GODAWARI COLLEGE

AFFILIATED TO

TRIBHUVAN UNIVERSITY

LAB REPORT

CSC 321 Image Processing

SUBMITTED TO
Department of B.Sc.CSIT
Godawari College
Itahari-9, Sunsari

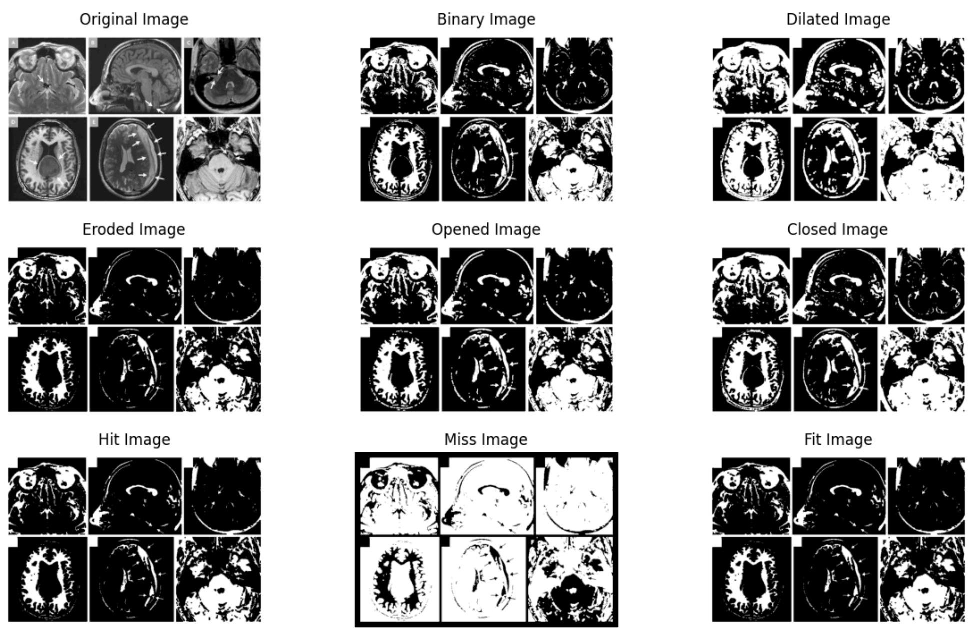
SUBMITTED BY
Name: **Dilli Hang Rai**
Roll No: **29677/078**
Batch & Semester: 2078 5thSemester

Source Code of Morphological Operations: Hit,Miss, Closing,Opening, dilation,erosion

```
import cv2

import numpy as np

from matplotlib import pyplot as plt
# Load a real image (example: an image named 'example_image.jpg')

image = cv2.imread('lab1Img1.jpg', cv2.IMREAD_GRAYSCALE)


# Threshold the image to make it binary (black and white)

_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)


# Define a 3x3 kernel (structuring element)

kernel = np.ones((3, 3), np.uint8)


# Dilation: Expands the boundaries of foreground pixels (white pixels)

dilated_image = cv2.dilate(binary_image, kernel, iterations=1)


# Erosion: Shrinks the boundaries of foreground pixels (white pixels)

eroded_image = cv2.erode(binary_image, kernel, iterations=1)


# Opening: Erosion followed by Dilation, removes noise (small white spots)

opened_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)


# Closing: Dilation followed by Erosion, removes small black holes

closed_image = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)


# Hit and Miss Operation (Custom Implementation using OpenCV functions)

hit_image = cv2.morphologyEx(binary_image, cv2.MORPH_HITMISS, kernel)


# Miss is essentially the inverse of Hit, so we can use the NOT of the result

miss_image = cv2.bitwise_not(hit_image)

# Fit is generally defined as the overlap of dilation and erosion

# So we calculate the intersection of dilation and erosion

fit_image = cv2.bitwise_and(dilated_image, eroded_image)

# Plotting the results
```

```
titles = ['Original Image', 'Binary Image', 'Dilated Image', 'Eroded Image', 'Opened Image', 'Closed Image', 'Hit Image', 'Miss Image', 'Fit Image']
images = [image, binary_image, dilated_image, eroded_image, opened_image, closed_image, hit_image, miss_image, fit_image]
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()
```

