فیلتر گذاری روی عکس:

فیلترگذاری روی تصویر به فرایندی گفته می‌شود که در آن با اعمال تغییرات ریاضی یا گرافیکی روی پیکسل‌های تصویر، ظاهر یا ویژگی‌های آن تغییر می‌کند. این تغییرات می‌توانند برای بهبود کیفیت تصویر (مثل حذف نویز یا افزایش وضوح) یا ایجاد جلوه‌های هنری (مثل سیاه‌وسفید کردن یا تار کردن بخش‌هایی از تصویر) به کار بروند. در واقع، هر فیلتر الگوریتمی دارد که مشخص می‌کند هر پیکسل چگونه با توجه به خودش یا پیکسل‌های اطرافش تغییر یابد.

فیلترها به‌طور کلی به دو دسته‌ی اصلی تقسیم می‌شوند :**فیلترهای مکانی (spatial filters)** و **فیلترهای فرکانسی (frequency filters)**. در فیلترهای مکانی، تغییرات مستقیماً روی پیکسل‌های تصویر و همسایه‌های آن‌ها اعمال می‌شود؛ مثل فیلترهای محوکننده (blur) یا تیزکننده .(sharpen) در مقابل، فیلترهای فرکانسی ابتدا تصویر را به حوزه‌ی فرکانس (مثلاً با تبدیل فوریه) منتقل می‌کنند و سپس تغییرات روی بسامدهای مختلف اعمال می‌شود؛ مثل فیلتر پایین‌گذر (low-pass) برای حذف جزئیات ریز یا بالاگذر (high-pass) برای برجسته‌سازی لبه‌ها.

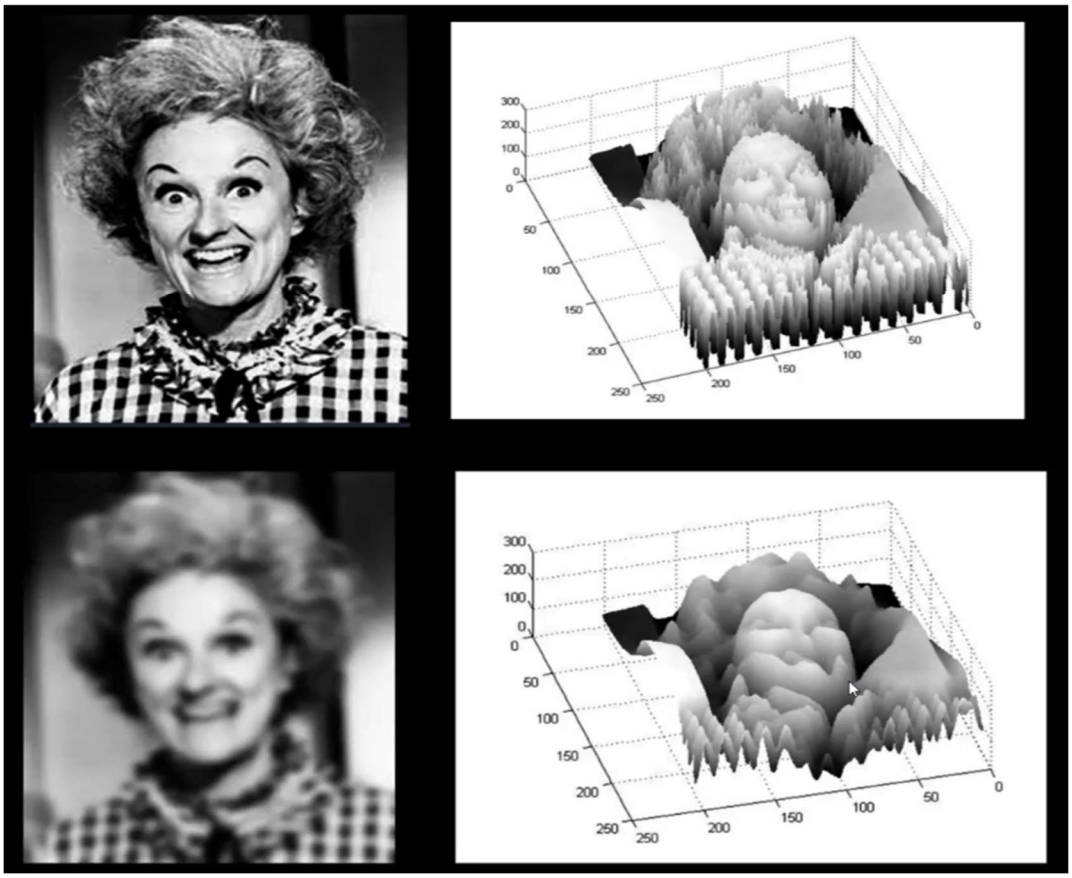
کاربردهای فیلترگذاری بسیار گسترده است. در پردازش تصویر پزشکی برای وضوح بهتر جزئیات، در عکاسی دیجیتال برای ایجاد افکت‌های هنری، در سیستم‌های بینایی ماشین برای تشخیص لبه‌ها و اشیا، و حتی در شبکه‌های اجتماعی برای زیباسازی عکس‌ها از آن استفاده می‌شود. علاوه بر این، فیلترگذاری می‌تواند به‌عنوان مرحله‌ی پیش‌پردازش داده‌ها پیش از اعمال الگوریتم‌های هوش مصنوعی نقش کلیدی داشته باشد، چراکه کیفیت داده‌ی ورودی را بهبود می‌بخشد.

تصویر به عنوان یک تابع:

وقتی در ریاضیات یا پردازش تصویر درباره‌ی «تصویر به‌عنوان یک تابع» صحبت می‌کنیم، منظور این است که تصویر را مجموعه‌ای از مقادیر شدت روشنایی یا رنگ در نقاط مختلف صفحه در نظر بگیریم. به‌طور ساده، می‌توان تصویر خاکستری را به صورت یک تابع دو متغیره $f(x,y)$ تعریف کرد که در آن x و y مختصات مکان در صفحه هستند و مقدار $f(x,y)$ شدت روشنایی پیکسل در آن نقطه است. برای تصاویر رنگی هم به جای یک مقدار، معمولاً سه مقدار (قرمز، سبز و آبی) داریم.

