

Хэв танилтын үндэс Лаборатори 2-2 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 2-2 тайлан

Эрдэнэ Тэмүүжин - 20B1NUM1970

2023 оны 11-р сарын 3

МУИС-ХШУИС Хэрэглээний Математикийн тэнхим

Удиртгал

Өмнөх лабораторийн ажилд дижитал зургийн пиксел нэг бүрийн утгад хандаж зурагт босго тогтоох эсвэл зургийн ялгарлыг нэмэгдүүлж байсан бол энэ лабораторийн ажилд зургийн тодорхой орчин дахь пикселийн утгуудад хандана.

1. Histogram Processing

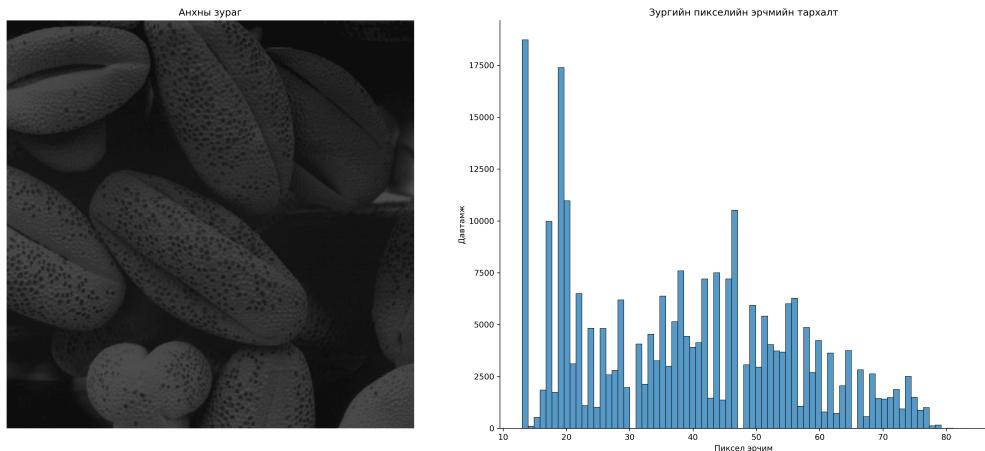
Дижитал зургийн гистограм нь уг зургийн пикселийн эрчмийн үзэгдэх магадлалыг буцаадаг функц юм [1].

$$p_r(r_k) = \frac{n_k}{MN}, \quad k = 0, 1, \dots, L - 1 \quad (1)$$

Энд n_k нь r_k эрчимтэй пикселийн тоо, харин MN нь зургийн нийт пикселийн тоо. L нь зургийн авч болох боломжит эрчмийн тоо. Эндээс зургийн эрчмийн хуримтлагдсан тархалтын функцийг тооцно [1].

$$s_k = \frac{L - 1}{MN} \sum_{j=0}^k n_j, \quad k = 0, 1, \dots, L - 1 \quad (2)$$

Тэгэхээр оролтын зургийн пикселийн эрчмийг хуримтлагдсан тархалтын утга бүрд нь харгалзуулснаар гаралтын зураг үүсэх юм.



Зураг 1: Оролтын зураг ба эрчмийн тархалт

```
def imhist(img):
    row, col = img.shape
    h = [0] * 256

    for i in range(row):
        for j in range(col):
            h[img[i, j]] += 1

    return np.array(h)/(row*col)

def cumsum(h):
```

```

        return [sum(h[:i+1]) for i in range(len(h))]

def hist_eq(img):
    h = imhist(img)
    cdf = np.array(cumsum(h))
    sk = np.uint8(255 * cdf)
    s1, s2 = img.shape
    Y = np.zeros_like(img)

    for i in range(s1):
        for j in range(s2):
            Y[i, j] = sk[img[i, j]]

    H = imhist(Y)
    return Y, h, H, sk

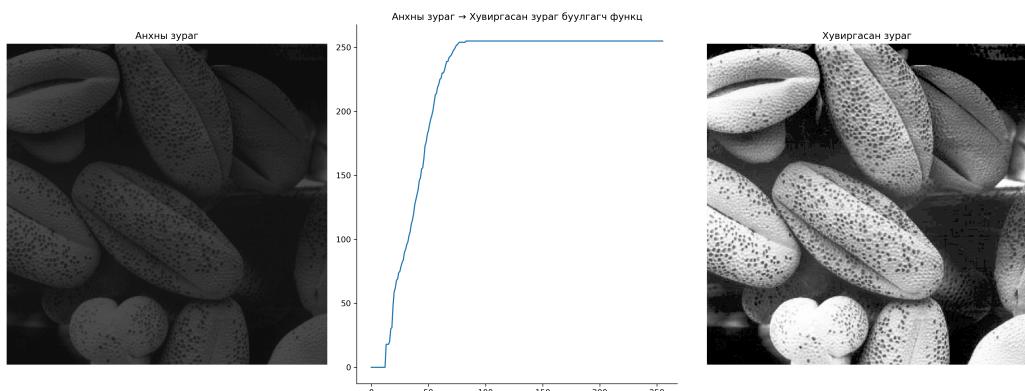
```

imhist функц болон *cumsum* функцуудэд пикселийн эрчмийн гистограм болон хуримтлагдсан тархалтыг гараар тооцож байгаа. Үүний дараагаар *hist_eq* функц ашиглан гаралтын зургийг тооцож байна.

