Lab 2

Loading gray scale image and apply the following gray level transformation on it

1. **Image Negatives**

**import cv2**

**def image\_negative(image\_path):**

**# Read the image**

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

**# Check if image was successfully loaded**

**if img is None:**

**print("Error: Unable to load image")**

**return**

**# Convert image to negative**

**negative\_img = 255 - img**

**# Display original and negative images**

**cv2.imshow('Original Image', img)**

**cv2.imshow('Negative Image', negative\_img)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**# Save the negative image**

**cv2.imwrite('negative\_image.jpg', negative\_img)**

**# Provide the path to the image**

**image\_path = 'path\_to\_your\_image.jpg'**

**# Call the function to perform image negation**

**image\_negative(image\_path)**

1. **Log Transformations**

log transformation is a common technique used in image processing, especially for enhancing the contrast of images. In Python, you can achieve this using libraries like OpenCV or PIL (Python Imaging Library)

**import cv2**

**import numpy as np**

**# Read the image**

**image = cv2.imread('input\_image.jpg', cv2.IMREAD\_GRAYSCALE)**

**# Apply log transformation**

**c = 1 # Constant for tuning the contrast**

**log\_transformed = c \* np.log(1 + image)**

**# Normalize the result to the range [0, 255]**

**log\_transformed = cv2.normalize(log\_transformed, None, 0, 255, cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)**

**# Display the original and log transformed images**

**cv2.imshow('Original Image', image)**

**cv2.imshow('Log Transformed Image', log\_transformed)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

1. **Power-Law Transformations**

**import numpy as np**

**import cv2**

**def power\_law\_transform(image, gamma):**

**# Normalizing the image to 0-1 range**

**normalized\_image = image / 255.0**

**# Applying the power-law transformation**

**transformed\_image = np.power(normalized\_image, gamma)**

**# Scaling back to 0-255 range**

**transformed\_image = np.uint8(transformed\_image \* 255)**

**return transformed\_image**

**# Load image**

**image = cv2.imread('input\_image.jpg', cv2.IMREAD\_GRAYSCALE)**

**# Define gamma value**

**gamma = 0.5**

**# Apply power-law transformation**

**transformed\_image = power\_law\_transform(image, gamma)**

**# Display original and transformed images**

**cv2.imshow('Original Image', image)**

**cv2.imshow('Transformed Image', transformed\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

1. **Piecewise-Linear Transformation Functions**

**Contrast stretching**

Gray level transformation, also known as intensity transformation or contrast enhancement, is a fundamental image processing technique used to adjust the contrast and brightness of an image. Here's a simple Python code snippet implementing gray level transformation.

**import numpy as np**

**import cv2**

**def gray\_level\_transformation(image, r1, s1, r2, s2):**

**"""**

**Apply gray level transformation to the input image.**

**Args:**

**- image: Input grayscale image**

**- r1: Lower input intensity level**

**- s1: Lower output intensity level**

**- r2: Upper input intensity level**

**- s2: Upper output intensity level**

**Returns:**

**- transformed\_image: Transformed image**

**"""**

**transformed\_image = np.zeros\_like(image)**

**# Iterate through each pixel in the image**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**pixel\_value = image[i, j]**

**# Apply piecewise linear transformation**

**if pixel\_value < r1:**

**transformed\_image[i, j] = s1 / r1 \* pixel\_value**

**elif r1 <= pixel\_value < r2:**

**transformed\_image[i, j] = ((s2 - s1) / (r2 - r1)) \* (pixel\_value - r1) + s1**

**else:**

**transformed\_image[i, j] = ((255 - s2) / (255 - r2)) \* (pixel\_value - r2) + s2**

**return transformed\_image**

**# Load an image (grayscale)**

**image = cv2.imread("input\_image.jpg", cv2.IMREAD\_GRAYSCALE)**

**# Define parameters for transformation**

**r1, s1 = 70, 0**

**r2, s2 = 180, 255**

**# Apply gray level transformation**

**transformed\_image = gray\_level\_transformation(image, r1, s1, r2, s2)**

**# Display original and transformed images**

**cv2.imshow("Original Image", image)**

**cv2.imshow("Transformed Image", transformed\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

* **gray\_level\_transformation** function performs the transformation based on the input parameters **r1**, **s1**, **r2**, and **s2**.
* **r1**, **s1** define the lower threshold input and output intensity levels respectively.
* **r2**, **s2** define the upper threshold input and output intensity levels respectively.
* The transformation is piecewise linear, adjusting the intensity values within the specified ranges.
* **cv2.imshow()** is used to display the original and transformed images.

