

МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

Бадарчийн Бат-Энх

Хэв танилтын үндэс (ICSI380)
(Лаборатори №2)

Компьютерын Ухаан (D061301)
Лабораторийн даалгавар

Улаанбаатар

2025 оны 10 сар

МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

Хэв танилтын үндэс (ICSI380)
(Лаборатори №2)

Компьютерын Ухаан (D061301)
Лабораторийн даалгавар

Удирдагч: _____ Др. Б. Сувдаа

Гүйцэтгэсэн: _____ Б. Бат-Энх (22B1NUM7226)

Улаанбаатар

2025 оны 10 сар

ГАРЧИГ

УДИРТГАЛ	1
1. ОНОЛЫН СУДАЛГАА	2
1.1 Spatial and Intensity Resolution	2
1.2 Image Interpolation	2
1.3 Contrast Stretching	2
1.4 Gray-Level Slicing	2
1.5 Bit-plane Slicing	3
1.6 Histogram Processing	3
1.7 Local Enhancement	3
1.8 Enhancement using Arithmetic / Logic Operations.....	3
1.9 Smoothing Linear Filters.....	3
1.10 Median Filter	4
1.11 Using Second-Derivative for Image Sharpening	4
1.12 Unsharp Masking and High-boost Filtering	4
1.13 Combining Spatial Enhancement Methods	4
1.14 Notch Filters	4
2. ХЭРЭГЖҮҮЛЭЛТ	6
2.1 Spatial and Intensity Resolution	6
2.2 Image Interpolation	7
2.3 Contrast Stretching.....	8
2.4 Gray-Level Slicing	10
2.5 Bit-plane Slicing	12
2.6 Histogram Processing	14
2.7 Local Enhancement	16
2.8 Enhancement using Arithmetic / Logic Operations.....	18
2.9 Smoothing Linear Filters.....	19
2.10 Median Filter	21
2.11 Using Second-Derivative for Image Sharpening	22
2.12 Unsharp Masking and High-boost Filtering	23
2.13 Combining Spatial Enhancement Methods	25

2.14 Notch Filters	25
3. ҮР ДҮН	27
3.1 Intensity quantization.....	27
3.2 Зургийн томруулалт (Image Interpolation).....	28
3.3 Contrast Stretching.....	28
3.4 Gray-Level Slicing	29
3.5 Bit-Plane Slicing	29
3.6 Histogram Processing	30
3.7 Local Enhancement	31
3.8 Logic Operations.....	31
3.9 Smoothing Linear Filters.....	32
3.10 Median Filter	33
3.11 Sharpening using Laplacian	33
3.12 Combined methods.....	34
3.13 Notch Filtering	34
ДҮГНЭЛТ	36
НОМ ЗҮЙ	37
ХАВСРАЛТ.....	38

ЗУРГИЙН ЖАГСААЛТ

3.1	Эх зураг болон кванталсан хувилбарууд. Кванталлалын түвшин багасах тусам posterization илүү тод харагдана.....	27
3.2	Bilinear interpolation аргаар томруулсан үр дүн. Анхны хэмжээ ихсэх тусам чанар сайжирна.	28
3.3	Contrast stretching техник ашиглан бүдэг зургийг сайжруулсан үр дүн. Бараан ба цайвар хэсгүүд илүү тод ялгарах болсон.	28
3.4	Gray-level slicing	29
3.5	8 bit plane харуулав. Bit 7 (MSB) нь зургийн үндсэн бүтцийг агуулж, bit 0 (LSB) нь чимээ болон нарийн texture-ийг агуулдаг.	30
3.6	Зургийн histogram ба normalized histogram. Normalized histogram нь утга тус бүрийн магадлалыг харуулдаг.	30
3.7	Local contrast хийж сайжруулсан үр дүн.	31
3.8	Examples of logic/ mask images used in the experiments.....	31
3.9	Янз бүрийн хэмжээтэй averaging filter ашиглан smoothing хийсэн үр дүн. Kernel хэмжээ ихсэх тусам зураг илүү бүдэгрэх боловч чимээ багасдаг.....	32
3.10	Median filter ашиглан salt-and-pepper noise арилгасан үр дүн. Ирмэгүүд хадгалагдаар чимээ бараг бүрэн устсан байна.	33
3.11	Laplacian оператор ашиглан зургийг хурцалсан үр дүн. Ирмэгүүд тодорхой болж, бүдэг зураг илүү нарийвчилсан болсон байна.	34
3.12	Sobel оператор ашиглан ирмэг илрүүлсэн үр дүн. Объектын хил, хэлбэр тодорхой илэрч байна.....	35
3.13	Notch filter ашиглан periodic noise арилгасан үр дүн. Frequency domain дээр чимээний цэгүүдийг илрүүлж шүүснээр цэвэр зураг гаргаж авсан.....	36

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

2.1	Image Quantization Logic	6
2.2	Bilinear Interpolation Core Logic	7
2.3	Контраст сунгалтын логик	8
2.4	Binary threshold логик	9
2.5	Параметрууд болон гүйцэтгэл	9
2.6	Flat slicing логик	11
2.7	Background preservation slicing логик	11
2.8	Параметрууд болон гүйцэтгэл	12
2.9	Bit plane slicing логик	13
2.10	Histogram тооцоолох логик	14
2.11	Нормчлогдсон Histogram логик	14
2.12	Matplotlib ашиглан харуулах	15
2.13	Local Enhancement 3x3 Logic	17
2.14	Image Inversion	18
2.15	Logic AND Operation	18
2.16	Logic OR Operation	18
2.17	Image Smoothing Logic	19
2.18	Create Smoothing Kernels	20
2.19	Average Filter Logic	21
2.20	Median Filter Logic	21
2.21	Applying Filters	22
2.22	Core Logic	23
2.23	Unsharp and high boost	23
2.24	Applying Filters	25
2.25	Applying Filters	25

УДИРТГАЛ

Компьютерын ухааны нэг хэсэг болох хэв танилт нь судалгааны ажил болон өдөр тутмын амьдралд хэрэглэгдэх чухал сэдвүүдийг судалсаар ирсэн билээ. Хэв танилт сэдэв нь өнөөдрийн эрчимтэй хөгжих буй талбарууд болох Машин сургалт, Хиймэл оюуны судалгааны суурь онол болдог учир өнөөдрийг хүртэл хэв танилтын судалгааны чиглэл эрчимтэй явагдсаар байна. Харийн дүрс боловсруулалт нь хэв танилтын судалгааны томоохон суурь сэдвүүдийн нэг бөгөөд хэв танилтын хамгийн хуучин талбаруудын нэг гэдгээрээ онцлогтой.

Зорилго

Энэхүү лабораторийн хүрээнд ”Хэв танилтын үндэс” хичээлийн хүрээнд судалсан дүрс боловсруулах арга техникийдийг практикт хэрэгжүүлж дүн шинжилгээ хийнэ. Түүнчлэн дүрс боловсруулалтын үндсэн суурь ойлголтуудыг бэхжүүлж, бодит асуудалд тохицуулан параметр турших, үр дүнг үнэлэх чадварыг хөгжүүлэхд тусална.

Тус лабораторыг гүйцэтгэхдээ python хэл болон python хэл дээрх OpenCV, NumPy сангүүдиг ашигласан болно

Зорилт

Дээрх зорилгийн хүрээнд дүрс боловсруулалтын алгоритм болон шинжлэх аргачлалуудыг тус бүрт нь дараах байдлаар судалж үзнэ.

- Онолын судалгаа: Ажиллах зарчмыг ойлгож бататган хэрэгжүүлэх алхмыг хөнгөвчлөх.
- Хэрэгжүүлэлт: Онолын ойлголтыг практикт ашиглаж болох аргуудыг судалж хэрэгжүүлнэ.
- Дүгнэлт: Хэрэгжүүлэлтийн үр дүнг нээлттэй сангүүдийн хэрэгжүүлэлт болон оновчтой үр дүнтэй харьцуулан үр дүнг шинжилнэ.

1. ОНОЛЫН СУДАЛГАА

1.1 Spatial and Intensity Resolution

Spatial resolution нь тухайн зургийн хамгийн жижиг нэгжийн дурсэлж чадах хэсгийн хэлдэг. Өндөр spatial resolution-тэй зураг нь жижиг элементүүдийг нарийвчлалтайгаар харуулах боломжтой. Intensity resolution нь пиксел бүр хэдэн өөр өнгө, туяаг илэрхийлэхийг заадаг бөгөөд өндөр intensity resolution-тай зураг гэрлийн ялгарлыг илүү нарийн үзүүлнэ. Эдгээр хоёр үзүүлэлт нь зураг боловсруулах, дүрсийг сайжруулах, анализ хийх үндсэн шалгуур болдог. Эдгээр хэмжүүрүүд нь анагаахын зураг болон астрономийн зайнаас авах зурагт зайлшгүй хяналт сайжруулалтад байх ёстой үзүүлэлтүүд юм.

1.2 Image Interpolation

Image interpolation нь зургийг масштаблаж хэмжээс өөрчлөгдөх үед тухайн зургийн гол "feature"-үүдийг хадгалж үлдэх зорилготой хэрэглэгддэг. Interpolation хамгийн энгийн бөгөөд түгээмэл арга нь nearest-neighbor арга юм. Хамгийн ойрын пикселийн утгыг шинээр үүсэх пикселд хуваарилах бөгөөд хурдан боловч жигд бус хэт хүрц ирмэгтэй зураг үүсгэж болно. Bilinear interpolation нь ойрын пикселүүдээс жингийн дундажийг авч жигд үр дүн гаргахыг зоридог. Bicubic interpolation нь арван зургаан пикселээс жингийн дундажийг тооцоолж, ирмэгийг хадгалах ба нарийвчилсан мэдээллийг илүү сайн хадгална. Энэхүү арга нь дүрс сайжруулах процесст өргөн хэрэглэгддэг.

1.3 Contrast Stretching

Contrast stretching нь зургийг бүрдүүлж буй гэрэлтэй ба бараан хэсгүүдээс тодорхой мужид орших утгуудыж илүү тодотгон харуулах арга юм. Бага contrast-той буюу бүрсгэх зургуудын элемэнтүүлийг илрүүлэхэд чухал хэрэгтэй техник юм. Мөн зураг дах ялгаралтыг нэмэгдүүлснээр, segmentation болон edge detection зэрэг ахисан түвшний боловсруулалтуудын бэлтгэл алхам болдог.

1.4 Gray-Level Slicing

Gray-level slicing нь зурагт тодорхой intensity дэх мужийг бусад мужаас салган дүрслэх техник юм. Энэ арга нь тодорхой муж дахь пикселүүдийг цайруулж эсвэл тод өнгөтэй болгож, бусад пикселүүдийг хэвээр үлдээх эсвэл бараан болгох зарчмаар ажиллана. Мөн тодруулах явцад background-ийг хэр өөрчилж буйгаас хамаарч хадгалах буюу арилгах гэсэн хоёр төрөлд хуваан авч үздэг. Gray level slicing техник нь зурагт тодорхой объект,

Үл мэдэгдэх хэсгийг илрүүлэхэд үр дүнтэй тул рентген зураг, хэт авиан дүрс сайжруулахад ашиглагддаг.

1.5 Bit-plane Slicing

Bit-plane slicing нь пикселийн бит бүрийг мужид хуваан салгах техник юм. Чимээ үүсгэж буй пикселүүд ихэвчлэн нэг өнгөний мужид оршдог тул feature агуулж буй пиксел болон чимээ үүсгэж буй пикселүүдийг салгахад үр дүнтэй байдаг. Иймд bit plane slicing техник нь дурсийн мэдээлэлд нууц мэдээ нуух болон илрүүлэхэд ашиглагддаг.