این دیدگاه تابعی کمک می‌کند تا بتوانیم از ابزارهای ریاضی مثل جبر خطی، حسابان یا تبدیل‌ها (مثل تبدیل فوریه) برای تحلیل تصویر استفاده کنیم. برای مثال، وقتی می‌گوییم فیلتر محو (blur) اعمال می‌کنیم، در واقع داریم تابع تصویر را با یک «کرنل» (یک تابع کوچک دیگر) کانولوشن می‌کنیم. یا وقتی صحبت از لبه‌ها می‌شود، منظور تغییرات شدید تابع در اطراف یک نقطه است که می‌توان با مشتق‌گیری از تابع تصویر آن را پیدا کرد.

مزیت نگاه تابعی به تصویر این است که تصویر دیگر صرفاً یک عکس دیجیتال یا مجموعه‌ای از پیکسل‌ها نیست، بلکه یک داده‌ی ریاضی است که می‌توان روی آن عملیات محاسباتی انجام داد. همین نگاه باعث شده که پردازش تصویر و بینایی ماشین با روش‌های ریاضی و الگوریتمی پیشرفت کند. به‌طور خلاصه: هر تصویر = یک تابع دوبعدی (یا چندبعدی برای رنگ) که می‌توان روی آن محاسبات و تبدیل‌ها انجام داد.

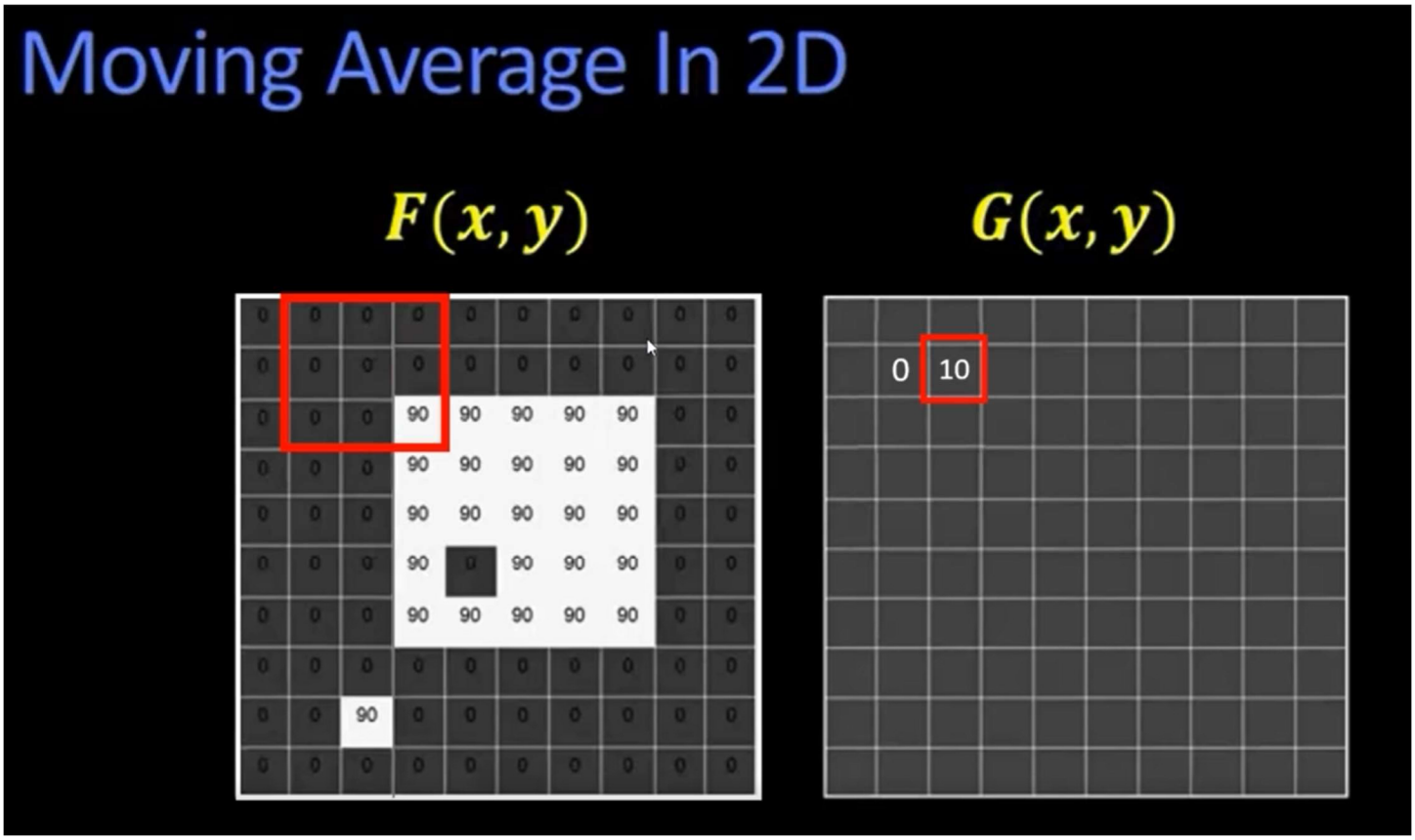


فیلتر متحرک در عکس *Moving Average*:

یکی از روش‌های پرکاربرد برای کاهش نویز در تصاویر دیجیتال، استفاده از فیلتر میانگین متحرک یا همان ***Moving Average Filter*** است. این فیلتر بر اساس یک ایده‌ی ساده کار می‌کند: هر پیکسل تصویر با میانگین مقدار خودش و پیکسل‌های اطرافش جایگزین می‌شود. به این ترتیب، تغییرات ناگهانی و نویزهای تصادفی کاهش یافته و تصویر حالت نرم‌تر و یکنواخت‌تری (smooth) پیدا می‌کند.