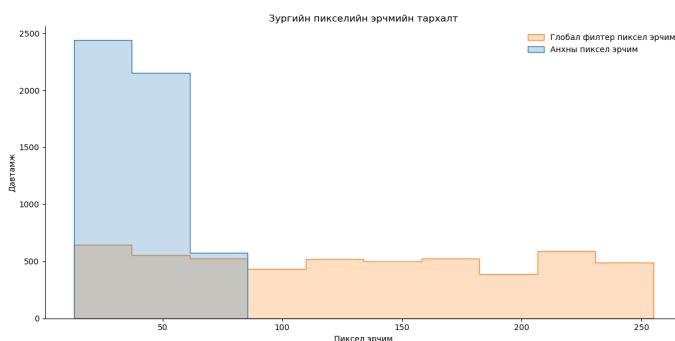
```

for i in range(s1):
    for j in range(s2):
        Y[i, j] = sk[img[i, j]]

```

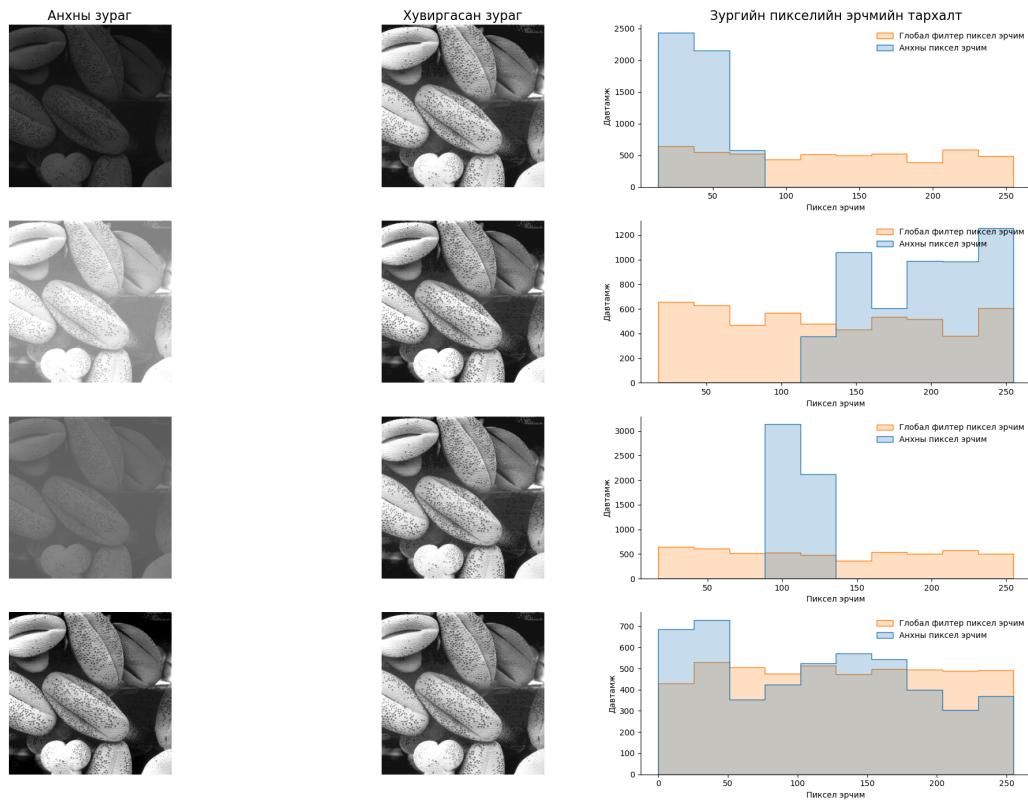


Зураг 2: Гаралтын зураг ба буулгагч функц

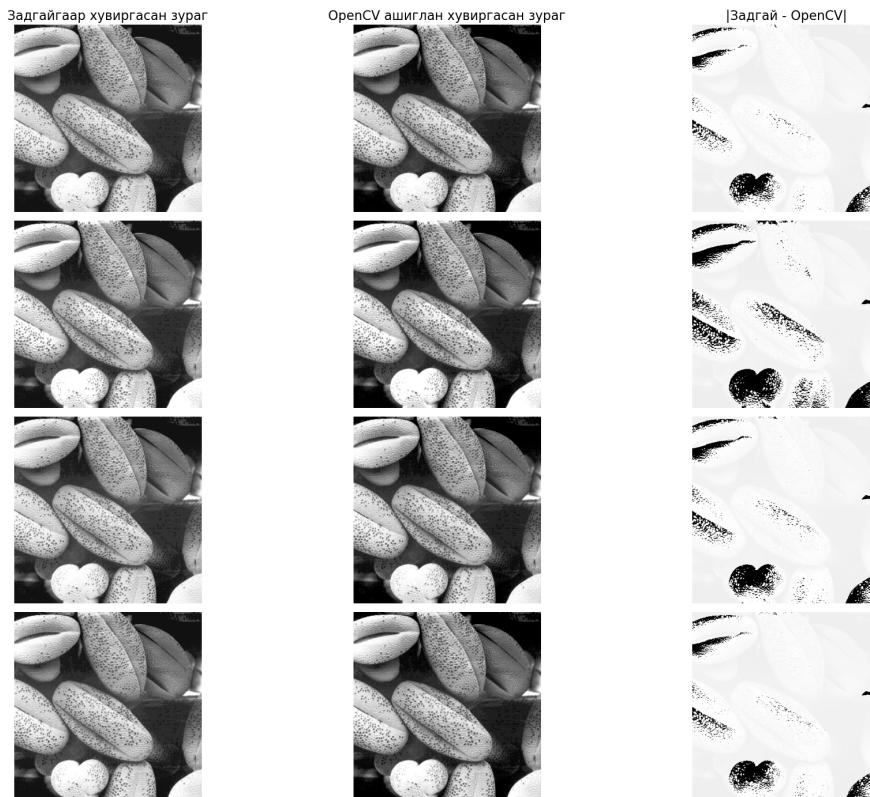


Зураг 3: Оролтын болон гаралтын зургийн эрчмуудийн тархалт

Одоо 4 ялгаатай эрчмийн тархалттай зургууд дээр уг аргыг ашигласан үр дүнг харуулъя. Энд зураг бүрийн эрчмийн тархалт нь голдуу зүүн эсвэл баруун хэсэгтээ буюу зургийн боломжит эрчим бүрийн хувьд жигд тархаагүй хэсэг газартаа бөөгнөрсөн хэв маягтай байсан.

