IT5437 – Assignment 1: Intensity Transformations and Neighbourhood Filtering

**Name**: Kaumadi I. A. S.

**Index No:** 249311P

**Module:** IT5437 - Computer Vision

**GitHub Repository:** [Click here](https://github.com/kaumadi123/IT5437_Assignment_249311P)

## Q1. Intensity Transformation and Visualization

points = [(0,0), (49,49), (50,100), (150,150), (255,255)]

xs, ys = zip(\*points)

lut = np.interp(np.arange(256), xs, ys).astype('uint8')

out = cv2.LUT(img, lut)

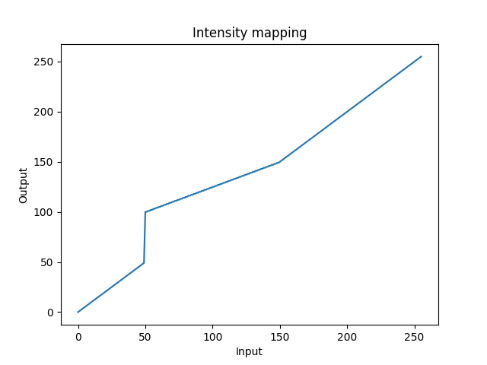


Figure 2: Transformed image

Figure : LUT (Input - Output)

## Q2. Accentuating White/Gray Matter in Brain PD Images

def make\_piecewise(points):

    """points = [(x\_in, x\_out), ...]"""

    xs, ys = zip(\*points)

    lut = np.interp(np.arange(256), xs, ys).astype('uint8')

    return lut

points\_white = [(0,0), (100,60), (140,180), (200,255), (255,255)]

lut\_white = make\_piecewise(points\_white)

out\_white = cv2.LUT(img, lut\_white)

points\_gray = [(0,0), (60,60), (120,200), (160,220), (255,255)]

lut\_gray = make\_piecewise(points\_gray)

out\_gray = cv2.LUT(img, lut\_gray)

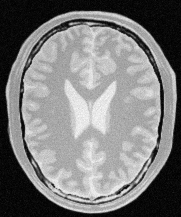
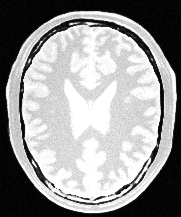


Figure 5: Gray Matter

Figure 4: White Matter

Figure 3: Original Brain Slice

## Q3. Gamma Correction on L\* Channel in Lab Color Space

Figure 6: Intensity Transformations

lab = cv2.cvtColor(img, cv2.COLOR\_BGR2LAB)

L, a, b = cv2.split(lab)

L\_norm = L / 255.0

gamma = 0.8

L\_gamma = np.power(L\_norm, gamma)

L\_corrected = np.clip(L\_gamma \* 255.0, 0, 255).astype("uint8")

lab\_corrected = cv2.merge([L\_corrected, a, b])

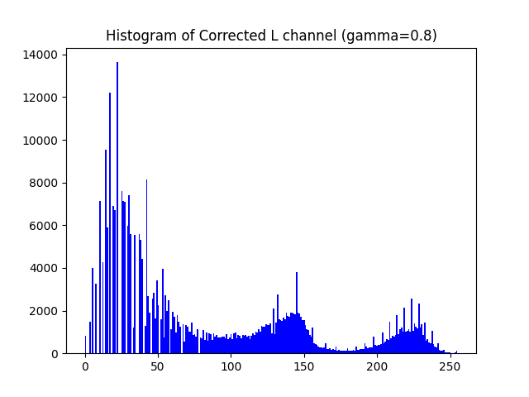
img\_corrected = cv2.cvtColor(lab\_corrected, cv2.COLOR\_LAB2BGR)

Figure 9: Histogram of Corrected L channel (gamma=0.8)

