**台科大 111學年度「彩色影像處理」 作業二：影像分析應用**

M11125016 蕭強

Programming language: Python

第一部分「景深擴張」

import numpy as np

import cv2

\_\_lapFil\_\_ = np.array([[-1, -1, -1],

                       [-1, 8, -1],

                       [-1, -1, -1]])

def imgProc(img):

    """

    讀取影像

    """

    img = img / 255

    img = img.astype(np.float32)

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

    return img

def laplacian\_filter(img):

    """

    使用Laplacian濾鏡進行影像銳利化

    """

    img = imgProc(img)

    h, w = img.shape

    kh, kw = \_\_lapFil\_\_.shape

    pad = int((kh - 1) / 2)

    pad\_img = np.pad(img, (pad, pad), "symmetric")

    output = np.zeros([h, w], dtype=np.float32)

    for y in range(h):

        for x in range(w):

            ci = pad\_img[y:y+3, x:x+3]  # ci is crop image

            result = (ci \* \_\_lapFil\_\_).sum()

            output[y, x] = result

    return abs(output) \* 3

def median\_filter(img):

    """

    均值綠波濾鏡使影像不破碎化

    """

    mean\_kernel = np.ones([23, 23]) / 23\*\*23

    h, w = img.shape

    kh, kw = mean\_kernel.shape

    pad = int((kh - 1) / 2)

    pad\_img = np.pad(img, (pad, pad), "symmetric")

    output = np.zeros([h, w], dtype=np.float32)

    for y in range(h):

        for x in range(w):

            cp = pad\_img[y:y+23, x:x+23]

            result = (cp \* mean\_kernel).sum()

            output[y, x] = result

    return output

def imgTreshold(img):

    """

    二值化函數

    """

    h, w = img.shape

    output = np.zeros([h, w], dtype=np.float32)

    for y in range(h):

        for x in range(w):

            pixel = img[y, x]

            if pixel > 0:

                pixel = 1

            else:

                pixel = 0

            output[y, x] = pixel

    return output

def cvtDtype(img):

    output = (img \* 255).astype(np.uint8)

    return output

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

    # 2. 讀取並顯示對焦在前景(fg)與背景(bg)的兩幅影像，並轉換至float格式。

    fgimg = cv2.imread('./depthOfField/2fg.jpg')

    bgimg = cv2.imread('./depthOfField/2bg.jpg')

    # 4. 高通濾波：將兩幅影像由彩色轉換至灰階格式，並分別做 Laplacian 高通濾波後，取絕對值。

    fg\_hipass = laplacian\_filter(fgimg)

    bg\_hipass = laplacian\_filter(bgimg)

    # 6. 製作前景遮罩mask = fg\_hipass - bg\_hipass

    mask = fg\_hipass - bg\_hipass

    # 7. 將遮罩做「均值濾波」，濾鏡尺寸要很大，才不至於使區塊破碎。

    mask = median\_filter(mask)

    # 8. 以0為門檻，將前景遮罩二值化。

    img\_tresh = imgTreshold(mask)

    # 10. 根據二值遮罩分別取前景(fg)與背景(bg)的清晰像素，組成景深擴增影像。

    img\_tresh\_index = np.argwhere(img\_tresh > 0)

    new\_img = bgimg.copy()

    new\_img[img\_tresh\_index[:, 0], img\_tresh\_index[:, 1]] = fgimg[img\_tresh\_index[:, 0], img\_tresh\_index[:, 1]]

    # 12. 儲存景深擴增影像。

    fg\_hipass = cvtDtype(fg\_hipass)

    bg\_hipass = cvtDtype(bg\_hipass)

    img\_tresh = cvtDtype(img\_tresh)

    img\_tresh = cv2.cvtColor(img\_tresh, cv2.COLOR\_GRAY2BGR)

    print(img\_tresh.shape)

    cv2.imwrite("hw1/fg\_hipass.jpg", fg\_hipass)

    cv2.imwrite("hw1/bg\_hipass.jpg", bg\_hipass)

    cv2.imwrite("hw1/img\_tresh.jpg", img\_tresh)

    cv2.imwrite("hw1/new\_img.jpg", new\_img)

    # cv2.imshow("a", new\_img)

    # cv2.waitKey(0)

    # cv2.destroyAllWindows

|  |  |
| --- | --- |
| A picture containing text, reptile, indoor, black  Description automatically generated  Figure 1. 對焦在前景的圖片套用Laplacian濾鏡 | Figure 2. 對焦在後景圖片套用Laplacian濾鏡 |
| Figure 3. 二值化後的情景遮罩 | Figure 4. 景深擴增影像 |

