OpenCV 없이 구현한 C++ 영상처리

[Intel] 엣지 AI SW 아카데미
객체지향 프로그래밍

허찬욱

프로젝트 개요

프로젝트 목표

이미지 파일을 C++로 구현한
알고리즘으로 보정하기

프로젝트 기간

2024.03.25 ~ 2024.04.04

프로그램명

Color Image Processing
(Version 2.0)

주요 기능

화소점 처리, 기하학 처리, 화소영역 처리,
히스토그램 처리, HSI 모델

개발 환경

Windows 10 Pro 64bit
Visual Studio 2022

전체 소스코드 링크

https://blog.naver.com/hew0916/

특이사항

다양한 형식의 파일로 저장 가능
자율 비율 & Color Image 사용

구조도

D:/Mypic

mypet.jpg
(300X400)

out1.png
(300x400)

OnOpenDocument()

CImage

CImage

OnSaveDocument()

메모리(RAM)

m_inW

m_inH

m_inImageR
m_inImageG
m_inImageB

영상처리 알고리즘

m_outH

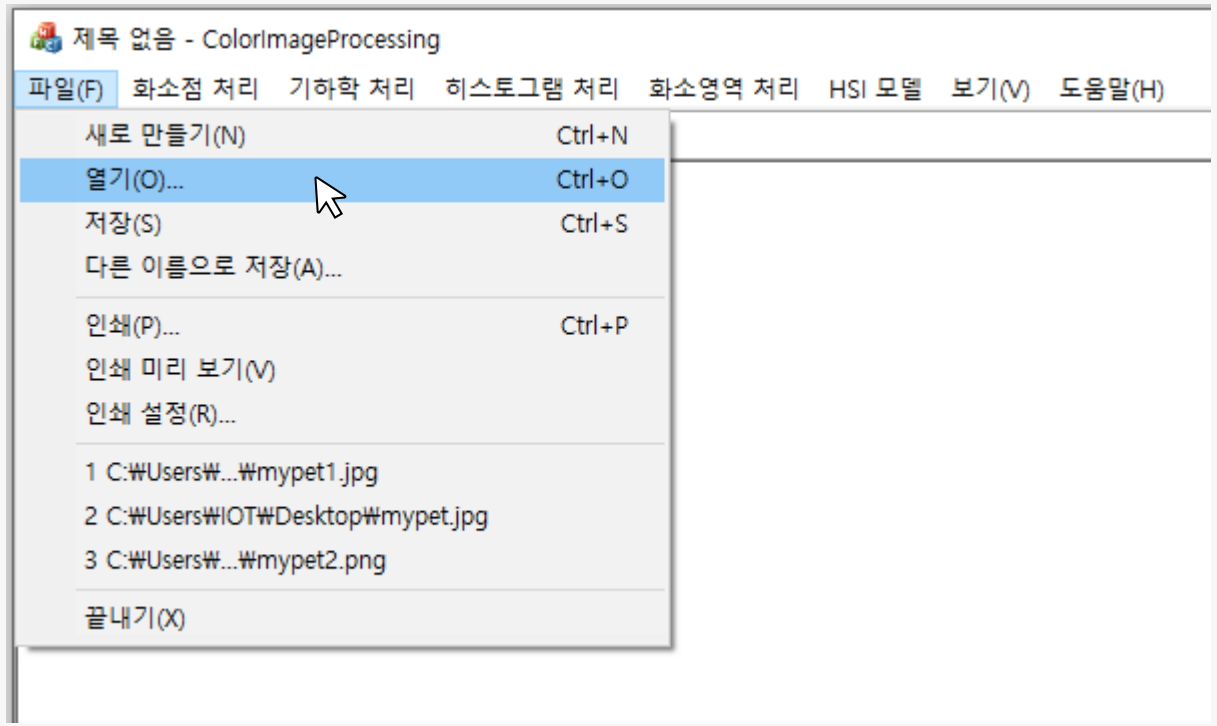
m_outImageR
m_outImageG
m_outImageB

m_outW

OnDraw()

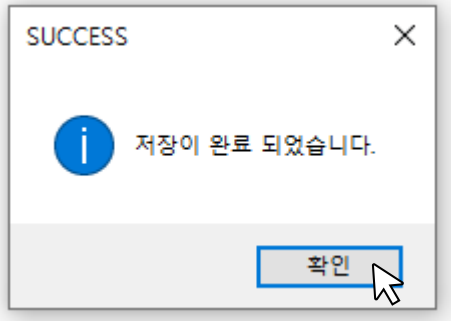
Display

메인화면 구성



[파일]-[열기]를 누른 후 이미지 파일을 선택하면,
해당 이미지가 Print 됨

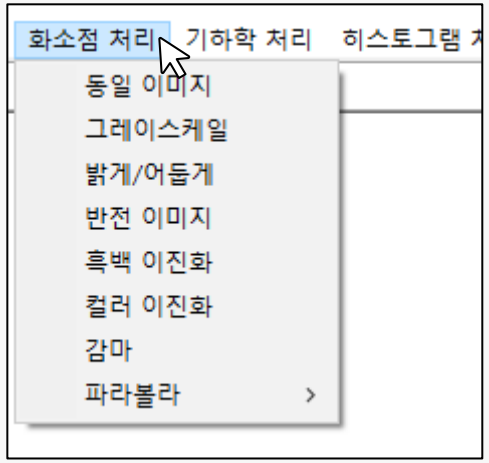
저장 알림 창



영상처리 알고리즘을 활용해 이미지 편집 후
[파일] – [저장]을 눌러 원하는 곳에
원하는 파일 형식으로 저장 가능

1. 화소점 처리

- 원 화소의 값이나 위치를 바탕으로 단일 화소값을 변경하는 기술
- 종류 :
  - 동일
  - 그레이스케일
  - 밝게/어둡게
  - 반전
  - 흑백 이진화
  - 칼라 이진화
  - 감마
  - 파라볼라(CAP/CUP)



동일

원본 이미지

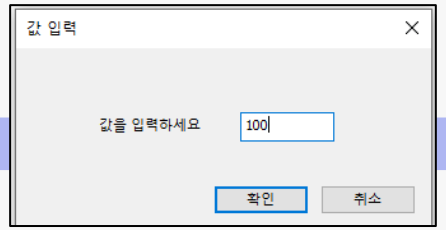
OnEqualImage()

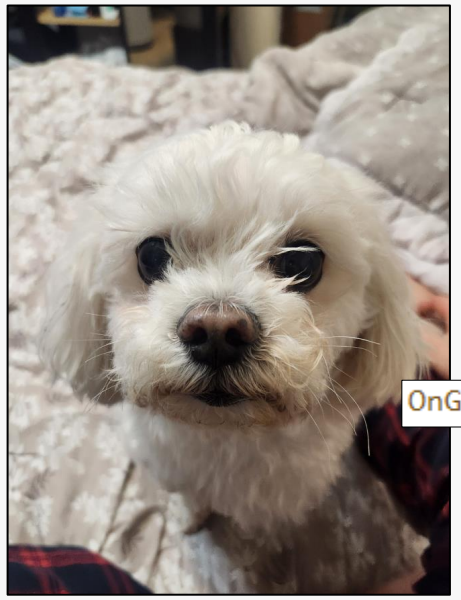
출력 이미지

```
m_outImageR[i][k] = m_inImageR[i][k];
m_outImageG[i][k] = m_inImageG[i][k];
m_outImageB[i][k] = m_inImageB[i][k];
```