1.6 Histogram Processing

Histogram нь зургийн пикселүүдийн тархалтыг графикаар илэрхийлэх арга юм. Энэ нь тухайн дүрсэд дун шинжилгээ хийхэд ашиглагддаг бөгөөд дотроо өөр өөр хэрэглээтэй олон аргачлалд салдаг. Жишээ нь, Энгийн histogram, пикселүүдийн утгын авч тархалтыг харуулах бол Normalized histogram нь аливаа пикселийн авч болох утгын магадлалыг харуулдаг. Түүнчлэн histogram дээр үйлдэл хийснээр эх зурагт боловсруулалт хийх боломжтой болдог. Жишээ нь, "outlier" чимээ илрүүлж хасах, хэт тод болон хэт бүдэг пикселүүдийн утгыг өөрчлөх.

1.7 Local Enhancement

Local enhancement нь тухайн пикселийн ойрын хүрээтэй харьцуулан тэдгээрийн дундаж болон стандарт хазайлтад үндэслэн боловсруулалт хийх техник юм. Жишээ нь 3×3 хүрээ ашиглан төвийн пикселийг орчны утгатай харьцуулж эх дүрсэд шингэсэн feature-үүдийг салгаж авдаг. Local enhancement нь дурсийн пикселүүд нийтдээ өргөн хүрээг хамарсан үед ашиглагддаг.

1.8 Enhancement using Arithmetic / Logic Operations

Зургийг арифметик буюу логик үйлдлээр сайжруулах гэдэг нь пикселийн тоон утга дээрх үйлдэл юм. Logic үйлдлүүд болон AND, OR, XOR, NOT-ийг ашиглан маск үүсгэж зургыг хэсэгчлэн салгах боломжтой. Энэ аргыг ашиглан зурагт background нэмэх, хасах мөн хоёр зураг нийлүүлэх, зургын ялгааг олох гэх мэт үйлдлүүдиг хийж болдог.

1.9 Smoothing Linear Filters

Smoothing linear filters нь зургийн чимээг багасгах, дүрс дэх ирмэгүүдийг зөөлрүүлэх арга юм. Averaging kernel эсвэл Gaussian kernel ашиглан бүдэгрүүлэхдээ төвийн пикселийг ойрын пикселүүдийн дундажтай сольдог.

Kernel-ийн хэмжээ ихсэх тусам бүдгэрэлт ихсэж, чимээ багасдаг боловч ирмэгүүд дагаад бүдгэрэх учир feature extraction хүндрэлтэй болох магадлалтай. Smooth linear filter-ийг noise reduction, edge detection-ийн бэлтгэл шат болгон ашигладаг.

1.10 Median Filter

Median filter smooth linear filter-тэй төстэй зарчимтай бөгөөд дундаж утга ашиглахын оронд медиан утгыг ашигладаг. Ингэснээр хэт тод эсвэл бүдэг пикселүүдийн нөлөөлөл багасдаг. Иймд salt-pepper төрлийн noise-ийг маш сайн арилгадаг. Энэ арга нь ирмэгийг хадгалах чадвартай тул linear averaging-тэй харьцуулахад илүү нарийвчилсан үр дүн өгдөг.

1.11 Using Second-Derivative for Image Sharpening

Зургийг хурцлахад ашиглагдах second-derivative буюу Laplacian операторууд нь дүрс дэх ирмэгүүдийг илрүүлж, гэрлийн хурц өөрчлөлтийг тодруулдаг. Laplacian нь хоёр дугаар зэргийн уламжлал авч, утга өөрчлөгдөх буюу уламжлалын тэмдэг өөрчлөгдөх мөчийг ирмэг гэж үзэж болно гэх онол дээр үндэслэн ажилладаг. Илэрсэн ирмэгийн лүрсийг эх дурсэд нэмж хурцалсан зураг үүсгэдэг. Энэ арга нь spatial domain sharpening-д хамгийн түгээмэл бөгөөд edge detection болон texture enhancement-д ашиглагддаг.

1.12 Unsharp Masking and High-boost Filtering

Unsharp masking нь Gaussian blur-aac үүсэх дүрс болон эх дурс хоёрыг нэмж буюу давхарлана, харин High-boost filtering нь mask-ын коэффициентийг өндөрсгөж илүү хүчтэй хурцуулах боломж олгодог. Энэхүү арга нь local contrast ба ирмэгийг илрүүрэх процессыг хялбарчилж, бүрсгэр зургийг тодруулдаг.

1.13 Combining Spatial Enhancement Methods

Олон spatial enhancement аргуудыг хослуулснаар дүрсний чанарыг хамгийн оновчтой байдлаар сайжруулж болно. Жишээ нь, Laplacian edge sharpening, histogram equalization, local enhancement-г нэгтгэж, зургийн ирмэг, гэрэл, contrast-ийг нэг дор сайжруулах эсвэл эх дүрсээс эхлэн сайжруулсан дүрс хүртлэх автомат Pipeline үүсгэх гэх мэтээр хэрэгжүүлж болно. Composite methods нь ашиглан уян хатан байдлаар боловсруулалт хийх нь ихэнх тохиолдолд хамгийн ашигтай үр дүхг гаргана.

1.14 Notch Filters

Notch filter нь давтамжийн орчинд тодорхой давтамжийн чимээг арилгах зориулалттай. Fourier transform

ашиглан давтамжийн дурслэлийг гаргаж, чимээ болон periodic noise оршин буй цэгүүдийг олж хасаад Inverse Fourier Transform-г ашиглан пикселийн орчинд эргэн хөрвүүлж noise багассан зургийг гаргадаг. Notch filter нь давтамжийн орчинд тодорхой давтамжийн чимээг арилгах хэрэглэгддэг.

2. ХЭРЭГЖҮҮЛЭЛТ

2.1 Spatial and Intensity Resolution

Энэ хэсэгт 256 gray level-тэй эхний grayscale зургийг тодорхой тооны gray level-д багасгах (quantization) аргыг хэрэгжүүлэв. Энэхүү үйлдэл нь зурган дахь пикселийн утгыг хүссэн түвшинд хязгаарлаж, хязгаар давсан утгатай пикселүүдийг бүлэглэх логикоор ажиллаж байна.

2.1.1 Python кодын хэрэгжилт

```
1 # Түвшнүүд
2 levels = [128, 64, 32, 16, 8, 4, 2]
3
4 # Түвшинтусбүрт
5 for L in levels:
6     quantized_img = img.copy()
7
8     # Бүлэглэжбуйлогик
9     factor = 256 / L
10    quantized_img = np.floor(quantized_img / factor) * (255 / (L - 1))
11
12    quantized_img = quantized_img.astype(np.uint8)
```

Код 2.1 Image Quantization Logic

2.1.2 Тайлбар

- **factor:** Тухайн Level-ийн буюу шатлалын хэмжээ. Дээрх жишээнд шатлалын хэмжээ 16 болно.
- **np.floor(I / factor):** Пикселийн утгыг бүхэл тоонд шилжүүлж, gray level-ийг шатлана.
- *** (255 / (L-1)):** Шатласан утгыг 0–255 хүрээнд дахин масштаблана.
- **astype(np.uint8):** Эцсийн зургийг 8-bit формат руу хөрвүүлж хадгална.

2.2 Image Interpolation

Доорх хэсэг bilinear interpolation аргыг ашиглан зургийн хэмжээг томруулах үйлдлийг хэрэгжүүлэв. Хэрэгжүүлэлтэд эх зургийн 128×128 хэмжээг 1024×1024 болгон өрөгтгөв.

2.2.1 Python кодын хэрэгжсилт

```

1 old_height, old_width = image.shape
2 output = np.zeros((new_height, new_width), dtype=image.dtype)
3
4 scale_x = (old_width - 1) / (new_width - 1)
5 scale_y = (old_height - 1) / (new_height - 1)
6
7 for i in range(new_height):
8     for j in range(new_width):
9         x = j * scale_x
10        y = i * scale_y
11
12        x1 = int(x)
13        y1 = int(y)
14        x2 = min(x1 + 1, old_width - 1)
15        y2 = min(y1 + 1, old_height - 1)
16
17        wx = x - x1
18        wy = y - y1
19
20        I11 = image[y1, x1] # Дээдэзүүн
21        I12 = image[y1, x2] # Дээдбаруун
22        I21 = image[y2, x1] # Доодэзүүн
23        I22 = image[y2, x2] # Доодбаруун
24
25        I1 = I11 * (1 - wx) + I12 * wx
26        I2 = I21 * (1 - wx) + I22 * wx
27
28        output[i, j] = I1 * (1 - wy) + I2 * wy
29

```

30 | `return output`

Код 2.2 Bilinear Interpolation Core Logic

2.2.2 Тайлбар

- **scale_x, scale_y:** Хуучин болон шинэ зургийн харьцааг илэрхийлэх харьцаа
- **x = j * scale_x, y = i * scale_y:** Шинэ зурагт байгаа (i, j) пикселийн байршлыг хуучин зургийн (x, y) координат руу шилжүүлнэ.
- **x1, y1, x2, y2:** Координатын эргэн тойрон дахь дөрвөн хамгийн ойрын бүхэл тоот пикселүүдийн байршил. x1, y1 нь зүүн дээд булан, x2, y2 нь баруун доод булан.
- **wx, wy:** Жингийн коэффициентүүд. Жишээ нь: wx = 0.3 гэдэг нь зүүн талын пикселээс 30% зйттай байна гэсэн үг.
- **I11, I12, I21, I22:** Дөрвөн хөрш пикселийн intensity утгууд. Эдгээрийг жинлэсэн дунджаар нийлүүлж шинэ утгыг гаргана.
- **output[i,j] = I1 * (1-wy) + I2 * wy:** Эцсийн алхам. Дээд болон доод шугамын утгуудыг босоо чиглэлд wy жингээр нийлүүлж эцсийн пикселийн утгыг гаргана.

Алгоритмын давуу тал: Nearest neighbor-oos илүү гөлгөр, үр дүн нь дунд зэргийн чанартай. Өргөтгөсөн зурагт блоктой эффект багатай, өнгөний шилжилт зөвлөн байна.

2.3 Contrast Stretching

Энэ хэсэгт зургийн contrast stretching болон binary threshold аргуудыг хэрэгжүүлэв.

2.3.1 Python кодын хэрэгжилт

(Contrast Stretching)

```

1 def contrast_stretch(image, r1, s1, r2, s2):
2     output = np.zeros_like(image)
3     height, width = image.shape
4
5     # Пикселүүрийг боловсруулах
6     for i in range(height):

```

```

7     for j in range(width):
8         pixel = image[i, j]
9         # Гурванмуруйнхэсэг
10        if 0 <= pixel <= r1:
11            output[i, j] = (s1 / r1) * pixel
12        elif r1 < pixel <= r2:
13            output[i, j] = ((s2 - s1) / (r2 - r1)) * (pixel - r1) + s1
14        elif r2 < pixel <= 255:
15            output[i, j] = ((255 - s2) / (255 - r2)) * (pixel - r2) + s2
16
17    return output

```

Код 2.3 Контраст сунгалтын логик

(Binary Threshold)

```

1 def threshold_binary(image, threshold_value):
2     output = np.zeros_like(image)
3     height, width = image.shape
4
5     # Пикселбүрийг босготой харьцуулах
6     for i in range(height):
7         for j in range(width):
8             if image[i, j] >= threshold_value:
9                 output[i, j] = 255 # Цагаан
10            else:
11                output[i, j] = 0 # Хар
12
13    return output

```

Код 2.4 Binary threshold логик

Параметрүүд

```

1 # Контраст сунгалтын оролцогч цэгүүд
2 r1, s1 = 80, 50
3 r2, s2 = 120, 255
4 threshold_val = int(np.sum(img) // (img.shape[0] * img.shape[1]))

```

```

5  stretched_image = contrast_stretch(img, r1, s1, r2, s2)
6  binary_thresh = threshold_binary(img, threshold_val)
Код 2.5 Параметрүүд болон гүйцэтгэл

```

2.3.2 Тайлбар

(Contrast Stretching)

- **r1, s1, r2, s2:** Хяналтын цэгүүд. (r_1, s_1) болон (r_2, s_2) нь хувиргалтын муруйн гол цэгүүд бөгөөд, эдгээр цэгүүдээр үүсэх муруйг ашиглан дурсэд хувиргалт хийнэ:
- **Эхний хэсэг ($0 \leq \text{pixel} \leq r_1$):** Харанхуй пикселүүдийн бүс. $(s_1/r_1) \times \text{pixel}$ томъёогоор тооцоолж, харанхуй хэсэгт өөрчлөлт оруулна. Налуу бага учир бага өөрчлөлт орно.
- **Дунд хэсэг ($r_1 < \text{pixel} \leq r_2$):** Гол ажлын бүс. $\frac{s_2 - s_1}{r_2 - r_1} \times (\text{pixel} - r_1) + s_1$ томъёогоор өндөр налуугаар тэлнэ. Энэ хэсэгт contrast хамгийн их нэмэгддэг.
- **Эцсийн хэсэг ($r_2 < \text{pixel} \leq 255$):** Гэрэлтэй пикселүүдийн бүс. $\frac{255 - s_2}{255 - r_2} \times (\text{pixel} - r_2) + s_2$ Харанхуй бүстэй төстэй налуутай байх учир их өөрчлөлт орохгүй.

