DIP 4

Image Restoration

Image restoration involves improving the appearance of an image by removing noise, correcting distortions, or recovering lost details. Here are a few common techniques:

Adding Noise

```
from skimage.util import random_noise

fruit_image = plt.imread('./dataset/fruits_square.jpg')

# Add noise to the image
noisy_image = random_noise(fruit_image)

# Show the original and resulting image
plot_comparison(fruit_image, noisy_image, 'Noisy image')
```

Reducing Noise

```
from skimage.restoration import denoise_tv_chambolle

noisy_image = plt.imread('./dataset/miny.jpeg')

# Apply total variation filter denoising
denoised_image = denoise_tv_chambolle(noisy_image, multichannel=True)

# Show the noisy and denoised image
plot_comparison(noisy_image, denoised_image, 'Denoised Image')
```

Reducing Noise while preserving edges
Use bilateral denoising filter

```
from skimage.restoration import denoise_bilateral

landscape_image = plt.imread('./dataset/noise-noisy-nature.jpg')

# Apply bilateral filter denoising
denoised_image = denoise_bilateral(landscape_image, multichannel=True)

# Show original and resulting images
plot_comparison(landscape_image, denoised_image, 'Denoised Image')
```

Denoising: You can use libraries like OpenCV and scikit-image for denoising images.

References

- A. Buades, B. Coll, J.-M. Morel, “A non-local algorithm for image denoising”, CVPR, 2005.
- A. Chambolle, “An Algorithm for Total Variation Minimization and Applications”, *Journal of Mathematical Imaging and Vision*, vol. 20, p. 89–97, 2004.
- L.B. Lucy, “An iterative technique for the rectification of observed distributions”, *Astronomical Journal*, vol. 79, no 6, p. 745–754, 1974.
- W.H. Richardson, “Bayesian-based iterative method of image restoration”, *Journal of the Optical Society of America*, vol. 62, no 1, p. 55–59, 1972.
- L.I. Rudin, S. Osher, E. Fatemi, “Nonlinear total variation based noise removal algorithms”, *Physica D*, vol. 60, no. 1–4, p. 259–268, 1992.


```
import cv2
import numpy as np

# Load the image
img = cv2.imread('noisy_image.jpg')

# Apply Gaussian Blur
denoised_img = cv2.GaussianBlur(img, (5, 5), 0)

# Save the result
cv2.imwrite('denoised_image.jpg', denoised_img)
```

Deblurring: Using Wiener filter from scikit-image.

Python

```
from skimage import restoration, io

# Load the image
img = io.imread('blurry_image.jpg', as_gray=True)

# Apply Wiener filter
deblurred_img = restoration.wiener(img, psf=np.ones((5, 5)) / 25,
balance=0.1)

# Save the result
io.imsave('deblurred_image.jpg', deblurred_img)
```

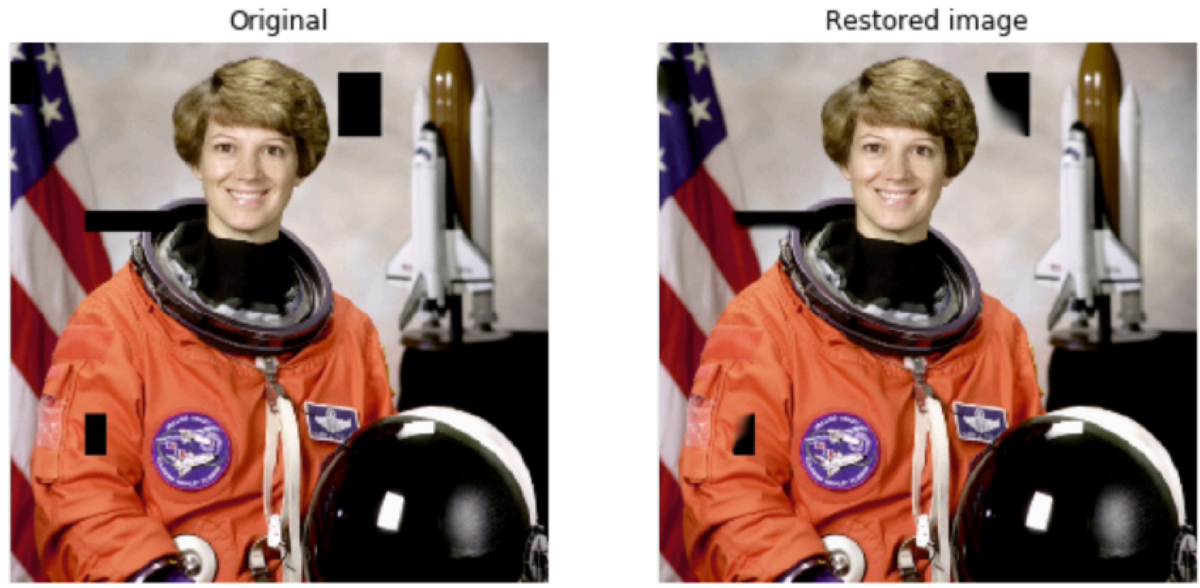
Inpainting is the process of reconstructing lost or deteriorated parts of images and videos.

```
Things to do:

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rcParams['figure.figsize'] = (10, 8)
```