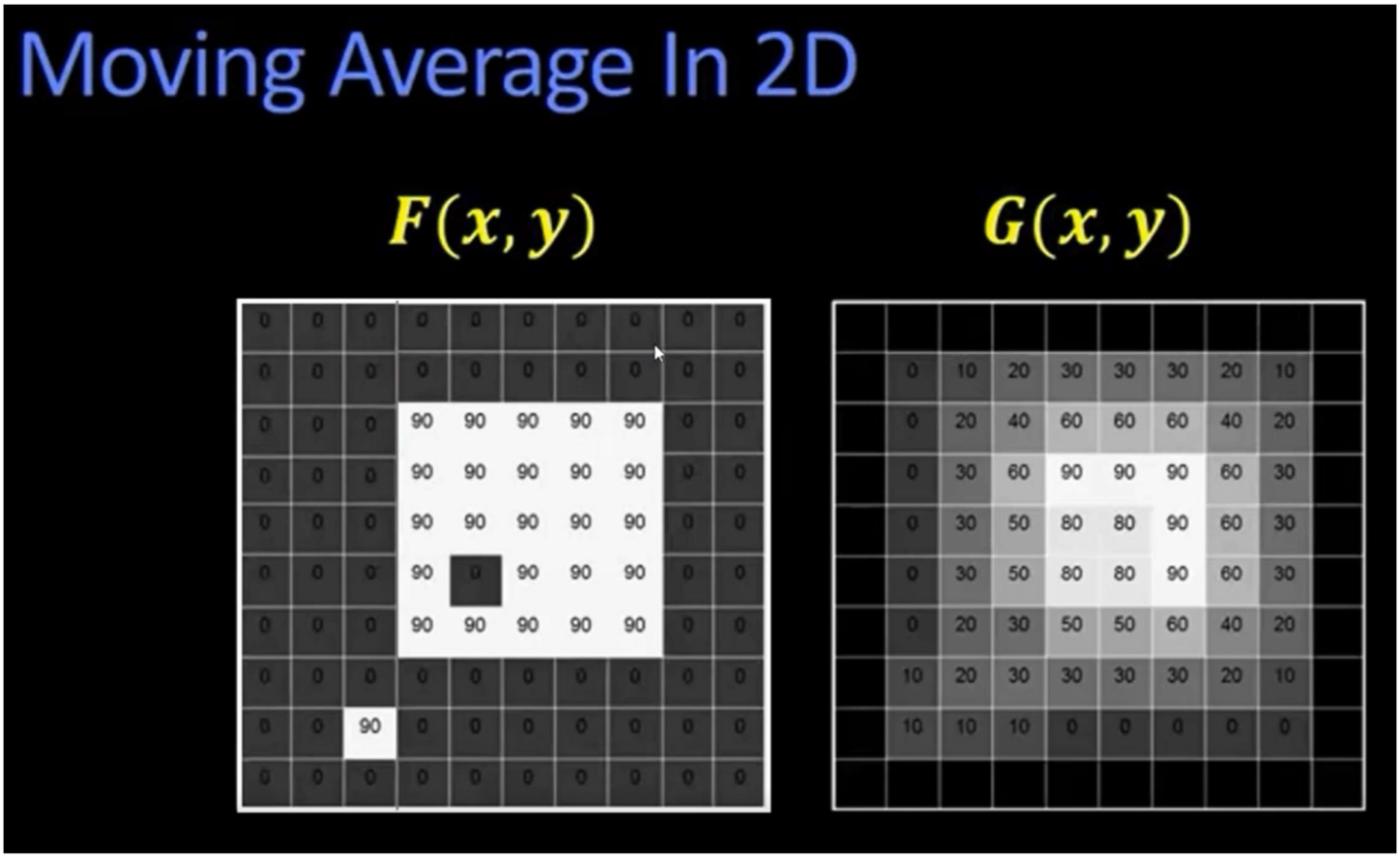
برای اعمال این فیلتر، ابتدا یک پنجره یا ماسک با اندازه‌ی مشخص (مانند ۳×۳ یا ۵×۵) انتخاب می‌شود. این پنجره به صورت متحرک روی تصویر حرکت می‌کند دلیل استفاده از اعداد فرد در این پنجره این است که ما میخواهیم یک مرکز نسبت به همسایه ها داشته باشیم. در هر موقعیت، همه‌ی پیکسل‌های موجود در پنجره با هم جمع می‌شوند و میانگین آن‌ها محاسبه می‌شود. مقدار به‌دست‌آمده سپس جایگزین مقدار پیکسل مرکزی در آن ناحیه خواهد شد. این فرآیند برای تمام نقاط تصویر تکرار می‌شود.

برای روشن‌تر شدن موضوع، فرض کنید پنجره‌ای ۳×۳ داریم که مقادیر پیکسل‌های داخل آن به شکل زیر است: 10، 50، 80، 20، 70، 90، 30، 40، 100 .مجموع این مقادیر برابر 490 بوده و میانگین آن‌ها برابر با 54 می‌شود. بنابراین پیکسل مرکزی که قبلاً مقدار 70 داشت، اکنون با عدد 54 جایگزین خواهد شد. این تغییر باعث می‌شود تأثیر نویزهای اتفاقی کمتر شود.



این فیلتر مزایایی مانند سادگی، سرعت و کارایی در کاهش نویزهای تصادفی به‌ویژه نویز گاوسی دارد. همچنین به دلیل هموارسازی، تصویری نرم‌تر و یکنواخت‌تر ایجاد می‌کند. با این حال، نقطه ضعف آن این است که علاوه بر نویز، جزئیات ریز و لبه‌های تصویر نیز تا حدی محو می‌شوند و وضوح تصویر کاهش می‌یابد.

در نهایت می‌توان گفت فیلتر میانگین متحرک یک ابزار پایه‌ای در پردازش تصویر به شمار می‌رود. این فیلتر در بسیاری از مواقع به‌عنوان یک مرحله‌ی پیش‌پردازش برای کاهش نویز به کار می‌رود، هرچند که در شرایطی که نیاز به حفظ جزئیات و لبه‌ها باشد، استفاده از فیلترهای دیگری مانند فیلتر میانه مناسب‌تر خواهد بود.



مراحل کار

1. ابتدا یک پنجره (ماسک یا کرنل) با اندازه مشخص، مثلاً 3×3 \times 33×3یا 5×5 \times 55×5، انتخاب می‌کنیم.
2. پنجره را روی تصویر حرکت می‌دهیم (به همین دلیل به آن "متحرک" می‌گویند).
3. برای هر پیکسل، همه مقادیر داخل پنجره جمع شده و بر تعدادشان تقسیم می‌شود.
4. نتیجه به‌عنوان مقدار جدید پیکسل مرکزی قرار می‌گیرد.
5. این روند برای همه‌ی نقاط تصویر تکرار می‌شود.