OUTPUT:



Conclusion:

This lab report on Morphological operations, using structuring elements, is essential for modifying and analyzing shapes in binary images. Dilation and erosion adjust the size of objects, while opening and closing refine shapes by removing noise or filling gaps. These operations are fundamental in image preprocessing for various applications like object detection
and noise removal.

Source Code of Image Segmentation using Region-Growing Algorithm

```python
import numpy as np
import cv2

def region_growing(image, seed, threshold):
   """
   Region growing algorithm

   :param image: Input image (2D numpy array)
   :param seed: Seed point (tuple of y, x coordinates)
   :param threshold: Intensity threshold for region growing
   :return: Segmented binary image
   """
   rows, cols = image.shape
   segmented = np.zeros((rows, cols), dtype=np.uint8)
   segmented[seed] = 1
   seed_value = image[seed]

   def _get_neighbors(y, x):
       return [(y-1, x), (y+1, x), (y, x-1), (y, x+1)]

   stack = [seed]
   while stack:
       y, x = stack.pop()
       for ny, nx in _get_neighbors(y, x):
           if 0 <= ny < rows and 0 <= nx < cols:
               if segmented[ny, nx] == 0 and abs(int(image[ny, nx]) - int(seed_value))
<= threshold:
                   segmented[ny, nx] = 1
                   stack.append((ny, nx))

   return segmented

def load_image_from_file(file_path):
   """
   Load an image from a file path.

   :param file_path: Path to the image file
   :return: Image as a numpy array
   """
   image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
   return image
```

```python
def main():
   file_path = 'medicalcombine.jpeg'  # Path to the local image file

   # Load the image from the local file
   image = load_image_from_file(file_path)

   # Check if the image is loaded successfully
   if image is None:
       print("Error: Image could not be loaded from the specified file path.")
       return

   seed = (100, 100)  # Example seed point (y, x)
   threshold = 45
   result = region_growing(image, seed, threshold)

   # Save the segmented image
   cv2.imwrite('FinalSegementedRegionGrow.png', result * 255)
   print("Segmented image saved as 'FinalSegementedRegionGrow.png'")

if __name__ == "__main__":
   main()
```

Source Code of Image Segmentation using laplacian,mexican filter

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter

# Load a real image
image = cv2.imread('lab3Image.png', cv2.IMREAD_GRAYSCALE)

# Load a real image
imagee = cv2.imread('lab3_1Image.jpeg', cv2.IMREAD_GRAYSCALE)

# Ensure the image was loaded correctly
if image is None or imagee is None:
   raise ValueError("Image not loaded. Ensure the image file path is correct.")

# 1. Point Detection using Laplacian (discontinuity based)
laplacian = cv2.Laplacian(image, cv2.CV_64F)
laplacian_abs = cv2.convertScaleAbs(laplacian)

# 2. Line Detection using Sobel operator (Gradient-based edge detection)
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
sobel_edge = cv2.magnitude(sobel_x, sobel_y)
sobel_edge = cv2.convertScaleAbs(sobel_edge)
```

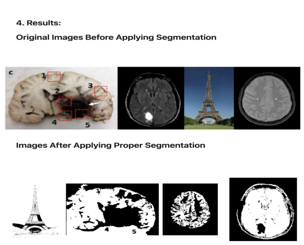
```python
# 3. Mexican Hat filter (Second derivative of Gaussian)
def mexican_hat_filter(image, sigma=1.0):
   gaussian = cv2.GaussianBlur(image, (0, 0), sigma)
   return cv2.Laplacian(gaussian, cv2.CV_64F)