Example:
using the **inpaint_biharmonic()** function to restore an image that has missing parts.
The **mask** is a black and white image with patches that have the position of the image bits that have been corrupted. We can apply the restoration function on these areas. Replace 1's binary mask to simulate the image.

```
def show_image(image, title='Image', cmap_type='gray'):
    plt.imshow(image, cmap=cmap_type)
    plt.title(title)
    plt.axis('off')

def plot_comparison(img_original, img_filtered,
img_title_filtered):
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 8),
sharex=True, sharey=True)
    ax1.imshow(img_original, cmap=plt.cm.gray)
    ax1.set_title('Original')
    ax1.axis('off')
    ax2.imshow(img_filtered, cmap=plt.cm.gray)
    ax2.set_title(img_title_filtered)
    ax2.axis('off')
from skimage.restoration import inpaint
from skimage.transform import resize
from skimage import color

defect_image = plt.imread('./dataset/damaged_astronaut.png')
defect_image = resize(defect_image, (512, 512))
defect_image = color.rgba2rgb(defect_image)

mask = pd.read_csv('./dataset/astronaut_mask.csv').to_numpy()
```

```python
# Apply the restoration function to the image using the mask
restored_image = inpaint.inpaint_biharmonic(defect_image, mask,
multichannel=True)

# Show ther defective image
plot_comparison(defect_image, restored_image, 'Restored image')
```



Removing Logos

create and set the mask to be able to erase the logo by inpainting this area

```python
image_with_logo = plt.imread('./dataset/4.2.06_w_logo_2_2.png')

# Initialize the mask
mask = np.zeros(image_with_logo.shape[:-1])

# Set the pixels where the logo is to 1
mask[210:272, 360:425] = 1

# Apply inpainting to remove the logo
image_logo_removed = inpaint.inpaint_biharmonic(image_with_logo,
                                                mask,
                                                multichannel=True)

# Show the original and logo removed images
plot_comparison(image_with_logo, image_logo_removed, 'Image with
logo removed')
```

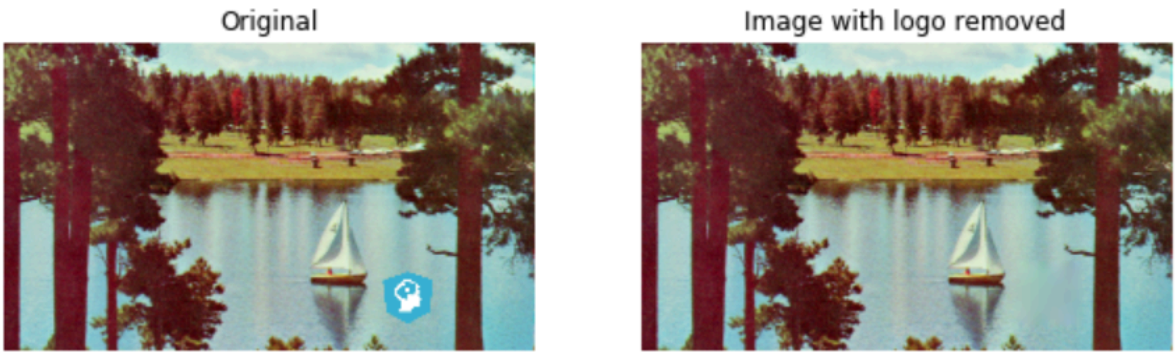


Image Compression

Image compression reduces the file size of an image without significantly affecting its quality. Here are a few methods:

1. Using Pillow:

Python

```
from PIL import Image

# Open an image file
with Image.open('image.jpg') as img:
    # Compress the image
    img.save('compressed_image.jpg', quality=85, optimize=True)
```

2. Using OpenCV:

Python

```
import cv2

# Load the image
img = cv2.imread('image.jpg')

# Compress the image
cv2.imwrite('compressed_image.jpg', 8img,
[int(cv2.IMWRITE_JPEG_QUALITY), 5])
```

Using Numpy and Singular Value Decomposition (SVD):

Python

```
import numpy as np
from PIL import Image

def compress_image(image_path, k):
    img = Image.open(image_path)
```

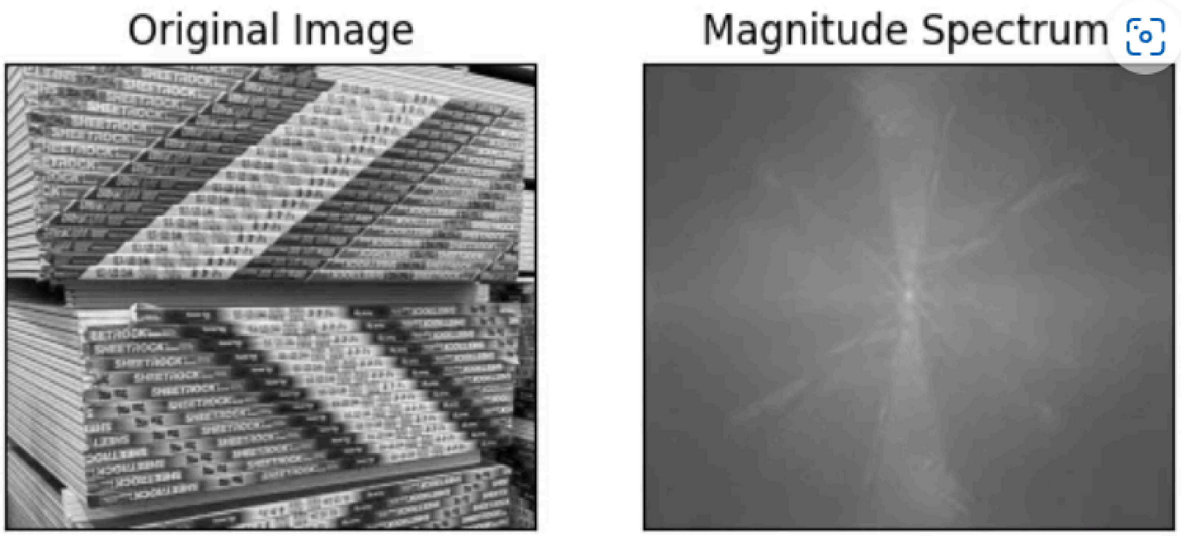
```
    img_array = np.array(img) / 255.0
    U, s, V = np.linalg.svd(img_array, full_matrices=False)
    S = np.diag(s[:k])
    compressed_img = np.dot(U[:, :k], np.dot(S, V[:k, :]))
    return Image.fromarray((compressed_img *
255).astype(np.uint8))

compressed_img = compress_image('image.jpg', 50)
compressed_img.save('compressed_image.jpg')
```

Fourier Transform for Image Compression:

1. Understanding Fourier Transform:
Fourier Transform decomposes an image into its frequency components. High-frequency components, representing details and edges, can be reduced without losing significant information.

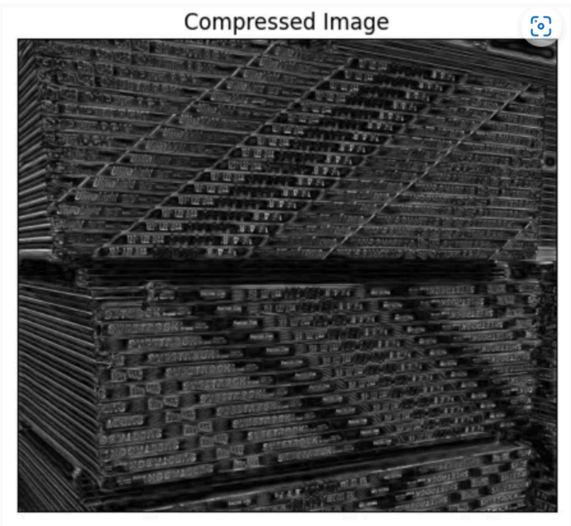
2. Implementation in Python:

```
#Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

#Read in the image
image = cv2.imread('sample_image.jpg', 0)

#Apply the fourier transform
f_transform = np.fft.fft2(image)
f_transform_shifted = np.fft.fftshift(f_transform)
magnitude_spectrum = np.log(np.abs(f_transform_shifted) + 1)

#Plot the images for comparison
plt.subplot(121), plt.imshow(image, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



3. Compression:
By discarding high-frequency coefficients, you can compress the image. Adjust the threshold to control the compression ratio.