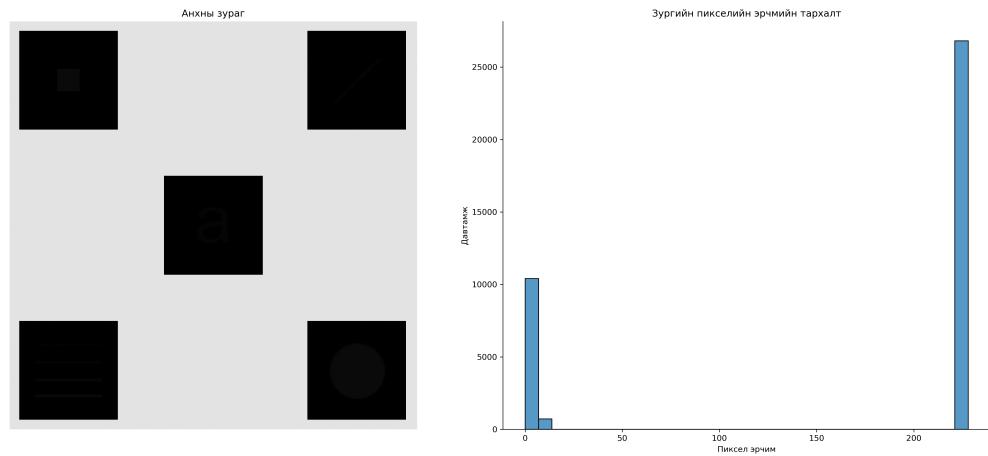


Зураг 4: Ялгаатай эрчмийн тархалттай зургууд дээрх үр дүн



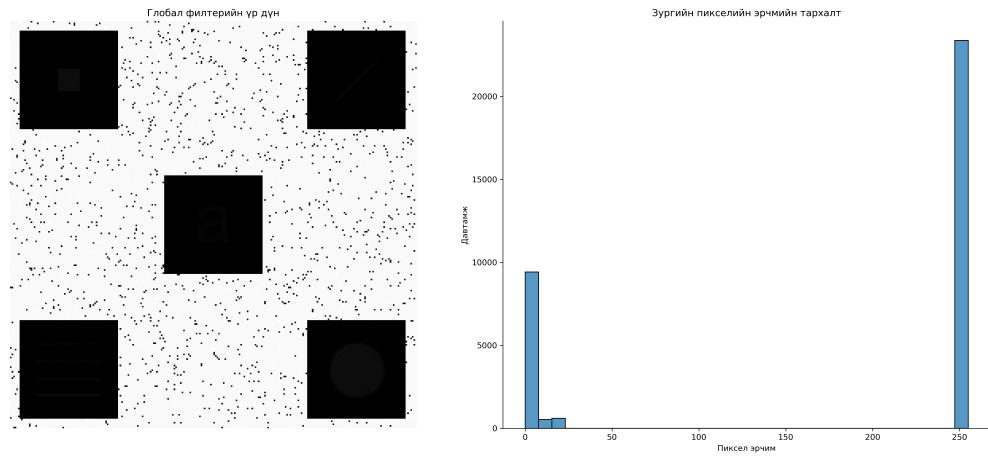
Зураг 5: Гараар хувиргасан функц болон бэлэн функцийн ялгаа

Тэгэхээр сая бид зургийн эрчмийн тархалтыг нийтээр нь нормчлох аргын талаар үзсэн бол одоо хэсэг орчны хувьд нормчлох талаар үзье.



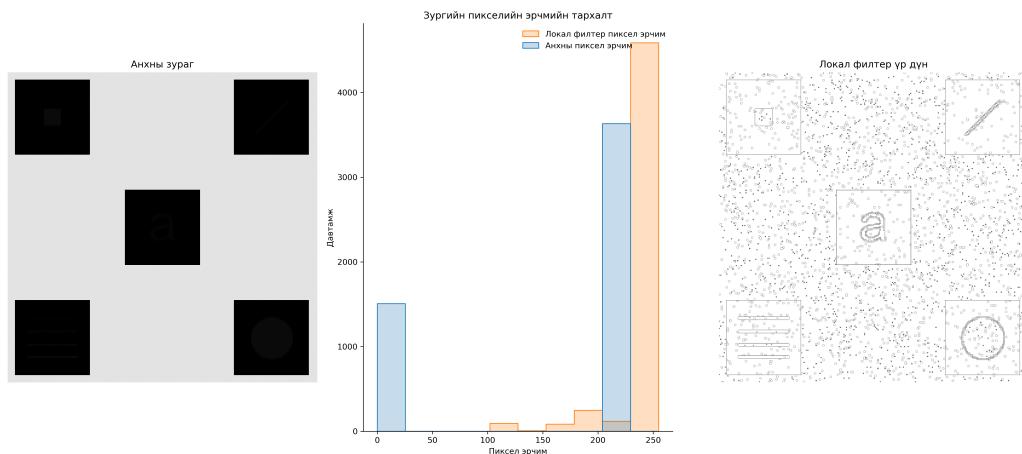
Зураг 6: Оролтын зураг болон эрчмийн тархалт

Энэхүү зургийн хувьд глобал гистограм арга ашиглах нь дараах үг дүнг буцаана.