mexican_hat = mexican_hat_filter(image)
# 4. Edge Linking and Boundary Detection using Canny edge detector
edges = cv2.Canny(image, 100, 200)
# Plot results of the edge detection methods
plt.figure(figsize=(12, 8))
plt.subplot(2, 3, 1)
plt.imshow(laplacian_abs, cmap='gray')
plt.title('Point Detection (Laplacian)')
plt.subplot(2, 3, 2)
plt.imshow(sobel_edge, cmap='gray')
plt.title('Line Detection (Sobel)')
plt.subplot(2, 3, 3)
plt.imshow(mexican_hat, cmap='gray')
plt.title('Mexican Hat Filter')
plt.subplot(2, 3, 4)
plt.imshow(edges, cmap='gray')
plt.title('Edge Linking (Canny)')

# 5. Thresholding (Global, Local, Adaptive)
_, global_thresh = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
local_thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 11, 2)
plt.subplot(2, 3, 5)
plt.imshow(global_thresh, cmap='gray')
plt.title('Global Thresholding')

plt.subplot(2, 3, 6)
plt.imshow(local_thresh, cmap='gray')
plt.title('Adaptive Thresholding')
plt.show()
```

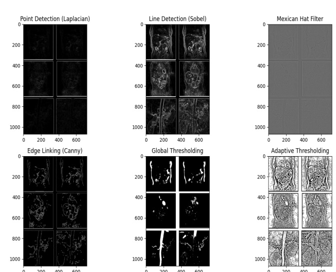
<u>Source code of Image Segmentation using Region-Splitting and Merge</u>

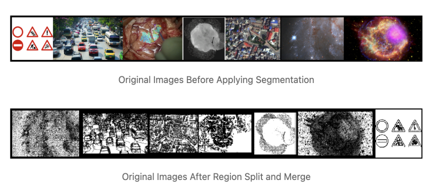
```python
import cv2
import numpy as np

def is_homogeneous(region, threshold):
   min_val, max_val = np.min(region), np.max(region)
   return (max_val - min_val) <= threshold

def split_and_merge(image, threshold):
   def recursive_split(region):
       rows, cols = region.shape
       if rows <= 1 or cols <= 1:
           return np.zeros_like(region, dtype=np.uint8)

       if is_homogeneous(region, threshold):
```

```
            return np.ones_like(region, dtype=np.uint8)

        mid_row, mid_col = rows // 2, cols // 2
        top_left = region[:mid_row, :mid_col]
        top_right = region[:mid_row, mid_col:]
        bottom_left = region[mid_row:, :mid_col]
        bottom_right = region[mid_row:, mid_col:]
        segmented_quadrants = np.zeros_like(region, dtype=np.uint8)
        segmented_quadrants[:mid_row, :mid_col] = recursive_split(top_left)
        segmented_quadrants[:mid_row, mid_col:] = recursive_split(top_right)
        segmented_quadrants[mid_row:, :mid_col] = recursive_split(bottom_left)
        segmented_quadrants[mid_row:, mid_col:] = recursive_split(bottom_right)

        return segmented_quadrants

    def merge_regions(segmented):
        return segmented

    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    segmented_image = recursive_split(image)
    segmented_image = merge_regions(segmented_image)
    return segmented_image

def main():
    file_path = 'R.png'  # Local file path to the image
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

    if image is None:
        print("Error loading image.")
        Return

    threshold = 20  # Adjust this value as needed
    result = split_and_merge(image, threshold)
    cv2.imwrite('segmented_image.png', result * 255)
    cv2.imshow('Segmented Image', result * 255)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

OUTPUT







Conclusion:

