**Practical 1**

Mathematical Tools for Image Processing:

Create PYTHON CODE to transform an image in special domain into its frequency domain using DFT

**I/O:-**

import numpy as np

import matplotlib.pyplot as plt

image\_filename = r"C:\Users\Downloads\Practs\Practs\mobile.webp"

def calculate\_2dft(input):

ft = np.fft.ifftshift(input)

ft = np.fft.fft2(ft)

return np.fft.fftshift(ft)

# Read and process image

image = plt.imread(image\_filename)

image = image[:, :, :3].mean(axis=2) # Convert to grayscale

plt.set\_cmap("gray")

ft = calculate\_2dft(image)

plt.subplot(121)

plt.imshow(image)

plt.axis("off")

plt.subplot(122)

plt.imshow(np.log(abs(ft)))

plt.axis("off")

plt.show()

**Practical 2**

Image Enhancement in Spatial Domain:

Create a python code to demonstrate the spatial domain image enhancement methods like: Image Contrast Stretching, Log Transform,

Power Law Transform, Image Averaging and Median Filter

**I/O :-**

**Contrast Stretching**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale

img = cv2.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\low\_contrast.jpg")

# Get min and max intensity values

min\_val = np.min(img)

max\_val = np.max(img)

# Apply contrast stretching formula

stretched = ((img - min\_val) / (max\_val - min\_val) \* 255).astype(np.uint8)

# Display original and stretched image side by side

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(img)

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Contrast Stretched")

plt.imshow(stretched, cmap='gray')

plt.axis('off')

plt.show()

**Log Transform**

# import Pillow modules

from PIL import Image

from PIL import ImageFilter

import math

# Compute log

def logTransform(c, f):

g = c \* math.log(float(1 + f),10);

return g;

# Apply logarithmic transformation for an image

def logTransformImage(image, outputMax = 255, inputMax=255):

c = outputMax/math.log(inputMax+1,10);

print(c)

# Read pixels and apply logarithmic transformation

for i in range(0, img.size[0]-1):

for j in range(0, img.size[1]-1):

# Get pixel value at (x,y) position of the image

f = img.getpixel((i,j));

# Do log transformation of the pixel

redPixel = round(logTransform(c, f[0]));

greenPixel = round(logTransform(c, f[1]));

bluePixel = round(logTransform(c, f[2]));

# Modify the image with the transformed pixel values

img.putpixel((i,j),(redPixel, greenPixel, bluePixel));

return image;

# Display the original image

imageFileName = r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\jungle.jpg";

img = Image.open(imageFileName);

#img.show();

plt.subplot(121)

plt.imshow(img)

plt.title('Original')

plt.xticks([])

plt.yticks([])

# Display the image after applying the logarithmic transformation

logTransformedImage = logTransformImage(img);

#logTransformedImage.show();

plt.subplot(122)

plt.imshow(logTransformedImage)

plt.title('logTransformedImage')

plt.xticks([])

plt.yticks([])

plt.show()

**Power Law Transform**

import cv2

import matplotlib.pyplot as plt

import math

import numpy as np

#this type of processing is suited for displaying image correctly for human eye based on monitor's display settings

# Read an image

image = cv2.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\jungle.jpg")

plt.imshow(image)

plt.show()

# Trying 7 gamma values.

for gamma in [0.1, 0.3, 0.4, 0.5, 1.2, 2.2, 3.2]:

# Apply gamma correction.

gamma\_corrected = np.array(255\*(image / 255) \*\* gamma, dtype = 'uint8')

plt.imshow(gamma\_corrected)

plt.show()

**Image Averaging**

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = cv.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\averaging.jpg")

assert img is not None, "file could not be read, check with os.path.exists()"

kernel = np.ones((5,5),np.float32)/25

dst = cv.filter2D(img,-1,kernel)

plt.subplot(121)

plt.imshow(img)

plt.title('Original')

plt.xticks([])

plt.yticks([])

plt.subplot(122)

plt.imshow(dst)

plt.title('Averaging')

plt.xticks([])

plt.yticks([])

plt.show()

**Median Filter**

from PIL import Image, ImageFilter

from matplotlib import pyplot as plt

im = Image.open(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\averaging.jpg")

im1 = im.filter(ImageFilter.MedianFilter(size = 7))

plt.subplot(121)

plt.imshow(im)

plt.title('Original')

plt.xticks([])

plt.yticks([])

plt.subplot(122)

plt.imshow(im1)

plt.title('Median')

plt.xticks([])

plt.yticks([])

plt.show()

**Practical 3**

Image Enhancement in Frequency Domain:

Create a python code to demonstrate the frequency domain image enhancement methods like: Ideal, Gaussian, Butterworth Low Pass & Ideal, Gaussian, and Butterworth High Pass Filter.