```
rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
f_transform_shifted[crow - 30:crow + 30, ccol - 30:ccol + 30] = 0
f_transform_inverse = np.fft.ifftshift(f_transform_shifted)

img_back = np.fft.ifft2(f_transform_inverse)
img_back = np.abs(img_back)


plt.imshow(img_back, cmap='gray')
plt.title('Compressed Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

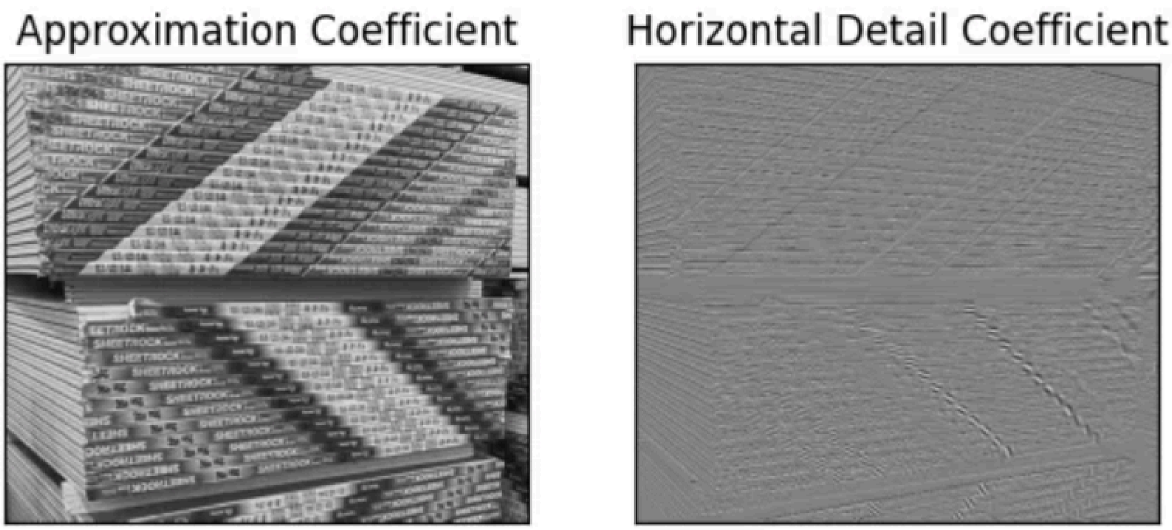


Wavelet Transform for Image Compression:

1. Understanding Wavelet Transform:
Wavelet Transform provides a multi-resolution analysis of an image. It decomposes the image into approximation and detail coefficients, allowing for efficient compression.

2. Implementation in Python:

```
#Import the pywt library
import pywt

#Read in the image
image = cv2.imread('sample_image.jpg', 0)

#Find the coefficients
coeffs = pywt.dwt2(image, 'bior1.3')
cA, (cH, cV, cD) = coeffs

#Plot the images
plt.subplot(121), plt.imshow(cA, cmap='gray')
plt.title('Approximation Coefficient'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(cH, cmap='gray')
plt.title('Horizontal Detail Coefficient'), plt.xticks([]),
plt.yticks([])
plt.show()
```



3. Compression:
Adjust the threshold to discard less significant coefficients and achieve compression.

```
#Apply a thresholding limit
threshold = 20
```

```
#Using the thresholding limit for the wavelet transform
cA_thresholded = pywt.threshold(cA, threshold, mode='soft')
cH_thresholded = pywt.threshold(cH, threshold, mode='soft')

#Compress the image
coeffs_thresholded = (cA_thresholded, (cH_thresholded, cV, cD))
img_compressed = pywt.idwt2(coeffs_thresholded, 'bior1.3')

#Show the resulting image after compression
plt.imshow(img_compressed, cmap='gray')
plt.title('Compressed Image'), plt.xticks([]), plt.yticks([])
plt.show()
```