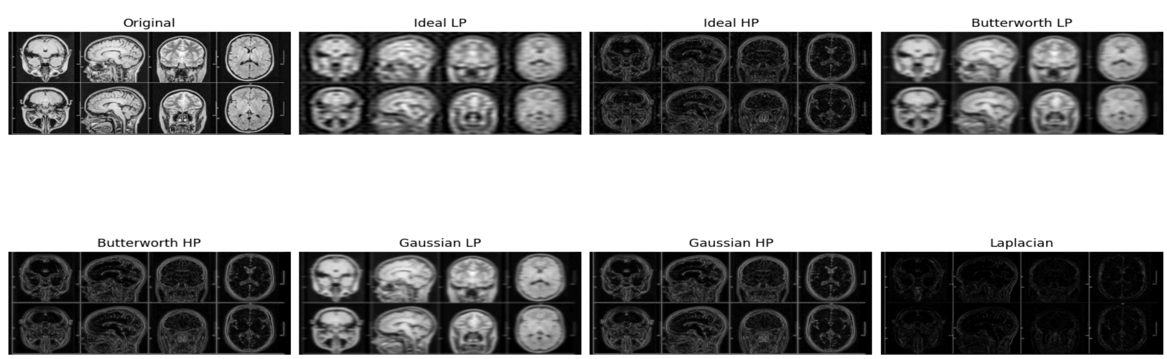
Hence this lab report presents the image segmentation (the partition of an image into components or regions) methods using various methods like region-split,merge,laplacian and mexican filters.

Source Code of High and Low Pass Filters: Image Smoothing

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def apply_all_filters(img_path, cutoff=30):
   # Read image
   img = cv2.imread(img_path, 0)
   rows, cols = img.shape
   crow, ccol = rows//2, cols//2

   # Compute DFT
   dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
   dft_shift = np.fft.fftshift(dft)

   # Create coordinate grid
   x = np.arange(rows) - crow
   y = np.arange(cols) - ccol
   X, Y = np.meshgrid(y, x)
   D = np.sqrt(X**2 + Y**2)

   # Define all filters
   filters = {
       'Ideal LP': D <= cutoff,
       'Ideal HP': D > cutoff,
       'Butterworth LP': 1 / (1 + (D/cutoff)**4),
       'Butterworth HP': 1 / (1 + (cutoff/D)**4),
       'Gaussian LP': np.exp(-D**2/(2*cutoff**2)),
       'Gaussian HP': 1 - np.exp(-D**2/(2*cutoff**2)),
       'Laplacian': -4*np.pi**2 * (X**2 + Y**2)
   }

   # Apply each filter
   results = {}
   plt.figure(figsize=(20, 10))

   # Original image
   plt.subplot(2, 4, 1)
   plt.imshow(img, cmap='gray')
   plt.title('Original')
   plt.axis('off')
```

```
    # Apply and display each filter
    for i, (name, mask) in enumerate(filters.items(), 2):
        # Apply filter
        fshift = dft_shift * mask[:,:,np.newaxis]
        f_ishift = np.fft.ifftshift(fshift)
        filtered = cv2.idft(f_ishift)
        magnitude = cv2.magnitude(filtered[:,:,0], filtered[:,:,1])

        # Normalize result
        result = cv2.normalize(magnitude, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
        results[name] = result

        # Display
        plt.subplot(2, 4, i)
        plt.imshow(result, cmap='gray')
        plt.title(name)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
    return results