◆مثال عددی

اگر پنجره 3×3 به شکل زیر باشد:

$$\begin{bmatrix} 80 & 50 & 10 \\ 90 & 70 & 20 \\ 100 & 40 & 30 \end{bmatrix}$$

جمع کل عناصر برابر با 490 است.
با تقسیم بر 9، میانگین حدود **54**به دست می‌آید.
بنابراین مقدار پیکسل مرکزی (که قبلاً 70 بود) با 54 جایگزین می‌شود.

◆مزایا

- ساده و قابل پیاده‌سازی.
- در کاهش نویزهای تصادفی (مانند نویز گاوسی) مؤثر است.
- باعث نرم و هموار شدن تصویر می‌شود.

◆معایب

- لبه‌ها و جزئیات تصویر محو و تار می‌شوند.
- برای نویزهای شدید مانند "نمک و فلفل" کارایی چندانی ندارد (فیلتر میانه بهتر عمل می‌کند).

کانولوشن (Convolution) و کرولیشن (Correlation) در فیلترگذاری تصویر

در پردازش تصویر، اعمال فیلتر یا ماسک بر روی تصویر معمولاً از طریق دو عمل ریاضی انجام می‌شود :**کرولیشن** (Correlation) و **کانولوشن** .(Convolution)این دو عمل در ظاهر شباهت زیادی به یکدیگر دارند، اما یک تفاوت بنیادی میان آن‌ها وجود دارد که باعث تمایز نتایج در برخی موارد می‌شود.

۱ .کرولیشن(Correlation)

در کرولیشن، ماسک (کرنل) به همان شکل اصلی خود بر روی تصویر حرکت داده می‌شود. مقدار هر پیکسل خروجی برابر است با مجموع حاصل‌ضرب عناصر ماسک در پیکسل‌های متناظر تصویر در همان ناحیه. رابطه‌ی ریاضی این عملیات به صورت زیر بیان می‌شود:

$$h(i,j) \cdot (j+i, y+f(x \sum_{j} \sum_{i} = g(x,y)$$

که در آن:

- f(x,y)f(x,y)f(x,y) تصویر ورودی،
- h(i,j) ماسک یا کرنل،
- g(x,y) تصویر خروجی است.

۲ .کانولوشن(Convolution)

کانولوشن از نظر محاسبات مشابه کرولیشن است، با این تفاوت که پیش از اعمال ماسک، کرنل باید **۱۸۰ درجه دوران داده شود** (یعنی هم در راستای افقی و هم در راستای عمودی معکوس گردد). فرمول این عمل به صورت زیر تعریف می‌شود:

$$h(-i,-j) \cdot (j+i, y+f(x \sum_{j} \sum_{i} = g(x,y)$$

بنابراین، کانولوشن همان کرولیشن است با این تفاوت که ماسک وارونه شده مورد استفاده قرار می‌گیرد.

تفاوت اصلی

تفاوت کلیدی میان این دو عمل در همین وارونه‌سازی ماسک نهفته است. اگر ماسک **متقارن** باشد (برای مثال ماسک میانگین یا ماسک گاوسی)، نتایج کانولوشن و کرولیشن کاملاً یکسان خواهد بود. اما در ماسک‌های **نامتقارن** (مانند ماسک‌های تشخیص لبه یا گرادیان)، نتایج این دو روش متفاوت خواهد بود.

کاربردها

- در **ریاضیات و پردازش سیگنال**، کانولوشن تعریف اصلی و استاندارد محسوب می‌شود و در مباحث تئوری عمدتاً این عمل مورد استفاده قرار می‌گیرد.
- در **نرم‌افزارها و کتابخانه‌های پردازش تصویر**)مانند(OpenCV ، به دلایل محاسباتی ساده‌تر، اغلب کرولیشن پیاده‌سازی می‌شود.
- در بسیاری از کاربردهای عملی که از ماسک‌های متقارن استفاده می‌کنند، تفاوتی میان این دو عمل مشاهده نمی‌شود.

جمع‌بندی

به طور خلاصه:

- در **کرولیشن**، ماسک بدون تغییر استفاده می‌شود.
- در **کانولوشن**، ماسک ابتدا ۱۸۰ درجه چرخانده شده و سپس اعمال می‌گردد.
- اگر ماسک متقارن باشد، خروجی هر دو روش یکسان خواهد بود.
- در نظریه‌ی ریاضی، کانولوشن مبنا قرار می‌گیرد، در حالی که در بسیاری از پیاده‌سازی‌های نرم‌افزاری، کرولیشن مورد استفاده است.

فیلتر غیر خطی یا میانه median:

فیلتر میانه یکی از مهم‌ترین فیلترهای غیرخطی در پردازش تصویر است که بیشتر برای حذف نویزهای ضربه‌ای، به‌ویژه نویز نمک و فلفل، مورد استفاده قرار می‌گیرد. تفاوت اصلی آن با فیلترهای خطی این است که به جای محاسبه‌ی میانگین مقادیر پیکسل‌های همسایه، از مقدار میانه‌ی آن‌ها استفاده می‌کند. این ویژگی باعث می‌شود فیلتر میانه نسبت به فیلتر میانگین در حفظ جزئیات تصویر عملکرد بهتری داشته باشد.

روش کار فیلتر میانه به این صورت است که ابتدا یک پنجره یا کرنل با اندازه مشخص (مانند ۳×۳ یا ۵×۵) روی تصویر قرار می‌گیرد. سپس مقادیر پیکسل‌های موجود در این پنجره استخراج شده و به ترتیب صعودی مرتب می‌شوند. عدد میانه (یعنی مقداری که در وسط این لیست مرتب‌شده قرار دارد) به عنوان مقدار جدید پیکسل مرکزی انتخاب می‌شود. این روند برای تمام پیکسل‌های تصویر تکرار خواهد شد.

برای مثال اگر پنجره‌ای ۳×۳ شامل مقادیر زیر باشد: 10، 250، 15، 20، 255، 30، 25، 240 و 35، پس از مرتب‌سازی به شکل 10، 15، 20، 25، 30، 35، 240، 250 و 255 در می‌آیند. مقدار میانه در این مجموعه ۳۰ است. بنابراین پیکسل مرکزی که مقدار اولیه‌ی آن 255 بوده است، با مقدار ۳۰ جایگزین می‌شود. این جایگزینی باعث حذف اثر نویز شدید و نزدیک‌تر شدن مقدار پیکسل به همسایگی واقعی آن می‌شود.