1. 화소점 처리

OnAddImage()

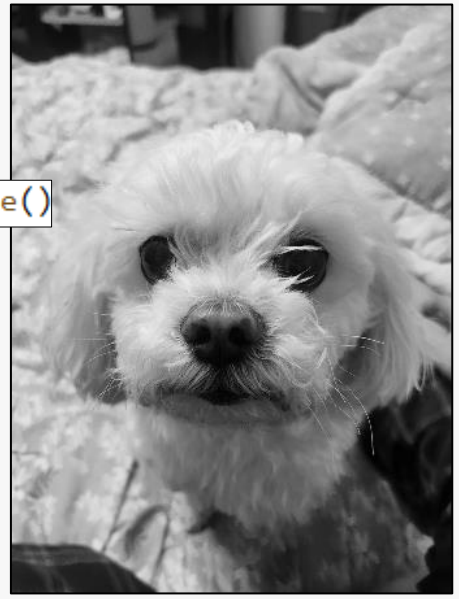




원본 이미지

OnGrayScale()

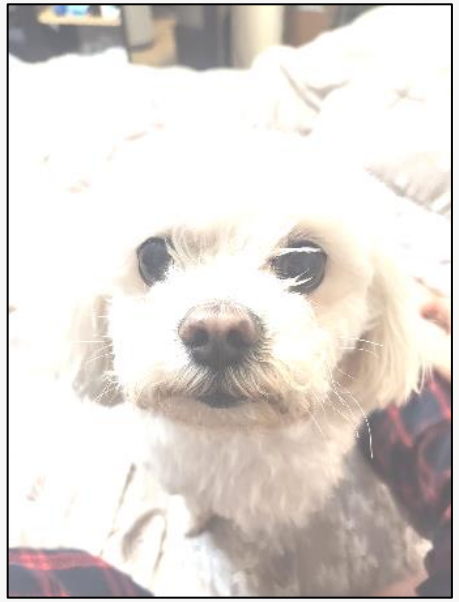
그레이스케일



출력 이미지

```
avg=(m_inImageR[i][k]+m_inImageG[i][k]+m_inImageB[i][k])/3.0;
m_outImageR[i][k]=m_outImageG[i][k]=m_outImageB[i][k]=(unsigned char)avg;
```

밝게(+100)



출력 이미지

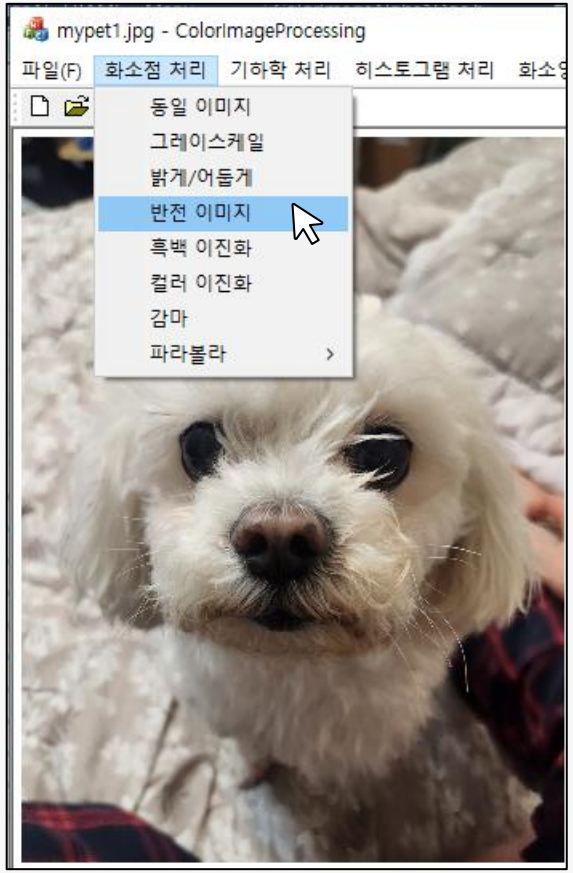
어둡게(-100)

출력 이미지

```
if (m_inImageR[i][k] + valueR > 255) m_outImageR[i][k] = 255;
else if (m_inImageR[i][k] + valueR < 0)m_outImageR[i][k] = 0;
else m_outImageR[i][k] = m_inImageR[i][k] + valueR;
```

//G, B 동일하게 반복

1. 화소점 처리



원본 이미지

반전

OnReverseImage()

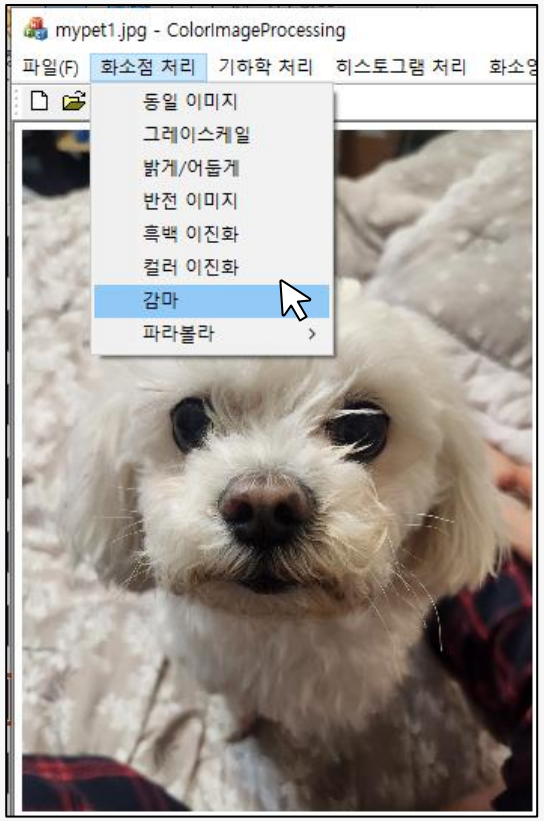
```
m_outImageR[i][k] = 255 - m_inImageR[i][k];
m_outImageG[i][k] = 255 - m_inImageG[i][k];
m_outImageB[i][k] = 255 - m_inImageB[i][k];
```

흑백

OnBwImage()

```
if (avg > 127) {
    m_outImageR[i][k] = 255;
    m_outImageG[i][k] = 255;
    m_outImageB[i][k] = 255;
}
else if (avg <= 127) {
    m_outImageR[i][k] = 0;
    m_outImageG[i][k] = 0;
    m_outImageB[i][k] = 0;
}
```