Figure 7: Original

Figure 8: Gamma Corrected

## Q4. Vibrance Enhancement via Nonlinear Saturation Transform

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

h, s, v = cv2.split(hsv)

sigma = 70.0

a = 0.6

def vibrance\_curve\_gaussian(a=0.6, sigma=70.0):

    x = np.arange(256, dtype=np.float32)

    bump = a \* 128.0 \* np.exp(-((x - 128.0) \*\* 2) / (2.0 \* sigma \*\* 2))

    y = np.minimum(x + bump, 255.0)

    return y.astype(np.uint8)

lut = vibrance\_curve\_gaussian(a=a, sigma=sigma)

s\_enhanced = cv2.LUT(s, lut)

hsv\_enhanced = cv2.merge([h, s\_enhanced, v])

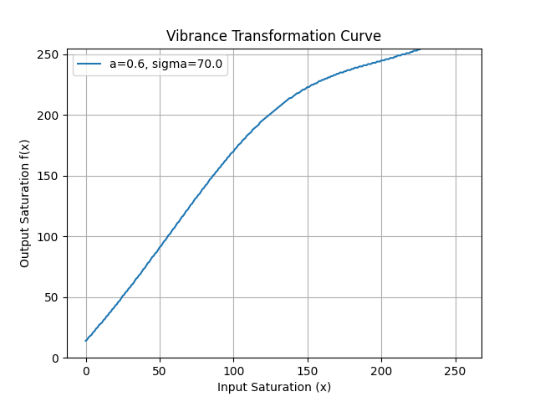
img\_enhanced = cv2.cvtColor(hsv\_enhanced, cv2.COLOR\_HSV2BGR)

Figure 1: Vibrance Transformation Curve

Figure 11: Vibrance

Figure 10: Original

## Q5. Foreground-Only Histogram Equalization in HSV

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

h, s, v = cv2.split(hsv)

\_, mask = cv2.threshold(v, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

cv2.imwrite("results/q5\_mask.png", mask)

fg = cv2.bitwise\_and(v, v, mask=mask)

cv2.imwrite("results/q5\_foreground.png", fg)

hist = cv2.calcHist([v], [0], mask, [256], [0, 256]).flatten()

cdf = hist.cumsum()

cdf\_min = cdf[cdf > 0][0]

N = mask.sum() / 255

lut = np.floor((cdf - cdf\_min) / (N - cdf\_min) \* 255).clip(0, 255).astype(np.uint8)

v\_eq = v.copy()

v\_eq[mask == 255] = lut[v[mask == 255]]

hsv\_eq = cv2.merge([h, s, v\_eq])

result = cv2.cvtColor(hsv\_eq, cv2.COLOR\_HSV2BGR)

Figure 1: Hue (H)

Figure 14: Saturation (S)

Figure 15: Value (V)

Q6. Sobel Filtering Implementation Methods

Figure 16: Result

gx\_a = cv2.filter2D(img, cv2.CV\_32F, sobel\_x)

gy\_a = cv2.filter2D(img, cv2.CV\_32F, sobel\_y)

mag\_a = cv2.magnitude(gx\_a, gy\_a)

cv2.imwrite("results/q6\_sobel\_a.png", np.uint8(mag\_a))

def conv2d(image, kernel):

    kh, kw = kernel.shape

    pad\_h, pad\_w = kh // 2, kw // 2

    padded = np.pad(image, ((pad\_h, pad\_h), (pad\_w, pad\_w)), mode='constant')

    out = np.zeros\_like(image, dtype=np.float32)

    for i in range(image.shape[0]):

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

            region = padded[i:i+kh, j:j+kw]

            out[i, j] = np.sum(region \* kernel)

    return out

gx\_b = conv2d(img, sobel\_x)

gy\_b = conv2d(img, sobel\_y)

mag\_b = np.sqrt(gx\_b\*\*2 + gy\_b\*\*2)

cv2.imwrite("results/q6\_sobel\_b.png", np.uint8(mag\_b))

kcol = np.array([1, 2, 1], dtype=np.float32).reshape(3,1)

