

Хэв танилтын үндэс Лаборатори 2-1 тайлан

Эрдэнэ Тэмүүжин

20b1num1970@stud.num.edu.mn



ХШУИС Хэрэглээний Математикийн тэнхим
Монгол Улсын Их Сургууль

2023 оны 10-р сарын 12

Хэв танилтын үндэс Лаборатори 2-1 тайлан

Эрдэнэ Тэмүүжин - 20B1NUM1970

2023 оны 10-р сарын 26

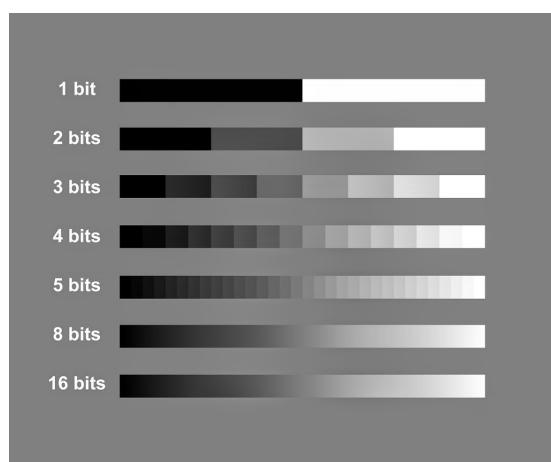
МУИС-ХШУИС Хэрэглээний Математикийн тэнхим

Удиртгал

Энэ лабораторийн ажилд дижитал зургийн пиксел утгуудын эрчмийг өөрчлөх (Intensity Resolution), Интерполяцийн арга ашиглан зургийн мэдэгдэхгүй утгуудыг тооцох (Image Interpolation), зургийн пиксел утгуудын далайцыг сунгах (Contrast Stretching), зургийн пиксел утгуудын сонирхсон завсрлын утгуудыг ихэсгэх (Gray level slicing) болон зургийн шахахад ашиглагдах арга (Bit-plane Slicing) туршсан.

1. Intensity Resolution

Зургийн нарийвчлал (intensity) нь зургийн пикселийг хэдэн битээр тодорхойлж байгаагаас хамаардаг. Жишээ нь хар цагаан зургийн хувьд пиксел бүрийг 1-бит буюу 0 эсвэл 1 гэсэн тоогоор илэрхийлж чадна. Харин 2-бит зургийн хувьд пиксел бүрийг нийт 4 янзаар илэрхийлж болно. 00 - хар, 11 - цагаан, 01 - бараан саарал, 10 - цайвар саарал гэсэн сонголтууд байна.



Зураг 1: Зургын бит гүн [1]

Энэ хэсгийн даалгавар нь анхны зураг болох 8-бит, саарал зургийн пикселийг 1-бит хүртэл багасгах юм.

```
a = 0
b = 255

def func(img, k):
    row, col = img.shape
    out_img = np.zeros((row, col))

    c = 0
    d = 2**k

    mult = (d - c)/(b - a)

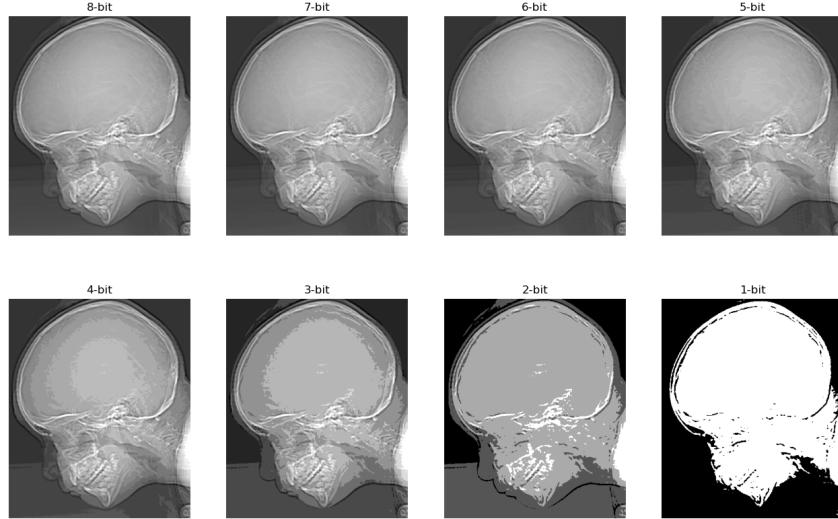
    for i in range(row):
        for j in range(col):
            out_img[i][j] = img[i][j]*mult

    return np.uint8(out_img), c, d - 1
```