Жишээ: $r_1=80, s_1=50, r_2=120, s_2=255$ параметрүүдээр $\text{pixel}=100$ утгыг боловсруулбал:

- $80 < 100 \leq 120$ тул дунд хэсгийн томъёо ашиглана
- $\frac{255 - 50}{120 - 80} \times (100 - 80) + 50 = \frac{205}{40} \times 20 + 50 = 152.5 \approx 153$
- Ахны 100 утга 153 болж, контраст нэмэгдсэн байна

(Binary Threshold)

Binary threshold нь зургийг хар цагаан хоёр өнгө рүү зургыг хөрвүүлнэ. Contrast stretch хийсэн зурагт нэмэлт боловсруулалт болгон ашиглана.

- **threshold_value:** Босго.
- **Шалгах нөхцөл:** Хэрэв пикセル босгоос их буюу тэнцүү бол 255 (цагаан), бусад тохиолдолд 0 (хар) гэж тогтооно.

2.4 Gray-Level Slicing

Энэ хэсэгт (gray level slicing) аргыг хэрэгжүүлэв.

2.4.1 Python кодын хэрэгжсилт

(Flat Slicing)

```

1 def flat_slice(img, mini, maxi):
2     width, height = img.shape
3     out = img.copy()
4
5     # Пикселбүрийгшалгах
6     for i in range(0, width):
7         for j in range(0, height):
8             if out[i][j] < mini:
9                 out[i][j] = 20
10            elif out[i][j] > maxi:
11                out[i][j] = 20
12            # mini <= pixel <= maxi болхэвээрүлдээнэ
13
14     return out

```

Код 2.6 Flat slicing логик

(Background Preservation Slicing)

```

1 def slicer(img, mini, maxi):
2     width, height = img.shape
3     out = img.copy()
4
5     for i in range(0, width):
6         for j in range(0, height):
7             if out[i][j] > mini and out[i][j] < maxi:
8                 out[i][j] = 235
9             else:
10                 out[i][j] = out[i][j]
11
12     return out

```

Код 2.7 Background preservation slicing логик

Параметрүүд

```

1 # Intensity хүрээгтодорхойлох
2 mini, maxi = 155, 255
Код 2.8 Параметрүүд болон гүйцэтгэл

```

2.4.2 Тайлбар

(Flat Slicing)

Flat slicing нь сонгосон intensity хүрээг онцлон харуулж, бусад бүх пикселүүдийг нэг тогтмол бага утгаар солино.

- **mini, maxi:** Сонгосон intensity хүрээний доод болон дээд хязгаар. Жишээ: [155, 255] гэдэг нь цайвар пикселүүдийг онцолж байна.
- **Доод нөхцөл (pixel < mini):** Хүрээнээс доогуур пикселүүдийг бараан утгаар солино.
- **Дээд нөхцөл (pixel > maxi):** Хүрээнээс дээгүүр пикселүүдийг бараан утгаар солино.
- **Хүрээний дотор (mini ≤ pixel ≤ maxi):** Эдгээр пикселүүд хэвээрээ үлдэж, зургийн гол объект болно.

Жишээ: Хэрэв pixel=200 бол $155 < 200 < 255$ учир хэвээр үлдэнэ. Харин pixel=100 бол $100 < 155$ учир 20 болно.

(Background Preservation Slicing)

Энэ арга нь сонгосон хүрээний пикселүүдийг гэрэлтүүлж онцолдог боловч арын талын мэдээллийг устгахгүй, ерөнхий бүтцийг хадгалж, зөвхөн сонгосон хэсгийг тодруулна.

- **Хүрээний нөхцөл (mini < pixel < maxi):** Энэ хүрээний пикселүүдийг 235 гэсэн маш гэрэл утгаар солино. Тэдгээр тод, цагаан өнгөтэй болж онцогдоно.
- **Бусад пикселүүд:** Өөрчлөлтгүй хэвээр үлдэнэ. Арын талын бүтэц, мэдээлэл алдагдахгүй.

2.5 Bit-plane Slicing

2.5.1 Ерөнхий тайлбар

Энэ хэсэгт bit plane slicing аргыг хэрэгжүүлэв. Bit plane slicing нь зургийн пикセル бүрийн 8 битийн төлөөлөлийг тусад нь салгаж харуулсан бөгөөд доорх жишээнд бага битийн чимээ ихтэй дурс ашигласан болно.

2.5.2 Python кодын хэрэгжилт

(Bit Plane Slicing)

```

1 def bit_plane_slice(img, bit_position):
2     output = np.zeros_like(img)
3     height, width = img.shape
4
5     # Пикселбүрийнтухайнбитийгялгах
6     for i in range(height):
7         for j in range(width):
8             bit_value = (img[i, j] >> bit_position) & 1
9             output[i, j] = bit_value * 255
10
11 return output

```

Код 2.9 Bit plane slicing логик

2.5.3 Тайлбар

Битийн давхаргын ойлголт

8-бит саарал зурагт пикセル бүр 0-255 утгатай бөгөөд үүнийг хоёртын тооллын системд 8 битээр илэрхийлнэ.

Жишээ нь: 197 = 11000101₂

- **Bit 7 (MSB):** $2^7 = 128$ - Хамгийн чухал бит, зургийн үндсэн бүтцийг агуулна
- **Bit 6:** $2^6 = 64$ - Дунд зэргийн чухал мэдээлэл
- **Bit 5:** $2^5 = 32$ - Нарийн ширхэгтэй дэлгэрэнгүй мэдээлэл
- **Bit 4:** $2^4 = 16$ - Бага зэргийн нарийн ширхэг
- **Bit 3:** $2^3 = 8$ - Mash бага нөлөөтэй
- **Bit 2:** $2^2 = 4$ - Дуу чимээтэй ойролцоо
- **Bit 1:** $2^1 = 2$ - Дуу чимээ
- **Bit 0 (LSB):** $2^0 = 1$ - Хамгийн бага чухал бит, санамсаргүй дуу чимээ

Алгоритмын тайлбар

- **bit_position:** Ялгах битийн оршиж буй муж (0-7). Bit 0 нь хамгийн баруун бит (LSB), Bit 7 нь хамгийн зүүн бит (MSB).
- **img[i, j] » bit_position:** Битийн баруун тийш шилжүүлэх үйлдэл. Пикселийн утгыг баруун тийш bit_position удаа шилжүүлж, хэрэгтэй битийг хамгийн баруун байрлалд авчирна.
- **& 1: AND үйлдэл.** Хамгийн баруун битийг ялгаж авна. Бусад битүүдийг 0 болгоно.
- **bit_value * 255:** Битийн утга 0 эсвэл 1 байна. Үүнийг 255-аар үржүүлж харагдахуйц болгоно ($0 \rightarrow \text{хар, } 1 \rightarrow \text{цагаан}$).

2.6 Histogram Processing

Энэ хэсэгт зургийн histogram болон normalized histogram тооцоолох аргыг хэрэгжүүлэв.

2.6.1 Python кодын хэрэгжилт

(Standard Histogram)

```

1 def calculate_histogram(image):
2     histogram = np.zeros(256, dtype=int)
3     height, width = image.shape
4
5     for i in range(height):
6         for j in range(width):
7             pixel_value = image[i, j]
8             histogram[pixel_value] += 1
9
10    return histogram

```

Код 2.10 Histogram тооцоолох логик

(Normalized Histogram)

```

1 def calculate_normal_histogram(image):
2     histogram = np.zeros(256, dtype=float)
3     height, width = image.shape
4     total_pixels = height * width # Нийтпикселийнтоо

```

```

5
6     for i in range(height):
7         for j in range(width):
8             pixel_value = image[i, j]
9             histogram[pixel_value] += 1
10
11    histogram = histogram / total_pixels
12
13    return histogram

```

Код 2.11 Нормчлогдсон Histogram логик

Дүрслэх

```

1 # Энгийн Histogram
2 hist = calculate_histogram(img)
3 plt.figure(figsize=(10, 6))
4 plt.bar(range(256), hist, width=1, color='black')
5 plt.xlabel('Pixel\u20d7Intensity')
6 plt.ylabel('Frequency')
7 plt.title('Histogram')
8 plt.xlim(0, 255)
9 plt.savefig('histogram_matplotlib.png')
10 plt.show()
11
12 # Нормчлогдсон Histogram
13 normal_hist = calculate_normal_histogram(img)
14 plt.figure(figsize=(10, 6))
15 plt.bar(range(256), normal_hist, width=1, color='red')
16 plt.xlabel('Pixel\u20d7Intensity')
17 plt.ylabel('Probability')
18 plt.title('Normalized\u20d7Histogram')
19 plt.xlim(0, 255)
20 plt.savefig('histogram_normalized.png')
21 plt.show()

```

Код 2.12 Matplotlib ашиглан харуулах

2.6.2 Тайлбар

(Standard Histogram)

- `histogram = np.zeros(256, dtype=int)`: 256 Утга авч болох хүснэгтүүдийн үүсгэнэ.
- `histogram[pixel_value] += 1`: Пиксел бүрийн утгыг уншиж, тухайн харгалзах хүснэгтийн нүдийг 1-ээр нэмэгдүүлнэ.
- **Үр дүн:** 256 элементтэй хүснэгт. Жишээ нь: `histogram[100] = 350` гэдэг нь зурагт intensity 100-тай 350 пиксел байна гэсэн үг.

Жишээ: 4×4 хэмжээтэй жижиг зурагт:

```
[50, 50, 100, 100]
[50, 150, 100, 200]
[50, 150, 150, 200]
[50, 150, 150, 200]
```

Histogram: `histogram[50]=5, histogram[100]=3, histogram[150]=4, histogram[200]=3, бусад=0`

(Normalized Histogram)

Нормчлогдсон Histogram нь давтамжийг магадлал болгон хувиргадаг.

- `total_pixels = height * width`: Зургийн нийт пикселийн тоо. Жишээ: $512 \times 512 = 262,144$ пиксел.
- `histogram = histogram / total_pixels`: Давтамж бүрийг нийт тоонд хувааж, магадлалын тархалт гаргана.

Жишээ: Дээрх 4×4 зурагт (нийт 16 пиксел):

- $p(50) = 5/16 = 0.3125$ (31.25%)
- $p(100) = 3/16 = 0.1875$ (18.75%)
- $p(150) = 4/16 = 0.25$ (25%)
- $p(200) = 3/16 = 0.1875$ (18.75%)

2.7 Local Enhancement

Энэхүү кодоор нь local enhancement техникийг хэрэгжүүлэв.