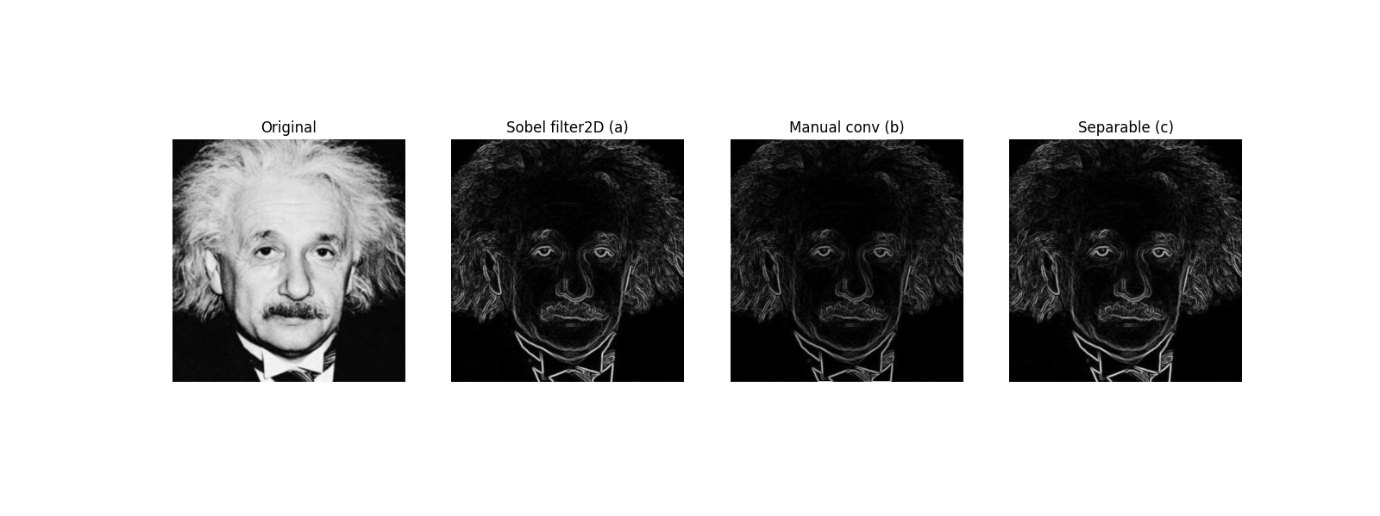
krow = np.array([1, 0, -1], dtype=np.float32).reshape(1,3)

Figure 17: Comparison

## Q7. Image Zoom with Interpolation Methods and SSD Comparison

def zoom(img, s, method="nearest"):

    h, w = img.shape[:2]

    new\_h, new\_w = int(h \* s), int(w \* s)

    out = np.zeros((new\_h, new\_w, img.shape[2]), dtype=img.dtype) if img.ndim == 3 else np.zeros((new\_h, new\_w), dtype=img.dtype)

    for y in range(new\_h):

        for x in range(new\_w):

            src\_x = x / s

            src\_y = y / s

            if method == "nearest":

                nx = int(round(src\_x))

                ny = int(round(src\_y))

                nx = min(nx, w - 1)

                ny = min(ny, h - 1)

                out[y, x] = img[ny, nx]

            elif method == "bilinear":

                x0 = int(np.floor(src\_x))

                x1 = min(x0 + 1, w - 1)

                y0 = int(np.floor(src\_y))

                y1 = min(y0 + 1, h - 1)

                dx = src\_x - x0

                dy = src\_y - y0

                if img.ndim == 2:

                    val = (img[y0, x0] \* (1 - dx) \* (1 - dy) +

                           img[y0, x1] \* dx \* (1 - dy) +

                           img[y1, x0] \* (1 - dx) \* dy +

                           img[y1, x1] \* dx \* dy)

                else:

                    val = (img[y0, x0, :] \* (1 - dx) \* (1 - dy) +

                           img[y0, x1, :] \* dx \* (1 - dy) +

                           img[y1, x0, :] \* (1 - dx) \* dy +

                           img[y1, x1, :] \* dx \* dy)

                out[y, x] = np.clip(val, 0, 255)

    return out.astype(img.dtype)