Make sure to replace **"input\_image.jpg"** with the path to your actual input image file. Also, adjust the parameters **r1**, **s1**, **r2**, and **s2** according to your specific requirements.

**Gray-level slicing**

Below is a Python code implementing Gray Level Slicing in image processing using Python. This code uses the OpenCV library to read and display images. Gray Level Slicing is a technique used to enhance certain features in an image by highlighting specific intensity ranges.

**import cv2**

**import numpy as np**

**def gray\_level\_slicing(image, lower\_thresh, upper\_thresh, background=0, foreground=255):**

**# Convert image to grayscale if it's not already**

**if len(image.shape) == 3:**

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

**# Create a mask of pixels within the specified intensity range**

**mask = cv2.inRange(image, lower\_thresh, upper\_thresh)**

**# Apply the slicing operation**

**result = np.where(mask == 255, foreground, background)**

**return result**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Read the image**

**image = cv2.imread('input\_image.jpg')**

**# Define the intensity range for slicing**

**lower\_thresh = 100**

**upper\_thresh = 200**

**# Apply gray level slicing**

**result = gray\_level\_slicing(image, lower\_thresh, upper\_thresh)**

**# Display the original and sliced images**

**cv2.imshow('Original Image', image)**

**cv2.imshow('Sliced Image', result)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

This code generates a random grayscale image for demonstration purposes. You can replace **image** with your own grayscale image array. Adjust **lower\_thresh** and **upper\_thresh** according to the intensity range you want to slice. The sliced image will have pixels within the specified intensity range set to 255 (white) and other pixels unchanged.

import numpy as np

import matplotlib.pyplot as plt

def gray\_level\_slicing(image, lower\_thresh, upper\_thresh):

result = np.copy(image)

result[(image >= lower\_thresh) & (image <= upper\_thresh)] = 255

return result

if \_\_name\_\_ == "\_\_main\_\_":

# Create a sample grayscale image

image = np.random.randint(0, 256, size=(100, 100), dtype=np.uint8)

# Define the intensity range for slicing

lower\_thresh = 100

upper\_thresh = 200

# Apply gray level slicing

result = gray\_level\_slicing(image, lower\_thresh, upper\_thresh)

# Display the original and sliced images

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

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

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 2, 2)

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

plt.title('Sliced Image')

plt.axis('off')

plt.show()

**Bit-plane slicing**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**def bit\_plane\_slice(image):**

**# Convert image to grayscale**

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

**# Initialize an array to store bit planes**

**bit\_planes = []**

**# Iterate over each bit plane**

**for i in range(8):**

**# Extract the i-th bit plane**

**bit\_plane = np.bitwise\_and(gray\_image, 2 \*\* i)**

**# Normalize the bit plane values to 0-255**

**bit\_plane = bit\_plane \* (255 / (2 \*\* i))**

**# Add the bit plane to the list**

**bit\_planes.append(bit\_plane.astype(np.uint8))**

**return bit\_planes**

**# Read an image**

**image\_path = 'image.jpg'**

**image = cv2.imread(image\_path)**

**# Check if the image was loaded successfully**

**if image is None:**

**print(f"Error: Unable to load image '{image\_path}'")**

**exit(1)**

**# Perform bit plane slicing**

**bit\_planes = bit\_plane\_slice(image)**

**# Display each bit plane using Matplotlib**

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

**for i, plane in enumerate(bit\_planes):**

**plt.subplot(2, 4, i + 1)**

**plt.imshow(plane, cmap='gray')**

**plt.title(f'Bit Plane {i}')**

**plt.axis('off')**

**plt.show()**