2.7.1 Python кодын хэрэгжилт

(Local Enhancement)

```

1 def local_enhance_3x3(img, x, y, enhancement_factor=15):
2     kernel_size = 1 # 3x3 орчин (1 пикселийн радиус )
3
4     neighborhood = []
5     for i in range(x - kernel_size, x + kernel_size + 1):
6         for j in range(y - kernel_size, y + kernel_size + 1):
7             neighborhood.append(img[i, j])
8
9     center_pixel = img[x, y] # Төвийн пиксел
10    local_mean = np.mean(neighborhood) # Орчны дундаж
11    local_std = np.std(neighborhood) # Орчны стандарт хазайлт
12
13    if center_pixel < local_mean:
14        enhanced_value = 0 # Харанхуй болгох
15    else:
16        enhanced_value = 255 # Гэрэл болгох
17
18    return enhanced_value

```

Код 2.13 Local Enhancement 3x3 Logic

2.7.2 Тайлбар

- **kernel_size = 1:** 3x3 хэмжээтэй цонх (kernel) ашиглана. 1 нь төвөөс 1 пиксел зайдай орчныг тодорхойлно.
- **neighborhood:** Тухайн пикселийн эргэн тойрны 3x3 хэмжээтэй 9 пикселийн жагсаалт. Эндээс утгуудаас дундаж болон стандарт хазайлт тооцоолно.
- **local_mean:** Орчны дундаж утга.
- **local_std:** Орчны стандарт хазайлт.
- **Босго:** Хэрэв төвийн пиксел орчны дундажаас бага бол 0 (хар), эс тэгвээс 255 (цагаан) болгоно. Энэ нэг ёсондоо binary thresholding болно.

2.8 Enhancement using Arithmetic / Logic Operations

2.8.1 *Image Logic Operations* алгоритмын өрөнхий тойм

Доорх хэсэгт логик үйлдлүүд (AND, OR) болон урвуу үйлдэл хийх код хэрэгжүүлэв.

2.8.2 *Python* кодын хэрэгжилт

Invert функц

```

1 def invert(img):
2     out = img.copy()
3     for i in range(height):
4         for j in range(width):
5             out[i][j] = 255 - img[i][j]
6
7     return out

```

Код 2.14 Image Inversion

Logic AND үйлдэл

```

1 def logic_and(img, mask):
2     out = img.copy()
3     for i in range(height):
4         for j in range(width):
5             if mask[i][j] & img[i][j]:
6                 out[i][j] = img[i][j]
7             else:
8                 out[i][j] = mask[i][j]
9
10    return out

```

Код 2.15 Logic AND Operation

Logic OR үйлдэл

```

1 def logic_or(img, mask):
2     out = img.copy()
3     for i in range(height):
4         for j in range(width):

```

```

5     if (mask[i][j]==0) | (img[i][j]==0):
6         out[i][j] = img[i][j]
7     else:
8         out[i][j] = mask[i][j]
9
return out

```

Код 2.16 Logic OR Operation

2.8.3 Тайлбар

Invert функц

- $255 - \text{img}[i][j]$: Пикселийн урвуу утга. Цагаан хар болж, хар цагаан болно.

Logic AND үйлдэл

- $\text{mask}[i][j] \& \text{img}[i][j]$: Хоёр утга хоёулаа 0-ээс ялгаатай бол үнэн.

Logic OR үйлдэл

- $(\text{mask}[i][j]==0) | (\text{img}[i][j]==0)$: Аль нэг нь 0 байвал нөхцөл биелнэ.

2.9 Smoothing Linear Filters

Энэхүү код нь зургийг smooth (гөлгөр) болгох алгоритм юм.

2.9.1 Python кодын хэрэгжилт

Manual Smooth функц

```

1 def manual_smooth(img, kernel_size):
2     output = np.zeros_like(img, dtype=np.float32)
3     half_size = kernel_size // 2
4
5     for i in range(height):
6         for j in range(width):
7             total = 0
8             count = 0
9
10            for ki in range(-half_size, half_size + 1):

```

```

11     for kj in range(-half_size, half_size + 1):
12         ni, nj = i + ki, j + kj
13
14         # Зургийнхүрээнд байгаа эсэхийг шалгах
15         if 0 <= ni < height and 0 <= nj < width:
16             total += img[ni, nj]
17             count += 1
18
19         # Дундажутгыг тооцоолох
20         output[i, j] = total / count if count > 0 else 0
21
22     return output.astype(np.uint8)

```

Код 2.17 Image Smoothing Logic

Kernel үүсгэх функц

```

1 def create_smoothing_kernels():
2
3     kernel_sizes = [3, 5, 9, 15, 35]    # Янзбүрийнхэмжээ
4     kernels = {}
5
6
7     for size in kernel_sizes:
8
9         # Бүхэлментнүүжиулутгатай      kernel
10        kernel = np.ones((size, size), dtype=np.float32) / (size * size)
11        kernels[size] = kernel
12
13
14    return kernels

```

Код 2.18 Create Smoothing Kernels

2.9.2 Тайлбар

Manual Smooth функц

- **half_size = kernel_size // 2:** Kernel-ийн радиус. Жишээ нь 5x5 kernel бол радиус нь 2 болно.
- **total / count:** Орчны пикселүүдийн дундаж утга. Энэ нь төвийн пикселийн шинэ утга болно.
- **0 <= ni < height:** Зургийн захаас гарахгүй байхыг шалгана.

Kernel үүсгэх функци

- `np.ones((size, size))`: Бүх элемент нь 1 байх матриц үүсгэнэ.
- `/ (size * size)`: Дундаж авахын тулд нийт элементийн тоонд хувааж normalize хийнэ.

Жишээ нь

- **3x3 kernel**: бүх элемент 1/9 утгатай болно.

2.10 Median Filter

Энэхүү код нь зураг дахь salt-and-pepper дуу чимээг арилгах хоёр фильтрийг хэрэгжүүлдэг. Average filter нь орчны дундажаар, Median filter нь орчны дунд утгаар пикселийг солино. Median filter нь дуу чимээг илүү сайн арилгадаг.

2.10.1 Python кодын хэрэгжсилт

Average Filter функци

```

1 def avg(img, x, y):
2     sum = 0
3     for i in range(x - 1, x + 2):
4         for j in range(y - 1, y + 2):
5             sum += img[i][j]
6     return sum / 9

```

Код 2.19 Average Filter Logic

Median Filter функци

```

1 def median(img, x, y):
2     buffer = []
3     for i in range(x - 1, x + 2):
4         for j in range(y - 1, y + 2):
5             buffer.append(img[i][j])
6
7     buffer.sort()
8     return buffer[4]

```

Код 2.20 Median Filter Logic

Filter-үүдийг хэрэглэх

```

1 buffer = np.zeros_like(img, dtype=np.float32)
2 buffer_med = np.zeros_like(img, dtype=np.float32)
3
4 for i in range(1, dim[0] - 1):
5     for j in range(1, dim[1] - 1):
6         buffer[i][j] = avg(img, i, j) # Average
7         buffer_med[i][j] = median(img, i, j) # Median
8
9 dst = cv2.medianBlur(img, 3)

```

Код 2.21 Applying Filters

2.10.2 Тайлбар

Average Filter

- **sum / 9:** 3x3 орчны 9 пикселийн дундаж утга. Энэ нь дуу чимээг зөөлрүүлнө.
- Алдаа: дуу чимээний утга (0 эсвэл 255) дундажд нөлөөлж, зураг бүдгэрнэ.

Median Filter

- **buffer.sort():** Орчны утгуудыг эрэмбэлнэ. Жишээ: [0, 50, 60, 70, 80, 90, 100, 110, 255].
- **buffer[4]:** Эрэмбэлсэн жагсаалтын дунд утга (5 дахь элемент). Жишээн дээрх дунд утга нь 80 болно.
- Давуу тал: хэт их эсвэл багатай утгуудыг үл тоомсорлож, salt-and-pepper дуу чимээг үр дүнтэй арилгана.

Харьцуулалт

- **cv2.medianBlur():** OpenCV-ийн optimize хийгдсэн median filter. Илүү хурдан ажиллана.
- **dst - buffer_med:** Manual болон built-in функцийн ялгаа. Захын пикселүүдийн боловсруулалтад ялгаа гарч болно.

2.11 Using Second-Derivative for Image Sharpening

Ирмэг илрүүлэхэд түгээмэл ашиглагддаг хоёр дугаар эрэмбийн уламжлал (Second-Derivative) ашиглан зургийг хурцлах аргыг хэрэгжүүлэв.

```

1 laplacian_kernel_1 = np.array([[0, 1, 0],
2                                 [1, -4, 1],
3                                 [0, 1, 0]])
4 laplacian_kernel_2 = np.array([[1, 1, 1],
5                                 [1, -8, 1],
6                                 [1, 1, 1]])
7 def sharpening(img, x, y, kernel=0):
8     if x < 1 or x >= img.shape[0] - 1 or y < 1 or y >= img.shape[1] - 1:
9         return img[x][y]
10
11     val = 0
12     for i in range(0, 3):
13         for j in range(0, 3):
14             val += laplacian[kernel][i][j] * img[x - 1 + i][y - 1 + j]
15
16     return np.clip(val, 0, 255)

```

Код 2.22 Core Logic

Тайлбар

- `laplacian_kernel_1`, `laplacian_kernel_2`: Хоёр төрлийн Laplacian kernel. Kernel ийн нийлбэр учир ижил хөрштэй пикселүүд буюу хар өөр буюу ирмэг байх магадлалтай пикселүүд 0 ээс өөр буюу гэрэлтэй болж хувирна.

2.12 Unsharp Masking and High-boost Filtering

Энэхүү код нь зургийг тодруулахын тулд ашиглагдах unsharp masking болон high-boost filtering аргуудыг хэрэгжүүлэв.

```

1 # ...existing code...
2 def unsharp_mask(img, ksize=3, amount=1.0):
3     """
4     @param img: 2D uint8 grayscale
5     @param ksize: odd blur kernel size (3,5,...)
6     @param amount: strength of added mask (e.g., 0.5..2.0)

```

```

8     """
9     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
10    mask = img.astype(np.int16) - blurred.astype(np.int16)
11    sharpened = img.astype(np.int16) + np.round(amount * mask).astype(np.int16)
12    return np.clip(sharpened, 0, 255).astype(np.uint8)
13
14 def high_boost(img, ksize=3, A=1.5):
15     """
16     Core logic: high-boost filtering
17     A: boost factor (A=1 -> unsharp mask baseline; A>1 -> boosted)
18     High-boost result = img + (A-1)*(img - blurred)
19     """
20
21     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
22     mask = img.astype(np.int16) - blurred.astype(np.int16)
23     highboosted = img.astype(np.int16) + np.round((A - 1) * mask).astype(np.int16)
24     return np.clip(highboosted, 0, 255).astype(np.uint8)
25
26
27 Brief: unsharp_mask adds a scaled high-frequency mask to the original; high_boost uses
28 an amplification factor A to emphasize the mask (A>1 increases sharpening).
29
30 def unsharp_mask(img, ksize=3, amount=1.0):
31
32     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
33     mask = img.astype(np.int16) - blurred.astype(np.int16)
34     sharpened = img.astype(np.int16) + np.round(amount * mask).astype(np.int16)
35     return np.clip(sharpened, 0, 255).astype(np.uint8)
36
37 def high_boost(img, ksize=3, A=1.5):
38     """
39     Core logic: high-boost filtering
40     A: boost factor (A=1 -> unsharp mask baseline; A>1 -> boosted)
41     High-boost result = img + (A-1)*(img - blurred)
42     """
43
44     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)

```

```

42     mask = img.astype(np.int16) - blurred.astype(np.int16)
43     highboosted = img.astype(np.int16) + np.round((A - 1) * mask).astype(np.int16)
44
45     return np.clip(highboosted, 0, 255).astype(np.uint8)