از مهم‌ترین مزایای فیلتر میانه می‌توان به توانایی بالای آن در حذف نویز نمک و فلفل اشاره کرد. علاوه بر این، برخلاف فیلترهای خطی نظیر فیلتر میانگین، این فیلتر باعث تار شدن شدید تصویر نمی‌شود و لبه‌ها و جزئیات اصلی تصویر را بهتر حفظ می‌کند. به همین دلیل در بسیاری از کاربردها، زمانی که حذف نویز همراه با حفظ ساختار تصویر مدنظر باشد، فیلتر میانه انتخاب مناسبی است.

با این حال، فیلتر میانه محدودیت‌هایی نیز دارد. از جمله این که محاسبات آن نسبت به فیلتر میانگین سنگین‌تر است، زیرا در هر پنجره نیاز به مرتب‌سازی داده‌ها وجود دارد. همچنین این فیلتر در کاهش نویزهای گاوسی یا نویزهای یکنواخت عملکرد بهتری نسبت به روش‌های خطی ندارد. در مجموع می‌توان گفت فیلتر میانه ابزاری کارآمد برای حذف نویز ضربه‌ای است که در بسیاری از سیستم‌های پردازش تصویر به عنوان روشی استاندارد مورد استفاده قرار می‌گیرد.

فیلتر یونیفرم(Uniform Filter)

فیلتر یونیفرم یا **فیلتر یکنواخت**، فیلتری است که در آن همه‌ی ضرایب کرنل (ماسک) **مقدار یکسان** دارند. به زبان ساده، یعنی وزن هر پیکسل در همسایگی برابر است.

- مثال: یک ماسک 3×3 یونیفرم این‌گونه است:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

در اینجا همه‌ی پیکسل‌های اطراف به یک اندازه در محاسبه‌ی مقدار جدید پیکسل مرکزی تأثیر دارند. این همان چیزی است که در **فیلتر میانگین متحرک ساده (Mean Filter)** استفاده می‌کنیم.

ویژگی‌ها:

- ساده و سریع در محاسبه.
- باعث هموارسازی و کاهش نویز می‌شود.
- اما لبه‌ها و جزئیات را محو می‌کند چون همه‌ی پیکسل‌ها اهمیت یکسان دارند.

فیلتر نان‌یونیفرم(Non-Uniform Filter)

فیلتر نان‌یونیفرم یا **غیر یکنواخت** فیلتری است که ضرایب کرنل آن **یکسان نیستند** .در این نوع فیلتر، به بعضی از پیکسل‌های همسایگی وزن بیشتری داده می‌شود و به برخی کمتر.

- مثال: ماسک گاوسی (Gaussian Kernel) یک فیلتر نان‌یونیفرم است:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

در اینجا پیکسل مرکزی (وزن ۴) اهمیت بیشتری دارد، پیکسل‌های نزدیک وزن متوسط دارند، و پیکسل‌های دورتر کمترین وزن را می‌گیرند.

ویژگی‌ها:

- نرم‌سازی هوشمندانه‌تر نسبت به فیلتر یونیفرم.
- نویز را کاهش می‌دهد ولی در عین حال لبه‌ها و جزئیات را بهتر حفظ می‌کند.
- از مهم‌ترین نمونه‌ها می‌توان به **Gaussian Blur** و فیلترهای تشخیص لبه (Sobel, Prewitt, Laplacian) اشاره کرد.

```
image = cv2.imread('images/noisy-cow.png')

# Creating our 3 x 3 kernel
kernel_3x3 = np.ones((3, 3), np.float32) / 9

# We use the cv2.fitler2D to conovlve the kernal with an image
blurred = cv2.filter2D(image, -1, kernel_3x3)


# Creating our 7 x 7 kernel
kernel_7x7 = np.ones((7, 7), np.float32) / 49

blurred2 = cv2.filter2D(image, -1, kernel_7x7)

plt.figure(figsize=[15,5])
plt.subplot(131);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(132);plt.imshow(blurred[...,::-1]);plt.title("blurred with 3x3 filter");
plt.subplot(133);plt.imshow(blurred2[...,::-1]);plt.title("blurred with 7x7 filter");
```

در این کد ابتدا تصویر noisy-cow.png از مسیر مشخص بارگذاری می‌شود. سپس برای فیلتر کردن تصویر از مفهوم "kernel" یا "هسته‌ی کانولوشن" استفاده می‌شود. کرنل در واقع یک ماتریس کوچک است که روی تصویر حرکت می‌کند و هر بخش تصویر را با آن ترکیب می‌کند. در اینجا ابتدا یک کرنل ۳×۳ ساخته شده است که تمام مقادیر آن برابر ۱ هستند و بعد بر ۹ تقسیم شده‌اند تا میانگین مقادیر محاسبه شود. این عمل نوعی فیلتر هموارسازی (smoothing) یا محوکننده ساده (blur) است.

برای اعمال این کرنل بر روی تصویر، از تابع `cv2.filter2D`استفاده شده است. این تابع تصویر اصلی را با کرنل تعریف شده کانولوشن می‌کند و نتیجه را در متغیر `blurred`ذخیره می‌نماید. همین فرآیند یک بار دیگر با کرنل ۷×۷ انجام شده است. در این حالت، کرنل بزرگ‌تر باعث محوی (blur) بیشتری در تصویر می‌شود، زیرا ناحیه‌ی بزرگ‌تری از همسایگی هر پیکسل برای محاسبه مقدار جدید آن در نظر گرفته می‌شود. نتیجه این عمل در متغیر `blurred2`ذخیره می‌شود.