Зураг 7: Оролтын зургийн глобал гистограм аргын үр дүн

Энэхүү арга нь зургийн шуугианыг нэмэгдүүлсэн хэдий ч өөр хэрэгтэй мэдээлэл гаргаж чадахгүй байна. Учир нь бидэнд энэхүү зургаас хэрэгтэй байгаа пикселийн эрчим нь орчныхоо эсвэл нийт зургийн эрчимтэй харьцуулахад нөлөө багатай гэж үзэж нийт зургийн хувьд гистограм арга ашиглах биш харин хэсэг орчны хувьд ашиглай.



Зураг 8: Оролтын зургийн орчны хувьд гистограм арга ашигласан үр дүн

```

def local_hist_eq(img, mask_size, stride = 1):
    height, width = img.shape
    mask_iter = mask_size // 2
    enhanced_image = np.zeros((height, width), dtype = np.uint8)

    padding = mask_iter * stride
    padded_img = np.pad(img, pad_width = ((padding, padding), (padding, padding)),
                        mode = 'constant', constant_values = 0)

    for i in range(mask_iter, height + mask_iter, stride):
        for j in range(mask_iter, width + mask_iter, stride):
            neighborhood = padded_img[i - mask_iter: i + mask_iter + 1, j - mask_iter: j + mask_iter + 1]
            hist = np.histogram(neighborhood, bins = np.arange(256))[0]
            cdf = hist.cumsum()

            if neighborhood.shape[1] <= mask_iter:
                enhanced_pixel = cdf[neighborhood[mask_iter, 0]] * 255 / cdf[-1]
            elif neighborhood.shape[0] <= mask_iter:
                enhanced_pixel = cdf[neighborhood[0, mask_iter]] * 255 / cdf[-1]
            else:
                enhanced_pixel = cdf[neighborhood[mask_iter, mask_iter]] * 255 / cdf[-1]
            enhanced_image[i - mask_iter, j - mask_iter] = enhanced_pixel

    return enhanced_image

```

```

img = Image.open(f"{img_dir}/Fig0326(a)(embedded_square_noisy_512).tif")
img = np.asarray(img)

enhanced_image = local_hist_eq(img, 3, stride = 1)