```

Код 2.23 Unsharp and high boost

2.13 Combining Spatial Enhancement Methods

```

1 def unsharp_mask(img, ksize=3, amount=1.0):
2
3     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
4
5     mask = img.astype(np.int16) - blurred.astype(np.int16)
6
7     sharpened = img.astype(np.int16) + np.round(amount * mask).astype(np.int16)
8
9     return np.clip(sharpened, 0, 255).astype(np.uint8)
10
11
12 def high_boost(img, ksize=3, A=1.5):
13
14     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
15
16     mask = img.astype(np.int16) - blurred.astype(np.int16)
17
18     highboosted = img.astype(np.int16) + np.round((A - 1) * mask).astype(np.int16)
19
20     return np.clip(highboosted, 0, 255).astype(np.uint8)
21
22
23 def unsharp_mask(img, ksize=3, amount=1.0):
24
25
26     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
27
28     mask = img.astype(np.int16) - blurred.astype(np.int16)
29
30     sharpened = img.astype(np.int16) + np.round(amount * mask).astype(np.int16)
31
32     return np.clip(sharpened, 0, 255).astype(np.uint8)
33
34
35 def high_boost(img, ksize=3, A=1.5):
36
37     blurred = cv2.GaussianBlur(img, (ksize, ksize), 0)
38
39     mask = img.astype(np.int16) - blurred.astype(np.int16)
40
41     highboosted = img.astype(np.int16) + np.round((A - 1) * mask).astype(np.int16)

```

Код 2.24 Applying Filters

2.14 Notch Filters

```

1 def compute_spectrum(img):

```

```

2     dft = np.fft.fft2(img)
3     dft_shift = np.fft.fftshift(dft)
4     mag = 20 * np.log(np.abs(dft_shift) + 1)
5     return dft_shift, mag
6
7
8 def gaussian_notch_mask(shape, notch_points, sigma=10):
9     rows, cols = shape
10    crow, ccol = rows // 2, cols // 2
11    y, x = np.ogrid[:rows, :cols]
12    mask = np.ones((rows, cols), dtype=np.float32)
13
14    for (u, v) in notch_points:
15        gauss1 = np.exp(-(((x - (ccol + u))**2 + (y - (crow + v))**2) / (2.0 * sigma *
16                           sigma)))
17        gauss2 = np.exp(-(((x - (ccol - u))**2 + (y - (crow - v))**2) / (2.0 * sigma *
18                           sigma)))
19
20        # attenuation: multiply by (1 - gaussian) so center of notch goes to 0
21        mask *= (1.0 - gauss1)
22        mask *= (1.0 - gauss2)

23    return mask

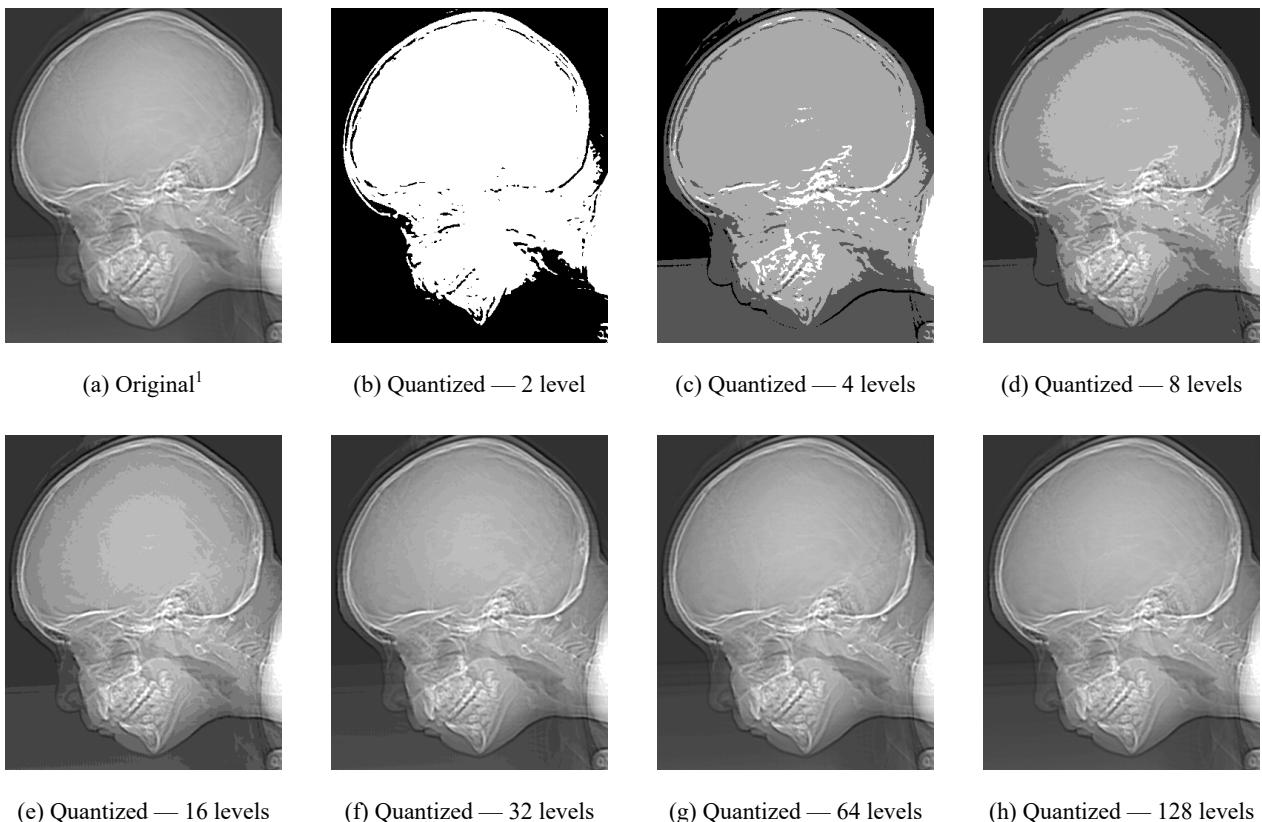
```

Код 2.25 Applying Filters

3. УРДУН

3.1 Intensity quantization

Энэ хэсэгт зургийн intensity түвшний тоог өөр өөр утгаар хязгаарласан үр дүнг харуулав. Gray-level quantization нь зургийн мэдээллийг багасгаж, эх зурагт posterization буюу блок үүсэх эфект харагдана.



Зураг 3.1 Эх зураг болон кванталсан хувилбарууд. Кванталлалын түвшин багасах тусам posterization илүү тод хаяргдана.

Үр дүн: Кванталсан level-ын тоо багасах тусам зурагт posterization шатлалт/блок эфект илүү тод илэрнэ: 2 болон 4 түвшин нь нягтаршил ихтэй битийн хязгаарлагдмал дүрслэл үүсгэж, нарийн градиент болон сульялгарлууд алга болно. 8–16 түвшин нь ихэнх градиентыг авч үлдэнэ, харин 64–128 түвшин нь эх зурагт ойр үр дүн өгнө.

¹Source: Digital Image Processing (Gonzalez & Woods), Chapter 2, Figure 2.21(a)

3.2 Зургийн томруулалт (Image Interpolation)

Bilinear interpolation аргыг ашиглан өөр өөр анхны хэмжээтэй зургуудыг 1024×1024 хүртэл томруулав. Жижиг хэмжээтэй анхны зургаас томруулах тусам блур болон aliasing эфект илүү тод харагдана.

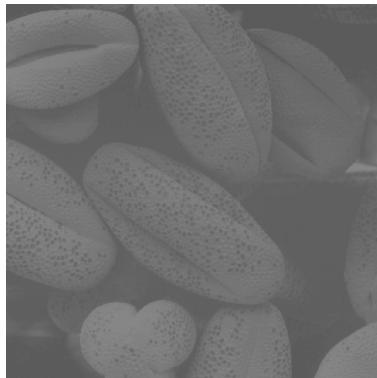
(a) $32 \times 32 \rightarrow 1024 \times 1024$ (b) $64 \times 64 \rightarrow 1024 \times 1024$ (c) $128 \times 128 \rightarrow 1024 \times 1024$

Зураг 3.2 Bilinear interpolation аргаар томруулсан үр дүн. Анхны хэмжээ ихсэх тусам чанар сайжирна.

Үр дүн: 32×32 эх зураг хамгийн бүдэг ир дүн гаргасан бөгөөд pixelation илэрхийй байна. 64×64 нь зарим нарийвчлалыг хадгалж чадсан боловч блур бага зэрэг үлдсэн. 128×128 эх зураг нь хамгийн тод үр дүн өгч, ирмэг болон текстурын мэдээлэл илүү сайн хадгалагдсан байна.

3.3 Contrast Stretching

Contrast stretching нь бага contrast-той зургийн гэрэл, бараан хэсгүүдийг өргөсгөн тод болгох арга юм. Зураг 3.3-д эх зураг болон contrast stretching хийсэн үр дүнг харуулав.

(a) Эх зураг (бүдэг)²

(b) Contrast stretching хийсэн



(c) Threshold хийсэн

Зураг 3.3 Contrast stretching техник ашиглан бүдэг зургийг сайжруулсан үр дүн. Бараан ба цайвар хэсгүүд илүү тод ялгарах болсон.

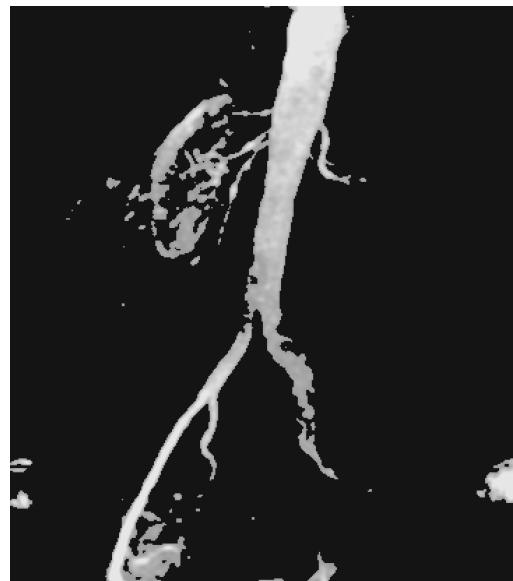
Үр дүн: Эх зураг нь intensity value-үүд нарийн мужид (жишээ нь 50–150) төвлөрсөн байсан бол contrast stretching дараа 0–255 мужид тархаж гэрэлтэй болон бараан хэсгүүдийн ялгарал тодорхой болсон байна.

3.4 Gray-Level Slicing

Gray-level slicing нь зургийн тодорхой интенситийн мужид байгаа пикселүүдийг илрүүлж тусгаарлах арга юм. Зураг ??-д хоёр төрлийн gray-level slicing үр дүнг харуулав.



(a) Background хадгалсан



(b) Background арилгасан

Зураг 3.4 Gray-level slicing

Үр дүн: Тодорхой intensity мужид орших объектуудыг цайруулж тодруулсан. Зүүн талд background хадгалсан, баруун талд арилгасан хувилбарыг үзүүлэв.