1. 화소점 처리

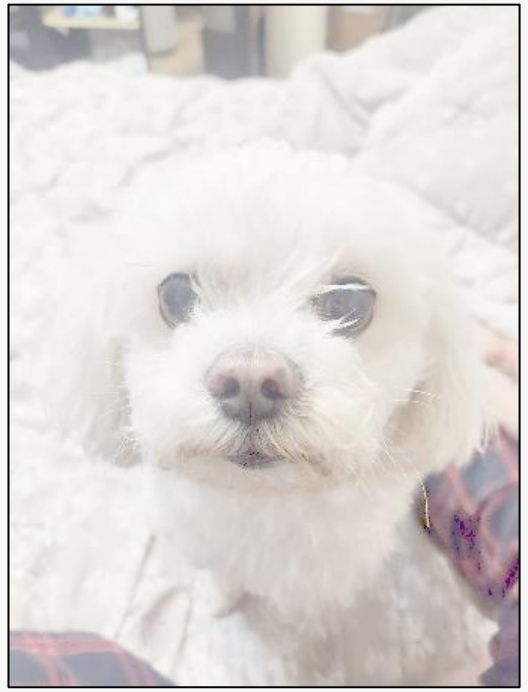


원본 이미지

OnGammaImage()

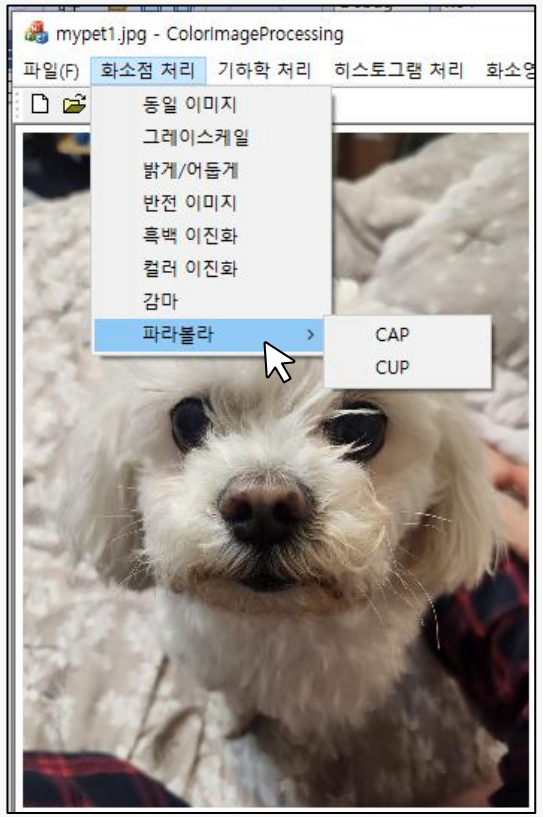
감마(1.8입력)

감마(0.2입력)



```
// pow(a,b) ; a의 b 제곱
m_outImageR[i][k] = (double)255.0 * pow(m_inImageR[i][k] / 255.0, value);
m_outImageG[i][k] = (double)255.0 * pow(m_inImageG[i][k] / 255.0, value);
m_outImageB[i][k] = (double)255.0 * pow(m_inImageB[i][k] / 255.0, value);
```

1. 화소점 처리



원본 이미지

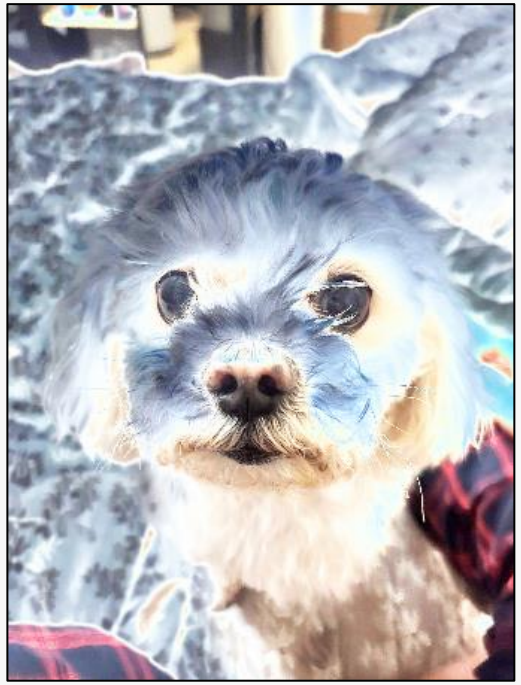
OnCapImage()

파라볼라 (CAP)

```
LUT[i] = 255.0 * pow((i / 127.0 - 1), 2);
```

OnCupImage()

파라볼라 (CUP)



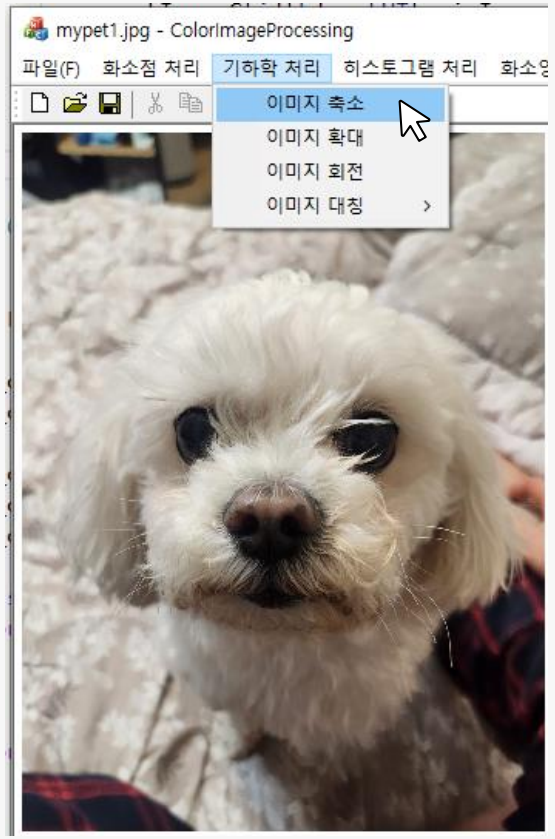
```
LUT[i] = -255.0 * pow((i / 127.0 - 1), 2) + 255;
```

```
m_outImageR[i][k] = LUT[m_inImageR[i][k]];
m_outImageG[i][k] = LUT[m_inImageG[i][k]];
m_outImageB[i][k] = LUT[m_inImageB[i][k]];
```

2. 기하학 처리

- 영상을 구성하는 화소의 공간적 위치나 배열을 재배치 하는 기술
- 종류 :
  - 축소
  - 확대
  - 회전
  - 대칭

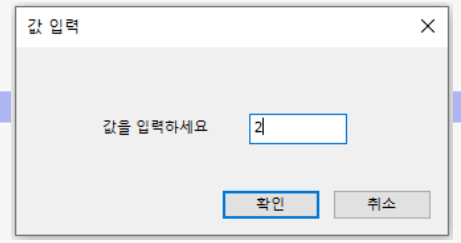
  (좌우,상하,상하좌우)



원본 이미지(300x400)