**I/O:-**

**Ideal Low and High Pass Filter**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# original image

f = cv2.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\averaging.jpg",0)

plt.imshow(f, cmap='gray')

plt.axis('off')

plt.show()

# image in frequency domain

F = np.fft.fft2(f)

Fshift = np.fft.fftshift(F)

plt.imshow(np.log1p(np.abs(Fshift)),

cmap='gray')

plt.axis('off')

plt.show()

# Filter: Low pass filter

M,N = f.shape

H = np.zeros((M,N), dtype=np.float32)

D0 = 25

for u in range(M):

for v in range(N):

D = np.sqrt((u-M/2)\*\*2 + (v-N/2)\*\*2)

if D <= D0:

H[u,v] = 1

else:

H[u,v] = 0

plt.imshow(H, cmap='gray')

plt.axis('off')

plt.show()

# Ideal Low Pass Filtering

Gshift = Fshift \* H

# Inverse Fourier Transform

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()

# Filter: High pass filter

H = 1 - H

plt.imshow(H, cmap='gray')

plt.axis('off')

plt.show()

# Ideal High Pass Filtering

Gshift = Fshift \* H

# Inverse Fourier Transform

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()

**Butterworth Low and High Pass Filter**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# open the image

f = cv2.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\averaging.jpg",0)

# transform image into freq. domain and shifted

F = np.fft.fft2(f)

Fshift = np.fft.fftshift(F)

plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')

plt.axis('off')

plt.show()

# Butterwort Low Pass Filter

M,N = f.shape

H = np.zeros((M,N), dtype=np.float32)

D0 = 10 # cut of frequency

n = 10 # order

for u in range(M):

for v in range(N):

D = np.sqrt((u-M/2)\*\*2 + (v-N/2)\*\*2)

H[u,v] = 1 / (1 + (D/D0)\*\*n)

plt.imshow(H, cmap='gray')

plt.axis('off')

plt.show()

# frequency domain image filters

Gshift = Fshift \* H

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.imshow(g, cmap='gray') plt.axis('off')

plt.show()

# Butterworth High Pass Filter

HPF = np.zeros((M,N), dtype=np.float32)

D0 = 10

n = 1

for u in range(M):

for v in range(N):

D = np.sqrt((u-M/2)\*\*2 + (v-N/2)\*\*2)

HPF[u,v] = 1 / (1 + (D0/D)\*\*n)

plt.imshow(HPF, cmap='gray') plt.axis('off')

plt.show()

# frequency domain image filters

Gshift = Fshift \* HPF

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()

**Gaussian Low and High Pass Filter**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# open the image f

f = cv2.imread(r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\averaging.jpg",0)

plt.figure(figsize=(5,5))

plt.imshow(f, cmap='gray')

plt.axis('off')

plt.show()

# transform the image into frequency domain, f --> F

F = np.fft.fft2(f)

Fshift = np.fft.fftshift(F)

plt.figure(figsize=(5,5))

plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')

plt.axis('off')

plt.show()

# Create Gaussin Filter: Low Pass Filter

M,N = f.shape

H = np.zeros((M,N), dtype=np.float32)

D0 = 10

for u in range(M):

for v in range(N):

D = np.sqrt((u-M/2)\*\*2 + (v-N/2)\*\*2)

H[u,v] = np.exp(-D\*\*2/(2\*D0\*D0))

plt.figure(figsize=(5,5))

plt.imshow(H, cmap='gray')

plt.axis('off')

plt.show()

# Image Filters

Gshift = Fshift \* H

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.figure(figsize=(5,5))

plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()

# Gaussian: High pass filter

HPF = 1 - H

plt.figure(figsize=(5,5))

plt.imshow(HPF, cmap='gray')

plt.axis('off')

plt.show()

# Image Filters

Gshift = Fshift \* HPF

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))

plt.figure(figsize=(5,5))

plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()

**Practical 4**

Illustration of Segmentation techniques:

Create a python code to demonstrate image segmentation using varying techniques like Global Thresholding and Region Growing.