第二部分「視覺異常模擬」

import cv2

import numpy as np

M = np.array([[0.412453, 0.357580, 0.180423],  # Matrix of RGB to XYZ

              [0.212671, 0.715160, 0.072169],

              [0.019334, 0.119193, 0.950227]])

def \_\_f\_\_(img\_XYZ):

    return np.power(img\_XYZ, 1 / 3) if img\_XYZ > 0.00856 else (7.787 \* img\_XYZ) / (16 / 116)

def \_\_anti\_f\_\_(img\_XYZ):

    return np.power(img\_XYZ, 3) if img\_XYZ > (6 / 29) else 3 \* ((6 / 29) \*\* 2) \* (img\_XYZ - 4 / 29)

# region of RGB to Lab

def \_\_BGR2XYZ\_\_(pixel):

    b, g, r = pixel[0], pixel[1], pixel[2]

    rgb = np.array([r, g, b])  # The order list is BGR via opencv. So I transform the order form BGR to RBG

    XYZ = np.dot(M, rgb.T)

    XYZ = XYZ / 255  # normalize

    return XYZ[0] / 0.950456, XYZ[1] / 1, XYZ[2] / 1.088754

def \_\_XYZ2Lab\_\_(XYZ):

    F\_XYZ = [\_\_f\_\_(X) for X in XYZ]

    L = 116 \* F\_XYZ[1] - 16 if XYZ[1] > 0.00856 else 903.3 \* XYZ[1]

    a = 500 \* (F\_XYZ[0] - F\_XYZ[1])

    b = 200 \* (F\_XYZ[1] - F\_XYZ[2])

    return L, a, b

def BGR2Lab(img):

    h = img.shape[0]

    w = img.shape[1]

    Lab = np.zeros([h, w, 3])

    for y in range(h):

        for x in range(w):

            XYZ = \_\_BGR2XYZ\_\_(img[y, x])

            result = \_\_XYZ2Lab\_\_(XYZ)

            Lab[y, x] = result[0], result[1], result[2]

    return Lab

# end region

# region of Lab to BGR

def \_\_Lab2XYZ\_\_(Lab):

    fY = (Lab[0] + 16.0) / 116.0

    fX = (Lab[1] / 500.0) + fY

    fZ = fY - Lab[2] / 200.0

    X = \_\_anti\_f\_\_(fX)

    Y = \_\_anti\_f\_\_(fY)

    Z = \_\_anti\_f\_\_(fZ)

    X = X \* 0.95047

    Y = Y \* 1

    Z = Z \* 1.0883

    return X, Y, Z

def \_\_XYZ2RGB\_\_(XYZ):

    XYZ = np.array(XYZ)

    XYZ = XYZ \* 255

    rgb = np.dot(np.linalg.inv(M), XYZ.T)

    rgb = np.uint8(np.clip(rgb, 0, 255))

    return rgb

def Lab2BGR(img):

    h = img.shape[0]

    w = img.shape[1]

    new\_img = np.zeros([h, w, 3])

    for y in range(h):

        for x in range(w):

            XYZ = \_\_Lab2XYZ\_\_(img[y, x])

            RGB = \_\_XYZ2RGB\_\_(XYZ)

            new\_img[y, x] = RGB[2], RGB[1], RGB[0]

    new\_img = new\_img.astype(np.uint8)

    return new\_img

# end region

def rg\_blind(Lab):

    h = Lab.shape[0]

    w = Lab.shape[1]

    rg\_img = np.zeros([h, w, 3])

    for y in range(h):

        for x in range(w):

            pixel = Lab[y, x]

            if pixel[1] != 0:

                pixel[1] = 0

            else:

                pixel[1] = pixel[1]

            rg\_img[y, x] = (pixel[0], pixel[1], pixel[2])

    return rg\_img

def yb\_blind(Lab):

    h = Lab.shape[0]

    w = Lab.shape[1]

    yb\_img = np.zeros([h, w, 3])

    for y in range(h):

        for x in range(w):

            pixel = Lab[y, x]

            if pixel[2] != 0:

                pixel[2] = 0

            else:

                pixel[2] = pixel[2]

            yb\_img[y, x] = (pixel[0], pixel[1], pixel[2])

    return yb\_img

def matlab\_style\_gauss2D(shape=(5, 5), sigma=0.5):

    """

    2D gaussian mask - should give the same result as MATLAB's

    fspecial('gaussian',[shape],[sigma])