축소

```
OnZoomoutImage()
```

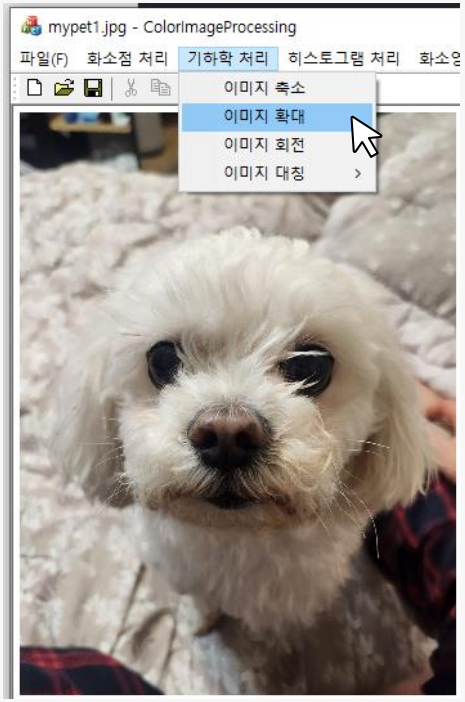


출력 이미지(150x200)

```
int valAngle = (int)dlg.m_constant;
m_outH = m_inH / valAngle;
m_outW = m_inW / valAngle;
```

```
m_outImageR[i][k] = m_inImageR[i * valAngle][k * valAngle];
m_outImageG[i][k] = m_inImageG[i * valAngle][k * valAngle];
m_outImageB[i][k] = m_inImageB[i * valAngle][k * valAngle];
```

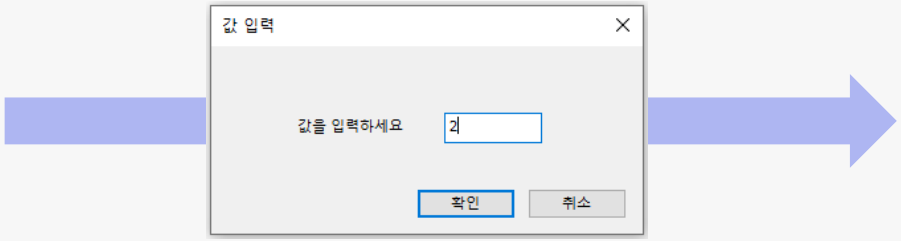
2. 기하학 처리



원본 이미지(300x400)

확대

```
OnZoominImage()
```

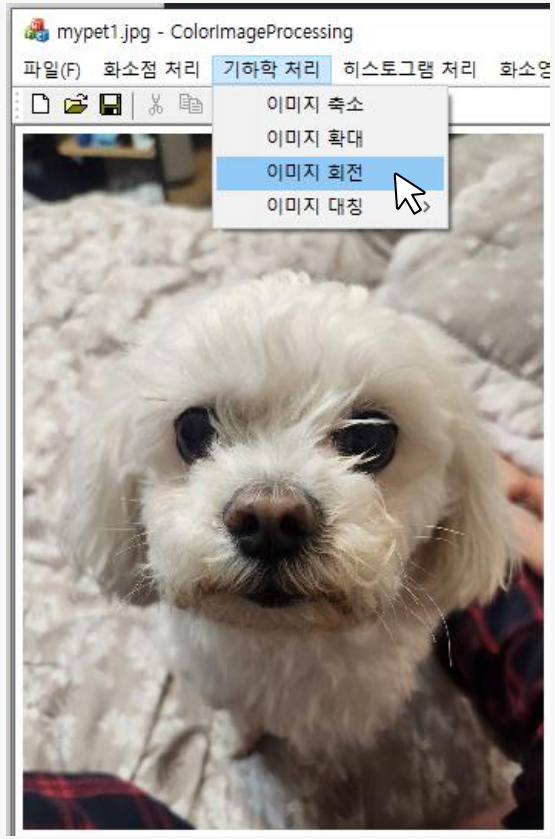



```
int valAngle = (int)dlg.m_constant;
m_outH = m_inH * valAngle;
m_outW = m_inW * valAngle;
```

```
m_outImageR[i][k] = m_inImageR[i / valAngle][k / valAngle];
m_outImageG[i][k] = m_inImageG[i / valAngle][k / valAngle];
m_outImageB[i][k] = m_inImageB[i / valAngle][k / valAngle];
```

출력 이미지(600x800)

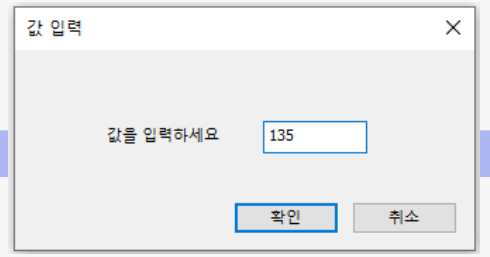
2. 기하학 처리



원본 이미지

회전

OnRotateImage()



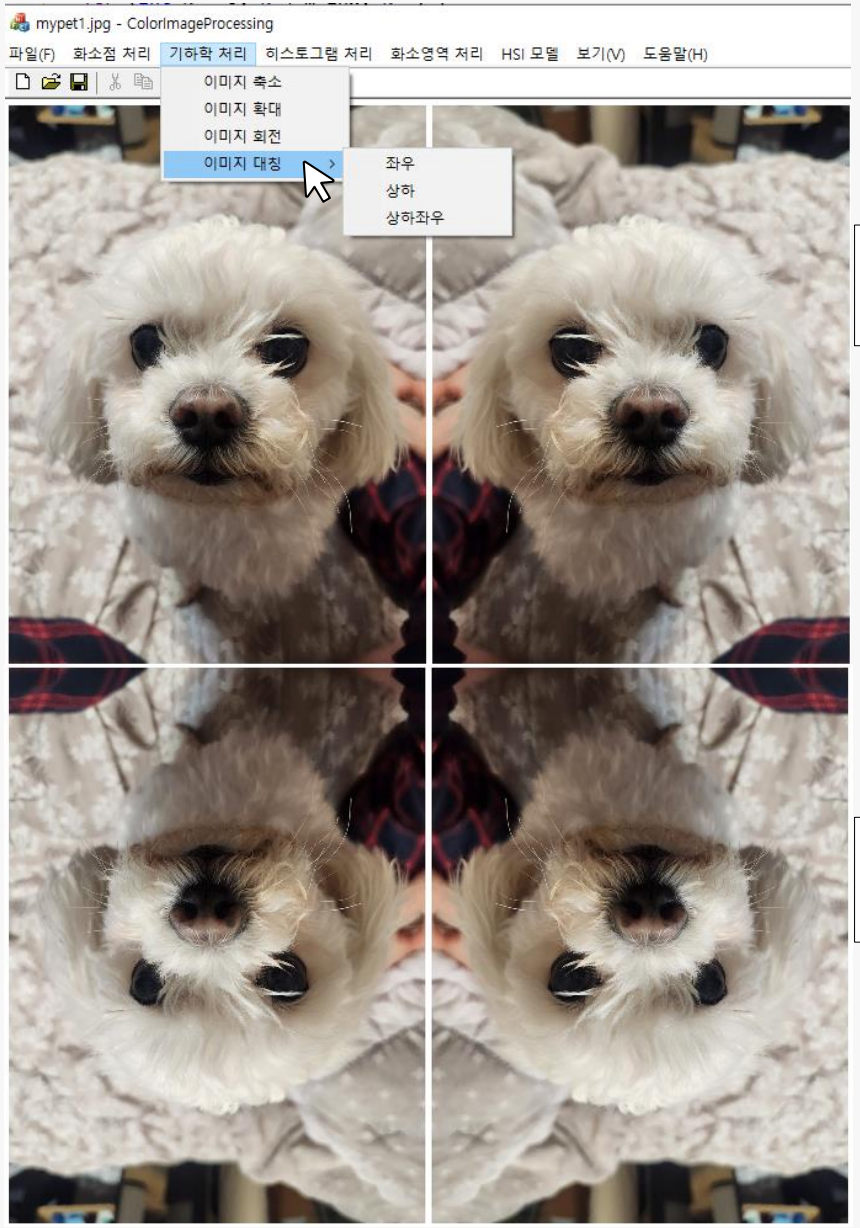

```
int degree = dlg.m_constant;
double radian = degree * 3.141592 / 180.0;
```

```
int cx = m_inH / 2;
int cy = m_inW / 2;
for (int i = 0; i < m_outH; i++) {
    for (int k = 0; k < m_outW; k++) {
        int xd = i;
        int yd = k;

        int xs = (int)(cos(radian) * (xd - cx) + sin(radian) * (yd - cy));
        int ys = (int)(-sin(radian) * (xd - cx) + cos(radian) * (yd - cy));
        xs += cx;
        ys += cy;

        if ((0 <= xs && xs < m_outH) && (0 <= ys && ys < m_outW)) {
            m_outImageR[xd][yd] = m_inImageR[xs][ys];
            m_outImageG[xd][yd] = m_inImageG[xs][ys];
            m_outImageB[xd][yd] = m_inImageB[xs][ys];
        }
    }
}
```