Анхны буюу 8-бит зургийн пиксел бүр $p_{ij} \in [0, 255]$ завсраас утгаа авдаг. Зургийг 7-битээр илэрхийлэхийн тулд $\hat{p}_{ij} \in [0, 127]$ харин 6-бит $p_{ij} \in [\hat{p}_{ij} \in 0, 127]$ гэх мэтчилэн хувиргалтуудыг хийх шаардлагатай. Иймд $\hat{p}_{ij} = f(p_{ij}) = c + (\frac{d-c}{b-a})(p_{ij} - a)$ Функц зарлаж зургийг илэрхийлэх битийн хэмжээг өөрчилсөн. Энд бит бүрийн хувьд пикселийн доод утга $c = a = 0$ тул функцийг дараахаар тодорхойлсон:

$$\hat{p}_{ij} = f(p_{ij}) = p_{ij} \frac{d - c}{b - a} \quad (1)$$

Энд $d = 2^k$ буюу пикселийн бит, $b = 255$ 8-бит зургийн пикселийн авах боломжтой утгын тоо.



Зураг 2: Image Resolution үр дүн

2. Image Interpolation

Энэхүү лабораторид *Nearest-Neighbor* болон *Bilinear* интерполяцийн аргуудыг туршиж үзсэн. *Nearest-Neighbor* нь зургийг томруулах үед мэдэгдэхгүй пикселийн утгыг тооцоходо утга нь мэдэгддэг хамгийн ойр пикселийн утгыг хувилдаг арга. Жишээ авьяа, дараах зургийг 2 дахин томруульяа:

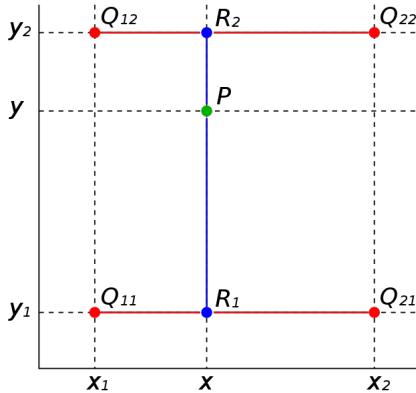
$$I = \begin{bmatrix} 10 & 15 \\ 20 & 40 \end{bmatrix} \rightarrow I_{2x} = \begin{bmatrix} 10 & ? & ? & 15 \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 20 & ? & ? & 40 \end{bmatrix} \quad (2)$$

I_{2x} зургийн $?$ бүрд ойр, мэдэгдэж байгаа пикселийн утгуудыг хувилъяа.

$$I_{2x} = \begin{bmatrix} 10 & 10 & 15 & 15 \\ 10 & 10 & 15 & 15 \\ 20 & 20 & 40 & 40 \\ 20 & 20 & 40 & 40 \end{bmatrix} \quad (3)$$

Харин *Bilinear* интерполяци нь бага зэрэг ажиллагаатай. Энэ аргыг ашиглаж зураг томруулах үед мэдэгдэхгүй пикселийн утгыг хамгийн ойр, мэдэгдэж байгаа 4 пикселийн жинлэсэн дунджаар тооцдог [2]. Анхны зургийг f функц илэрхийлдэг гэе. $Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1), Q_{22} = (x_2, y_2)$ цэгүүд дээрх функцийн утгыг мэддэг гэе. Манай жишээн дээр $Q_{11} = 10, Q_{12} = 15, Q_{21} = 20, Q_{22} = 40$ байгаа. Хэвтээ тэнхлэгийн дагуу шугаман интерполяци хийнэ. [2]

$$\begin{aligned} f(x, y_1) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \\ f(x, y_2) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \end{aligned} \quad (4)$$



Зураг 3: Bilinear Interpolation [2]

Эндээс босоо тэнхлэгийн дагуу шугаман интерполяци ашигласны дараагаар дараах ойролцооллын функцийг гаргана. [2]

$$f(x, y) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y} f(x, y_2) \quad (5)$$

Өмнөх жишээг ахин авч үзье.

$$I = \begin{bmatrix} 10 & 15 \\ 20 & 40 \end{bmatrix} \rightarrow I_{2x} = \begin{bmatrix} 10 & ? & ? & 15 \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 20 & ? & ? & 40 \end{bmatrix} \quad (6)$$

Анхны зургийн хувьд бидэнд пиксел бүрийн координатууд болон утгууд нь мэдэгдэж байгаа.

$$\begin{pmatrix} 0 & 1 \\ f(0,0) & f(1,0) \\ f(1,0) & f(1,1) \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 & 15 \\ 20 & 30 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (7)$$

