OpenCV-Image Filtering

김성영교수
금오공과대학교
컴퓨터공학과

학습 내용

- 픽셀 처리 기반의 Image filtering 구현
- 라이브러리 함수 기반의 Image filtering 구현
- 칼라 영상에 대한 영상처리 방법

픽셀 처리기반의 image filtering 구현

- Mean filtering (3 x 3 mask 사용)

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

void meanFiltering(const Mat &image, Mat &result);

int main(void)
{
   Mat image = imread( "trees_gray.jpg", 0 );
   if( image.data == NULL ) return -1;

   Mat result;
   meanFiltering( image, result );

   // Display the images
   namedWindow( "Image" );
   namedWindow( "Result" );
   imshow( "Image", image );
   imshow( "Result", result );

   waitKey();

   return 0;
}
```

```
void meanFiltering( const Mat &image, Mat &result ) {
    // allocate if necessary
   result.create( image.size(), image.type() );
   for( int r=1; r<image.rows-1; r++ ) {
      const uchar* previous = image.ptr<const uchar>( r-1 );
      const uchar* current = image.ptr<const uchar>( r );
      const uchar* next = image.ptr<const uchar>( r+1 );
      uchar* output = result.ptr<uchar>( r );
      for( int c=1; c<image.cols-1; c++ ) {
         int sumP=0, sumC=0, sumN=0;
         for( int m=c-1; m<=c+1; m++ ) {
            sumP += previous[m];
            sumC += current[m];
            sumN += next[m];
         }
         *output++ = saturate_cast<uchar>(
                           (sumP+sumC+sumN)/9.+0.5 );
      }
   }
```

```
    // Set the unprocess pixels to 0
    result.row(0).setTo( Scalar(0) );
    result.row( result.rows-1 ).setTo( Scalar(0) );
    result.col(0).setTo( Scalar(0) );
    result.col( result.cols-1 ).setTo( Scalar(0) );
}
```

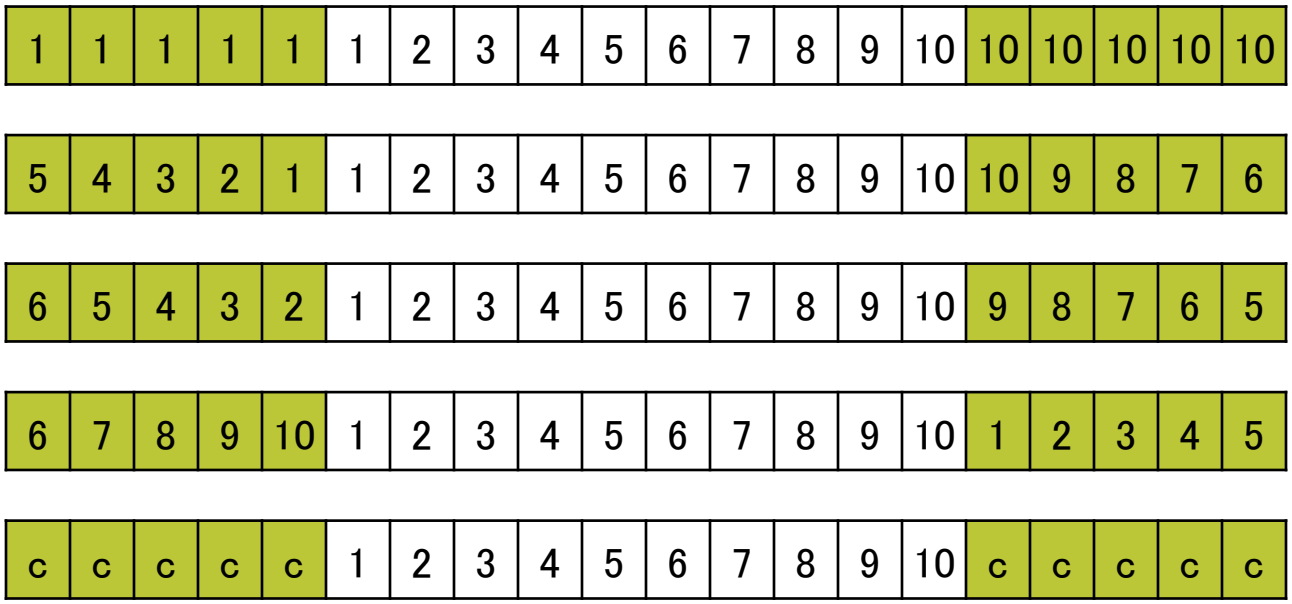
라이브러리 함수 기반의 image filtering 구현

