Lab 3: Image Processing

Name: Gavin Abrigo Student ID: 205260732

1. Histogram equalization

Compute and visualize histogram and cumulative distance function (CDF) of an input gray-scale image

*# Useful code snippet:*

*# read an image as gray-scale image*

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread('./bay.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

raise FileNotFoundError('Could not load ./bay.png')

*# compute image histogram*

hist, bins = np.histogram(img.flatten(), 256, [0, 256])

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

plt.bar(range(256), hist, width=1.0, color='gray')

plt.title('Histogram of bay.png')

plt.xlabel('Pixel intensity')

plt.ylabel('Frequency')

*# compute cumulative density function*

cdf = hist.cumsum()

*# normalize cdf*

cdf\_normalized = cdf / cdf.max()

*# histogram equalization using CDF*

cdf\_masked = np.ma.masked\_equal(cdf, 0)

cdf\_scaled = (cdf\_masked - cdf\_masked.min()) \* 255 / (cdf\_masked.max() - cdf\_masked.min())

cdf\_scaled = np.ma.filled(cdf\_scaled, 0).astype(np.uint8)

eq\_img = cdf\_scaled[img]

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

plt.plot(cdf\_normalized)

plt.xlabel('Pixel intensity')

plt.ylabel('Normalized cumulative frequency')

plt.title('CDF of bay.png')

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

plt.subplot(1, 2, 1)

plt.imshow(img, cmap='gray', vmin=0, vmax=255)

plt.title('Original')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(eq\_img, cmap='gray', vmin=0, vmax=255)

plt.title('Histogram equalized')

plt.axis('off')

plt.tight\_layout()

plt.show()

1. Apply histogram equalization using obtained CDF on the input image

A graph of a line

AI-generated content may be incorrect.

1. Compute and visualize histogram of output image

A graph of a person with a long line

AI-generated content may be incorrect.

A collage of a mountain slope

AI-generated content may be incorrect.

1. Image denoising
2. Read the input image and convert to a grayscale image

img = cv2.imread("./lena.png")

if img is None:

raise FileNotFoundError("Could not load ./lena.png")

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

1. Add two types of noise including Gaussian noise and Salt/Pepper noise (Implement your own functions to add noise to an image)

def add\_gaussian\_noise(

image: np.ndarray,

mean: float = 0.0,

std: float = 15.0,

rng: np.random.Generator | None = None,

) -> np.ndarray:

"""Add Gaussian noise to a grayscale image."""

if rng is None:

rng = np.random.default\_rng()

noise = rng.normal(mean, std, size=image.shape)

noisy = image.astype(np.float32) + noise

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

def add\_salt\_pepper\_noise(

image: np.ndarray,

amount: float = 0.04,

salt\_vs\_pepper: float = 0.5,

rng: np.random.Generator | None = None,

) -> np.ndarray:

"""Add Salt & Pepper noise to a grayscale image."""

if rng is None:

rng = np.random.default\_rng()

noisy = image.copy()

total\_pixels = image.size

num\_salt = int(total\_pixels \* amount \* salt\_vs\_pepper)

num\_pepper = int(total\_pixels \* amount \* (1.0 - salt\_vs\_pepper))

salt\_coords = (

rng.integers(0, image.shape[0], num\_salt),

rng.integers(0, image.shape[1], num\_salt),

)

pepper\_coords = (

rng.integers(0, image.shape[0], num\_pepper),

rng.integers(0, image.shape[1], num\_pepper),

)

noisy[salt\_coords] = 255

noisy[pepper\_coords] = 0

return noisy

1. Implement mean and median filtering in 5x5 windows

gaussian\_noisy = add\_gaussian\_noise(lena\_gray, std=20.0, rng=rng)

*# Gaussian noise denoised with Gaussian blur*

*# 5x5 kernel keeps parity with other operations*

gaussian\_denoised = cv2.GaussianBlur(gaussian\_noisy, (5, 5), sigmaX=0)

salt\_pepper\_noisy = add\_salt\_pepper\_noise(lena\_gray, amount=0.03, salt\_vs\_pepper=0.6, rng=rng)

*# Salt & pepper noise denoised with median blur (distinct method)*

salt\_pepper\_denoised = cv2.medianBlur(salt\_pepper\_noisy, 5)

1. Check if mean or median filtering is able to completely remove Gaussian noise or Salt/Pepper noise. Compare original image and denoised image.

fig, axes = plt.subplots(2, 3, figsize=(12, 8))

fig.suptitle("Noise addition and denoising on lena.png", fontsize=14)

images = [

(lena\_gray, "Original"),

(gaussian\_noisy, "Gaussian noise"),

(gaussian\_denoised, "Gaussian blur (5x5)"),

(lena\_gray, "Original"),

(salt\_pepper\_noisy, "Salt & pepper noise"),

(salt\_pepper\_denoised, "Median blur (5x5)"),

]

A collage of a person wearing a hat

AI-generated content may be incorrect.

1. Image gradient
2. Compute image gradient in x and y direction respectively

img = cv2.imread("./lena.png")

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

*# compute gradient*

gx = cv2.Sobel(lena\_gray, cv2.CV\_64F, 1, 0, ksize=5)

gy = cv2.Sobel(lena\_gray, cv2.CV\_64F, 0, 1, ksize=5)

1. Read the input image and convert to a grayscale image

img = cv2.imread("./lena.png")

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

1. Compute magnitude of image gradient for each pixel
2. magnitude = cv2.magnitude(gx, gy)

magnitude\_norm = cv2.normalize(magnitude, None, 0, 255, cv2.NORM\_MINMAX)

magnitude\_uint8 = magnitude\_norm.astype(np.uint8)

1. Thresholding on magnitude to determine image edges, try various thresholds.

plt.imshow(magnitude\_uint8, cmap="gray")

plt.title("Gradient magnitude of lena.png")

plt.xlabel("Column (pixels)")

plt.ylabel("Row (pixels)")

plt.colorbar(label="Gradient magnitude")

plt.show()

A black and white image of a person wearing a hat

AI-generated content may be incorrect.