Figure 19: Pair 02 Comparison

Figure 18: Pair 01 Comparison

## Q8. GrabCut Segmentation with Background Blur

mask = np.zeros(img.shape[:2], np.uint8)

bgdModel = np.zeros((1, 65), np.float64)

fgdModel = np.zeros((1, 65), np.float64)

h, w = img.shape[:2]

rect = (int(w\*0.2), int(h\*0.1), int(w\*0.6), int(h\*0.8))  # (x,y,w,h)

# Apply GrabCut

cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

# Final mask: probable/definite foreground -> 1, background -> 0

mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")

foreground = img \* mask2[:, :, np.newaxis]

background = img \* (1 - mask2[:, :, np.newaxis])

blurred\_bg = cv2.GaussianBlur(img, (25, 25), 0)

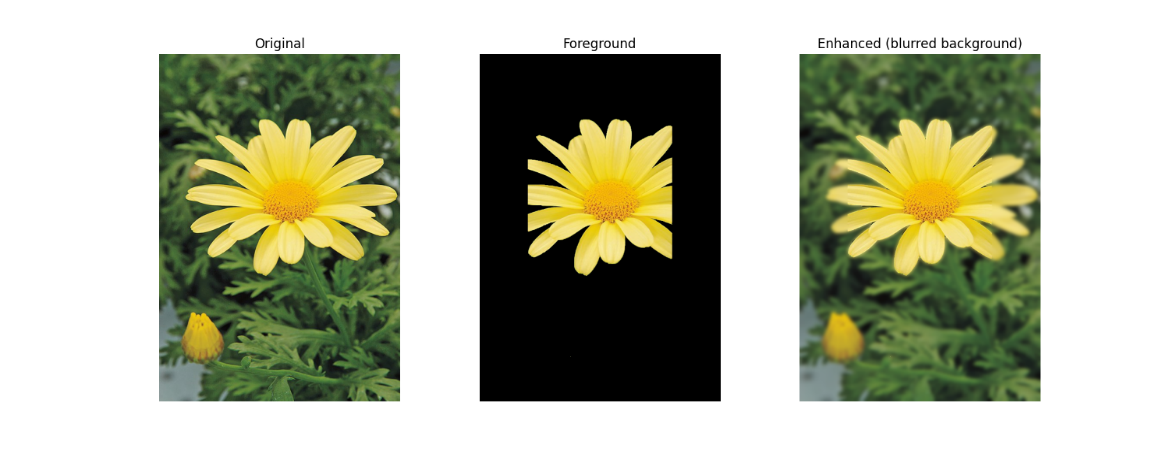
enhanced = blurred\_bg \* (1 - mask2[:, :, np.newaxis]) + foreground

Figure 20: Background Blur Comparison

## Q9. Rice Image Denoising, Otsu Segmentation, and Grain Counting

g\_d = cv2.GaussianBlur(img, (3, 3), 0.8)

k = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (31, 31))

bg = cv2.morphologyEx(g\_d, cv2.MORPH\_OPEN, k)

flat = cv2.subtract(g\_d, bg)

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

\_, th\_raw = cv2.threshold(flat, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

if np.mean(flat[th\_raw > 0]) < np.mean(flat[th\_raw == 0]):

    th\_raw = cv2.bitwise\_not(th\_raw)

se = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (3, 3))

mask = cv2.morphologyEx(th\_raw, cv2.MORPH\_OPEN, se, iterations=1)

mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, se, iterations=2)

num, lab = cv2.connectedComponents(mask)

areas = np.bincount(lab.ravel())

keep = np.ones(num, np.uint8)

keep[0] = 0

minA, maxA = 30, 10000

for i, a in enumerate(areas):

    if i == 0:

        continue

    if a < minA or a > maxA:

        keep[i] = 0

mask = (keep[lab] \* 255).astype(np.uint8)

def fill\_holes(bin255):

    inv = cv2.bitwise\_not(bin255)

    h, w = inv.shape

    ffmask = np.zeros((h + 2, w + 2), np.uint8)

    flood = inv.copy()

    cv2.floodFill(flood, ffmask, (0, 0), 255)

    holes = cv2.bitwise\_not(flood)

    return cv2.bitwise\_or(bin255, holes)

mask\_filled = fill\_holes(mask)

bin8 = (mask\_filled > 0).astype(np.uint8)

dist = cv2.distanceTransform(bin8, cv2.DIST\_L2, 5)

markers = (dist > 0.45 \* dist.max()).astype(np.uint8)