    """

    m, n = [(ss - 1.) / 2. for ss in shape]

    y, x = np.ogrid[-m:m + 1, -n:n + 1]

    h = np.exp(-(x \* x + y \* y) / (2. \* sigma \* sigma))

    h[h < np.finfo(h.dtype).eps \* h.max()] = 0

    sumh = h.sum()

    if sumh != 0:

        h /= sumh

    return h

def glaucoma(img):

    h = img.shape[0]

    w = img.shape[1]

    fil = matlab\_style\_gauss2D(shape=(h, w), sigma=100)

    fil = fil / np.nanmax(fil)

    b, g, r = cv2.split(img)

    b = np.multiply(b, fil)

    g = np.multiply(g, fil)

    r = np.multiply(r, fil)

    result = cv2.merge([b, g, r])

    result = result.astype(np.uint8)

    return result

def rgBlindSim(img):

    lab = BGR2Lab(img)

    rg = rg\_blind(lab)

    new\_img = Lab2BGR(rg)

    return new\_img

def ybBlindSim(img):

    lab = BGR2Lab(img)

    yb = yb\_blind(lab)

    new\_img = Lab2BGR(yb)

    return new\_img

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

    img = cv2.imread('cry1.jpg')

    # 1. 紅綠色盲：自行找一張色彩豐富的圖片，將RGB影像轉換至浮點格式，再轉換至LAB空間，將a\*設為0，再轉回RGB空間。

    rg\_blind\_img = rgBlindSim(img)

    # 2. 黃藍色盲：將RGB影像轉換至浮點格式，再轉換至LAB空間，將b\*設為0，再轉回RGB空間。

    yb\_blind\_img = ybBlindSim(img)

    # 3. 青光眼：讀取RGB 影像的尺寸，利用fspecial 函式建立與影像同尺寸的2D高斯濾鏡(Gaussain filter)，sigma 值必須很高，才有效果。

    #    將濾鏡數值矩陣的每個數值除以其最大值。再將濾鏡點對點乘上影像的RGB 值。模擬青光眼患者視野狹窄的現象。

    gl\_img = glaucoma(img)

    cv2.imwrite("hw2/rg\_blind\_img.jpg", rg\_blind\_img)

    cv2.imwrite("hw2/yb\_blind\_img.jpg", yb\_blind\_img)

    cv2.imwrite("hw2/gl\_img.jpg", gl\_img)

    # cv2.imshow("a", img)

    # cv2.imshow("b", rg\_blind\_img)

    # cv2.imshow("c", yb\_blind\_img)

    # cv2.imshow("d", gl\_img)

    # cv2.waitKey(0)

    # cv2.destroyAllWindows

|  |  |
| --- | --- |
| Figure 5. 一般人 | Figure 6. 紅綠色盲 |
| Figure 7. 黃藍色盲 | Figure 8. 青光眼 |

第三部分「以多維空間分析樹葉的差異」

import numpy as np

import cv2

import os

def imgMask(img, threshold):

    """

    :param img:

    :param threshold:

    :return: image mask in float32, [0, 1]

    """

    img\_ = img.copy()

    h, w = img\_.shape[0], img\_.shape[1]

    output = np.zeros([h, w]).astype(np.float32)

    img\_ = cv2.cvtColor(img\_, cv2.COLOR\_BGR2GRAY)

    img\_ = img\_.astype(np.float32)

    for y in range(img\_.shape[0]):

        for x in range(img\_.shape[1]):

            pixel = img\_[y, x]

            if pixel > threshold:

                new\_pixel = 0

            else:

                new\_pixel = 1

            output[y, x] = new\_pixel

    return output

def maskingImg(img, threshold):

    img\_ = img.copy()

    img\_mask = imgMask(img\_, threshold)

    img\_mask = np.argwhere(img\_mask != 1)

    img\_[img\_mask[:, 0], img\_mask[:, 1]] = 0

    img\_ = (img\_ / 255).astype(np.float32)

    return img\_

def imgCenter(img):

    img\_ = img.copy()

    img\_ = imgMask(img\_, 250)

    h, w = img\_.shape[0], img\_.shape[1]

    img\_area = img\_.sum()

    sum\_x = 0

    sum\_y = 0

    for y in range(h):

        for x in range(w):

            if img\_[y, x] == 1.0:

                sum\_y += y

                sum\_x += x

    cy = round(sum\_y / img\_area)

    cx = round(sum\_x / img\_area)

    return cy, cx

def plotCenter(img):

    img\_ = img.copy()

    h, w = img\_.shape[0], img\_.shape[1]

    cy, cx = imgCenter(img\_)

    img\_masked = imgMask(img\_, 250)

    img\_masked = cv2.cvtColor(img\_masked, cv2.COLOR\_GRAY2BGR)  # 3-D

    new\_img = cv2.line(img\_masked, (0, cy), (w, cy), (0, 0, 1), 1)

    new\_img = cv2.line(new\_img, (cx, 0), (cx, h), (0, 0, 1), 1)

    return new\_img

def sigCurve(img, bins=60):

    img\_ = img.copy()

    img\_masked = imgMask(img\_, 250)

    interval = 360 / bins

    r\_histogram = np.zeros([bins, 1])