```

Дээрх зурагт 3×3 маскийг 1 пикселийн зйтай шилжүүлж байсан бол одоо 5×5 маскийг 1-с олон пикселийн зйтай шилжүүлэх процесс хэрхэн харагдахыг сонирхье.

```

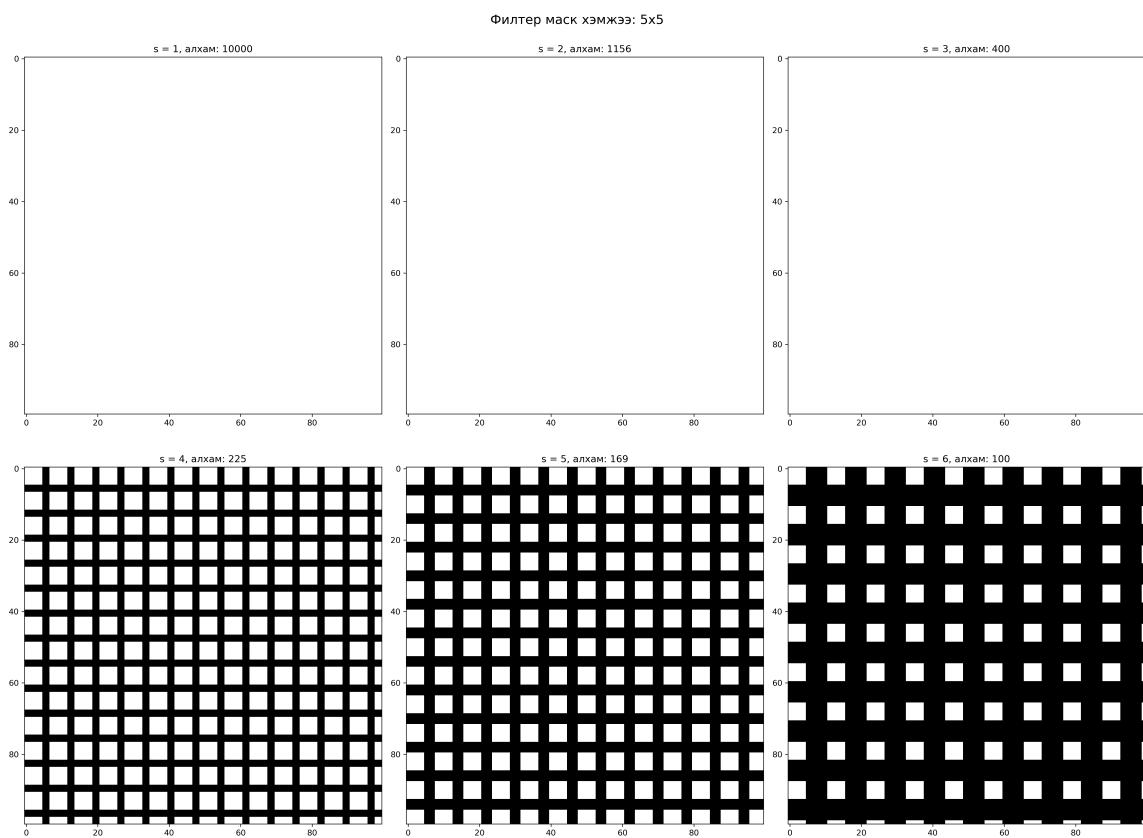
def stride_viz(height, width, stride, mask_size):
    iter_cnt = 0
    mask_iter = mask_size // 2
    img = np.zeros((height, width))

    for i in range(mask_iter, height + mask_iter, stride):
        for j in range(mask_iter, width + mask_iter, stride):
            img[i - mask_iter: i + mask_iter + 1, j - mask_iter: j + mask_iter + 1] = 1
            iter_cnt += 1

    return img, iter_cnt

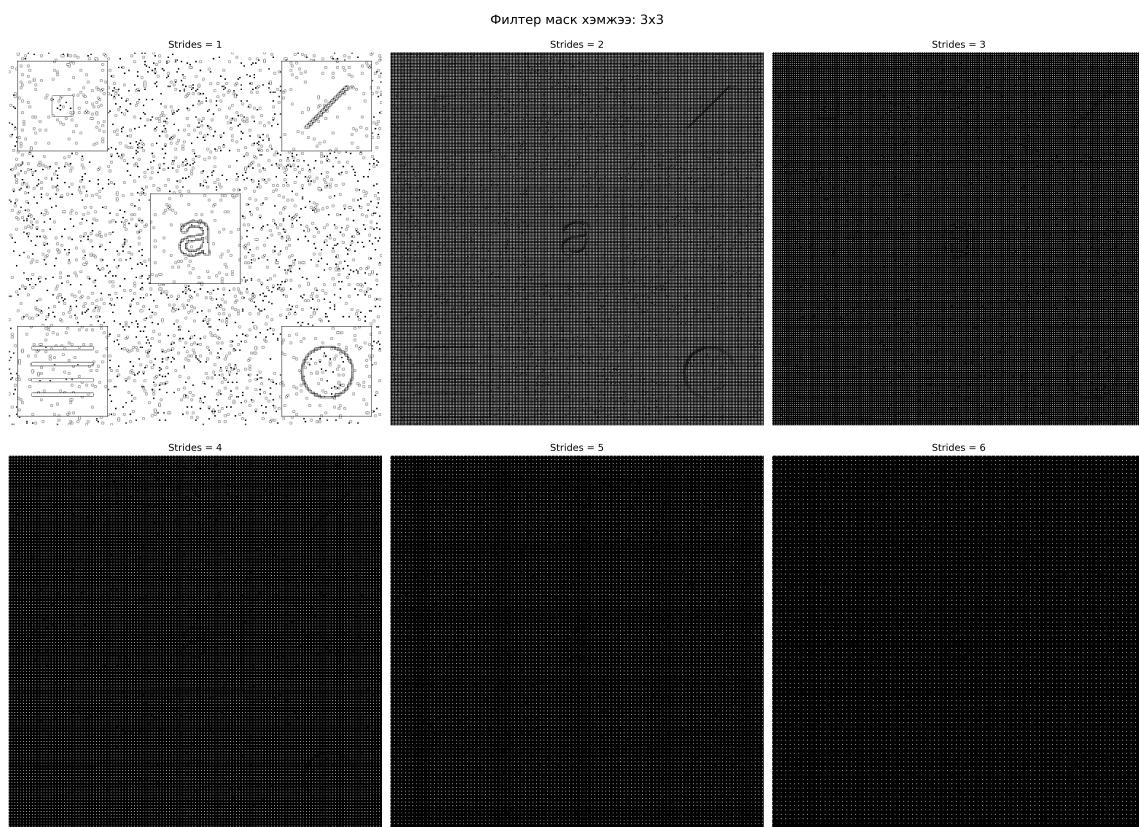
height, width = (100, 100)
img_1, iter_cnt_1 = stride_viz(height, width, stride = 1, mask_size = 5)
img_2, iter_cnt_2 = stride_viz(height, width, stride = 3, mask_size = 5)
img_3, iter_cnt_3 = stride_viz(height, width, stride = 5, mask_size = 5)
img_4, iter_cnt_4 = stride_viz(height, width, stride = 7, mask_size = 5)
img_5, iter_cnt_5 = stride_viz(height, width, stride = 8, mask_size = 5)
img_6, iter_cnt_6 = stride_viz(height, width, stride = 11, mask_size = 5)

```



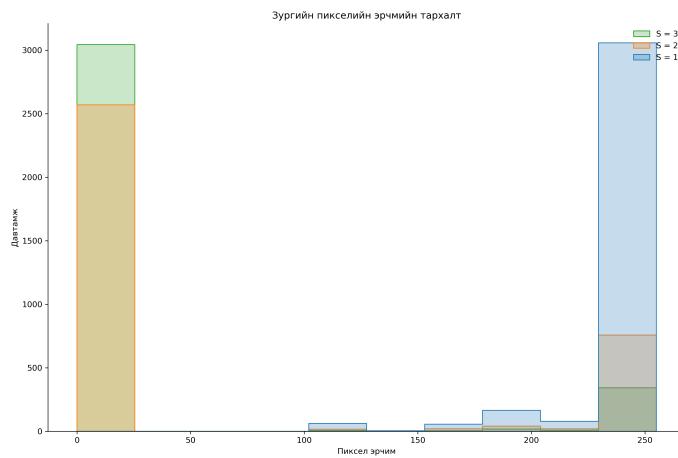
Зураг 9: 5×5 маск ялгаатай зайгаар шилжүүлэх нь

Одоо тэгвэл оролтын зургийн хувьд ялгаатай зайгаар маск шилжүүлэхэд үүсэх зургийг сонирхъё.