출력 이미지

2. 기하학 처리

원본 이미지



좌우반전

OnLeftrightImage()

```
m_outImageR[i][m_inW-1-k]=m_inImageR[i][k];
m_outImageG[i][m_inW-1-k]=m_inImageG[i][k];
m_outImageB[i][m_inW-1-k]=m_inImageB[i][k];
```

상하반전

OnUpdownImage()

```
m_outImageR[m_inH-i-1][k]=m_inImageR[i][k];
m_outImageG[m_inH-i-1][k]=m_inImageG[i][k];
m_outImageB[m_inH-i-1][k]=m_inImageB[i][k];
```

상하좌우반전

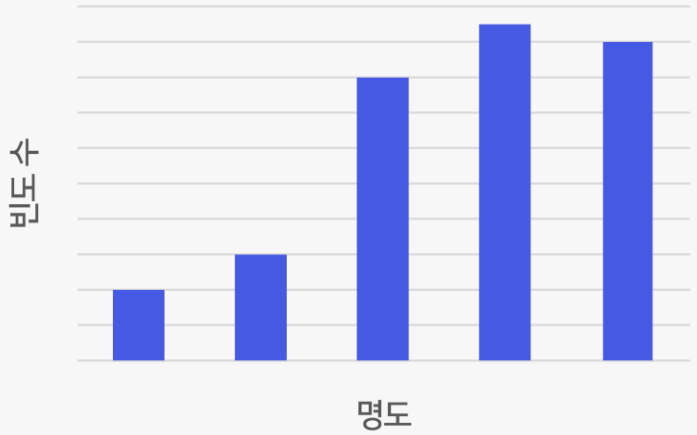
OnUpdownleftrightImage()

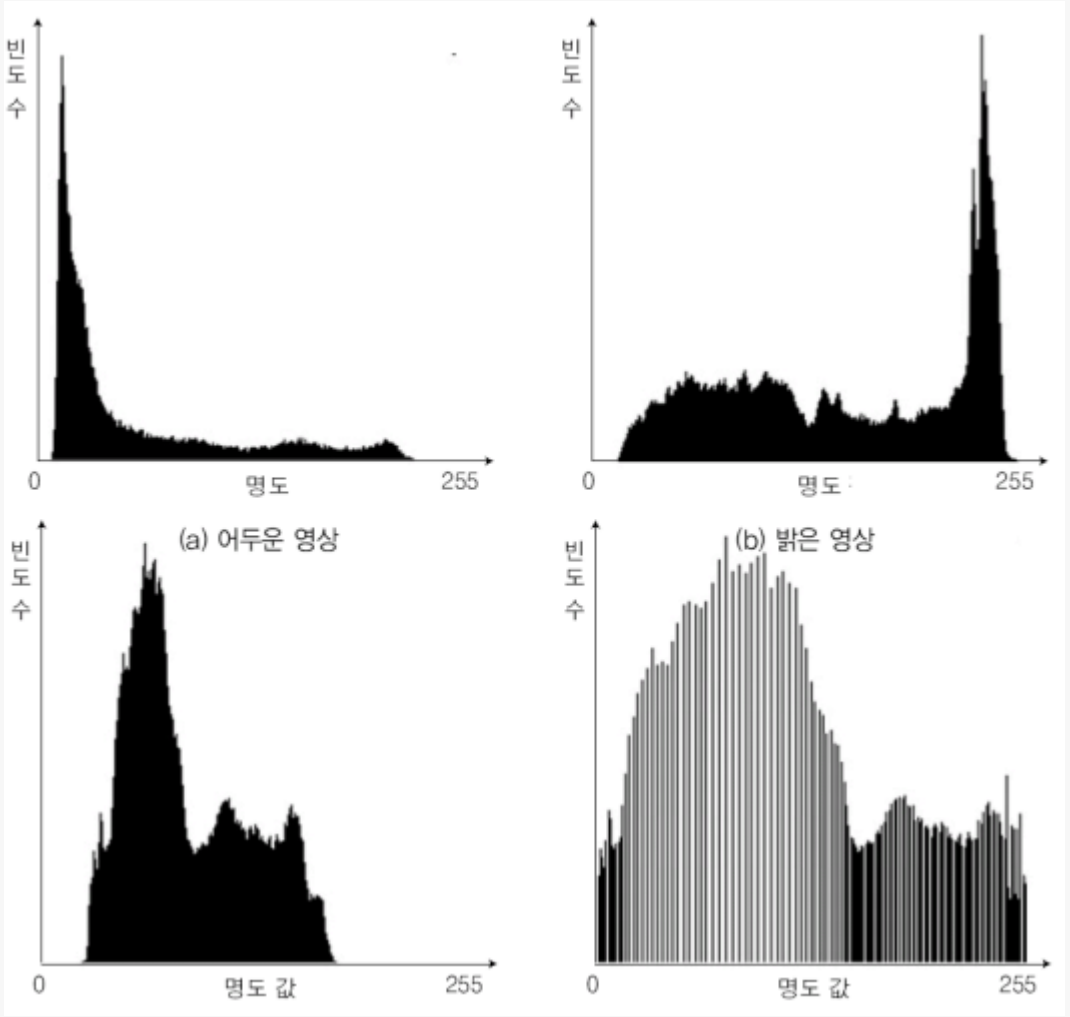
```
m_outImageR[m_inH-i-1][m_inW-1-k]=m_inImageR[i][k];
m_outImageG[m_inH-i-1][m_inW-1-k]=m_inImageG[i][k];
m_outImageB[m_inH-i-1][m_inW-1-k]=m_inImageB[i][k];
```

3. 히스토그램이란?