**I/O:-**

**Image Segmentation using Global Thresholding**

import numpy as np

import matplotlib.pyplot as plt

from skimage.io import imread

from skimage.color import rgb2gray

import cv2

# Load image

image\_path = r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\chico.jpg"

chico = imread(image\_path)

# Convert to grayscale

chico\_gray = rgb2gray(chico)

# Display original image

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

plt.imshow(chico)

plt.title("Original Image")

plt.axis("off")

plt.show()

# Generate multiple thresholded versions

th\_values = np.linspace(0, 1, 10)

fig, axis = plt.subplots(2, 5, figsize=(15, 8))

for th, ax in zip(th\_values, axis.flatten()):

chico\_binarized = chico\_gray < th

ax.imshow(chico\_binarized, cmap='gray')

ax.set\_title(f'Threshold = {th:.2f}')

ax.axis("off")

plt.tight\_layout()

plt.show()

# Ask user for final threshold input

user\_input = input("Enter the threshold value you liked (e.g., 0.3): ")

try:

final\_th = float(user\_input)

except ValueError:

print("Invalid input. Using default threshold 0.5.")

final\_th = 0.5

# Apply final threshold

final\_binarized = chico\_gray < final\_th

# Show and save the final result

plt.figure()

plt.imshow(final\_binarized, cmap='gray')

plt.title(f'Final Thresholded Image @ {final\_th:.2f}')

plt.axis('off')

plt.show()

# Save final binary image

save\_path = r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\chico\_final\_binarized.png"

cv2.imwrite(save\_path, (final\_binarized \* 255).astype('uint8'))

print(f"Saved final binarized image to {save\_path}")

**Image Segmentation using Region Growing algorithm**

import numpy as np

import matplotlib.pyplot as plt

from scipy.ndimage import binary\_dilation

# Binary input image (1: object, 0: background)

A = np.array([

[0, 0, 0, 0, 0, 0, 0],

[0, 0, 1, 1, 0, 0, 0],

[0, 1, 0, 0, 1, 0, 0],

[0, 1, 0, 0, 1, 0, 0],

[0, 0, 1, 0, 1, 0, 0],

[0, 1, 0, 0, 1, 0, 0],

[0, 0, 1, 0, 0, 1, 0],

[0, 1, 0, 0, 0, 1, 0],

[0, 1, 1, 1, 1, 0, 0],

[0, 0, 0, 0, 0, 0, 0]

])

# Complement of image

Ac = 1 - A

# 4-connected structuring element

B = np.array([

[0, 1, 0],

[1, 1, 1],

[0, 1, 0]

])

# Initial seed point

x = np.zeros\_like(A)

x[2, 2] = 1 # Seed point inside the shape

# Visualization

fig, axs = plt.subplots(1, 3, figsize=(12, 4))

plt.gray()

# Display original image

axs[0].imshow(A)

axs[0].set\_title("Original Image")

# Region Growing Process

k = 0

while True:

k += 1

axs[1].imshow(x)

axs[1].set\_title(f"Filling iteration {k}")

plt.pause(0.5)

xnew = np.logical\_and(binary\_dilation(x, structure=B), Ac)

if np.array\_equal(xnew, x):

break

x = xnew

# Combine filled region with original

y = np.logical\_or(x, A)

axs[2].imshow(y)

axs[2].set\_title("Final Filled Region")

plt.tight\_layout()

plt.show()

**Practical 5**

Image Compression Technique:

Create python code for demonstrating Image Compression using Run Length Coding algorithm.

**I/O:-**

import numpy as np

import cv2 # or use imageio if you prefer

def rle\_encode(arr):

# Encode 1D numpy array using Run-Length Encoding.

# Returns list of (value, count) pairs.

if len(arr) == 0:

return []

encoded = []

prev\_value = arr[0]

count = 1

for val in arr[1:]:

if val == prev\_value:

count += 1

else:

encoded.append((prev\_value, count))

prev\_value = val

count = 1

encoded.append((prev\_value, count))

return encoded

def rle\_decode(encoded):

# Decode RLE-encoded data back to 1D numpy array.

decoded = []

for value, count in encoded:

decoded.extend([value] \* count)

return np.array(decoded, dtype=np.uint8)

# Load grayscale image

image = cv2.imread(r"C:\Users\Downloads\Practs\Practs\ico\_warning.png", cv2.IMREAD\_GRAYSCALE)

# Flatten to 1D

flat\_image = image.flatten()

print("Original Length:",len(flat\_image))

# Encode

encoded = rle\_encode(flat\_image)

print("Encoded length:", len(encoded))

# Decode

decoded\_flat = rle\_decode(encoded)

# Reshape back to original shape

decoded\_image = decoded\_flat.reshape(image.shape)

# Save reconstructed image to verify

cv2.imwrite(r"C:\Users\Downloads\Practs\Practs\ico\_warning\_decode.png", decoded\_image)

**Practical 6**

Various Image Retrieval Techniques:

Create Python code for demonstrating techniques for Feature Detection in an Image.