در بخش پایانی کد، از کتابخانه Matplotlib برای نمایش سه تصویر استفاده شده است: تصویر اصلی، تصویر محوشده با کرنل ۳×۳، و تصویر محوشده با کرنل ۷×۷. هر تصویر در یک ساب‌پلات جداگانه نمایش داده می‌شود تا بتوان تفاوت‌ها را مشاهده کرد. همان‌طور که انتظار می‌رود، تصویر محوشده با کرنل ۳×۳ فقط کمی هموارتر از تصویر اصلی است، در حالی که تصویر محوشده با کرنل ۷×۷ بسیار نرم‌تر و تارتر به نظر می‌رسد، چون اثر میانگین‌گیری روی نواحی بزرگ‌تری اعمال شده است.

```
blurred2 = cv2.filter2D(image, -1, kernel_7x7)
```

1. **ورودی اول :(image)** این همان تصویر اصلی است که قرار است روی آن عملیات کانولوشن (اعمال فیلتر) انجام شود. می‌تواند تصویر رنگی یا خاکستری باشد.
2. **ورودی دوم :(-1)** این پارامتر مشخص می‌کند "عمق تصویر خروجی (depth) "چه باشد. اگر مقدار `-1`قرار دهیم، یعنی خروجی باید همان عمق و نوع داده‌ای (مثل 8 بیتی یا 32 بیتی) تصویر ورودی را داشته باشد. در واقع اینجا می‌گوید خروجی از نظر نوع داده مثل `image`باشد. اگر عدد دیگری (مثلاً `cv2.CV_32F`)بدهیم، خروجی با آن نوع داده ساخته می‌شود.

ورودی دوم در متد `cv2.filter2D`با نام **ddepth** شناخته می‌شود. این پارامتر تعیین می‌کند **تصویر خروجی چه نوع داده‌ای (Data Type) داشته باشد.**

به زبان ساده‌تر: وقتی یک کرنل روی تصویر اعمال می‌شود، مقادیر پیکسل‌ها تغییر می‌کنند و ممکن است نتیجه عددی خارج از محدوده‌ی اصلی پیکسل‌ها باشد (مثلاً مقدار منفی یا بزرگ‌تر از ۲۵۵ در تصاویر ۸ بیتی) .به `ddepth` OpenCV می‌گوید که با این مقادیر چه کار کند و نوع داده‌ی تصویر خروجی چه باشد.

- اگر مقدار **-1**بدهیم، یعنی :*خروجی باید همان نوع داده‌ی تصویر ورودی باشد* .مثلاً اگر ورودی از نوع `uint8`(یعنی مقادیر پیکسل‌ها بین ۰ تا ۲۵۵) باشد، خروجی هم همان خواهد بود. این حالت معمولاً برای فیلترهای ساده مثل blur مناسب است.
- اگر یک مقدار مثبت مشخص بدهیم، می‌توانیم نوع داده خروجی را تغییر دهیم. مثلاً:
  - **cv2.CV_8U** → تصویر خروجی با نوع داده‌ی ۸ بیتی بدون علامت (۰ تا ۲۵۵).
  - **cv2.CV_16S** → خروجی با ۱۶ بیتی علامت‌دار (برای نگهداری مقادیر منفی هم).

- **cv2.CV_32F** یا **cv2.CV_64F** → خروجی با اعداد اعشاری(Float) ، که دقت بالاتری دارند و می‌توانند مقادیر خیلی بزرگ یا کوچک را هم ذخیره کنند. این حالت مثلاً در فیلترهای لبه‌یابی (Sobel, Laplacian)کاربرد زیادی دارد.

3. **ورودی سوم :(kernel_7x7)** این همان کرنلی است که قبلاً تعریف کرده‌ایم. یک ماتریس ۷×۷ از مقادیر عددی که مشخص می‌کند چطور مقادیر پیکسل‌های همسایه با هم ترکیب شوند. در این مثال کرنل میانگین‌گیر است، یعنی مقدار هر پیکسل جدید برابر میانگین همسایه‌هایش می‌شود، و همین باعث ایجاد اثر محوی (blur) می‌شود.

```
4. image = cv2.imread('images/noisy-cow.png')
5.
6. # Averaging done by convolving the image with a normalized box filter.
7. # This takes the pixels under the box and replaces the central element
8. # Box size needs to odd and positive
9. blur = cv2.blur(image, (7,7))
10.
11.# Instead of box filter, gaussian kernel
12.gaussian = cv2.GaussianBlur(image, (7,7), 0)
13.
14.# Takes median of all the pixels under kernel area and central
15.# element is replaced with this median value
16.median = cv2.medianBlur(image, 5)
17.
18.# Bilateral is very effective in noise removal while keeping edges sharp
19.bilateral = cv2.bilateralFilter(image, 9, 75, 75)
20.
21.plt.figure(figsize=[15,10])
22.plt.subplot(231);plt.imshow(image[...,::-1]);plt.title("Original");
23.plt.subplot(232);plt.imshow(blur[...,::-1]);plt.title("After uniform filter");
24.plt.subplot(233);plt.imshow(gaussian[...,::-1]);plt.title("After non-uniform gaussian
   filter");
25.plt.subplot(234);plt.imshow(median[...,::-1]);plt.title("After median filter");
26.plt.subplot(235);plt.imshow(bilateral[...,::-1]);plt.title("After bilateral filter");
```

در این کد:

۱) cv2.blur(image, (7,7))

این متد یک فیلتر ساده‌ی میانگین‌گیر (Averaging) را روی تصویر اعمال می‌کند. ورودی اول (image) همان تصویر اصلی است. ورودی دوم (7, 7) اندازه‌ی کرنل (kernel size) یا همان پنجره‌ای است که روی تصویر حرکت می‌کند. در اینجا یعنی

یک مربع ۷×۷ پیکسل روی تصویر می‌چرخد و مقدار هر پیکسل خروجی برابر با میانگین همه‌ی پیکسل‌های داخل آن مربع خواهد بود. نتیجه‌ی این متد یک تصویر محوشده (blur) است که نویز کم می‌شود اما جزئیات تصویر نیز نرم و محو می‌شوند.

۲ `cv2.GaussianBlur(image, (7,7), 0)`

این متد مانند blur عمل می‌کند ولی به جای یک کرنل ساده از یک کرنل **گاوسی** استفاده می‌کند که وزن‌های مختلفی به پیکسل‌ها اختصاص می‌دهد. ورودی اول (`image`) همان تصویر اصلی است. ورودی دوم (`7, 7`) اندازه‌ی کرنل است که باید فرد (odd) باشد. ورودی سوم `0`مقدار انحراف معیار (sigmaX) در محور X است. اگر مقدار ۰ داده شود، OpenCVمقدار مناسب را به صورت خودکار محاسبه می‌کند. مزیت GaussianBlur نسبت به blur این است که پیکسل‌های نزدیک‌تر به مرکز وزن بیشتری دارند، بنابراین تصویر محوشده طبیعی‌تر و نرم‌تر به نظر می‌رسد.