- 표로 되어 있는 분포데이터를 막대 그래프로 나타낸 것
- 종류 :
  - 스트래칭
  - 엔드-인
  - 평활화

6	6	6	7
4	5	5	3
2	1	1	3
0	0	1	3

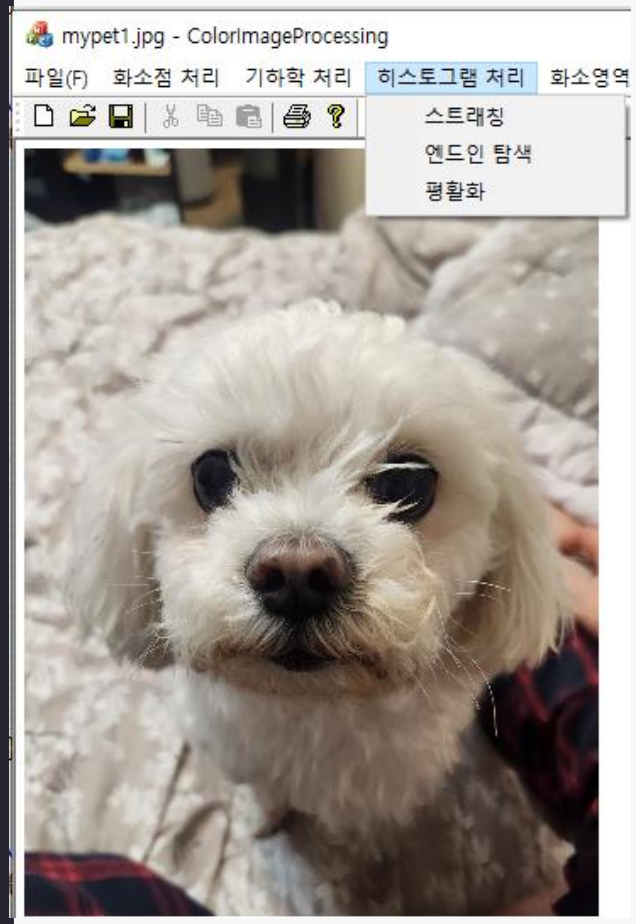




(C)명암 대비가 낮은 영상 (d)명암 대비가 높은 영상

3. 히스토그램 처리

- 원본 비교



원본 이미지

스트래칭

출력 이미지

엔드-인

출력 이미지

평활화

출력 이미지

3. 히스토그램 처리

- 스트래칭

- 히스토그램을 모든 영역으로 확장시켜 영상의 모든 범위의 화소값을 포함

$$outImage = \frac{inImage - low}{high - low} \times 255$$

OnStretchImage()

원본 이미지

출력 이미지

```
double outR, inR;
double outG, inG;
double outB, inB;

for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        inR = m_inImageR[i][k];
        outR = ((inR - lowR) / (double)(highR - lowR)) * 255.0;
        m_outImageR[i][k] = (unsigned char)outR;

        inG = m_inImageG[i][k];
        outG = ((inG - lowG) / (double)(highG - lowG)) * 255.0;
        m_outImageG[i][k] = (unsigned char)outG;

        inB = m_inImageB[i][k];
        outB = ((inB - lowB) / (double)(highB - lowB)) * 255.0;
        m_outImageB[i][k] = (unsigned char)outB;
    }
}
```

3. 히스토그램 처리

- 엔드-인 탐색

- 명암 대비 스트레칭에서 일정한 양의 화소를 흑백으로 지정하여 히스토그램의 분포를 더 균일하게 만드는 방법

원본 이미지

출력 이미지

$$outImage = \begin{cases} 0 & inImage \le low \\ \dfrac{inImage - low}{high - low} \times 255 & low \le inImage \le high \\ 255 & high \le inImage \end{cases}$$

OnEndinImage()

```
highR -= 50; lowR += 50;
highG -= 50; lowG += 50;
highB -= 50; lowB += 50;
```

//스트래칭에서 이부분만 추가

```
inR = m_inImageR[i][k];
outR = ((inR - lowR) / (double)(highR - lowR)) * 255.0;
if (outR > 255.0)
    outR = 255.0;
else if (outR < 0)
    outR = 0;
m_outImageR[i][k] = (unsigned char)outR;
```

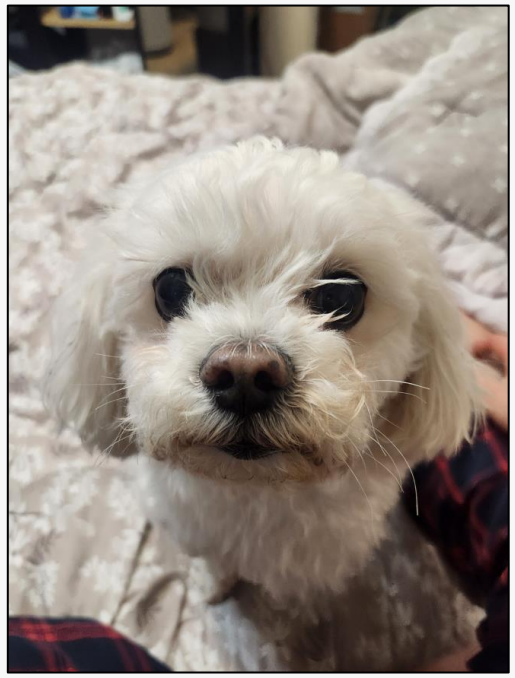
//G, B 동일하게 반복

3. 히스토그램 처리

- 평활화

- 어둡게 촬영된 영상의 히스토그램을 조절하여 명암 분포가 빈약한 영상을 균일하게 만드는 방법

$$sum[i] = \sum_{j=0}^{i} hist[j] \qquad n[i] = sum[i] \times \frac{i}{N(\text{총화소수})} \times I_{\max}(\text{최대 명도값})$$



원본 이미지

출력 이미지

OnHistoequalImage()

```
// 1단계 : 빈도수 (hist[]) 배열 완성
int histR[256] = { 0, };
```

//G, B 동일하게 반복

```
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        histR[m_inImageR[i][k]]++;
```

```
// 2단계 : 누적 히스토그램
int sumHistR[256] = { 0, }; sumHistR[0] = histR[0];
```

```
for (int i = 1; i < 256; i++) {
    sumHistR[i] = sumHistR[i - 1] + histR[i];
```

```
// 3단계 : 정규화된 히스토그램
double normalHistoR[256];
```

```
for (int i = 0; i < 256; i++) {
    normalHistoR[i] = sumHistR[i] * (1.0 / (m_inH * m_inW)) * 255.0;
```

```
// 4단계 : inImage를 정규화된 값으로 치환.
```

```
m_outImageR[i][k] = (unsigned char)normalHistoR[m_inImageR[i][k]];
```