Харин энэхүү зургийг 2 дахин томруулахад пиксел хоорондын утга болон координатуудыг тооцоолох шаардлагатай.

$$I_{2x} = \begin{pmatrix} 0 & ? & ? & 1 \\ 10 & ? & ? & 15 \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 20 & ? & ? & 40 \end{pmatrix} \begin{pmatrix} 0 \\ ? \\ ? \\ ? \end{pmatrix} \quad (8)$$

Эхлээд тооцоолох пиксел бүрийн координатыг ольё. Анхны зураг $(m, n) = (2, 2)$ хэмжээтэй матриц, томорсон зураг $(M, N) = (4, 4)$ хэмжээтэй матриц. Иймд бидэнд 4 цэгийн координат хэрэгтэй. Анхны зураг 2 мөр, баганатай. Тэгэхээр $x = 0$ эхлээд $x = 1$ дуусаж байгаа. Иймд x цэгийн хувьд хоорондоо ижил зйттай 0, 0.33, 0.67, 1 цэгүүдтэй болно. Мөн адил y цэгийн хувьд 0, 0.33, 0.67, 1 цэгүүдтэй болно.

$$\begin{aligned} x \text{ тэнхлэгийн дагуу: } \Delta x &= \frac{m-1}{M-1} = \frac{2-1}{4-1} = \frac{1}{3} \\ x_i &= i \cdot \Delta x, \text{ энд } i = 0, 1, 2, 3 \\ y \text{ тэнхлэгийн дагуу: } \Delta y &= \frac{n-1}{N-1} = \frac{2-1}{4-1} = \frac{1}{3} \\ y_j &= j \cdot \Delta y, \text{ энд } j = 0, 1, 2, 3 \end{aligned} \quad (9)$$

Тэгэхээр эндээс бид дараах цэгүүд дээрх пикселийн утгуудыг олно.

$$I_{2x} = \begin{bmatrix} f(0,0) & f(0.33,0) & f(0.67,0) & f(1,0) \\ f(0,0.33) & f(0.33,0.33) & f(0.67,0.33) & f(1,0.33) \\ f(0,0.67) & f(0.33,0.67) & f(0.67,0.67) & f(1,0.67) \\ f(0,1) & f(0.33,1) & f(0.67,1) & f(1,1) \end{bmatrix} \quad (10)$$

1. **f(0.33,0)**: Энэ цэгтэй хамгийн ойр мөн утга нь мэдэгдэж байгаа цэгүүд: $f(0,0) = 10, f(1,0) = 15, f(0,1) = 20, f(1,1) = 40$. Эхлээд $(0.33, 0)$ цэг дээрх жингүүдийг тооцно

$$w_x = \frac{x_2 - x}{x_2 - x_1} = \frac{1 - 0.33}{1 - 0} = 0.67,$$

$$w_y = \frac{y_2 - y}{y_2 - y_1} = \frac{1 - 0}{1 - 0} = 1$$

Одоо $f(x, y_1) = f(0.33, 0)f(x, y_2) = f(0.33, 1)$ дээрх утгуудыг тооцьё

$$f(x, y_1) = f(0.33, 0) = w_x \cdot f(0, 0) + (1 - w_x) \cdot f(1, 0) = 0.67 \cdot 10 + 0.33 \cdot 15 = 11.65$$

$$f(x, y_2) = f(0.33, 1) = w_x \cdot f(1, 0) + (1 - w_x) \cdot f(1, 1) = 0.67 \cdot 20 + 0.33 \cdot 40 = 26.6$$

Эндээс одоо $f(x, y) = f(0, 0.33)$ утгыг тооцох боломжтой боллоо.

$$f(x, y) = f(0.33, 0) = w_y \cdot f(0.33, 0) + (1 - w_y) \cdot f(0.33, 1) = 1 \cdot 11.65 + 0 \cdot 26.6 = 11.65$$

2. **f(0.67, 0)**:

$$w_x = \frac{x_2 - x}{x_2 - x_1} = \frac{1 - 0.67}{1 - 0} = 0.33,$$

$$w_y = \frac{y_2 - y}{y_2 - y_1} = \frac{1 - 0}{1 - 0} = 1$$

$y = 0$ үед $f(x, y_2)$ утга бодох шаардлагагүй тул зөвхөн $f(x, y_1)$ утгыг тооцьё.

$$f(x, y_1) = f(0.67, 0) = w_x \cdot f(0, 0) + (1 - w_x) \cdot f(1, 0) = 0.33 \cdot 10 + 0.67 \cdot 15 = 13.35$$

$$f(x, y) = f(0.67, 0) = w_y \cdot f(0.67, 0) + (1 - w_y) \cdot f(0.67, 1) = 13.35$$