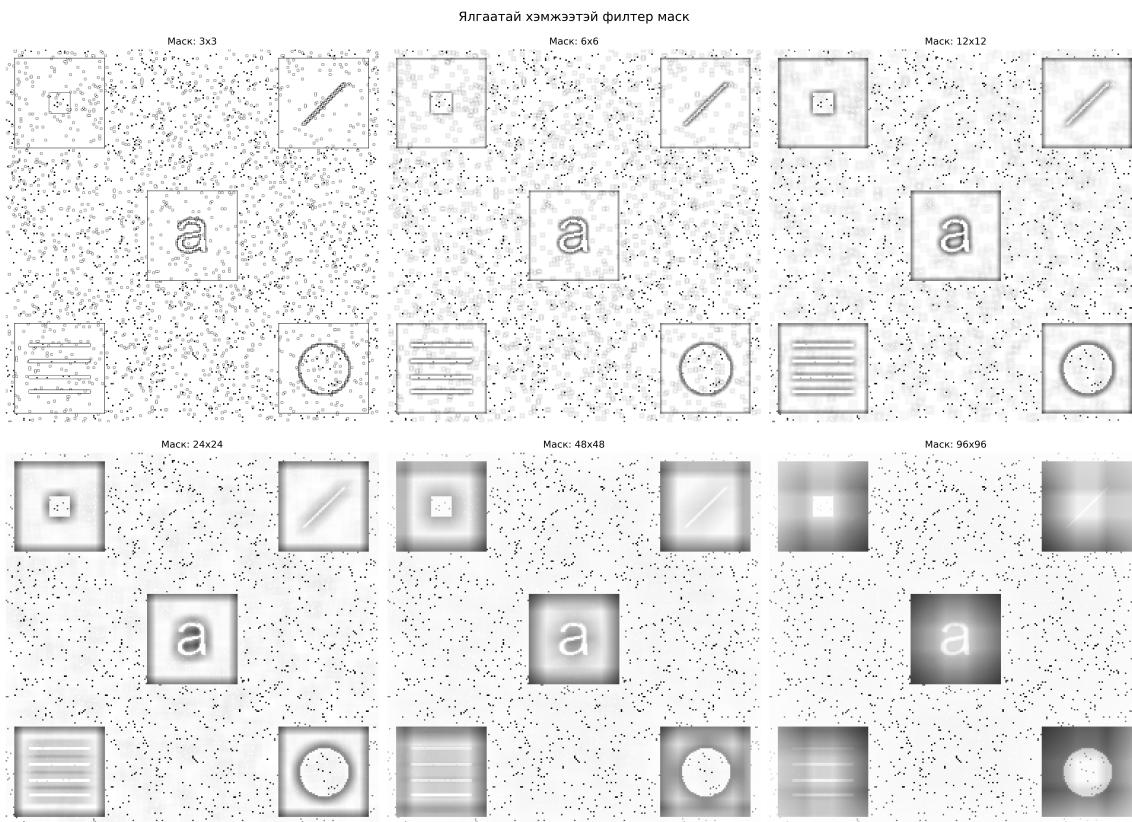
Зураг 10: Оролтын зургийн хувьд маск ялгаатай зайгаар шилжүүлэх нь

Эндээс $Strides = 1, 2, 3$ хувьд үүсэх зургийн тархалтуудыг харьж.

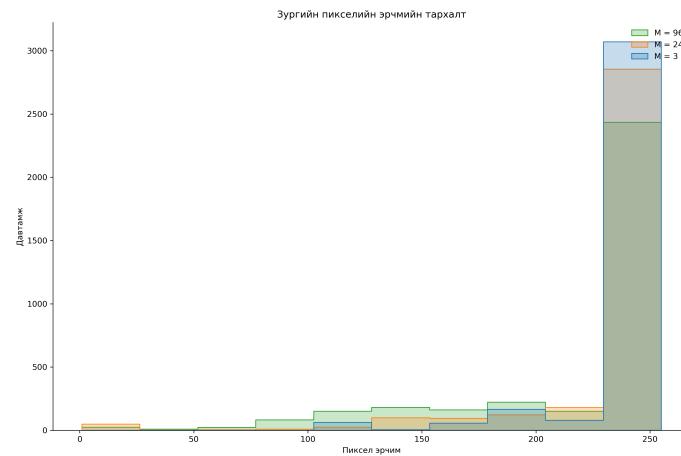


Зураг 11: Оролтын зургийн хувьд маскийг 1, 2, 3 зайгаар шилжүүлсэн нь

Одоо харин маскийн хэмжээг өөрчилж үзье.