```
void    filter2D( … )
void    copyMakeBorder( … )
void    sepFilter2D( … )
Mat     getGaussianKernel( … )
void    blur( … )
void    boxFilter( … )
void    GaussianBlur( … )
void    medianBlur( … )
void    Sobel( … )
void    Laplacian( … )
```

```
void filter2D(
    InputArray src,     // Source image
    OutputArray dst,    // Destination image
    int ddepth,         // Desired depth of dst
    InputArray kernel, // Convolution kernel
    Point anchor=Point(-1,-1),
    double delta=0,     // Optional value
    int borderType=BORDER_DEFAULT
)
```

borderType

BORDER_REPLICATE
BORDER_REFLECT
BORDER_REFLECT_101
BORDER_WRAP
BORDER_CONSTANT



```
void copyMakeBorder(
      InputArray src,
      OutputArray dst,
      int top, int bottom, int left, int right,
      int borderType,
      const Scalar& value=Scalar()
)
```

```
// let border be the same in all directions
int border=2;

// constructs a larger image
// to fit both the image and the border
Mat input_buf( input.rows + border*2, input.cols + border*2,
               input.depth() );

// forms a border around the input image
// copies the source image into the middle of
// the destination image
copyMakeBorder( input, input_buf,
                border, border, border, border,
                BORDER_REPLICATE );

// now do some custom filtering ...

// select the middle part of it w/o copying data (ROI)
Mat result( result_buf,
            Rect(border, border, input.cols, input.rows) );
```

```
void meanFiltering( const Mat &image, Mat &result )
{
   // allocate if necessary
   result.create( image.size(), image.type() );

   // Construct kernel (all entries initialized to 0)
   Mat kernel( 3, 3, CV_32F, cv::Scalar(0) );

   // assigns kernel values
   int r, c;
   for( r=0; r<=2; r++ )
   {
      for( c=0; c<=2; c++ )
      {
         kernel.at<float>( r, c ) = 1.f/9.f;
      }
   }

   // filter the image
   filter2D( image, result, image.depth(), kernel );
}
```

```
void blur(
        InputArray src,
        OutputArray dst,
        Size ksize,
        Point anchor=Point(-1,-1),
        int borderType=BORDER_DEFAULT
)
```

example

```
blur( image, result, 3 )
```

```
void boxFilter(
        InputArray src,
        OutputArray dst,
        int ddepth,
        Size ksize,
        Point anchor=Point(-1,-1),
        bool normalize=true,
        int borderType=BORDER_DEFAULT
)
```

example

```
boxFilter( image, result,
           image.depth(),
           3
)

≡ blur( image, result, 3 )
```

```
void GaussianBlur(
        InputArray src,
        OutputArray dst,
        Size ksize,
        double sigmaX,
        double sigmaY=0,
        int borderType=BORDER_DEFAULT
)
```

sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8

example

```
GaussianBlur( image, result,
              Size(3,3),
              1.0
)
```

```
Mat getGaussianKernel(
      int ksize,
      double sigma,
      int ktype=CV_64F
)
```

$$G_i = \alpha \times e^{-(i-(\text{ksize}-1)/2)^2/(2\times\sigma)^2}$$

$i = 1 \ldots$ ksize-1

α is the scale factor chosen so that $\sum_i G_i = 1$

```
void sepFilter2D(
      InputArray src,
      OutputArray dst,
      int ddepth,
      InputArray kernelX,
      InputArray kernelY,
      Point anchor=Point(-1,-1),
      double delta=0,
      int borderType=BORDER_DEFAULT
)
```

example

```
Mat kernel = getGaussianKernel( 3, 1. );
sepFilter2D( image, result,
             image.depth(),
             kernel, kernel.t()
);
```