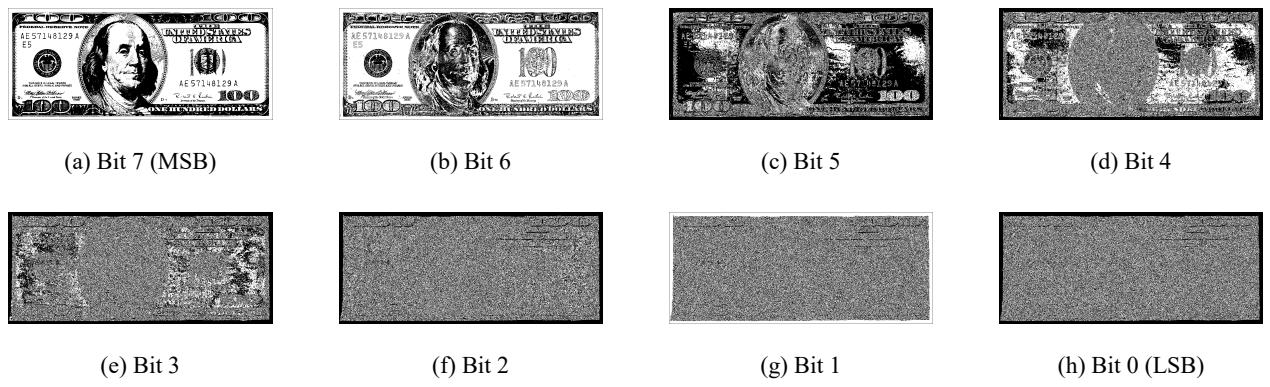
Хэрэглээ: Энэ арга нь медицины зураг дээр тодорхой төрлийн эд, эсвэл industrial inspection-д гэмтсэн хэсгийг илрүүлэхэд өргөн хэрэглэгддэг.

3.5 Bit-Plane Slicing

Bit-plane slicing нь пикселийн binary төлөөлөлийн тус бүр битийг салгаж Зураг 3.5-д харуулав.

Үр дүн: Өндөр bit plane-үүд (6, 7) нь зургийн үндсэн мэдээллийг тодорхой харуулж байхад доод bit plane-үүд (0, 1, 2) нь чимээ шиг санагддаг. Энэ техник нь data compression, steganography-д ашиглагддаг.

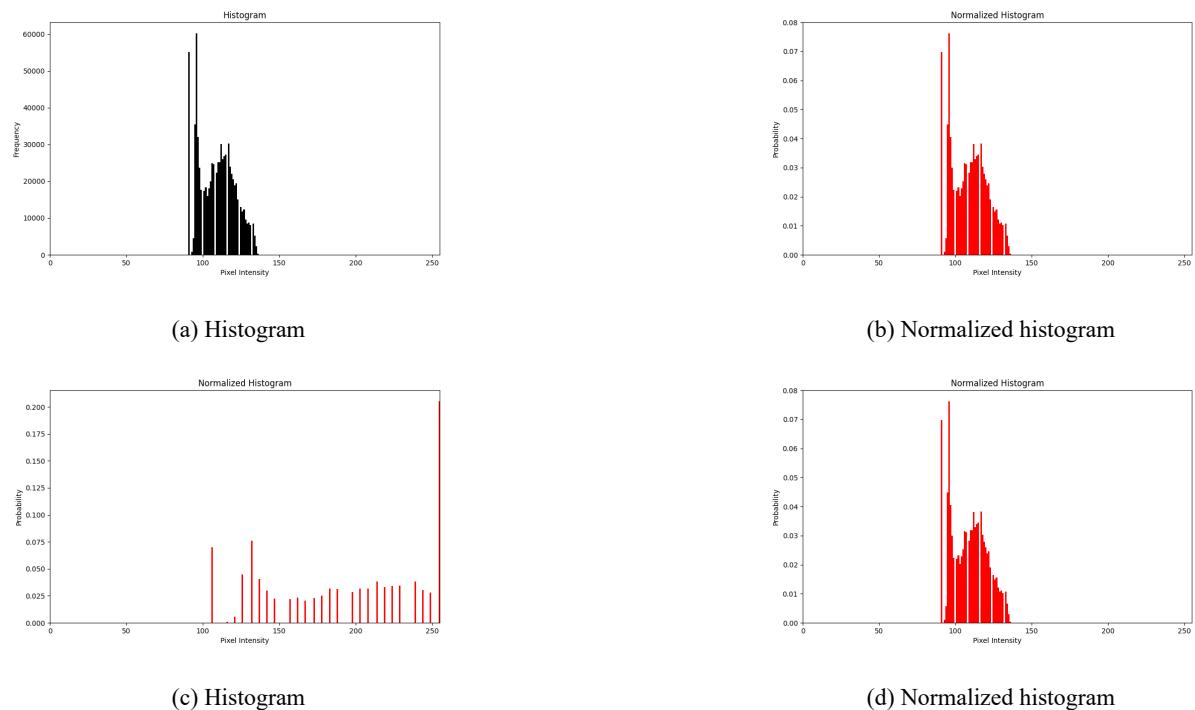
²Source: Digital Image Processing (Gonzalez & Woods), Chapter 3, Figure 3.10(b)



Зураг 3.5 8 bit plane харуулав. Bit 7 (MSB) нь зургийн үндсэн бүтцийг агуулж, bit 0 (LSB) нь чимээ болон нарийн texture-ийг агуулдаг.

3.6 Histogram Processing

Histogram нь зургийн intensity утгуудын тархалтыг харуулах график юм. Histogram equalization нь тархалтыг жигдүүлж contrast-ийг нэмэгдүүлдэг. Зураг 3.6-д histogram болон түүний normalized хэлбэрийг үзүүлэв.

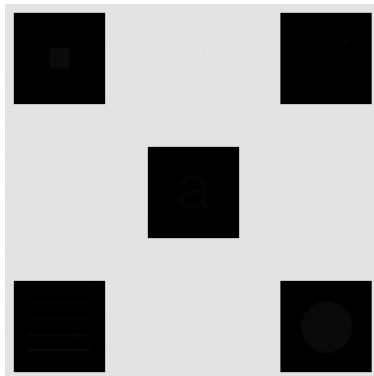
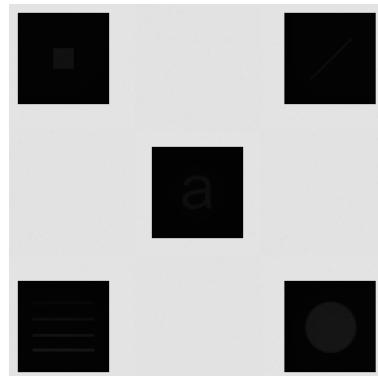


Зураг 3.6 Зургийн histogram ба normalized histogram. Normalized histogram нь утга тус бүрийн магадлалыг харуулдаг.

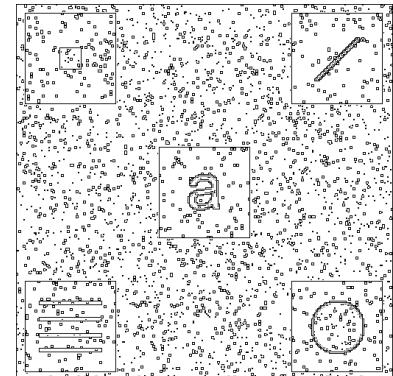
Хэрэглээ: Histogram analysis нь зургийн чанар үнэлэх, автомат threshold олох, түүх болон adaptive enhancement хийхэд ашиглагддаг.

3.7 Local Enhancement

Local enhancement нь зургийн локал хэсэг тус бүрийг орчны статистикт үндэслэн боловсруулах арга юм. CLAHE (Contrast Limited Adaptive Histogram Equalization) нь хамгийн түгээмэл local enhancement арга. Зураг 3.7-д үр дүнг харуулав.

(a) Эх зураг³

(b) CLAHE хийсэн



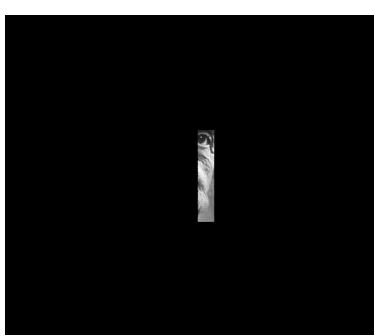
(c) Гараар хийсэн

Зураг 3.7 Local contrast хийж сайжруулсан үр дүн.

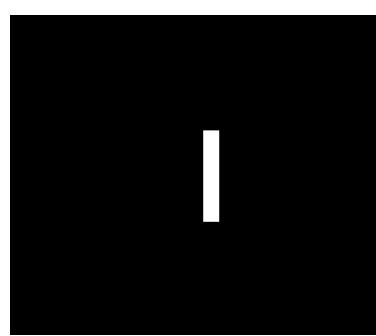
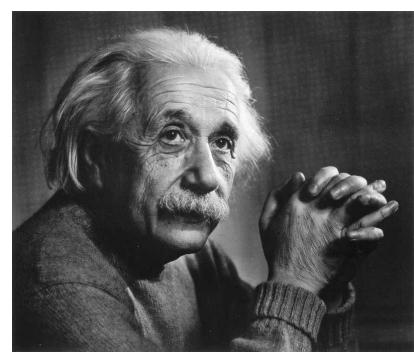
Үр дүн: Global histogram equalization-аас ялгаатай нь CLAHE нь локал хэсэг бүрийн contrast-ийг тусад нь сайжруулдаг учир чимээ багасгаж, нарийн дэлгэрэнгүй мэдээллийг илүү сайн гаргаж авдаг. Гэвч энэ тохиолдолд Гараар хийсэн local enhancement-тай харьцуулахад сүл үр дүн гаргасан байна.

3.8 Logic Operations

Logic operations Зураг ??-д AND үйлдлийн үр дүнг харуулав.



(a) Logic AND — results.

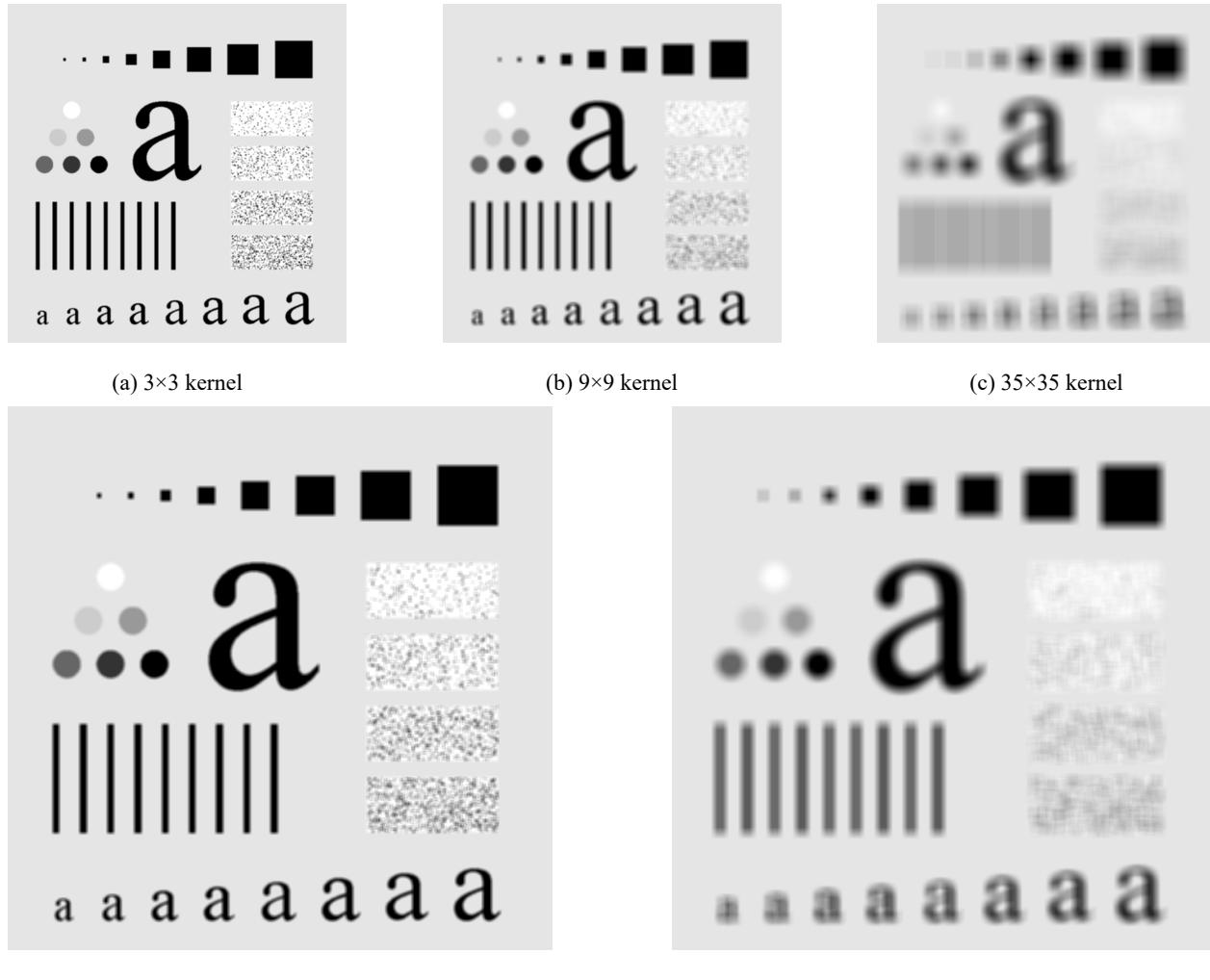
(b) Mask³.(c) Einstein (high contrast)⁴.