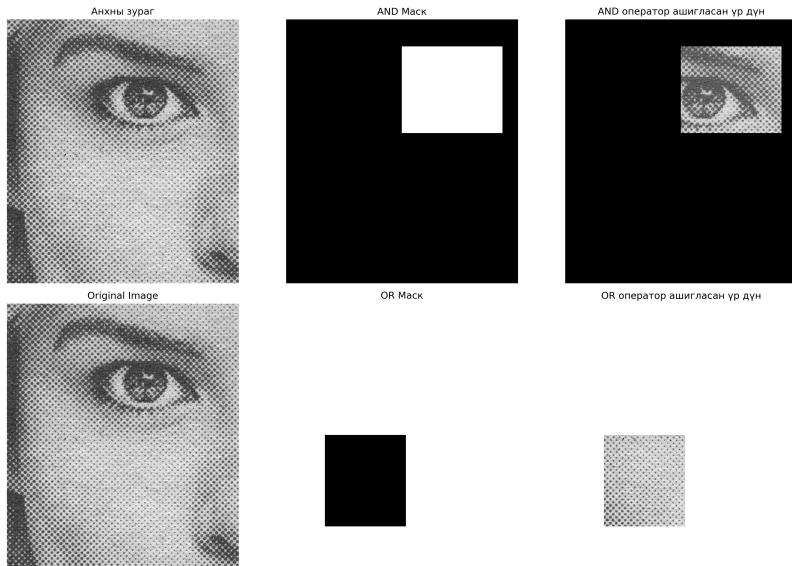
Зураг 12: Оролтын зургийн хувьд маскийг өөрчилсөн нь



Зураг 13: Оролтын зургийн хувьд маскийг өөрчилж шилжилтийн зайд тогтмол барьсан нь

2. Enhancement using Arithmetic / Logic Operations

Энд бид өгөгдсөн маскийг зурагтай уржуулэх үйлдлүүдийг хийнэ.

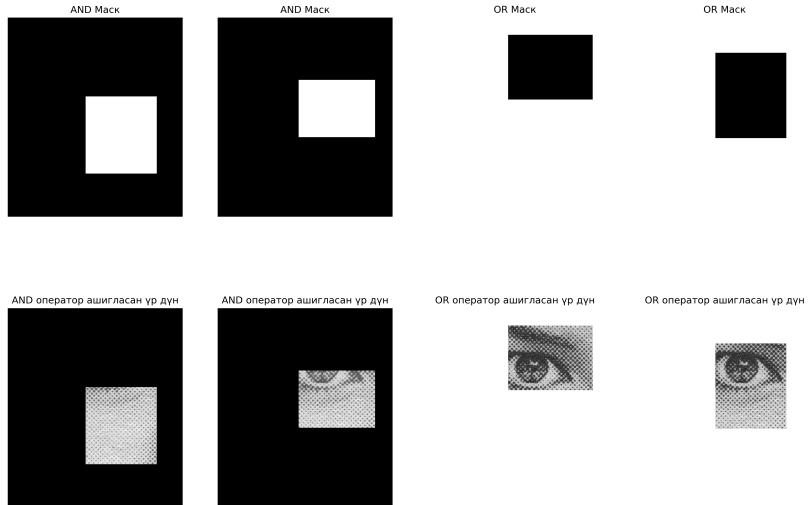


Зураг 14: Оролтын зураг дээрх AND болон OR масктай үржүүлсэн нь

```
def bitwise_op(img, mask, op):
    h, w = img.shape
    img = img / 255
    res = np.zeros((h, w))
    for i in range(h):
        for j in range(w):
            if ((op == 'and') and (img[i][j] and mask[i][j])):
                res[i][j] = img[i][j]
            elif ((op == 'or') and (img[i][j] or mask[i][j])):
                if (mask[i][j] > img[i][j]):
                    res[i][j] = mask[i][j]
                else:
                    res[i][j] = img[i][j]
    return res
```

Дурын маск үүсгэх

```
def transform_mask(img, mask, val):
    a = np.random.randint(img.shape[0]//2)
    b = a + np.random.randint(img.shape[0]//4, img.shape[0]//2)
    c = np.random.randint(img.shape[1]//2)
    d = c + np.random.randint(img.shape[0]//4, img.shape[1]//2)
    mask[a:b, c:d] = val
    return mask
```

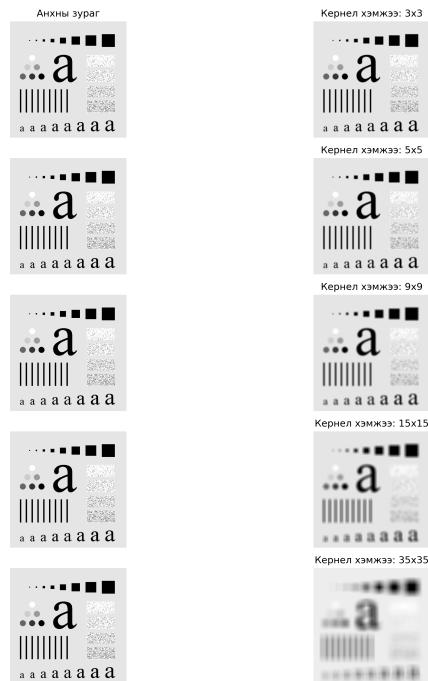


Зураг 15: Ялгаатай AND болон OR маск ашигласан үр дүн

3. Smoothing Linear Filters

Энэ хэсэгт зургийн пиксел бүрийн утгыг хөрш пикселийн утгуудын дунджаар солино [1].