3. **f(0, 0.33)**: Одоо y тэнхлэгийн дагуу хөдөльье.

$$w_x = \frac{x_2 - x}{x_2 - x_1} = \frac{1 - 0}{1 - 0} = 1,$$

$$w_y = \frac{y_2 - y}{y_2 - y_1} = \frac{1 - 0.33}{1 - 0} = 0.67$$

Бидэнд $f(x, y_1) = f(0, 0)$ болон $f(x, y_2) = f(0, 1)$ утгууд мэдэгдэж байгаа тул шууд $f(0, 0.33)$ утгыг тооцьё.

$$f(x, y) = f(0, 0.33) = w_y \cdot f(0, 0) + (1 - w_y) \cdot f(0, 1) = 0.67 \cdot 10 + 0.33 \cdot 20 = 13.3$$

Матрицын төвийн утгуудыг тооцохын тулд бид эхлээд захын бүх утгуудыг тооцох хэрэгтэй. Дээрх алхмуудыг захын мэдэгдэхгүй цэгүүд дээр гүйцэтгэснээр дараах утгуутай болно.

| Цэг | f утга |
|-------------|----------|
| $(0.33, 0)$ | 11.65 |
| $(0.67, 0)$ | 13.35 |
| $(0, 0.33)$ | 13.3 |
| $(0, 0.67)$ | 16.7 |
| $(0.33, 1)$ | 26.6 |
| $(0.67, 1)$ | 33.4 |
| $(1, 0.33)$ | 23.25 |
| $(1, 0.67)$ | 31.75 |

Хүснэгт 1: 4x томруулсан зургийн захын цэгүүд дээрх утгууд

Одоо төвийн утгуудыг тооцьё.

4. **f(0.33, 0.33)**:

$$\begin{bmatrix} f(0, 0) & \textcolor{red}{f(0.33, 0)} & f(0.67, 0) & f(1, 0) \\ f(0, 0.33) & \textcolor{blue}{f(0.33, 0.33)} & f(0.67, 0.33) & f(1, 0.33) \\ f(0, 0.67) & f(0.33, 0.67) & f(0.67, 0.67) & f(1, 0.67) \\ f(0, 1) & \textcolor{red}{f(0.33, 1)} & f(0.67, 1) & f(1, 1) \end{bmatrix} \quad (11)$$

Тэгэхээр $(0.33, 0.33)$ цэгийн утгыг олоход дээрх матрицаад улаанаар тэмдэглэсэн цэгүүдийн утгыг ашиглана.

$$w_y = \frac{y_2 - y}{y_2 - y_1} = \frac{1 - 0.33}{1 - 0} = 0.67$$

$$f(0.33, 0.33) = w_y \cdot f(0.33, 0) + (1 - w_y) \cdot f(0.33, 1) = 0.67 \cdot 11.65 + 0.33 \cdot 26.6 = 16.5835$$

гэх мэтчилэн байдлаар төвийн цэгүүдийн утгыг олж матрицыг дуургэнэ.

$$I_{2x} = \begin{bmatrix} f(0, 0) & f(0.33, 0) & f(0.67, 0) & f(1, 0) \\ f(0, 0.33) & f(0.33, 0.33) & f(0.67, 0.33) & f(1, 0.33) \\ f(0, 0.67) & f(0.33, 0.67) & f(0.67, 0.67) & f(1, 0.67) \\ f(0, 1) & f(0.33, 1) & f(0.67, 1) & f(1, 1) \end{bmatrix} = \begin{bmatrix} 10 & 11.65 & 13.35 & 15 \\ 13.3 & 16.58 & 26.78 & 23.25 \\ 16.7 & 19.88 & 31.21 & 31.75 \\ 20 & 26.6 & 33.4 & 40 \end{bmatrix} \quad (12)$$

