Name: Kaumadi I. A. S.
Index No: 249311P
Module: IT5437 - Computer Vision
GitHub Repository: Click here

IT5437 – Assignment 1: Intensity Transformations and Neighbourhood Filtering

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)
```

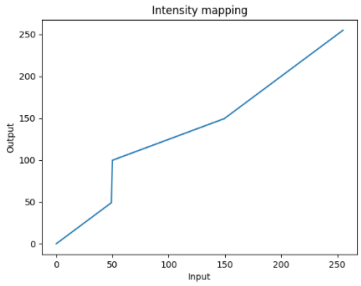

Figure 1: LUT (Input - Output)

Figure 2: Transformed image

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 3: Original Brain Slice

Figure 4: White Matter

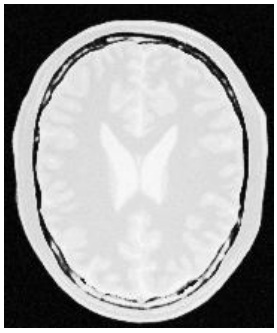

Figure 5: Gray Matter

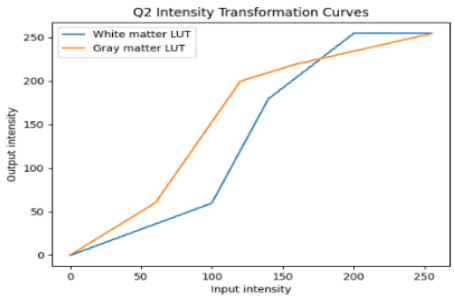

Figure 6: Intensity Transformations

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

```
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 7: Original

Figure 8: Gamma Corrected

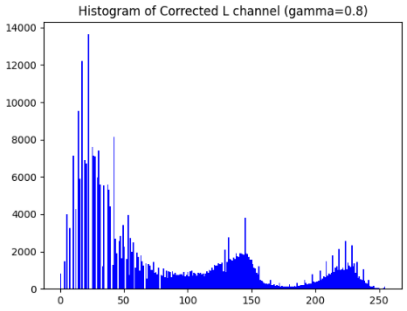

Figure 9: Histogram of Corrected L channel (gamma=0.8)

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 10: Original

Figure 11: Vibrance

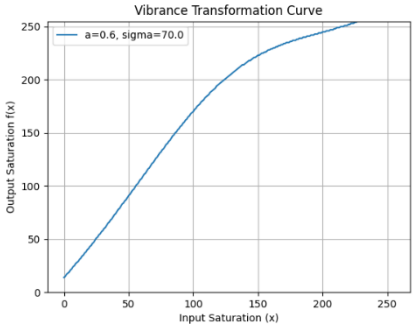

Figure 12: Vibrance Transformation Curve

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 13: Hue (H)

Figure 14: Saturation (S)

Figure 15: Value (V)

Figure 16: Result

Q6. Sobel Filtering Implementation Methods

```
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)
```

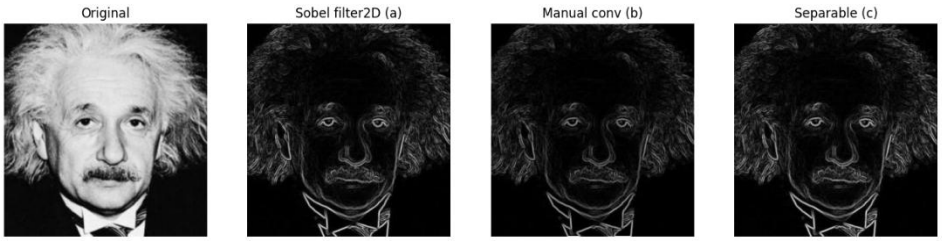

Figure 17: Comparison

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 18: Pair 01 Comparison

Figure 19: Pair 02 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
```

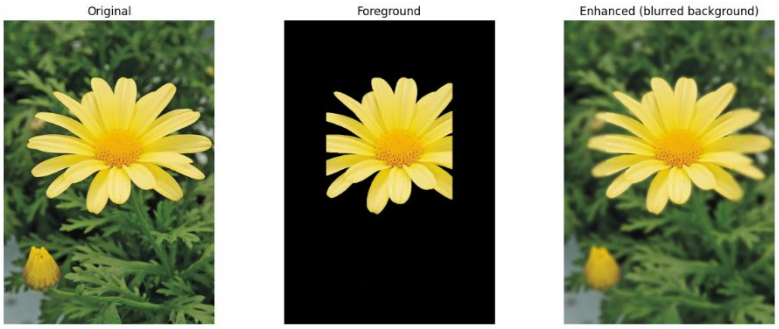

Figure 20: Background Blur Comparison

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
```

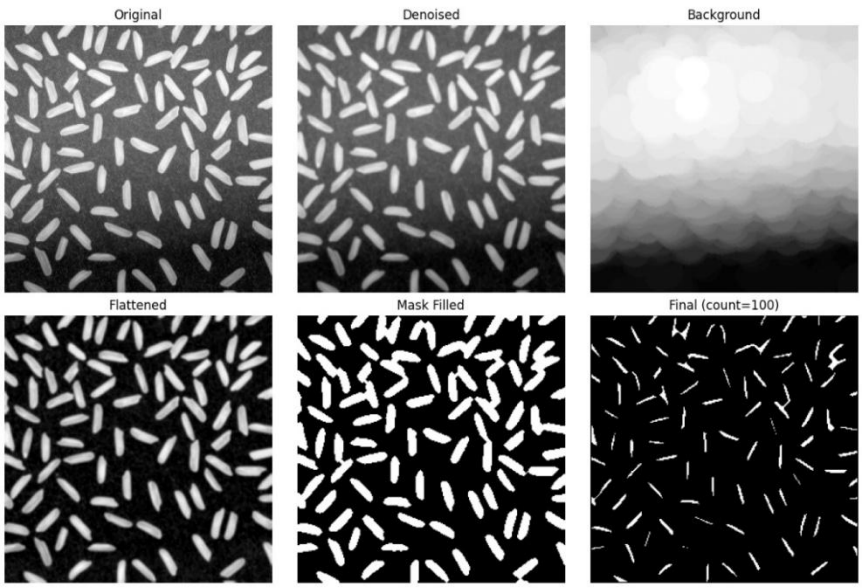

Figure 21: Comparison

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]
```

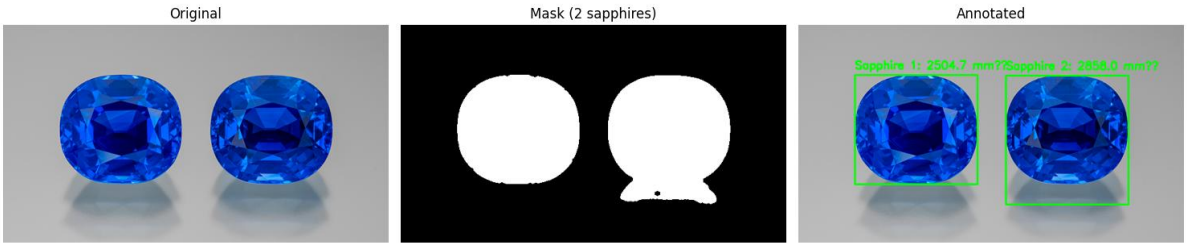

Figure 22: Sapphire Area Measurement