Filters and Convolution

★ Overview

⇒ In this chapter we will look into image processing.

★ Filters, Kernels and Convolution

(Filter)

↳ A filter is any algorithm that starts with some image $I(x,y)$ and computes a new image $I'(x,y)$ by computing for each pixel location x,y in I' some function of the pixels in I that are in some area around the same x,y location.

(Kernel)

↳ The template that defines both this small area's shape as well as how the elements of that small area are combined

(Linear Kernel) → Value assigned to point x,y in I' can be expressed as a weighted sum of the points around x,y in I.

$$I'(x,y) = \sum_{i,j \in Kernel} K_{i,j} \cdot I(x+i, y+j)$$

⇒ Any filter that can be expressed with linear kernel is known as a convolution.

↳ Though the term is often used somewhat casually in the Computer Vision community to include the application of any filter over an entire image.

Anchor Point

The anchor point defines how your kernel is positioned with respect to the pixel currently being processed during the filter operation.

* Border Extrapolation and Boundary Conditions

⇒ An issue that will come up with some frequency as we look at how image are processed in OpenCV is how borders are handled.

★ Making borders yourself

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

(top, bottom, left, right: Number of Pixel for padding)

⇒ Boder types:

- → cv::BORDER_CONSTANT
- → cv::BORDER_WRAP
- → cv::BORDER_REPLICATE
- → cv::BORDER_REFLECT
- → cv::BORDER_REFLECT_101
- → cv::BORDER_DEFAULT

★ Manual extrapolation

⇒ This function is typically used internally to OpenCV but it can come in handy in your own algorithms as well.

* Threshold Operations

⇒ The OpenCV function cv::threshold() accomplishes this task.

⇒ The basic idea is that an ~~element~~ array is given, along with a threshold, and then something happens to every element of the array depending on whether it is below or above the threshold.

```
double cv::threshold(
    cv::InputArray  src,
    cv::OutputArray dst,
    double  thresh,
    double  maxValue,
     int    thresholdType
);
```

(Table 10-2)

* Otsu's Algorithm

⇒ It is also possible to have cv::threshold() attempt to determine the optimal value of the threshold for you.

↳ You do this by passing the special value cv::THRESH_OTSU as the value of thresh.

⇒ This is not particularly fast process.

★ Adaptive Threshold

⇒ There is a modified threshold technique in which the threshold level is itself variable (across the image).

```
Void cv::adaptiveThreshold(
    cv::InputArray  src,
    cv::OutputArray dst,
    double          maxValue,
    int             adaptiveMethod,
    int             thresholdType
    int             blockSize
    double          C
);
```

adaptive Method
- cv::ADAPTIVE_THRESH_MEAN_C
- cv::ADAPTIVE_THRESH_GAUSSIAN_C

cv::ADAPTIVE_THRESH_GAUSSIAN_C → Pixel in the region around (x,y) are weighted according to Gaussian function of their distance from the center point.

cv::ADAPTIVE_THRESH_MEAN_C → Pixel in the area are weighted Equally

* Smoothing

⇒ Smoothing also called blurring.

⇒ Smoothing is also important when we wish to reduce the resolution of an image in a principled way.

⇒ OpenCV offers five different smoothing operations

* Simple Blur and the Box Filter

```
void cv::blur(
    cv::InputArray  src,
    cv::OutputArray dst,
    cv::Size        Ksize
    cv::Point       anchor = cv::Point(-1,-1),
    int             borderType = cv::BORDER_DEFAULT
);
```

⇒ The simple blur is a specialized version of a box filter.

↳ A box filter is any filter that has a rectangular profile and for which the values $K_{i,j}$ are all equal.

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

* Median Filter

⇒ The median filter replaces each pixel by the median in a rectangular neighborhood around the center pixel.

```
Void cv::medianBlur(
    cv::InputArray  src,
    cv::OutputArray dst,
    cv::Size  Ksize
);
```

* Gaussian Filter

⇒ Most useful smoothing filter.

```
Void cv::GaussianBlur(
    cv::InputArray  src,
    cv::OutputArray dst,
    cv::Size        ksize,
    double          sigmaX,
    double          sigmaY = 0.0
    int   borderType = cv::BORDER_DEFAULT
);
```

⇒ If you specify only the x value and set the y value to 0 (as default), then the y and x values will be taken to be equal.

* Bilateral Filter

```
void cv::bilateralFilter(
    cv::InputArray  src,
    cv::OutputArray dst,
    int     d,
    double  sigmaColor,
    double  sigmaSpace,
    int     borderType = cv::BORDER_DEFAULT
```

⇒ Bilateral filtering is one operation from a somewhat larger class of image analysis operators known as edge-preserving smoothing.