۳ `cv2.medianBlur(image, 5)`

این متد به جای گرفتن میانگین، مقدار **میانه (median)** پیکسل‌های داخل کرنل را محاسبه می‌کند. ورودی اول (`image`) همان تصویر اصلی است. ورودی دوم (`5`) اندازه‌ی کرنل است که باید یک عدد فرد مثبت باشد (مثل ۳، ۵، ۷). در اینجا کرنل ۵×۵ استفاده شده است. برای هر پیکسل، تمام مقادیر داخل این پنجره مرتب می‌شوند و مقدار میانه جایگزین پیکسل مرکزی می‌گردد. این روش برای حذف نویزهای خاص مثل "نمک و فلفل" (نقاط سفید و سیاه پراکنده) بسیار مؤثر است و در عین حال لبه‌ها را بهتر از blur و Gaussian حفظ می‌کند.

۴ `cv2.bilateralFilter(image, 9, 75, 75)`

این متد یک فیلتر پیچیده‌تر است که برای حذف نویز استفاده می‌شود ولی همزمان لبه‌های تصویر را حفظ می‌کند. ورودی اول (`image`)همان تصویر اصلی است. ورودی دوم (`9`) اندازه‌ی پنجره‌ی فیلتری است که اطراف هر پیکسل در نظر گرفته می‌شود. ورودی سوم (`75`) سیگمای فضایی (sigmaColor) است که تعیین می‌کند تفاوت رنگ‌ها تا چه حد تأثیرگذار باشند؛ هرچه بزرگ‌تر باشد، رنگ‌های متفاوت بیشتری با هم ترکیب می‌شوند. ورودی چهارم (`75`) سیگمای مکانی (sigmaSpace) است که

شعاع فضایی فیلتر را مشخص می‌کند، یعنی تا چه فاصله‌ای از پیکسل مرکزی تأثیر در محاسبه وجود داشته باشد. این فیلتر نسبت به روش‌های قبلی سنگین‌تر (کندتر) است، اما خروجی آن خیلی بهتر است چون نویز را کاهش می‌دهد و همزمان مرز اجسام واضح باقی می‌ماند.

```
image = cv2.imread('images/noisy-cow.png')

gaussian2 = cv2.GaussianBlur(image, (11,11), 2)
gaussian3 = cv2.GaussianBlur(image, (11,11), 3)
gaussian5 = cv2.GaussianBlur(image, (11,11), 5)

plt.figure(figsize=[10,10])
plt.subplot(221);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(222);plt.imshow(gaussian2[...,::-1]);plt.title("Gaussian filter, sigma=2");
plt.subplot(223);plt.imshow(gaussian3[...,::-1]);plt.title("Gaussian filter, sigma=3");
plt.subplot(224);plt.imshow(gaussian5[...,::-1]);plt.title("Gaussian filter, sigma=5");
```

در این کد از متد **cv2.GaussianBlur** استفاده شده که سه ورودی اصلی دارد. ورودی اول (image) همان تصویر اصلی است که قرار است محو شود. ورودی دوم (11,11) اندازه کرنل (kernel size) را مشخص می‌کند؛ این کرنل باید مقادیر فرد داشته باشد و در اینجا یک پنجره ۱۱×۱۱ پیکسلی است که روی تصویر حرکت می‌کند. ورودی سوم مقدار **انحراف معیار گاوسی** یا همان سیگما (sigmaX) در محور X است؛ این عدد تعیین می‌کند شدت محوشدگی چقدر باشد. هرچه این مقدار بزرگ‌تر باشد، پخش‌شدن مقادیر در کرنل بیشتر می‌شود و تصویر محوتر به نظر می‌رسد. در کد بالا سه بار متد فراخوانی شده است، با سیگماهای ۲، ۳ و ۵ تا تفاوت میزان محوی تصویر مقایسه شود. به طور خلاصه: تصویر ورودی تعیین می‌کند چه چیزی فیلتر شود، اندازه کرنل میزان ناحیه‌ی در نظر گرفته‌شده را مشخص می‌کند و سیگما شدت محوشدگی را کنترل می‌نماید.

```
image = cv2.imread('images/noisy-cow.png')

# Parameters, after None are - the filter strength 'h' (5-10 is a good range)
# Next is hForColorComponents, set as same value as h again
#
dst = cv2.fastNlMeansDenoisingColored(image, None, 6, 6, 7, 21)

plt.figure(figsize=[10,10])
plt.subplot(221);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(222);plt.imshow(dst[...,::-1]);plt.title("result");
```

متد **cv2.fastNlMeansDenoisingColored** پیاده‌سازی سریع «حذف نویز میانگین‌گیری غیرمحلی» (Fast Non-Local Means)برای تصاویر رنگی در OpenCV است. ایده‌ی اصلی این الگوریتم این است که به‌جای تکیه بر همسایه‌های خیلی نزدیک هر پیکسل، در یک ناحیه‌ی بزرگ‌تر به‌دنبال **وصله‌های (patch) مشابه** می‌گردد و با وزن‌دهی بر اساس شباهت بافتی، مقدار پیکسل مرکزی را بازسازی می‌کند. همین باعث می‌شود نویز کاهش یابد ولی **لبه‌ها و بافت‌ها** بهتر از فیلترهای ساده‌ای مثل میانگین یا گاوسی حفظ شوند. نسخه‌ی «Colored» این تابع نویز روشنایی و مؤلفه‌های رنگ را جداگانه مدل می‌کند تا رنگ‌ها دچار محوشدگی ناخواسته نشوند. ورودی تصویر معمولاً باید **۸ بیتی سه‌کاناله (BGR)** باشد.