# Apply filters
results = apply_all_filters('The-Results-Of-A-Head-CT-Scan.jpg', cutoff=30)
```

Output:



Conclusion:
Hence, the practical experiments confirmed that frequency domain filtering is an effective approach for image processing, with each filter type showing distinct advantages: Ideal filters for strict frequency cutoff, Butterworth for controllable transitions, and Gaussian for smooth, artifact-free results.

Source code of Image Restoration using Median,Mean,Arithmetic, Harmonic,Geometric Mean Filters

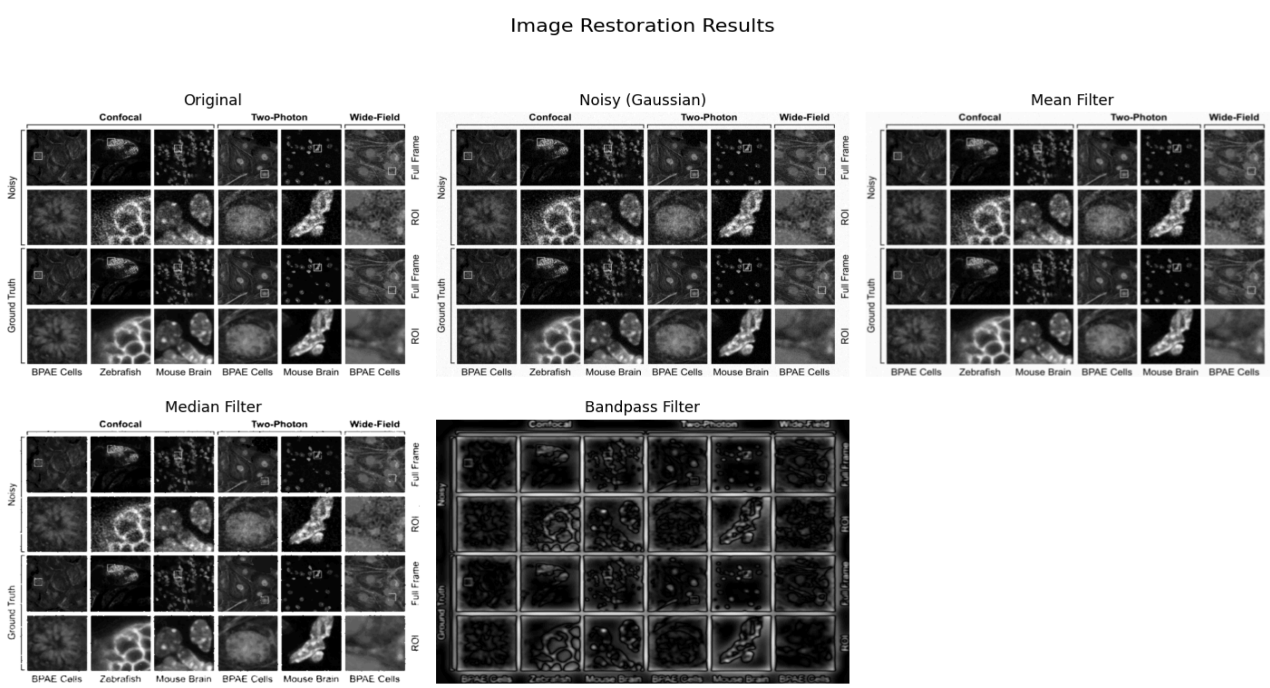
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

class ImageRestoration:
   def __init__(self, image_path):
       self.img = cv2.imread(image_path, 0)
       if self.img is None:
           raise FileNotFoundError("Image not found")

   def add_noise(self, noise_type='gaussian', params=None):
       """Add different types of noise to image"""
       noisy = np.float32(self.img.copy())

       if noise_type == 'gaussian':
           mean, var = params or (0, 50)
           noise = np.random.normal(mean, var**0.5, self.img.shape)
              noisy += noise

       elif noise_type == 'salt_pepper':
           prob = params or 0.05
           mask = np.random.random(self.img.shape) < prob
           noisy[mask] = 255
           mask = np.random.random(self.img.shape) < prob
           noisy[mask] = 0

       return np.uint8(np.clip(noisy, 0, 255))

   def mean_filters(self, img, kernel_size=3, filter_type='arithmetic'):
         """Apply different mean filters"""
       if filter_type == 'arithmetic':
           return cv2.blur(img, (kernel_size, kernel_size))

       elif filter_type == 'geometric':
           kernel = np.ones((kernel_size, kernel_size))
           dst = cv2.filter2D(np.float32(img), -1, kernel)
           dst = np.exp(dst/(kernel_size**2))
           return np.uint8(np.clip(dst, 0, 255))
```

```
        elif filter_type == 'harmonic':
            kernel = np.ones((kernel_size, kernel_size))
            return cv2.filter2D(1.0/img, -1, kernel)

        return img

    def order_statistics_filters(self, img, kernel_size=3, filter_type='median'):
        """Apply order statistics filters"""
        if filter_type == 'median':
            return cv2.medianBlur(img, kernel_size)

        elif filter_type == 'min':
            return cv2.erode(img, np.ones((kernel_size, kernel_size)))

        elif filter_type == 'max':
            return cv2.dilate(img, np.ones((kernel_size, kernel_size)))

        return img


    def bandpass_filter(self, low_cut, high_cut, filter_type='ideal'):
        """Apply bandpass filter in frequency domain"""
        dft = cv2.dft(np.float32(self.img), flags=cv2.DFT_COMPLEX_OUTPUT)