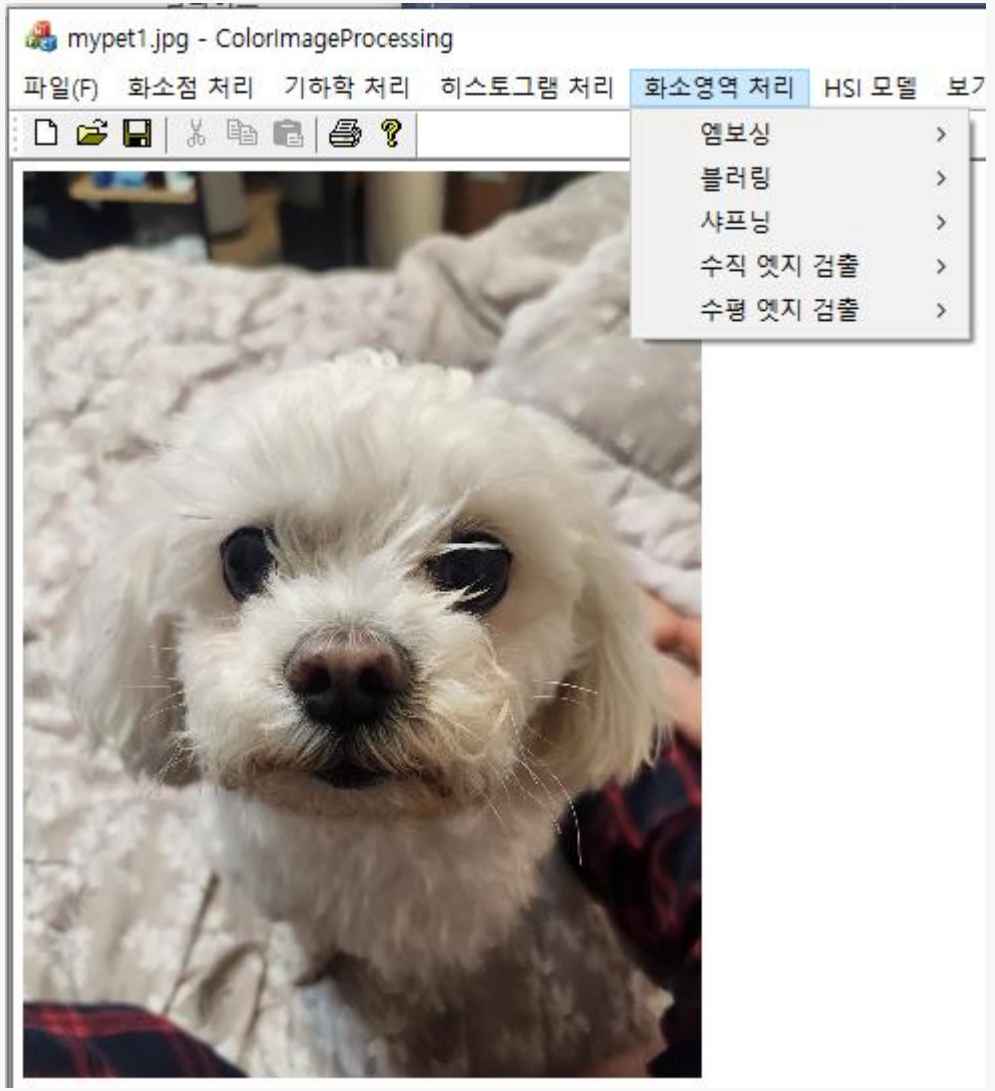
4. 화소영역 처리란?

- 화소점 처리(입력 화소) + 주변 화소 값도 고려하는 공간 영역 연산
- 회선 처리 또는 컨벌루션 처리(Convolution Processing)라고 함

$$Output\_pixel[x,y] = \sum_{m=(x-k)}^{x+k} \sum_{n=(y-k)}^{y+k} (I[m,n] \times M[m,n])$$

- **Output_pixel[x, y]: 회선 처리로 출력한 화소**
- **I[m, n]: 입력 영상의 화소**
- **M[m, n]: 입력 영상의 화소에 대응하는 가중치**

- 원시 화소와 이웃한 각 화소에 가중치를 곱한 합을 출력 화소로 생성
- 종류:
  - 엠보싱(RGB, HSI)
  - 블러링(RGB, HSI)
  - 샤프닝(RGB, HSI)
  - 경계선 검출(수직 엣지, 수평 엣지)



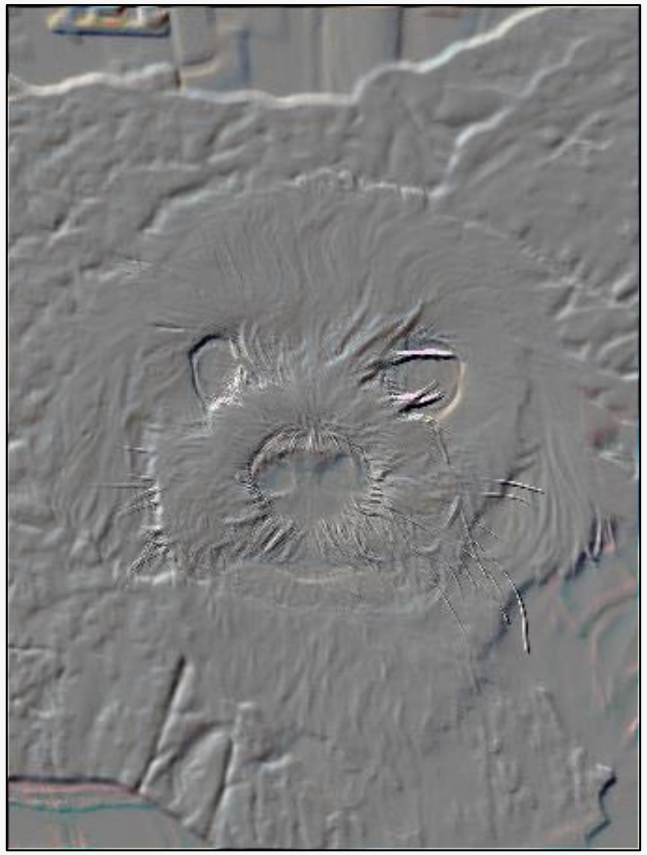
4. 화소영역 처리

- 엠보싱 (Embossing) 효과

- 입력 영상을 양각 형태로 보이게 하는 기술

원본 이미지

RGB 엠보싱



OnRgbEmbos()

HSI 엠보싱

OnHsiEmbos()

```
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {  // 엠보싱 마스크
    { -1.0, 0.0, 0.0 },
    {  0.0, 0.0, 0.0 },
    {  0.0, 0.0, 1.0 } };
```