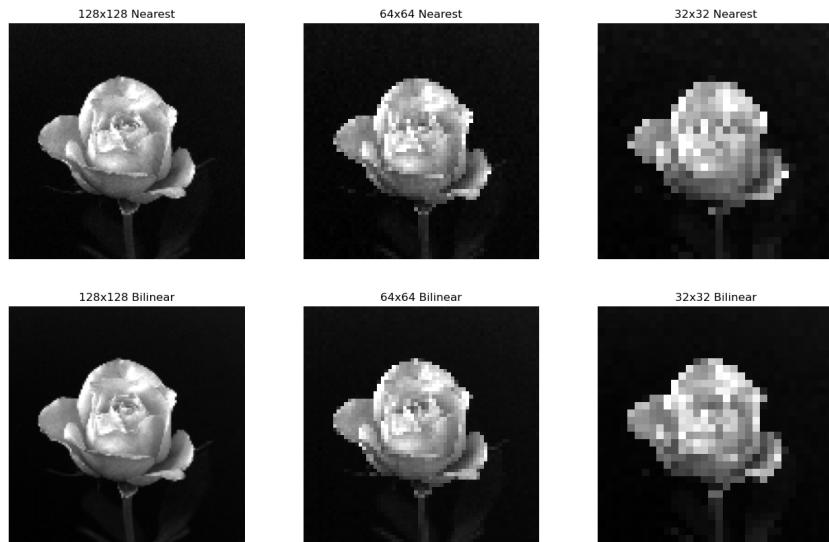
OpenCV сангийн Bilinear interpolation хийх функц ашиглан томруулсан матрицтай гараар бодсон үр дүнгээ харьцуульяа.

$$I_{opencv} = \begin{bmatrix} 10.00 & 11.25 & 13.75 & 15.00 \\ 12.50 & 14.69 & 19.06 & 21.25 \\ 17.50 & 21.56 & 29.69 & 33.75 \\ 20.00 & 25.00 & 35.00 & 40.00 \end{bmatrix} \quad (13)$$

$$I_{2x} - I_{opencv} = \begin{bmatrix} 0.00 & 0.40 & -0.40 & 0.00 \\ 0.80 & 1.8925 & 7.7175 & 2.00 \\ -0.80 & -1.6825 & 1.5225 & -2.00 \\ 0.00 & 1.60 & -1.60 & 0.00 \end{bmatrix} \quad (14)$$

```
# Nearest Neighbor Interpolation
img_128_nearest = cv2.resize(img, (128, 128),
                             interpolation = cv2.INTER_NEAREST)
img_64_nearest = cv2.resize(img, (64, 64),
                            interpolation = cv2.INTER_NEAREST)
img_32_nearest = cv2.resize(img, (32, 32),
                            interpolation = cv2.INTER_NEAREST)

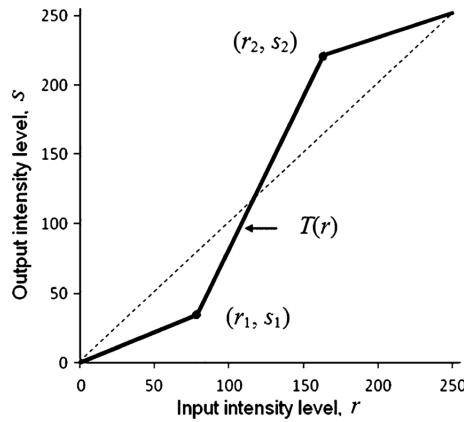
# Bilinear Interpolation
img_128_linear = cv2.resize(img, (128, 128),
                             interpolation = cv2.INTER_LINEAR)
img_64_linear = cv2.resize(img, (64, 64),
                           interpolation = cv2.INTER_LINEAR)
img_32_linear = cv2.resize(img, (32, 32),
                           interpolation = cv2.INTER_LINEAR)
```



Зураг 4: Image Interpolation үр дүн

3. Contrast Stretching

Зургийн contrast нь бараан болон цайвар пикселийн эрчмийн ялгаа. Тэгэхээр contrast stretching нь зургийн пикселийн авч болох утгуудын мужийг ихэсгэх/сунгах арга юм.



Зураг 5: Contrast Stretching функцийн даалгаварт өгөгдсөн хэлбэр [3]

Дээрх графд үзүүлсэн функцийг гаргаж авахын тулд зураг хамгийн бага болон хамгийн их эрчмүүд дээрээ ямар утга авч байгааг харах шаардлагатай болсон. Эндээс дараах функцийг гаргаж авсан.

$$f(x) = \begin{cases} \frac{s_1}{r_1}r, & r < r_1 \\ \frac{s_2-s_1}{r_2-r_1}(r - r_1) + s_1, & r_1 \leq r < r_2 \\ \frac{255-s_2}{255-r_2}(r - r_2) + s_2, & r_2 \leq r < 256 \end{cases} \quad (15)$$