        dft_shift = np.fft.fftshift(dft)

        rows, cols = self.img.shape
        crow, ccol = rows//2, cols//2

        mask = np.zeros((rows, cols))
        for x in range(rows):
            for y in range(cols):
                d = np.sqrt((x-crow)**2 + (y-ccol)**2)

                if filter_type == 'ideal':
                    if low_cut <= d <= high_cut:
                            mask[x,y] = 1

                elif filter_type == 'butterworth':
                    n = 2  # Order of filter
                    mask[x,y] = 1 / (1 + (d/low_cut)**(2*n)) * (1 - 1/(1 +
(d/high_cut)**(2*n)))
```

```python
            elif filter_type == 'gaussian':
                mask[x,y] = np.exp(-((d**2-low_cut**2)/(d*high_cut))**2)

    mask = np.float32(mask)
    fshift = dft_shift * mask[:,:,np.newaxis]
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:,:,0], img_back[:,:,1])

    return np.uint8(cv2.normalize(img_back, None, 0, 255, cv2.NORM_MINMAX))

def main():
    # Initialize
    restorer = ImageRestoration('Lab5Image.png')

    # Add noise
    noisy_gaussian = restorer.add_noise('gaussian', (0, 50))
    noisy_sp = restorer.add_noise('salt_pepper', 0.05)

    # Apply mean filters
    mean_filtered = restorer.mean_filters(noisy_gaussian, 3, 'arithmetic')
    geometric_filtered = restorer.mean_filters(noisy_gaussian, 3, 'geometric')

    # Apply order statistics filters
    median_filtered = restorer.order_statistics_filters(noisy_sp, 3, 'median')
    min_filtered = restorer.order_statistics_filters(noisy_sp, 3, 'min')

    # Apply bandpass filter
    bandpass = restorer.bandpass_filter(30, 80, 'gaussian')


     # Prepare results for display
    results = {
        'Original': restorer.img,
        'Noisy (Gaussian)': noisy_gaussian,
        'Mean Filter': mean_filtered,
        'Median Filter': median_filtered,
        'Bandpass Filter': bandpass
    }

     # Plot images in a 2x3 grid layout
```

```
    fig, axes = plt.subplots(2, 3, figsize=(12, 8))
    fig.suptitle('Image Restoration Results', fontsize=16)

    # List of titles and images
    titles = list(results.keys())
    images = list(results.values())

    for i, ax in enumerate(axes.flat):
        if i < len(images):
            ax.imshow(images[i], cmap='gray')
            ax.set_title(titles[i])
        ax.axis('off')  # Hide axis

    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
    plt.show()

if __name__ == "__main__":
    main()
```

Output:



Conclusion:

This lab demonstrates image restoration techniques like mean, median, and bandpass filters effectively reduce noise and enhance image quality by smoothing out distortions and highlighting details.