    cy, cx = imgCenter(img\_)

    for y in range(img\_masked.shape[0]):

        for x in range(img\_masked.shape[1]):

            if img\_masked[y, x] == 1:

                theta = np.mod(np.arctan2(cy - y, cx - x) \* 180 / np.pi, 360)

                i = int(np.ceil(theta / interval))

                i = int(np.floor(theta / interval))

                r = np.sqrt((cy - y) \*\* 2 + (cx - x) \*\* 2)

                if r > r\_histogram[i, 0]:

                    r\_histogram[i] = r

    # x = np.arange(r\_histogram.size)

    img\_gradient = abs(np.gradient(r\_histogram, axis=0)).sum()

    avg\_img\_gradient = img\_gradient / bins

    return avg\_img\_gradient

def avgLightness(img):

    img\_ = img.copy()

    img\_masked = maskingImg(img\_, 250)

    img\_masked = cv2.cvtColor(img\_masked, cv2.COLOR\_BGR2GRAY).astype(np.float32)

    gray\_index = np.argwhere(img\_masked != 0)

    avg = img\_masked.sum() / gray\_index.shape[0]

    return avg

def redRatio(img):

    img\_ = img.copy()

    img\_masked = maskingImg(img\_, 250)

    ch\_ratio = []

    for ch in img\_masked[:, :, 0], img\_masked[:, :, 1], img\_masked[:, :, 2]:  # BGR

        channel\_index = np.argwhere(ch > 0)

        ch\_avg = ch.sum() / channel\_index.shape[0]

        ch\_ratio.append(ch\_avg)

    output = ch\_ratio[2] / sum(ch\_ratio)

    return output

def avgHiPass(img):

    img\_ = img.copy()

    mask = imgMask(img\_, 250)

    mask\_indexing = np.argwhere(mask != 0)

    laplac\_img = cv2.Laplacian(img\_, -1, ksize=3)

    laplac\_img = abs(laplac\_img)

    output = laplac\_img.sum() / mask\_indexing.shape[0]

    return output

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

    y = 800

    x = 800

    blank\_img\_lr = np.ones([y, x, 3]).astype(np.float32)

    blank\_img\_ls = np.ones([y, x, 3]).astype(np.float32)

    folder\_dir = "C:/Users/cghsi/Desktop/HW2/leaves/"

    list1 = []

    for i in os.listdir(folder\_dir):

        img = cv2.imread(folder\_dir + i)

        img\_float = img.copy()

        img\_float = img\_float / 255

        masked\_img = imgMask(img, 250)

        masked\_img\_index = np.argwhere(masked\_img != 0)

        # 平均亮度

        img\_avg\_lightness = avgLightness(img)

        # 紅色平均占比

        img\_red\_ratio = redRatio(img)

        # 樹葉特徵分布圖1的座標

        by = np.floor(img\_avg\_lightness \* y).astype(np.int32)

        bx = np.floor(img\_red\_ratio \* x).astype(np.int32)

        # 平均高頻強度

        img\_avg\_lap = avgHiPass(img) / 400

        # 簽名曲線的梯度平均

        img\_avg\_sig = sigCurve(img) / 6.120549287353109

        # 樹葉特徵分布圖2的座標

        cy = np.floor(img\_avg\_lap \* y).astype(np.int32)

        cx = np.floor(img\_avg\_sig \* x).astype(np.int32)

        # 樹葉特徵分布圖1

        blank\_img\_lr[masked\_img\_index[:, 0] + (y - by), masked\_img\_index[:, 1] + bx] = img\_float[masked\_img\_index[:, 0],

                                                                                                 masked\_img\_index[:, 1]]

        # 樹葉特徵分布圖2

        blank\_img\_ls[masked\_img\_index[:, 0] + (y - cy), masked\_img\_index[:, 1] + cx] = img\_float[masked\_img\_index[:, 0],

                                                                                                 masked\_img\_index[:, 1]]

    blank\_img\_ls = (blank\_img\_ls \* 255).astype(np.uint8)

    blank\_img\_lr = (blank\_img\_lr \* 255).astype(np.uint8)

    cv2.imwrite("hw3/blank\_img\_ls.jpg", blank\_img\_ls)

    cv2.imwrite("hw3/blank\_img\_lr.jpg", blank\_img\_lr)

    # cv2.imshow("a", blank\_img\_lr)

    # cv2.imshow("b", blank\_img\_ls)

    # cv2.waitKey(0)

    # cv2.destroyAllWindows

|  |  |  |  |
| --- | --- | --- | --- |
| Lightness |  | Laplacian Texture |  |
|  | Redness (r = R / (R + G + B)) |  | Absolute gradient of the signature curve |

Figure 9. 多維樹葉特徵分析圖一