Энэ функцийг 2 янзаар бичсэн. Дараах граф гаргахад x утга бүр дээр итерацаар тооцоо хийдэг. Харин зураг буюу матриц дээр тооцоо хийхэд *numpy* сангийн вектор үйлдлүүдийг ашиглаж бичсэн.

```
def contrast_stretching_function(pixel_value, img_min, img_max, new_min, new_max):
    (r_1, s_1) = (img_min, new_min)
    (r_2, s_2) = (img_max, new_max)

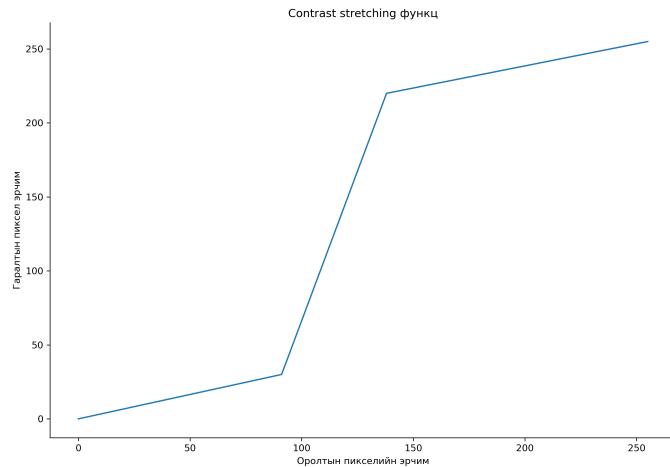
    assert r_2 - r_1 < new_max - new_min

    if pixel_value < r_1:
        return s_1/r_1*pixel_value
    elif (pixel_value >= r_1) and (pixel_value < r_2):
        return (s_2-s_1)/(r_2-r_1)*(pixel_value-r_1) + s_1
    elif r_2 <= pixel_value:
        return (255-s_2)/(255-r_2)*(pixel_value-r_2) + s_2

img_min, img_max = img.min(), img.max()
new_min, new_max = 30, 220

x_values = np.arange(256)
y_values = [contrast_stretching_function(x, img_min, img_max, new_min, new_max) for x in x_values]
```

Үп дүн:



Зураг 6: Contrast Stretching функц

Зураг дээр хувиргалт хийхэд бичэн функц:

```
def contrast_stretching_function_vectorized(img, new_min, new_max):
    r_1, s_1 = img.min(), new_min
    r_2, s_2 = img.max(), new_max

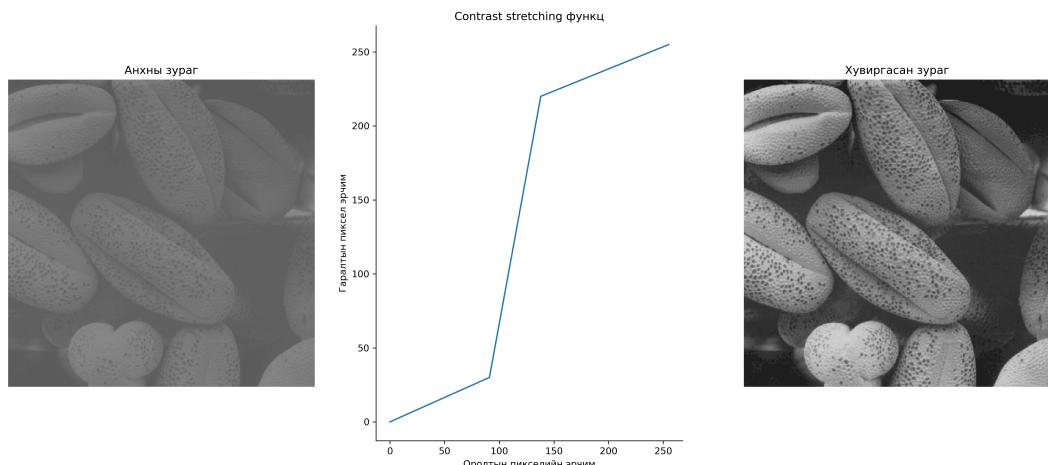
    assert r_2 - r_1 < new_max - new_min

    result = np.piecewise(img, [img < r_1, (img >= r_1) & (img < r_2), img >= r_2],
                          [lambda x: s_1 / r_1 * x,
                           lambda x: (s_2 - s_1) / (r_2 - r_1) * (x - r_1) + s_1,
                           lambda x: (255 - s_2) / (255 - r_2) * (x - r_2) + s_2])

    return result

img = Image.open("../DIP3E_Original_Images_CH03/Fig0310(b)(washed_out_pollen_image).tif")
img = np.asarray(img)
img_contrast_stretched = contrast_stretching_function_vectorized(img, 30, 220)
```

Үр дүн:



Зураг 7: Contrast Stretching функцийн анхны зураг дээрх хувиргалт

4. Gray-level Slicing

Саарал зургийн тодорхой мужийн утгуудыг тодруулдаг арга [4]. Энэ аргыг хамгийн бүдүүлгээр буюу зургийн пиксел бүрийн хувьд итераци гүйлгэж уг пикселийн утгыг тодруулах шаардлагатай эсэхийг шалгахаар бичсэн.

