的结果：

将变换结果f的绝对值取对数并乘以20，得到幅度谱。这一步的目的是将复数频域表示转换为实数表示，并放大幅度谱的显示。

（4）将零频率分量移到中心的结果：

使用np.fft.fftshift函数将傅里叶变换结果f中的零频率分量移动到频谱的中心。然后，对移动后的结果fshift取绝对值并进行对数运算，得到幅度谱。

（5）显示图像：

通过Matplotlib库绘制子图来展示结果。plt.subplot函数创建了三个子图，分别显示原始图像、未将零频率分量移到中心的傅里叶变换结果和将零频率分量移到中心的傅里叶变换结果。cmap="gray"指定了灰度图像的颜色映射，plt.axis("off")关闭坐标轴的显示。最后，使用plt.tight\_layout()调整子图的布局，并使用plt.show()显示图像。

通过对图像进行傅里叶变换，我们可以观察图像在频域中的频谱分布和特征。移动零频率分量到频谱中心可以更好地观察低频和高频信息的分布情况。这种变换可以用于图像处理和分析中的特征提取、滤波等应用。

**5.逆傅里叶变换处理：**

实验源代码：

**##逆傅里叶变换**

**#以灰度形式读入**

img = cv.imread('./img/cat.jpg',0)

**#使用**cv.dft（）**进行傅里叶变换**

dst = cv.dft(np.float32(img),flags = cv.DFT\_COMPLEX\_OUTPUT)

**#将变换后图像的低频部分转移到图像的中心**

dst\_center = np.fft.fftshift(dst)

**#中间可以定义相应的掩膜去留高频或者低频部分#**

**#使用**np.fft.ifftshift**将低频移动到原来的位置**

dst\_origin = np.fft.ifftshift(dst\_center)

**#使用**cv.idft**进行傅里叶的逆变化**

img\_idft = cv.idft(dst\_origin)

**#使用**cv.magnitude**转化为空间域内**

img\_idft = cv.magnitude(img\_idft[:,:,0],img\_idft[:,:,1])

**#显示图像**

plt.subplot(121)

plt.imshow(img,cmap = "gray")

plt.axis("off")

plt.title('original')

plt.subplot(122)

plt.axis("off")

plt.imshow(img\_idft,cmap = plt.cm.gray)

plt.title('NI'),plt.xticks([]),plt.yticks([])

plt.show()

实验结果图：



实验分析比较、结论：

这段代码展示了逆傅里叶变换的过程。逆傅里叶变换用于将频域信号转换回时域信号，从而恢复原始图像。

具体分析如下：

（1）读入图像：

使用cv.imread函数读取名为"cat.jpg"的图像，并以灰度形式存储在img变量中。

（2）进行傅里叶变换：

使用cv.dft函数对图像进行傅里叶变换，将图像从时域转换到频域。变换后的结果存储在变量dst中，它是一个复数数组，包含了图像在频域的信息

（3）将变换后图像的低频部分转移到图像的中心：

使用np.fft.fftshift函数将傅里叶变换结果dst中的低频部分移动到频谱的中心，得到dst\_center。

（4）定义掩膜：

在此部分代码中没有提供具体的掩膜定义，但是注释中提到可以在这个位置定义相应的掩膜，用于保留或去除图像的高频或低频部分。掩膜可以通过在dst\_center上进行操作来实现频域滤波。

（5）将低频移动到原来的位置：

使用np.fft.ifftshift函数将经过掩膜处理后的频域图像dst\_center中的低频部分移回原来的位置，得到dst\_origin。

（6）逆傅里叶变换：

使用cv.idft函数对dst\_origin进行逆傅里叶变换，将图像从频域恢复到时域。变换后的结果存储在img\_idft中。

（7）转换为空间域内：

使用cv.magnitude函数将逆傅里叶变换结果img\_idft的实部和虚部转换为幅度谱，得到空间域内的图像。

（8）显示图像：

通过Matplotlib库绘制子图来展示结果。plt.subplot函数创建了两个子图，分别显示原始图像和逆傅里叶变换后的图像。

通过逆傅里叶变换，我们可以将频域表示的图像转换回时域表示，从而恢复原始图像。逆傅里叶变换常用于图像处理、信号处理和通信领域中，可以用于去除图像中的噪声、图像复原、图像增强和特征提取等任务。通过在频域对图像进行滤波操作，可以选择保留或去除不同频率的信息，从而实现图像的频域处理。

以上代码展示了逆傅里叶变换的过程。首先，将图像进行傅里叶变换得到频域表示，然后可以通过掩膜操作对频域图像进行滤波处理。最后，将滤波后的频域图像进行逆傅里叶变换，将其转换回空间域表示，从而得到恢复或处理后的图像。

**五、实验总结与思考**

本实验通过使用OpenCV和NumPy库实现了离散余弦变换（DCT）、傅里叶变换（DFT）以及逆傅里叶变换的示例代码，并进行了分析和解释。

在离散余弦变换（DCT）的实验中，通过调整参数来控制压缩比和图片质量损失。在分块图DCT变换中，图像被分成多个小块，每个块进行独立的DCT变换，从而对图像进行分块处理。通过实验可以看出，DCT变换可以将图像从空域转换到频域，将图像的能量集中在较低的频率分量上，从而实现压缩效果。

在傅里叶变换的实验中，使用cv.dft和np.fft.fft2函数实现了傅里叶变换。通过将图像从时域转换到频域，我们可以观察图像的频谱特征。同时，使用np.fft.fftshift函数可以将零频率分量移到频谱的中心，使得频谱更直观地显示。

在逆傅里叶变换的实验中，通过将频域图像进行逆变换，可以将图像从频域恢复到时域。逆傅里叶变换常用于图像复原和图像增强等任务。在代码中，还提到了通过定义掩膜来选择保留或去除图像的高频或低频部分，从而实现频域滤波。