```
void medianBlur(
        InputArray src,
        OutputArray dst,
        int ksize
)
```

example

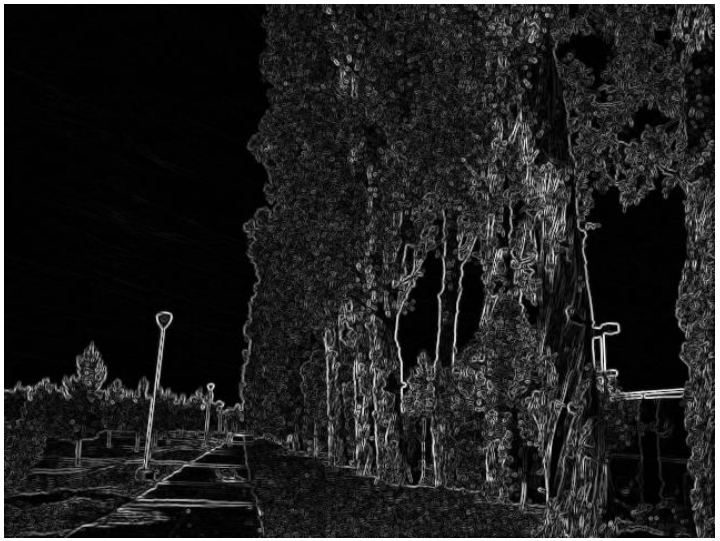
```
MedianBlur( image,
            result,
            3
)
```

Salt & pepper noise (0.03)

median Filtering (3 x 3)

mean Filtering (3 x 3)

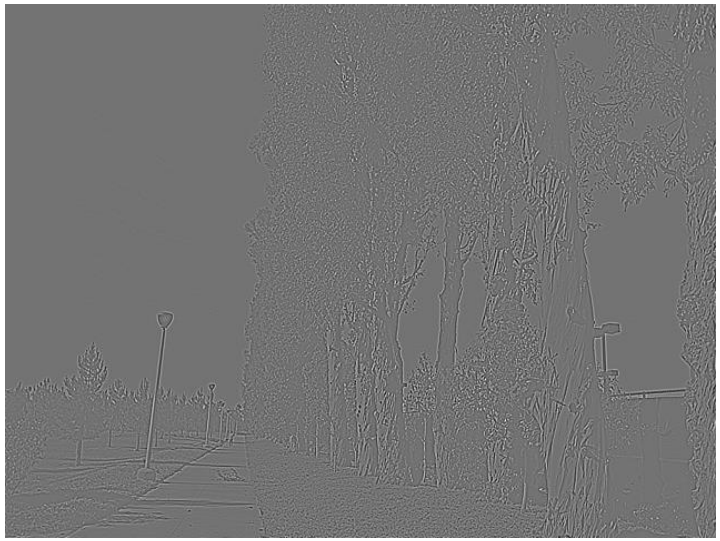
```
void Sobel(
        InputArray src,
        OutputArray dst,
        int ddepth,
        int dx, int dy,
        int ksize=3,
        double scale=1, double delta=0,
        int borderType=BORDER_DEFAULT
)
```



```
void edgeDetection( const Mat &image, Mat &result )
{
	// allocate if necessary
	result.create( image.size(), image.type() );

	// compute Sobel
	Mat sobelX, sobelY;
	Sobel( image, sobelX, CV_16S, 1, 0 );
	Sobel( image, sobelY, CV_16S, 0, 1 );

	// compute the L1 norm
	Mat sobel;
	sobel = abs( sobelX ) + abs( sobelY );

	// scaling
	double maxV;
	minMaxLoc( sobel, 0, &maxV );
	sobel.convertTo( result, image.type(), 255./maxV, 0 );
}
```

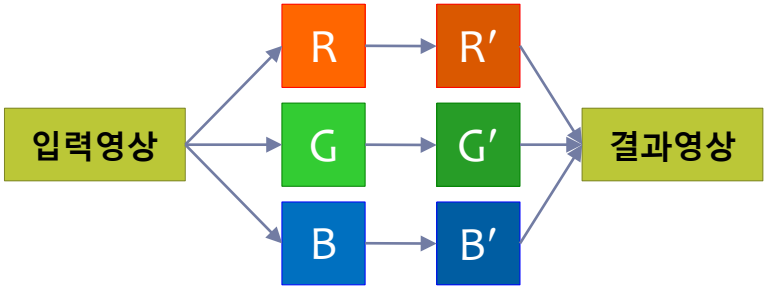
```
void edgeDetection( const Mat &image, Mat &result )
{
	// allocate if necessary
	result.create( image.size(), image.type() );

	// Sobel must be computed in floating point
	Mat sobelX, sobelY;
	Sobel( image, sobelX, CV_32F, 1, 0 );
	Sobel( image, sobelY, CV_32F, 0, 1 );

	// compute the L2 norm and direction
	Mat sobel, dir;
	cartToPolar( sobelX, sobelY, sobel, dir, true );

	// scaling
	double maxV;
	minMaxLoc( sobel, 0, &maxV );
	sobel.convertTo( result, image.type(), 255./maxV, 0 );
}
```