$$T_1(x) = \begin{cases} \text{range_val}, & \min_range \leq x \leq \max_range \\ \text{const_val}, & \text{бусад} \end{cases} \quad (16)$$

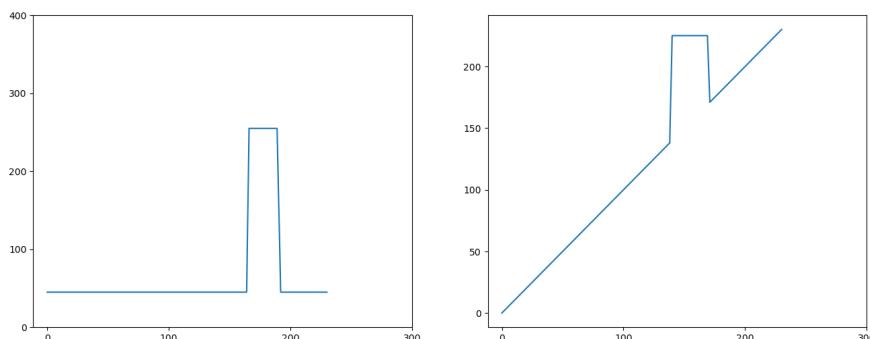
```

h, w = img.shape
img1 = np.zeros((h, w), dtype = 'uint8')

min_range = 165
max_range = 190
range_val = 255
const_val = 45
X = []
y = []

for i in range(h):
    for j in range(w):
        if(img[i, j] >= min_range and img[i, j] <= max_range):
            img1[i, j] = range_val
            X.append(img[i, j])
            y.append(range_val)

        else:
            img1[i, j] = const_val
            X.append(img[i, j])
            y.append(const_val)
    
```



Зураг 8: Gray level slicing функц

$$T_2(x) = \begin{cases} \text{range_val}, & \min_range \leq x \leq \max_range \\ x, & x < \min_range \text{ OR } x > \max_range \end{cases} \quad (17)$$

```

img2 = np.zeros((h, w), dtype = 'uint8')

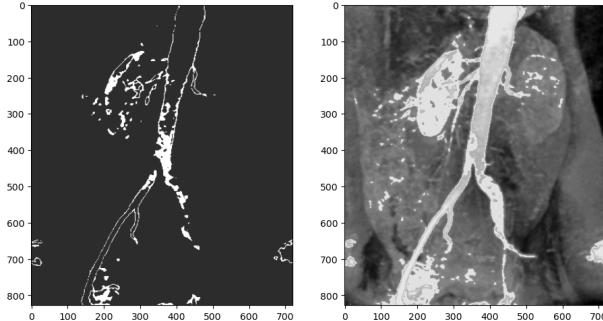
min_range = 140
max_range = 170
range_val = 225

X2 = []
y2 = []

for i in range(h):
    for j in range(w):
        if(img[i, j] >= min_range and img[i, j] <= max_range):
            img2[i, j] = range_val
            X2.append(img[i, j])
            y2.append(range_val)
        elif(img[i, j] < min_range or img[i, j] > max_range):
            img2[i, j] = img[i, j]
            X2.append(img[i, j])
            y2.append(img[i, j])
    
```

```
x2.append(img[i, j])
y2.append(img[i, j])
```

Үр дүн:



Зураг 9: Gray level slicing үр дүн

5. Bit-plane Slicing

Зураг нь пиксел гэх тоон утгын багц юм. Харин пиксел нь саарал зургийн хувьд 0 - 255 хооронд утга авдаг бодит too байна. Иймд 0-255 хооронд утгаа авдаг зураг 8-битээс тогтоно. Бит бүрийг хавтгайд задалж болох ба хавтгай бүр нь хамгийн чухал буюу хамгийн их мэдээлэл агуулахаас хамгийн бага мэдээлэл агуулдгаар нь эрэмбэлж чадна. Ингэснээр зургаа сайн илэрхийлж чадах цөөхөн битээр нь үүсгэснээр компьютер дээр эзлэх багтаамжийг бууруулах боломжтой болдог. Одоо жишээ авъя:

$$I = \begin{bmatrix} 10 & 15 \\ 20 & 40 \end{bmatrix} = \begin{bmatrix} 001010 & 001111 \\ 010100 & 101000 \end{bmatrix} \quad (18)$$