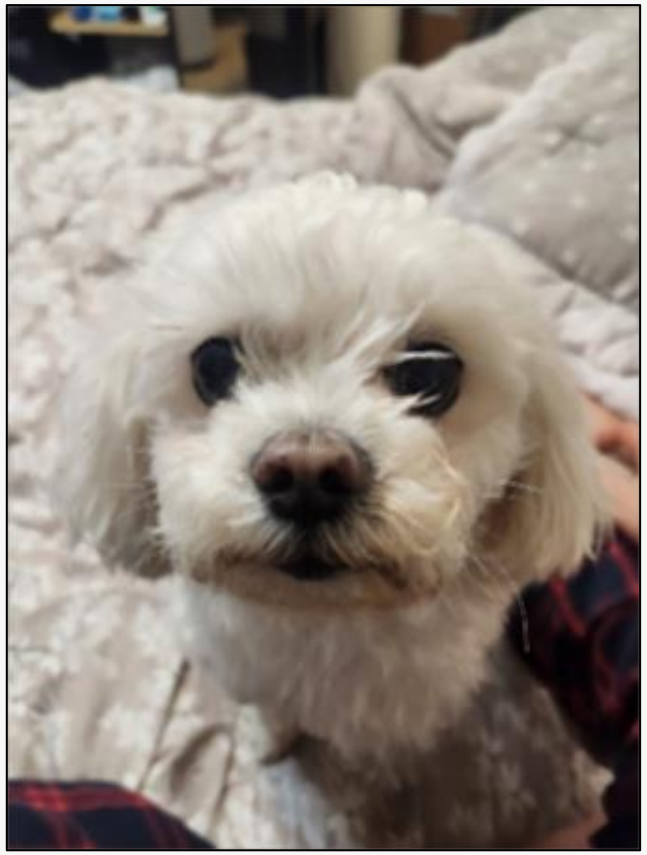
4. 화소영역 처리

- 블러링 (blurring) 효과

- 영상의 세밀한 부분을 제거하여 흐리게 하거나 부드럽게 하는 기술

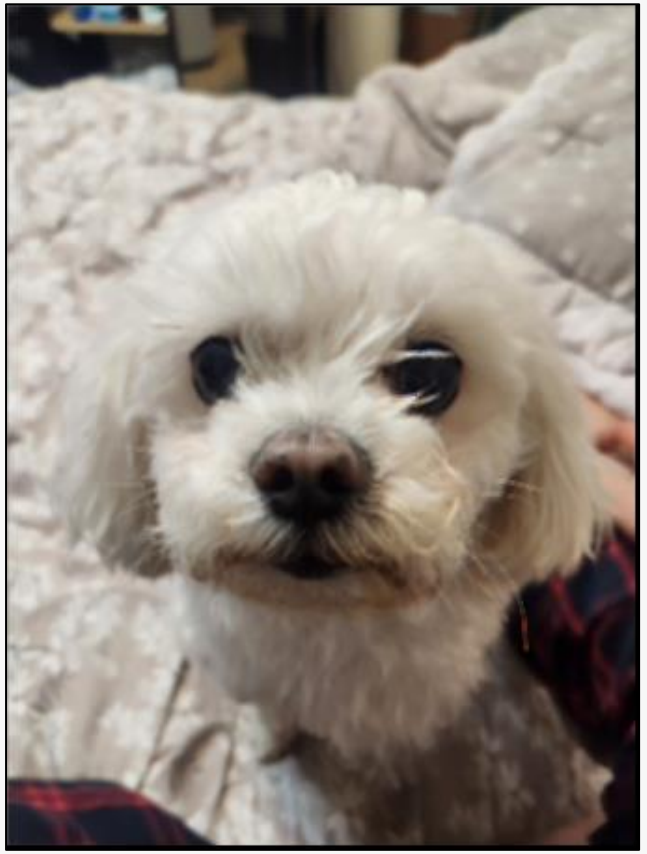
원본 이미지

RGB 블러링



OnRgbBlur()

HSI 블러링



OnHsiBlur()

```
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
    { 1 / 9., 1 / 9.,  1 / 9. },
    { 1 / 9., 1 / 9.,  1 / 9. },
    { 1 / 9., 1 / 9.,  1 / 9. }
};
```

4. 화소영역 처리

- 샤프닝 (Sharpening) 효과

- 블러링과는 반대로 디지털 영상에서 상세한 부분을 더욱 강조하여 표현하는 기술

원본 이미지

RGB 샤프닝

OnRgbSharp()

HSI 샤프닝

OnHsiSharp()

```
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
    { -1.0,  -1.0,  -1.0 },
    { -1.0,   9.0,  -1.0 },
    { -1.0,  -1.0,  -1.0 },
};
```

4. 화소영역 처리

- 경계선 검출

- 디지털 영상의 밝기가 낮은 값에서 높은 값으로, 또는 반대로 변하는 지점을 검출하는 과정

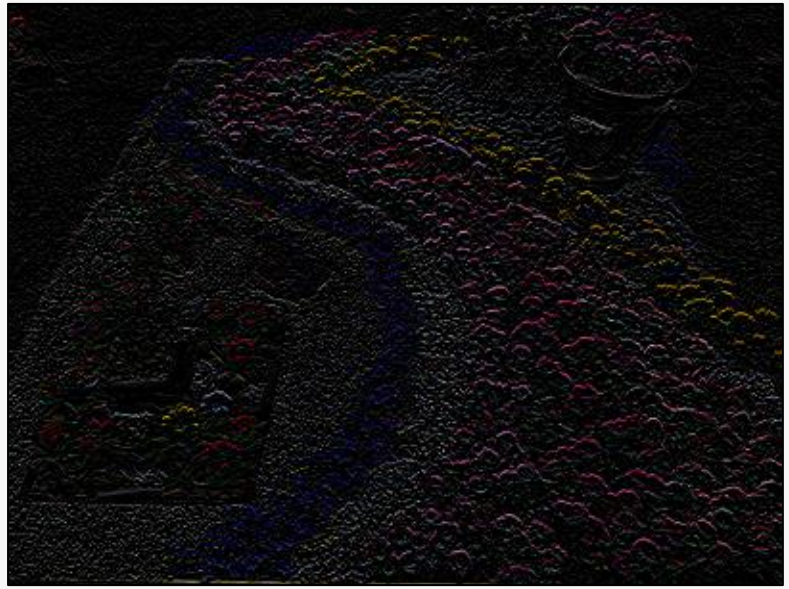
원본 이미지

수직 엣지 검출

OnRgbEdgeH()

```
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
        { 0. , 0. , 0. },
        {-1.0, 1.0, 0. },
        { 0.,  0. , 0. }
};
```

수평 엣지 검출



OnRgbEdgeL()

```
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
    {0., -1.0,  0. },
    {0.,  1.0,  0. },
    {0.,  0. ,  0. }
};
```

향후 발전 방향

- 더욱 다양한 영상처리 알고리즘 생성
- 효과 누적 및 취소(CtrI+Z) 기능 추가
- 단축키 기능 추가
- 인쇄 기능 추가
- 파일 이름 및 해상도 표시 기능 추가