```
                    double  delta = 0,
                    int  borderType = BORDER_DEFAULT
                    )
```

- src：待滤波图像。
- dst：输出图像，与输入图像 src 具有相同的尺寸、通道数和数据类型。
- ddepth：输出图像的数据类型（深度），根据输入图像的数据类型不同，拥有不同的取值范围，具体的取值范围在表 5-1 中给出。当赋值为−1 时，输出图像的数据类型自动选择。
- kernelX：X 方向的滤波器。
- kernelY：Y 方向的滤波器。
- anchor：内核的基准点（锚点），其默认值(−1, −1)代表内核基准点位于 kernel 的中心位置。基准点是卷积核中与进行处理的像素点重合的点，其位置必须在卷积核的内部。
- delta：偏值，在计算结果中加上偏值。
- borderType：像素外推法选择标志，取值范围在表 3-5 中给出。默认参数为 BORDER_DEFAULT，表示不包含边界值倒序填充。

该函数将可分离的线性滤波器分离成 X 方向和 Y 方向进行处理，与 filter2D()函数不同之处在于，filter2D()函数需要通过滤波器的尺寸区分滤波操作是作用在 X 方向还是作用在 Y 方向，例如滤波器尺寸为 $K\times1$ 时是 Y 方向滤波，$1\times K$ 尺寸的滤波器是 X 方向滤波，而 sepFilter2D()函数通过不同参数区分滤波器是作用在 X 方向还是作用在 Y 方向，无论输入滤波器的尺寸是 $K\times1$ 还是 $1\times K$，都不会影响滤波结果。

为了更加了解线性滤波的可分离性，在代码清单 5-17 中给出了利用 filter2D()函数和 sepFilter2D()函数实现滤波的示例程序。在该程序中，利用 filter2D()函数依次进行 Y 方向和 X 方向滤波，将结果与两个方向联合滤波器滤波结果相比较，验证两种方式计算结果的一致性。同时，将两个方向的滤波器输入 sepFilter2D()函数中，验证该函数计算结果是否与前面的计算结果一致。最后，利用自定义的滤波器对图像依次进行 X 方向滤波和 Y 方向滤波，查看滤波结果是否与使用联合滤波器的滤波结果一致。该程序的计算结果分别在图 5-19 和图 5-20 中给出。

代码清单 5-17 myselfFilter.cpp 可分离图像滤波

```
#include <opencv2\opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
    system("color F0");  //更改输出界面颜色
    float points[25] = { 1,2,3,4,5,
                         6,7,8,9,10,
                         11,12,13,14,15,
                         16,17,18,19,20,
                         21,22,23,24,25 };
    Mat data(5, 5, CV_32FC1, points);
    //X 方向、Y 方向和联合滤波器的构建
    Mat a = (Mat_<float>(3, 1) << -1, 3, -1);
    Mat b = a.reshape(1, 1);
    Mat ab = a*b;
    //验证高斯滤波的可分离性
    Mat gaussX = getGaussianKernel(3, 1);
    Mat gaussData, gaussDataXY;
    GaussianBlur(data, gaussData, Size(3, 3), 1, 1, BORDER_CONSTANT);
```