LAB # 3

DIGITAL IMAGE PROCESSING

Mentor: Dr. Irfan Ali, PhD
Assistant Professor (AI & MMG)

Aror University of Art, Architechture, Design & Heritage, Sukkur

Object/Task

Apply spatial domain based operations to enhance digital images

- **Spatial Domain Methods**

spatial domain refers to directly working with the pixels of an image. Here, the image is treated as a matrix of pixel intensity values. Operations are performed directly on pixel values rather than transform ing them into another domain (like frequency domain with Fourier transform).

- **Image negatives**

An image negative is one of the simplest and most common image enhancement techniques in digital image processing. It is mainly used to improve the visual quality of images by inverting their intensity values.

- **Image Thresholding**

Thresholding is a point processing technique in the spatial domain where pixel values are converted into binary (black & white) or into simplified intensity levels based on a chosen threshold value. It is widely used for image segmentation, where we separate objects (foreground) from the background.

- **Image contrast stretching**

Contrast stretching (also called normalization) is a point processing technique used to improve the contrast of an image by stretching th e range of intensity values.In many images, pixel values are concen trated in a narrow range (e.g., between 80–150 instead of full 0–25 5 for an 8-bit image). This makes the image look dull or low contra st. Contrast stretching spreads these values across the full dynamic range (0–255), making the image appear clearer and sharper.

Apply the image negative operation using built-in function.

[3]:
```
import cv2
```

[*]:
```
# Read the image
img = cv2.imread("sample2.jpg")

# Create negative using OpenCV
negative = cv2.bitwise_not(img)

# Show results
cv2.imshow("Original", img)
cv2.imshow("Negative", negative)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Apply the image negative operation using built-in function/Alternate.

[*]:
```
import cv2
import numpy as np

# Read the image
img = cv2.imread("your_image.jpg")

# Create negative manually
negative = 255 - img

# Show results
cv2.imshow("Original", img)
cv2.imshow("Negative", negative)

cv2.waitKey(0)
cv2.destroyAllWindows()
```


FREE

Apply the image negative operation on image using matplotlib.

[4]:
```
# Read the color image
img_color = cv2.imread(r"C:\Users\HP\Desktop\test\img2.jpg")

# Convert BGR → RGB for proper display with matplotlib
img_rgb = cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)

# Create negative (applied on RGB image)
negative = 255 - img_rgb

# Show results using matplotlib
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(1,2,2)
plt.title("Negative Image")
plt.imshow(negative)
plt.axis("off")

plt.show()
```

Original Image

Negative Image

Compare the results for negative images using Opencv and Numpy.

[5]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[6]:
```
# Read the image
img_color = cv2.imread("img2.jpg")

# Convert BGR → RGB for matplotlib display
img_rgb = cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)

# ------------------------
# Method 1: Using OpenCV
# ------------------------
negative_cv2 = cv2.bitwise_not(img_rgb)   # Inverts pixel values

# ------------------------
# Method 2: Using NumPy
# ------------------------
negative_numpy = 255 - img_rgb  # Subtraction applied element-wise
```

```
# ------------------------
# Display Results
# ------------------------
plt.figure(figsize=(15,5))

plt.subplot(1,3,1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis("off")
```

```
plt.subplot(1,3,2)
plt.title("Negative (OpenCV)")
plt.imshow(negative_cv2)
plt.axis("off")

plt.subplot(1,3,3)
plt.title("Negative (NumPy)")
plt.imshow(negative_numpy)
plt.axis("off")

plt.show()
```

Original Image

Negative (OpenCV)

Negative (NumPy)

Apply thresholding operation for enhance the image using Opencv

[8]:
```
# Read image in grayscale
img = cv2.imread("img2.jpg", cv2.IMREAD_GRAYSCALE)

# Apply simple binary threshold
ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

# Show results
plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(img, cmap="gray")
plt.axis("off")

plt.subplot(1,2,2)
plt.title("Thresholded Image")
plt.imshow(thresh, cmap="gray")
plt.axis("off")
```

Original Image

Thresholded Image

Apply Contrast stretching operation for enhance the image

[9]:
```
# Read image in grayscale
img = cv2.imread("img2.jpg", cv2.IMREAD_GRAYSCALE)

# Find minimum and maximum pixel values
min_val = np.min(img)
max_val = np.max(img)

# Apply contrast stretching formula
stretched = ((img - min_val) / (max_val - min_val)) * 255
stretched = stretched.astype(np.uint8)

# Show results
plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(img, cmap="gray")
plt.axis("off")

plt.subplot(1,2,2)
plt.title("Contrast Stretched Image")
plt.imshow(stretched, cmap="gray")
plt.axis("off")
```

Task : Apply Negative operation on any image using NumPy Subtraction (grayscale) by using waitKey and destroyAllWindow functions

Task : Apply Negative operation on image using NumPy invert() function

Hint : use (img = cv2.imread("your_image.jpg", 0)

# Using NumPy invert
negative = np.invert(img)

Task : Negative using OpenCV Built-in bitwise_not() with waitKey and destroyAllwindows functions

Task : Apply Negative operation on image for
Color Image Channels Separately

Hint : Use # Split channels
b, g, r = cv2.split(img) then apply negative on each channel

# Merge back
neg_img = cv2.merge([b_neg, g_neg, r_neg])