\_, markers = cv2.connectedComponents(markers)

markers = markers + 1

markers[bin8 == 0] = 0

rgb\_for\_ws = cv2.cvtColor(img, cv2.COLOR\_GRAY2RGB)

ws = cv2.watershed(rgb\_for\_ws.copy(), markers.astype(np.int32))

seg = (ws > 1).astype(np.uint8) \* 255

n\_final, \_ = cv2.connectedComponents((seg > 0).astype(np.uint8))

count = n\_final - 1

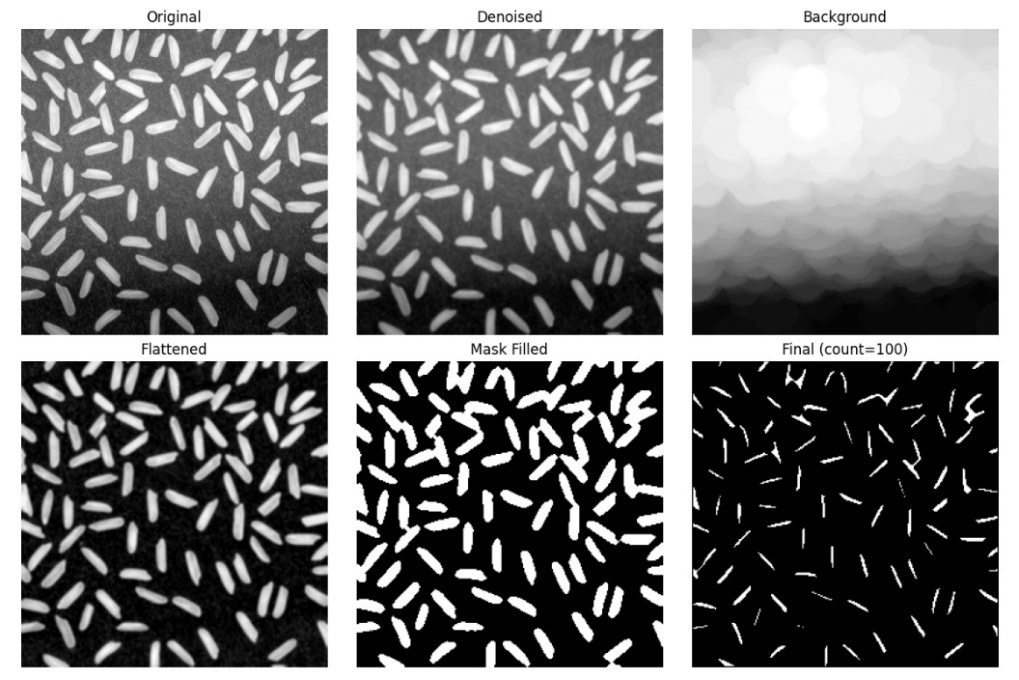


Figure 21: Comparison

## Q10. Sapphire Segmentation and Area Estimation

rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

# HSV range for blue sapphires

lower = np.array([100, 60, 40], dtype=np.uint8)

upper = np.array([140, 255, 255], dtype=np.uint8)

mask = cv2.inRange(hsv, lower, upper)

# Morphological cleanup

se = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))

mask = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, se, iterations=1)

mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, se, iterations=2)