Зураг 3.8 Examples of logic/ mask images used in the experiments.

³Source: Digital Image Processing (Gonzalez & Woods), Chapter 3, Figure 3.26(a)

3.9 Smoothing Linear Filters

Smoothing filters нь зургийн чимээг багасгах, ирмэгийг зөөлрүүлэх зориулалттай. Янз бүрийн хэмжээтэй averaging kernel ашиглан smoothing хийсэн үр дүнг Зураг 3.9-д харуулав.



Зураг 3.9 Янз бүрийн хэмжээтэй averaging filter ашиглан smoothing хийсэн үр дүн. Kernel хэмжээ ихсэх тусам зураг илүү бүдэгрэх боловч чимээ багасдаг.

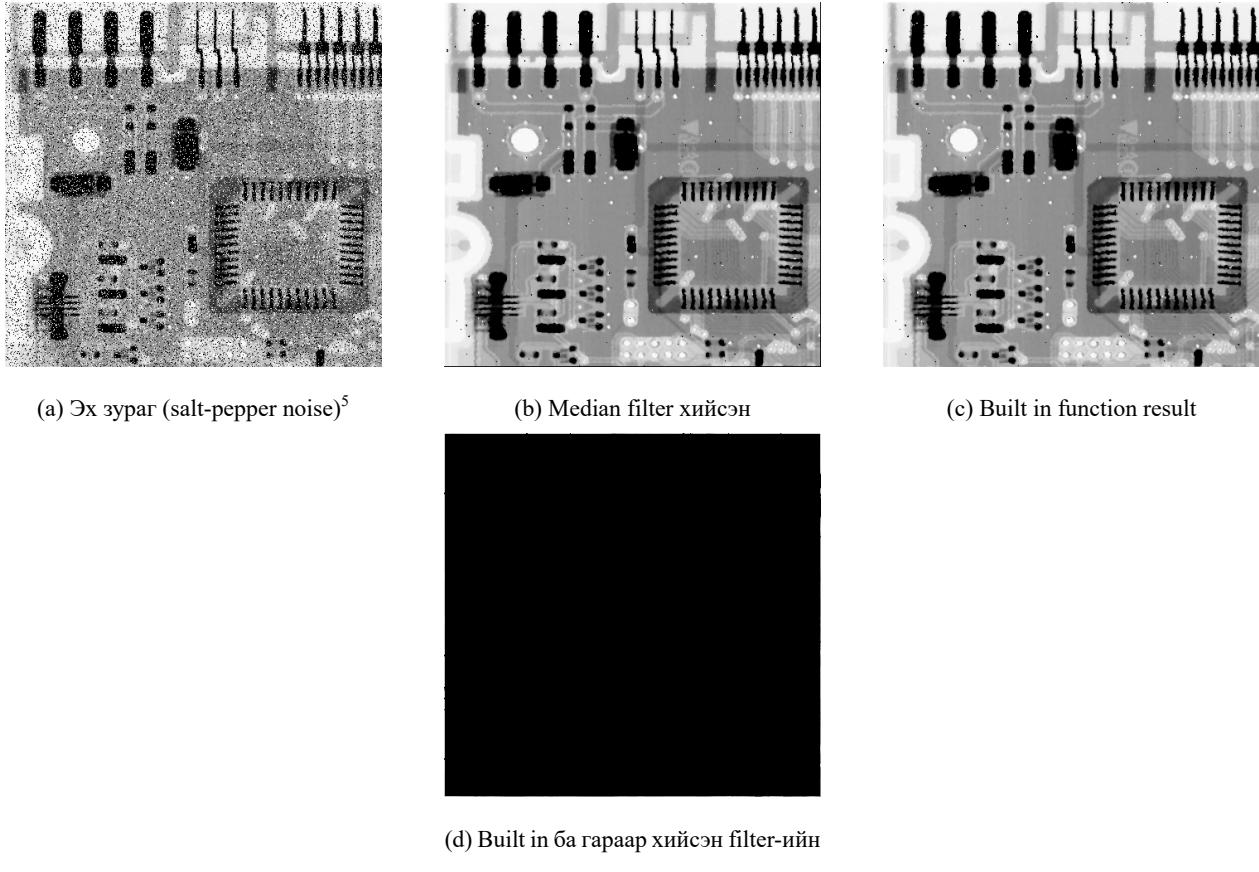
Үр дүн: Kernel хэмжээ томорхтой зэрэгцэн чимээ багасдаг боловч зургийн нарийн мэдээлэл, ирмэг алдагдах эрсдэл нэмэгддэг. Optimal хэмжээ сонгох нь хэрэглээнээс хамаарна.

⁴Source: Digital Image Processing (Gonzalez & Woods), Chapter 4, Figure 4.24(a)

⁴Source: Digital Image Processing (Gonzalez & Woods), Chapter 2, Figure 2.41(c)

3.10 Median Filter

Median filter нь salt-and-pepper noise арилгахад маш үр дүнтэй. Averaging filter-aас ялгаатай нь median утгыг ашигладаг учир ирмэгийг илүү сайн хадгалдаг. Зураг 3.10-д үр дүнг харуулав.



Зураг 3.10 Median filter ашиглан salt-and-pepper noise арилгасан үр дүн. Ирмэгүүд хадгалагдсаар чимээ бараг бүрэн устсан байна.

3.11 Sharpening using Laplacian

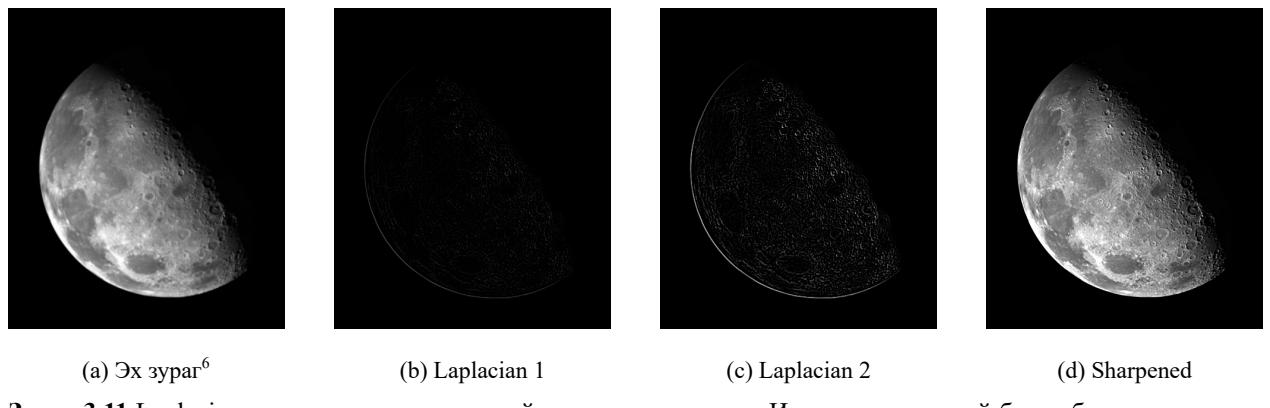
Laplacian оператор нь хоёр дахь зэргийн уламжлал ашиглан ирмэг илрүүлж зургийг хурцлах арга юм. Зураг 3.11-д янз бүрийн Laplacian kernel-ийн үр дүнг харуулав.

Үр дүн: Laplacian filter нь ирмэгийг илрүүлж, тэдгээрийг эх зурагт нэмснээр хурц, тод үр дүн гаргаж байна.

Энэ арга нь бүдэг зургийг сайжруулахад маш үр дүнтэй.

⁵Source: Digital Image Processing (Gonzalez & Woods), Chapter 3, Figure 3.35(a)

⁶Source: Digital Image Processing (Gonzalez & Woods), Chapter 3, Figure 3.38(a)



Зураг 3.11 Laplacian оператор ашиглан зургийг хурцалсан үр дүн. Ирмэгүүд тодорхой болж, бүдэг зураг илүү нарийвчилсан болсон байна.

3.12 Combined methods

Sobel оператор нь нэг дүгээр зэргийн уламжлал (gradient) ашиглан ирмэг илрүүлдэг. Зураг 3.12-д Sobel edge detection-ийн үр дүнг харуулав.

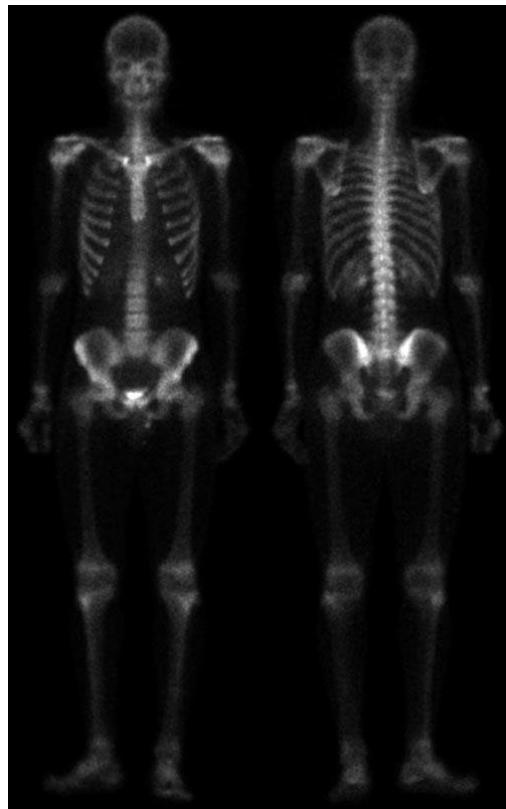
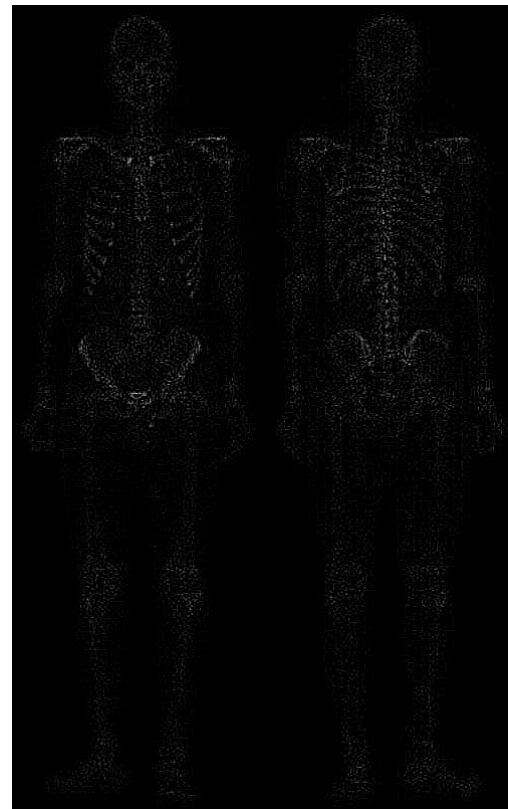
3.13 Notch Filtering

Notch filter нь давтамжийн орчинд (frequency domain) periodic noise арилгах арга юм. Fourier transform хийж, чимээний давтамжийг шүүж, inverse transform хийн цэвэр зураг гаргана. Зураг 3.13-д notch filter-ийн үр дүнг харуулав.