Энд 40 тооны хоёртын утга нь Тэгэхээр 40 тооны хувьд хамгийн чухал мэдээлэл агуулах (MSB) нь хамгийн

| MSB | 4 | 3 | 2 | 1 | LSB |
|-----|---|---|---|---|-----|
| 1 | 0 | 1 | 0 | 0 | 0 |

баруун гар талд байгаа ба энэ нь $2^5 = 32$. Харин хамгийн бага мэдээлэл агуулах (LSB) нь $2^0 = 1$ байна. Одоо бит бүрээр нь задалж харья: 5 бит (MSB):

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

4 бит:

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

3 бит:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

2 бит:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

1 бит:

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

0 бит (LSB):

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Бид яг дээрх байдлаар гараас өгсөн саарал зургийг 8 битийн түвшинд задална. Доорх функцэд `img » i` үйлдэл нь зургийн пикселийг *i*-битээр зүүн гар тал руу шилжүүлнэ (shift). Жишээ нь `001111 » 2` гэвэл `000011` гарна. Shift хийсний дараагаар AND оператор ашиглан хэрэггүй бит салгана.

```

def bit_plane(img, k):
    bit_planes = []
    for i in range(k):
        bit_plane = (img >> i) & 1
        bit_plane = bit_plane * 255
        bit_planes.append(bit_plane)

    return bit_planes

```

Үр дүн

```

img = Image.open("../DIP3E_Original_Images_CH03/Fig0314(a)(100-dollars).tif")
img = np.asarray(img)
bit_planes = bit_plane(img, 8)

fig, axes = plt.subplots(3, 3, figsize = (15, 10))
h, w = 0, 0

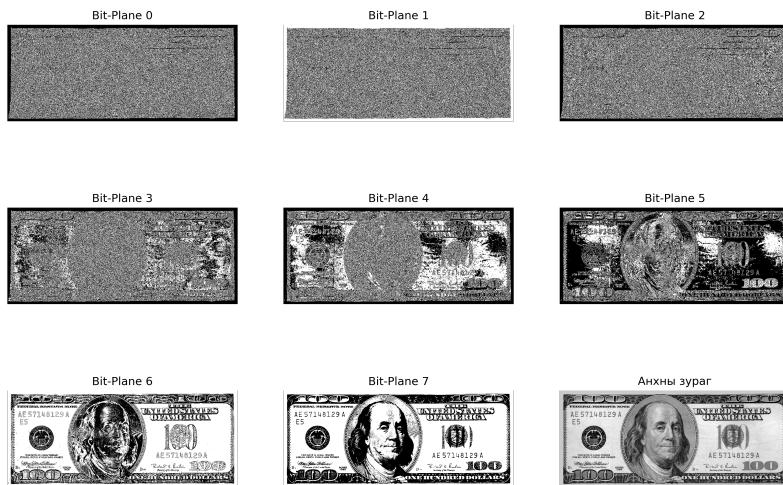
for j in range(8):
    axes[h][w].imshow(bit_planes[j], cmap = 'gray')
    axes[h][w].set_title(f'Bit-Plane {j}')
    axes[h][w].axis('off')

    w += 1
    if w == 3:
        w = 0
        h += 1

axes[2][2].imshow(img, cmap = 'gray')
axes[2][2].set_title(f'Анхны зураг') # latex code snipped, not supporting utf-8
axes[2][2].axis('off');

fig.savefig("bit_plane_slicing_1.png", dpi = 300)

```



Зураг 10: Bit plane slicing үр дүн

6. Дүгнэлт

Энэхүү лабораторид зураг сайжруулах аргуудын талаар үзлээ. Энэ нь 2-р хэсгийн эхний лаборатори байсан тул зургийг шугаман хувиргалтууд л ашиглан сайжруулах талаар үзлээ. Жишээ нь contrast stretching аргын шугаман хэлбэр нь энэхүү лаборатори дээр хийсэн бол Histogram Equalization гэх шугаман бус аргыг дараагийн лабораторийн ажил дээр хийх болно.

7. Ном зүй

- [1] T. Lumenera, *AN IN-DEPTH LOOK AT BIT DEPTH*, <https://www.lumenera.com/blog/bit-depth/>, [Online; accessed 13-October-2023], 2016.
- [2] Swienegel, *Bilinear interpolation*, <https://commons.wikimedia.org/w/index.php?curid=124957943>.
- [3] M. D. Saleh, C. Eswaran **and** A. Mueen, “An automated blood vessel segmentation algorithm using histogram equalization and automatic threshold selection,” *Journal of digital imaging*, **jourvol** 24, **pages** 564–572, 2011.
- [4] E. Baidoo **and** A. kwesi Kontoh, “Implementation of gray level image transformation techniques,” *International Journal of Modern Education and Computer Science*, **jourvol** 10, **number** 5, **page** 44, 2018.