# (b) Fill holes

def fill\_holes(bin255):

    inv = cv2.bitwise\_not(bin255)

    h, w = inv.shape

    ffmask = np.zeros((h + 2, w + 2), np.uint8)

    flood = inv.copy()

    cv2.floodFill(flood, ffmask, (0, 0), 255)

    holes = cv2.bitwise\_not(flood)

    return cv2.bitwise\_or(bin255, holes)

filled = fill\_holes(mask)

# Keep only the two largest components

num, lab, stats, \_ = cv2.connectedComponentsWithStats(filled, connectivity=8)

areas = [(i, stats[i, cv2.CC\_STAT\_AREA]) for i in range(1, num)]

areas\_sorted = sorted(areas, key=lambda t: t[1], reverse=True)[:2]

keep = {i for i, \_ in areas\_sorted}

mask\_two = np.where(np.isin(lab, list(keep)), 255, 0).astype(np.uint8)

mask\_two = fill\_holes(mask\_two)

# (c) Areas in pixels

num2, lab2, stats2, cents2 = cv2.connectedComponentsWithStats(mask\_two, connectivity=8)

areas\_px = [int(stats2[i, cv2.CC\_STAT\_AREA]) for i in range(1, num2)]

# (d) Convert to mm²

f\_mm = 8.0

Z\_mm = 480.0

pixel\_pitch\_mm = 0.0048

mm\_per\_px = (Z\_mm / f\_mm) \* pixel\_pitch\_mm

mm2\_per\_px = mm\_per\_px \*\* 2

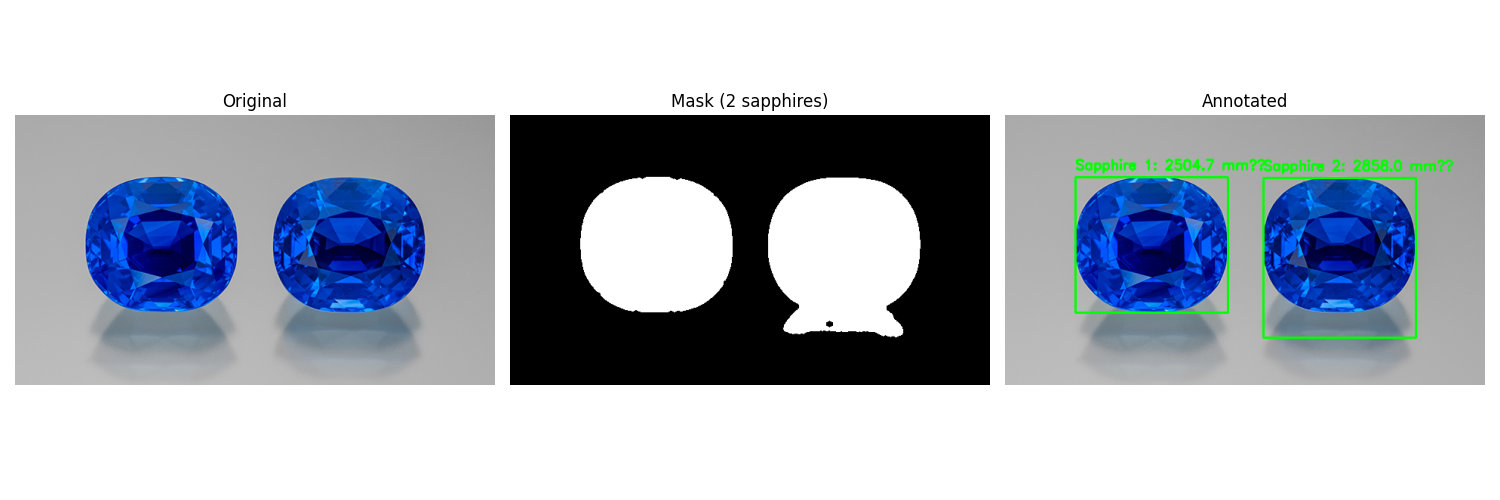
areas\_mm2 = [round(a \* mm2\_per\_px, 3) for a in areas\_px]

Figure 22: Sapphire Area Measurement