




ชื่อ__นาย ชีรธร อุดมศิลป์__รหัสนิสิต____6520503363_____

จาก texture ต้นแบบ จำนวน 3 ภาพ ให้เขียนโปรแกรมเพื่อดำเนินการตามลำดับ ดังต่อไปนี้

| Texture1 | Texture2 | Texture3 |
|---|---|---|
|  |  |  |

0. เปิดไฟล์ภาพ texture แปลงเป็น grayscale

1. เขียนฟังก์ชัน `calMeanGray(img)` รับอินพุตเป็นภาพ grayscale และตอบกลับเป็นค่าเฉลี่ยระดับสี (Mean intensity) ที่ทำการ normalize แล้ว

```
def calMeanGray(img):  
    h, w = img.shape[:2]  
    total_intensity = 0  
  
    for i in range(h):  
        for j in range(w):  
            total_intensity += np.float64(img[i][j][0])  
  
    mean_intensity = total_intensity / (h * w)  
  
    normalized_mean = mean_intensity / 255.0  
  
    return normalized_mean
```

เรียกใช้ฟังก์ชัน `calMeanGray` และบันทึกผลลัพธ์

| | texture 1 | texture 2 | texture 3 |
|--------------------------|-----------|-----------|-----------|
| Normalize Mean intensity | 0.331 | 0.486 | 0.410 |

2. เขียนฟังก์ชัน `calFEgedness(img, threshold)` คำนวณค่า **Fedgeness** โดยมีขั้นตอนการทำงานตามลำดับ ดังนี้

2.1 คำนวณค่าขอบภาพ โดยการทำ Sobel Edge Detection กับภาพอินพุต

2.2 ปรับภาพที่ได้เป็นค่าบวกทั้งหมด โดยใช้ `absolute`

2.3 นับจำนวน pixel ที่มีค่าขอบ มากกว่าค่า `threshold` ที่กำหนด

2.4 **Fedgeness** = จำนวน pixel ที่เป็นขอบภาพ / จำนวน pixel ทั้งหมด

```
def calFEgedness(img, threshold):  
    # 2.1  
    sobelxy = cv2.Sobel(img[:, :, 0], cv2.CV_64F, 1, 1, ksize=5)  
  
    # 2.2  
    magnitude = np.abs(sobelxy)
```

```
# 2.3
h, w = magnitude.shape[:2]
edge = magnitude > threshold
edge_pixels = np.sum(edge)
# 2.4
total_pixels = h * w
fedgeness = edge_pixels / total_pixels

return fedgeness
```

เรียกใช้ฟังก์ชัน และบันทึกผลลัพธ์

| | texture 1 | Texture 2 | texture 3 |
|-----------|-----------|-----------|-----------|
| Fedgeness | 0.808 | 0.825 | 0.569 |

3. เขียนฟังก์ชัน calTextureHis(img) รับภาพอินพุต เพื่อสร้าง Texture Histogram ของภาพ โดยประกอบด้วยค่า normalize mean intensity และ Fedgeness ในรูปแบบ numpy array
[normalize mean intensity , Fedgeness] เช่น [0.27, 0.15]

โดยมีขั้นตอนการทำงานตามลำดับดังนี้

3.1 สร้าง numpy array ว่าง

3.2 เพิ่มค่า Normalize Mean intensity ของภาพลงใน array โดยใช้ calMeanGray()

3.3 เพิ่มค่า Fedgeness ลงใน array โดยใช้ calFEedgeness() กำหนด threshold = 100

```
def calTextureHis(img):
# 3.1
textureHis = np.array([])

# 3.2
mean_intensity = calMeanGray(img)
textureHis = np.append(textureHis, mean_intensity)

# 3.3
fedgeness = calFEedgeness(img, threshold=100)
textureHis = np.append(textureHis, fedgeness)

return textureHis
```

เรียกใช้ฟังก์ชัน และบันทึกผลลัพธ์

| | texture 1 | Texture 2 | texture 3 |
|-------------------|---------------|----------------|---------------|
| Texture Histogram | [0.331,0.808] | [0.486, 0.825] | [0.410,0.569] |

4. เขียนฟังก์ชัน calL1Dist(his1,his2) รับ histogram 2 ชุด เพื่อคำนวณค่า L1 distance

```
def calL1Dist(his1, his2):
    return np.sum(np.abs(his1 - his2))
```

เรียกใช้ฟังก์ชัน และบันทึกผลลัพธ์

| | texture 1 vs texture2 | Texture 1 vs texture3 | Texture2 vs texture 3 |
|----|-----------------------|-----------------------|-----------------------|
| L1 | 0.171 | 0.318 | 0.332 |