```
void Laplacian(
        InputArray src,
        OutputArray dst,
        int ddepth,
        int ksize=1,
        double scale=1,
        double delta=0,
        int borderType=BORDER_DEFAULT
)
```

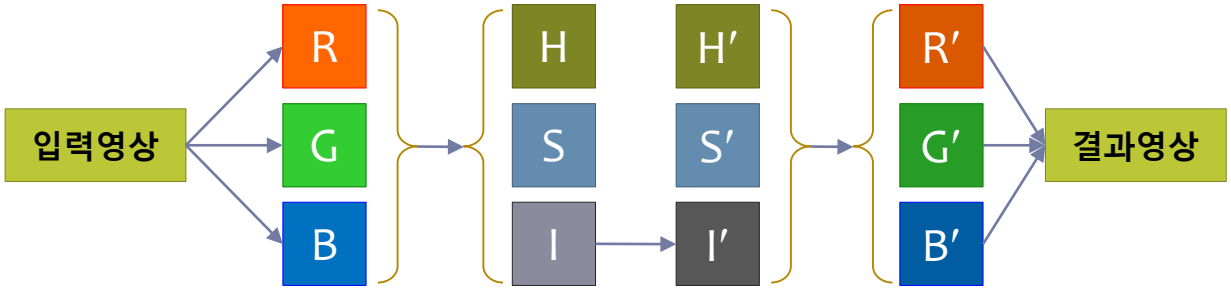


```
void computeLaplacian( const Mat &image, Mat &result )
{
	// allocate if necessary
	result.create( image.size(), image.type() );

	// compute Laplacian
	Mat laplace;
	Laplacian( image, laplace, CV_32F );

	// scaling
	double minvalue, maxvalue, scale;
	minMaxLoc( laplace, &minvalue, &maxvalue );
	scale = 127 / max(-minvalue, maxvalue);
	laplace.convertTo( result, image.type(), scale, 128 );
}
```

칼라 영상에 대한 처리 방법

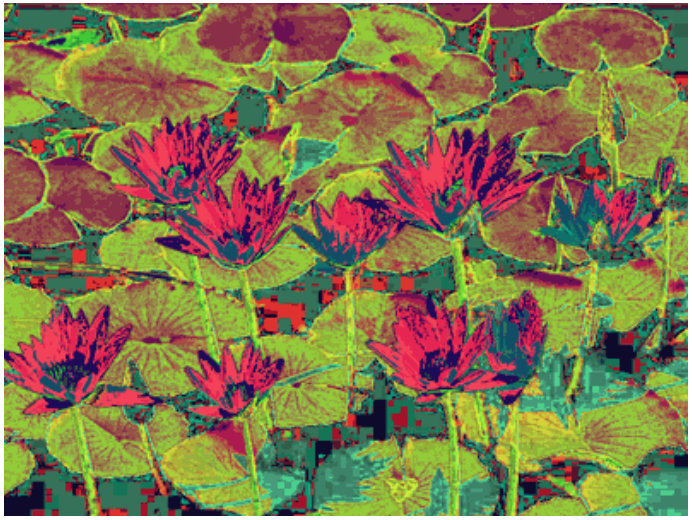


RGB 처리



HSI 처리

I 성분에 대한 histogram equalization



RGB 성분별 histogram equalization

```
void cvtColor(
      InputArray src,
      OutputArray dst,
      int code,
      int dstCn=0
)
```

RGB $\leftrightarrow$ CIE XYZ.Rec 709 with D65 white point

$\Rightarrow$ CV_BGR2XYZ, CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB

RGB $\leftrightarrow$ YCrCb JPEG (or YCC)

$\Rightarrow$ CV_BGR2YCrCb, CV_RGB2YCrCb, CV_YCrCb2BGR, CV_YCrCb2RGB

RGB $\leftrightarrow$ HSV

$\Rightarrow$ CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB

RGB $\leftrightarrow$ HLS

$\Rightarrow$ CV_BGR2HLS, CV_RGB2HLS, CV_HLS2BGR, CV_HLS2RGB

RGB $\leftrightarrow$ CIE L*a*b*

$\Rightarrow$ CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB

RGB $\leftrightarrow$ CIE L*u*v*

$\Rightarrow$ CV_BGR2Luv, CV_RGB2Luv, CV_Luv2BGR, CV_Luv2RGB

Reference

- R. Laganière, **OpenCV2 Computer Vision: Application Programming Cookbook,** PACKT Publishing, 2011
- G. Bradski and A. Kaebler, **Learning OpenCV: Computer Vision with the OpenCV Library**, O'REILLY, 2008
- http://docs.opencv.org