**I/O:-**

import cv2

import matplotlib.pyplot as plt

# === Load the Image ===

image\_path = r"D:\2023 Birla MSc-Bsc\MSc\MSc IT - IP\Practs\4shapes.png"

img\_color = cv2.imread(image\_path)

# Check if image is loaded successfully

if img\_color is None:

raise FileNotFoundError(f"Image not found at {image\_path}")

# Convert to grayscale

img\_gray = cv2.cvtColor(img\_color, cv2.COLOR\_BGR2GRAY)

# === Canny Edge Detection ===

edges = cv2.Canny(img\_gray, threshold1=100, threshold2=200)

# Display Canny edges

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.imshow(edges, cmap='gray')

plt.title('Canny Edge Detection')

plt.axis('off')

# === SIFT Keypoint Detection ===

sift = cv2.SIFT\_create()

keypoints, descriptors = sift.detectAndCompute(img\_gray, None)

# Print descriptor shape if needed

print(f"Number of keypoints: {len(keypoints)}")

print(f"Descriptor shape: {descriptors.shape if descriptors is not None else 'None'}")

# Draw keypoints on the original image

img\_sift = cv2.drawKeypoints(img\_color, keypoints, None, flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

# Display SIFT keypoints

plt.subplot(1, 2, 2)

plt.imshow(cv2.cvtColor(img\_sift, cv2.COLOR\_BGR2RGB))

plt.title('SIFT Keypoints')

plt.axis('off')

plt.tight\_layout()

plt.show()

**Practical 7**

Illustration of Image Forgery:

Create a Python code for demonstrating any image forgery method e.g. Copy-Move method.

**I/O:-**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load image

image = cv2.imread(r"C:\Users\Downloads\Practs\Practs\sunandmoon.png")

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Simulate forgery: Copy a region and paste it elsewhere

forged = image\_rgb.copy()

# Define region to copy (e.g., a 50x50 square)

x1, y1, w, h = 328, 358, 40, 40

region = forged[y1:y1+h, x1:x1+w]

# Define where to paste the copied region

x2, y2 = 200, 358

forged[y2:y2+h, x2:x2+w] = region

x3, y3 = 456, 358

forged[y3:y3+h, x3:x3+w] = region

# Display original and forged images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(image\_rgb)

plt.title("Original Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(forged)

plt.title("Forged Image (Copy-Move)-Image from Outer Space Planet")

plt.axis("off")

plt.tight\_layout()

plt.show()

cv2.imwrite(r"C:\Users\Downloads\Practs\Practs\sunandmoon\_forged.png",forged)

**Practical 8**

Illustration of Image Forgery Detection Method:

Create a python code to detect if any forgery is made in an image.

**I/O:-**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def embed\_watermark(image, watermark\_text):

# Convert image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

# Apply DCT

dct = cv2.dct(gray)

# Embed watermark text as ASCII into DCT coefficients

watermark = [ord(c) for c in watermark\_text] #ord gives ASCII code for every character in watermark\_text

for i, val in enumerate(watermark):

dct[0, i] += val % 10 # Embed only the last digit of the ASCII code into only in low-frequency band (top row)

# Apply inverse DCT

watermarked\_img = cv2.idct(dct)

watermarked\_img = np.uint8(np.clip(watermarked\_img, 0, 255))

return watermarked\_img

# Load original image

image = cv2.imread(r"C:\Users\Downloads\Practs\Practs\sunandmoon.png")

watermarked = embed\_watermark(image, "AUTHENTIC")

""" Simulate forgery by tampering with image"""

tampered = watermarked.copy()

tampered[358:398, 328:368] = 255 # White patch (fake object)

# Show images

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.title("Original Image")

plt.axis("off")

plt.subplot(1, 3, 2)

plt.imshow(watermarked, cmap='gray')

plt.title("Watermarked Image")

plt.axis("off")

plt.subplot(1, 3, 3)

plt.imshow(tampered, cmap='gray')

plt.title("Tampered Image")

plt.axis("off")

plt.tight\_layout()

plt.show()

"""

#Checking Image for Forgery

"""

def extract\_watermark(image, length):

# Convert to grayscale and DCT

gray = np.float32(image)

dct = cv2.dct(gray)

# Extract the embedded values

extracted = ""

for i in range(length):

char\_val = int(dct[0, i]) % 10

extracted += str(char\_val)

return extracted

# Try to extract watermark from both images

wm\_extracted\_clean = extract\_watermark(watermarked, len("AUTHENTIC"))

wm\_extracted\_tampered = extract\_watermark(tampered, len("AUTHENTIC"))

print("Extracted from watermarked image:", wm\_extracted\_clean)

print("Extracted from tampered image:", wm\_extracted\_tampered)

if wm\_extracted\_clean != wm\_extracted\_tampered:

print("\n⚠️ Forgery Detected: Watermark integrity compromised.")

else:

print("\n✅ Image appears to be authentic.")