5. เขียนฟังก์ชัน textureOverlay(img, texture, L1Threshold) รับภาพอินพุต ภาพ texture และค่า L1 distance Threshold เพื่อทำการ overlay ภาพอินพุต ที่ตรงกับภาพ texture ตามค่า threshold ที่กำหนด โดยมีการทำงานดังนี้

```
def textureOverlay(img, texture, threshold):
    #
    imgOut = np.copy(img)

    #
    hisTexture = calTextureHis(texture)

    #
    windowSize = 21

    #
    r, c = img.shape[:2]

    for i in range(r-windowSize+1):
        for j in range(c-windowSize+1):
            #
            subImg = img[i:i+windowSize, j:j+windowSize]

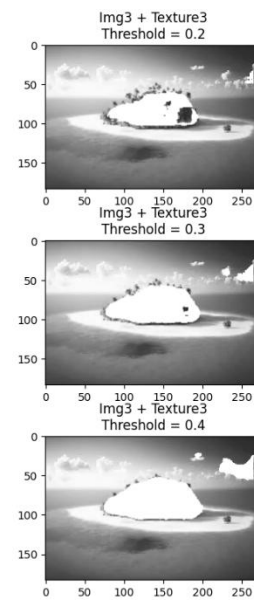
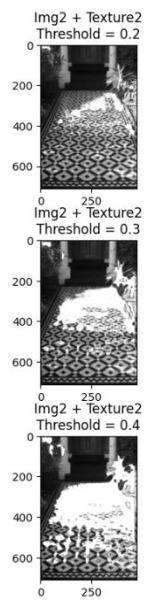
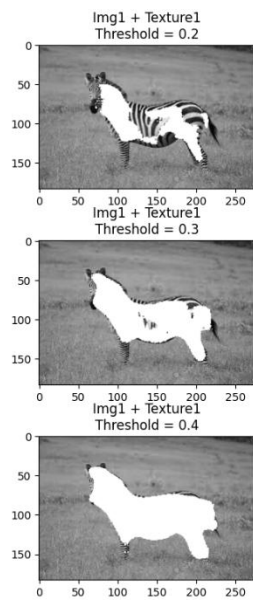
            #
            hisSubImg = calTextureHis(subImg)

            #
            if calL1Dist(hisTexture, hisSubImg) < threshold:
                #
                imgOut[i+windowSize//2][j+windowSize//2] = [255, 255, 255]

    return imgOut
```

เรียกใช้ฟังก์ชัน textureOverlay ระหว่างภาพ กับ texture โดยเปลี่ยนค่า L1Threshold และแสดงภาพผลลัพธ์ ดังนี้

| L1Threshold | Img1+texture 1 | Img2+Texture 2 | Img3+Texture 3 |
|-------------|----------------|----------------|----------------|
| 0.2 | | | |
| 0.3 | | | |
| 0.4 | | | |



NOTE : Numpy function
`np.average()` - หาค่าเฉลี่ย
`np.abs()` - ค่า absolute
`np.sum()` - ผลรวม
`np.copy()` - copy numpy array
`np.count_nonzero(condition)` นับจำนวน element ที่ตรงกับเงื่อนไข
`np.append` - เพิ่ม element ลงใน numpy array

Homework

ปรับปรุงการนำอัลกอริธึมจากโค้ดตัวอย่าง ไปใช้งานโดย

- กำหนดโจทย์ปัญหา ภาพตัวอย่าง 3-4 ภาพ
- แสดงภาพ Overlay ก่อนปรับปรุง
- เลือกดำเนินการปรับปรุงอย่างน้อยหนึ่งแนวทาง จากแนวทางดังต่อไปนี้
- เปรียบเทียบภาพ ต้นฉบับ ภาพ Overlay ก่อนปรับปรุง Overlay หลังปรับปรุง
- สรุปการดำเนินงานสั้นๆ 3-4 บรรทัด

แนวทางปรับปรุงอัลกอริธึม

| |
|--|
| 1. เพิ่ม feature Histogram ด้วยการวัดค่า texture ด้วยวิธีอื่นๆ แสดงภาพตัวอย่าง และ |
| 2. ใช้ภาพ texture ที่คล้ายกันหลายๆ ภาพ ปรับโปรแกรม โดยการทำ overlay จากหลาย Texture ร่วมกัน เพื่อให้การทำ segmentation ออกมาดีที่สุด |
| 3. ทดลองปรับค่า parameter ในหลายๆ แบบ และสรุปค่า parameter ที่เหมาะสม |
| 4 อื่นๆ ปรึกษากับอาจารย์ |

สิ่งที่ทำ: ใช้ adaptive threshold แทน ค่า static โดยใช้ ค่าmeanของรูป /255 แทนค่า threshold
โค้ดที่ลองปรับปรุง:

```
# Original overlay
original_overlay = textureOverlay(img1_gray, texture1_gray, 0.3)
plt.subplot(132)
plt.imshow(original_overlay)
plt.title('Original Overlay')

# Improved overlay (adaptive threshold)
improved_overlay = textureOverlay(img1_gray, texture1_gray, np.mean(img1_gray)/255)
plt.subplot(133)
plt.imshow(improved_overlay)
plt.title('Improved Overlay')
```