Үр дүн: Notch filtering нь spatial domain-д арилгах боломжгүй periodic pattern чимээг (жишээ нь scan lines, moiré patterns) маш үр дүнтэй арилгадаг.

⁷Source: Digital Image Processing (Gonzalez & Woods), Chapter 3, Figure 3.43(a)

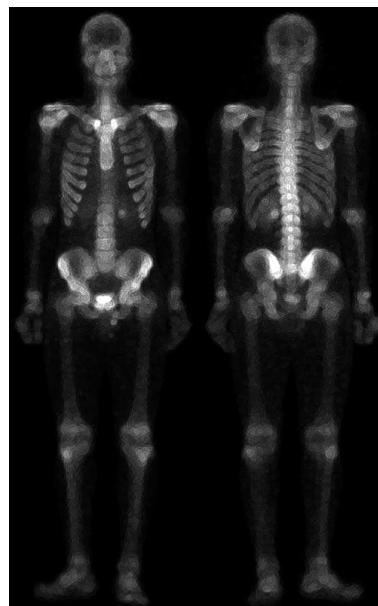
⁸Source: Digital Image Processing (Gonzalez & Woods), Chapter 4, Figure 4.64(a)

(a) Эх зураг⁷

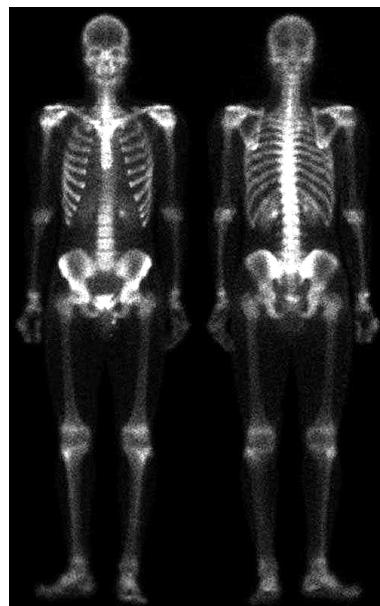
(b) Илрүүлсан ирмэг



(c) Sobel edge detection

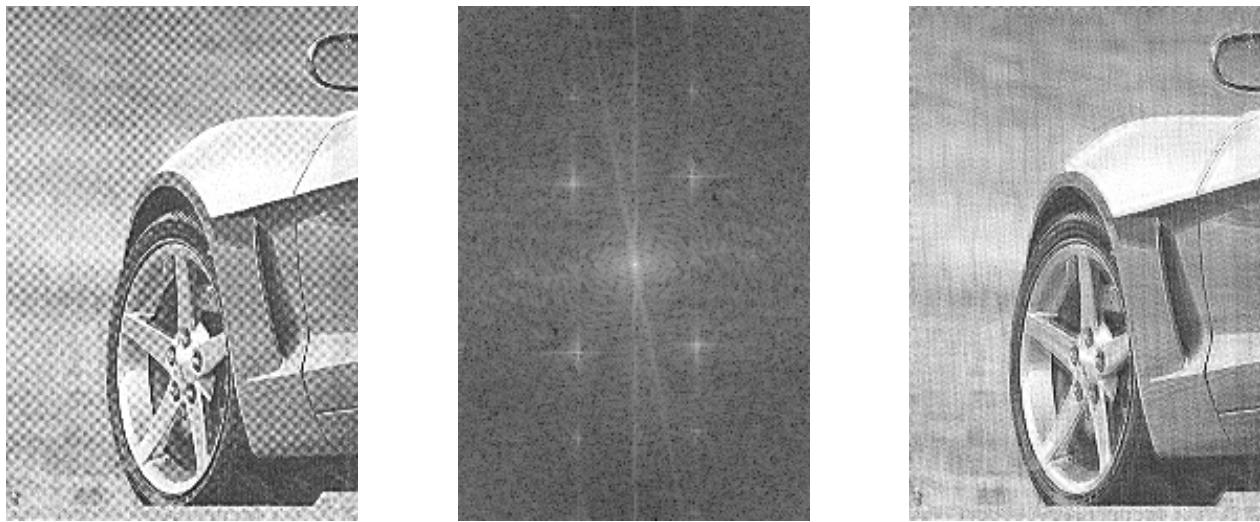


(d) Сайжруулсан зураг 1-р алхам



(e) Сайжруулсан зураг 2-р алхам

Зураг 3.12 Sobel оператор ашиглан ирмэг илрүүлсэн үр дүн. Объектын хил, хэлбэр тодорхой илэрч байна.

(a) Эх зураг (periodic noise)⁸

(b) Frequency spectrum

(c) Notch filter хийсэн

Зураг 3.13 Notch filter ашиглан periodic noise арилгасан үр дүн. Frequency domain дээр чимээний цэгүүдийг илрүүлж шүүснээр цэвэр зураг гаргаж авсан.

ДҮГНЭЛТ

Энэхүү лабораторийн ажил нь spatial болон frequency domain гэх мэт арван дөрвөн үндсэн дүрс боловсруулах техникуудийг хэрэгжүүлж хэрэглээ болон үр дүнг шинжлэв. Үүнд Python болон OpenCV ашигласан бөгөөд ихэнх онолын ойлголтуудыг өөрийн тодорхойлсон аргаар хэрэгжүүлсэн.

Spatial domain болон intensity domain-уудтай шууд харьцах нь харьцангуй энгийн байсан бөгөөд Үр дүн болон хэрэглээний хэрэгжүүлэлтэд зарцуулсан цагтай харьцуулахад маш ашигтай нь илт харагдсан.

Image interpolation техникууд нь зургийн spatial resolution сайжруулахад чухал үүрэгтэй техникууд юм. Тэдгээрээс хамгийн түгээмэл хэрэглэгдэх хоёр алгоритмуудыг хэрэгжүүлсэн. Эндээс Nearest-neighbor interpolation нь хурдан боловч чанарын хувьд сул үр дүн өгсөн бол bilinear interpolation тооцоолол ихтэй ч нь илүү нарийвчлалтай үр дүн гаргасан. Дээрх хоёр алгоритмуудыг хэрэгжүүлэх явцдаа, онолын хувьд хамгийн сайн ажиллаж болохуйц bicubic interpolation-ий тухай судалж мэдсэн боловч тус лабораторийн хүрээнд хэрэгжүүлэлт хийгээгүй болно.

Contrast stretching, gray-level slicing, bit-plane slicing зэрэг алгоритмууд нь бүгд дүрсийн тодорхой мужуудад боловсруулалт хийх техникууд. Тус алгоритмуудыг ашиглан дүрс дэх чимээг арилгаж, ашигтай мэдээллийг салгах чадвартай болохыг харуулсан. Gray-level slicing нь рентген болон хэт авианы зургийн анализад үр дүнтэй байсан бол bit-plane slicing нь steganography болон чимээг салгахад давуу талтай байв. Харин өнгө муутай дүрсийг сайжруулахад contrast stretching болон thresholding техникууд нь илүү өргөн хэрэглэгддэг, тэдгээр нь

чимээ арилгах гэхээс илүү feature тодотгох тал дээр тулгуурлан ажилладаг.

Histogram дээр тооцоолол хийх нь дүн шинжилгээний түгээмэл аргуудын нэг юм. Лабораторийн ажлыг гүйцэтгэх явцад нь зургийн тархалтыг шинжлэх болон contrast-ийг автоматаар сайжруулахад талаар хэрэглэсэн боловч histogram-ийг ашиглан дурс сайжруулах олон боломж байгааг анзаарсан. Жишээ нь: Outlier detection, image segmentation зэрэгт histogram-ийг ашиглаж болно. Мөн зөвхөн боловсруулалтаар хязгаарлагдахгүйгээр машин сургалтад feature extraction хийхэд чухал үүрэгтэй байж болохыг судалж мэдсэн.

Шүүлтийн аргуудын хувьд median filter нь salt-pepper чимээг арилгахад linear averaging filter-ээс илүү үр дүнтэй байсан бөгөөд ирмэгийг сайн хадгалдаг. Smoothing linear filters нь бүрсгэр буюу Гауссын чимээг багасгахад тохиromжтой байсан. Хэрэгжүүлэх явцад linear averaging kernel томрох тусам тооцоолол хийх хугацаа маш хурдтай өсөж буй нь анзаарагдсан. Харин median filter нь хэрэгжүүлэлтийг OpenCV сангийн хэрэгжүүлэлтэй жишиж үзсэн бөгөөд үр дүнгийн ялгаа үл мэдэгдэхүйц хэмжээд илэрсэн.

Sharpening техникуудийн хувьд Laplacian оператор нь ирмэгийг үр дүнтэй илрүүлж, unsharp masking болон high-boost filtering нь тухайн орчны хувьд contrast-ийг сайжруулахад илүү давуу тал олгосон. Эдгээр аргуудыг хослуулнаар нэг аргын сул талыг нөхөж, илүү чанартай үр дүн гаргаж чадсан.

Notch filters нь Histogram дээрх тооцоололтой санааны хувьд маш төстэй арга гэдгийг анзаарсан. Histogram нь энгийн пикселийн утга тоолох аргаар domain үүсгэж байсан бол Notch Filter нь Fourier transform ашиглан давтамжийн domain үүсгэж, чимээний эх үүсвэрийг шууд тодорхойлж арилгах аргаар ажилладгаараа онцлогтой. Notch filter нь тодорхой давтамж бүхий чимээг чимээг арилгахад онцгой үр дүнтэй байсан бөгөөд spatial domain-ийн аргуудаар хийхэд хүндрэлтэй тодорхой давтамжийн чимээг зайлцуулах боломжийг олгосон.

Судалгаа хийх явцад ”хамгийн сайн” алгоритм байхгүй гэдэг нь илт харагдаж байсан. Ихэнх судалгааны хүрээнд багтсан бүх алгоритмуудад хэрэгжүүлэлтийн энгийн болон нарийвчлалтайгаас үл хамааран хэрэглэгдэж болох тохиолдлууд баагүй байсан. Алгоритм сонгох нь зорилго тодорхойлолт, зургийн төрөл, чимээний шинж чанар, тодорхой хэрэглээний шаардлагаас хамааран хувьсах ёстой үйл явц байх шаардлагатай.

Энэхүү лабораторийн ажил нь дурс боловсруулах үндсэн техникуудийн онол, практик хэрэгжүүлэлтийг нэгтгэн, өөр өөр хэрэглээнд тохирсон арга сонгох чадварыг хөгжүүлэхэд чухал суурь болсон.

Bibliography

- [1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2008.
- [2] R. C. Gonzalez and R. E. Woods, "DIP3E Book Images," *The Image Processing Place*. [Online]. Available: https://www.imageprocessingplace.com/DIP-3E/dip3e_book_images_downloads.htm
- [3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
- [4] "OpenCV Documentation," OpenCV. [Online]. Available: <https://docs.opencv.org/>
- [5] "Interpolation (scipy.interpolate)," SciPy Documentation. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/interpolate.html>
- [6] "Histogram Equalization," OpenCV Tutorials. [Online]. Available: https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html
- [7] "Image Filtering," OpenCV Documentation. [Online]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
- [8] "Image Sharpening," SciKit-Image Documentation. [Online]. Available: https://scikit-image.org/docs/stable/auto_examples/filters/plot_unsharp_mask.html
- [9] "Fourier Transform," OpenCV Tutorials. [Online]. Available: https://docs.opencv.org/4.x/de/dbc/tutorial_py_fourier_transform.html
- [10] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
- [11] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [12] "Image Kernels Explained Visually," Setosa.io. [Online]. Available: <https://setosa.io/ev/image-kernels/>