$$I_3 = \frac{1}{3^2} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3)$$



Зураг 16: Оролтын зураг болон бэлэн функц ашигласан smoothing

```
kernel_sizes = [3, 5, 9, 15, 35]
```

```

num_rows = len(kernel_sizes)
num_cols = 2
fig, axes = plt.subplots(num_rows, num_cols, figsize = (14, 12))

for i, kernel_size in enumerate(kernel_sizes):
    kernel = np.ones((kernel_size, kernel_size), dtype = np.float32) / (kernel_size ** 2)
    smoothed_image = cv2.filter2D(img, -1, kernel)

    axes[i, 0].imshow(img, cmap = 'gray')
    if i == 0:
        axes[i, 0].set_title('Ahnii zurag')

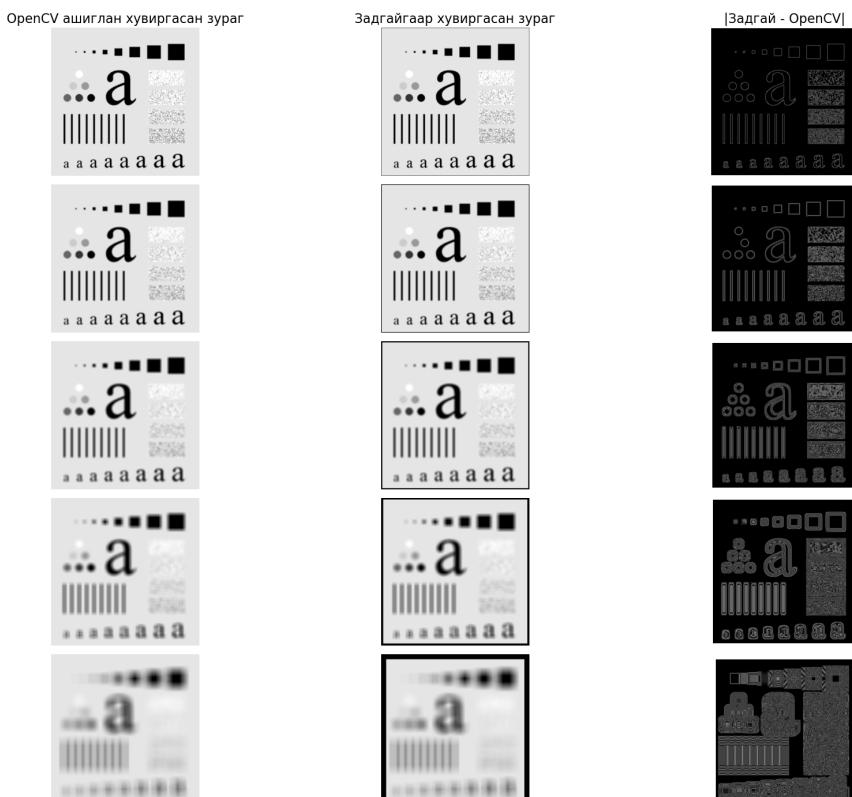
    axes[i, 1].imshow(smoothed_image, cmap='gray')
    axes[i, 1].set_title(f'Kernel hemjee: {kernel_size}x{kernel_size}')

    axes[i, 0].axis('off')
    axes[i, 1].axis('off')

plt.tight_layout()
plt.show()
fig.savefig("./plots/smoothing_1.png", dpi = 300)

```

Харин доорх зурагт задгайгаар бичиж smoothing функц болон бэлэн функцийн үр дунгүүдийн ялгааг харуулж байна.



Зураг 17: Задгайгаар бичиж smoothing функц болон бэлэн функцийн үр дунгийн ялгаа

```

def smooth_img(img, kernel_size):
    height, width = img.shape
    kernel = np.ones((kernel_size, kernel_size), dtype = np.float32) / (kernel_size ** 2)
    smoothed_image = np.zeros((height, width), dtype = np.float32)

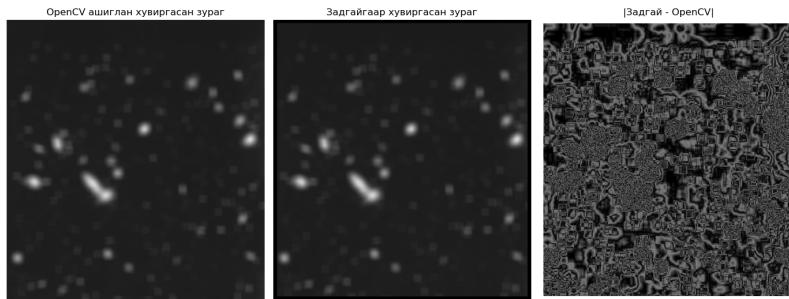
    kernel_iter = kernel_size // 2
    for i in range(kernel_iter, height - kernel_iter):
        for j in range(kernel_iter, width - kernel_iter):
            neighborhood = img[i - kernel_iter: i + kernel_iter + 1, j - kernel_iter: j + kernel_iter + 1]

            smoothed_pixel = np.sum(neighborhood * kernel)
            smoothed_image[i, j] = smoothed_pixel

```

```
return smoothed_image
```

Одоо оролтыг зургийг өөрчилж smoothing хийе. Харин smoothing хийсний дараагаар уг зурагтаа босго тогтоож узье.



Зураг 18: Задгай болон бэлэн smoothing функц

Пикселийн 128 утгаар босго тогтоо ё.

```
thresh = 128  
smoothed_image_cv_thresh = (smoothed_image_cv > thresh)*1  
smoothed_image_thresh = (smoothed_image > thresh)*1
```



Зураг 19: Задгай болон бэлэн smoothing функцийн үр дүнд босго тогтоосон нь

4. Дүгнэлт

Энэ лабораторийн ажлаар зурагт боловсруулалт хийхэд шууд нэг пикселийн хувьд биш харин хэсэг бүлэг пикселийн хувьд боловсруулалт хийж сурлаа. Үүнд ялгаатай хэмжээтэй маск зургаар гүйлгэхдээ ялгаатай зайгаар гүйлгэх мөн зургийн ирмэгийн пикселүүд дээр гүйлгэхийн тулд зурагт padding нэмэх гэх мэт аргуудтай танилцлаа.

5. Ном зүй

- [1] R. C. Gonzalez, *Digital image processing*. Pearson education india, 2009.