ورودی‌ها در فراخوانی `cv2.fastNlMeansDenoisingColored(image, None, 6, 6, 7, 21)`

- **image** تصویر ورودیِ رنگی(BGR) ، نوع داده `uint8`.اگر تصویر شما در فضای RGB است، برای نمایش با Matplotlibباید کانال‌ها را جابه‌جا کنید (در کد با `[...,::-1]` همین کار انجام شده.)
- **None** پارامتر `dst`:محلِ خروجی. وقتی `None`بدهید، OpenCV خودش خروجی را برمی‌گرداند و نیازی به پیش‌اختصاص بافِر نیست.
- **6 پارامتر h** :**قدرت فیلتر برای بخش روشنایی/با شدت** .هرچه بزرگ‌تر، حذف نویز قوی‌تر اما خطر «صاف‌شدن بیش از حد» (از بین رفتن جزئیات) بیشتر. بازه‌ی رایج تقریباً ۵ تا ۱۰ است؛ ۶ انتخاب ملایمی است.
- **6 پارامتر hColor** :**قدرت فیلتر برای مؤلفه‌های رنگی** .معمولاً نزدیک یا برابر با `h`تنظیم می‌شود؛ افزایش بیش از حد آن می‌تواند باعث به‌هم‌ریختگی یا پَختگی رنگ‌ها شود.
- **7 پارامتر templateWindowSize** :اندازه‌ی وصله‌ی مقایسه .(Patch) باید **فرد و مثبت** باشد (۳، ۵، ۷، ...). وصله‌ی ۷×۷ تعادل خوبی بین دقت شباهت‌یابی و سرعت دارد؛ کوچک‌تر حساس‌تر به نویز است، بزرگ‌تر کندتر و گاهی بیش‌ازحد هموار می‌کند.
- **21 پارامتر searchWindowSize** :اندازه‌ی پنجره‌ی جست‌وجو برای یافتن وصله‌های مشابه پیرامون هر پیکسل. باید **فرد** و معمولاً **بزرگ‌تر از template** باشد (مثلاً ۱۵، ۲۱، ۲۳). هرچه بزرگ‌تر، احتمال یافتن وصله‌های واقعاً مشابه بیشتر، اما محاسبات سنگین‌تر.

نکات کاربردی :اگر نویز شدید است، `h` و `hColor`را کمی بالا ببرید (با احتیاط برای جلوگیری از ازدست‌رفتن بافت‌ها). اندازه‌ها باید فرد باشند و افزایش `searchWindowSize`کیفیت را به بهای زمانِ بیشتر بهتر می‌کند. برای تصاویر تک‌کاناله از `cv2.fastNlMeansDenoising`استفاده کنید. این روش نسبت به فیلترهای ساده کندتر است، اما معمولاً لبه‌ها و جزئیات مهم را بسیار بهتر حفظ می‌کند.

Sharpen:

فیلتر **Unsharp Masking** یکی از روش‌های متداول شارپ‌سازی در پردازش تصویر است. ایده‌ی اصلی آن این است که ابتدا از تصویر یک نسخه‌ی محو (بلورشده) ساخته می‌شود. وقتی این نسخه‌ی محو از تصویر اصلی کم می‌گردد، فقط بخش‌هایی باقی می‌مانند که تغییرات سریع دارند، یعنی لبه‌ها و جزئیات. این بخش همان چیزی است که به آن ماسک جزئیات گفته می‌شود. سپس این ماسک با شدتی قابل تنظیم به تصویر اصلی اضافه می‌شود تا لبه‌ها پررنگ‌تر و تصویر واضح‌تر شود.

مزیت بزرگ این روش نسبت به شارپ‌سازی ساده این است که می‌توان میزان وضوح را کنترل کرد. اگر شدت تقویت زیاد انتخاب شود، تصویر بیش‌ازحد تیز و غیرطبیعی می‌شود، و اگر خیلی کم باشد، تأثیر محسوسی دیده نخواهد شد. به همین دلیل در نرم‌افزارها و کتابخانه‌هایی مثل OpenCV ، Unsharp Masking روشی انعطاف‌پذیر و پرکاربرد برای افزایش وضوح و برجسته‌سازی جزئیات تصاویر به حساب می‌آید.

```
image = cv2.imread('images/plate1.jpg')

kernel1 = np.array([[0,0,0],
                    [0,2,0],
                    [0,0,0]])

kernel2 = 1/9*np.array([[1,1,1],
                    [1,1,1],
                    [1,1,1]])

final_kernel = kernel1-kernel2
# applying different kernels to the input image
sharpened = cv2.filter2D(image, -1, final_kernel)

plt.figure(figsize=[12,7])
plt.subplot(121);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(122);plt.imshow(sharpened[...,::-1]);plt.title("Sharpened");
print(final_kernel)
```

این کد در حقیقت روشی برای **شارپ‌سازی تصویر بر اساس تکنیک Unsharp Masking** را پیاده‌سازی می‌کند. اساس این روش بر آن است که ابتدا نسخه‌ای محو از تصویر اصلی ایجاد شود و سپس با مقایسه‌ی آن با تصویر اصلی، بخش‌هایی که تغییرات سریع دارند (لبه‌ها و جزئیات) استخراج گردند. در این پیاده‌سازی، یک کرنل برای تقویت پیکسل مرکزی و کرنل دیگری برای محو کردن تصویر تعریف شده است. ترکیب این دو کرنل به صورت تفاضل، فیلتری را به وجود می‌آورد که قادر است جزئیات تصویر را برجسته‌تر نماید.

پس از ساخت این فیلتر، آن بر روی تصویر ورودی اعمال می‌شود و خروجی به صورت یک تصویر با وضوح بیشتر به دست می‌آید. در پایان، تصویر اصلی و تصویر شارپ‌شده در کنار یکدیگر نمایش داده می‌شوند تا مقایسه‌ی بصری امکان‌پذیر باشد. نتیجه‌ی نهایی این فرآیند، وضوح بیشتر لبه‌ها و شفاف‌تر شدن جزئیات تصویر است، در حالی که بخش‌های یکنواخت تصویر تقریباً بدون تغییر باقی می‌مانند. این روش یکی از شیوه‌های استاندارد و پرکاربرد در حوزه‌ی پردازش تصویر و بهبود کیفیت بصری محسوب می‌شود.

```
kernel1 = np.array([[0,0,0],
                    [0,2,0],
                    [0,0,0]])
```

این کرنل فقط روی پیکسل مرکزی تأکید می‌کند (ضریب ۲) و بقیه خانه‌ها صفر هستند. یعنی مقدار پیکسل اصلی را تقویت می‌کند.

```
kernel2 = 1/9*np.array([[1,1,1],
                    [1,1,1],
                    [1,1,1]])
```

این یک کرنل **میانگین‌گیر** است. همه‌ی عناصر برابر 19\tfrac{1}{9}91هستند و میانگین ۹ پیکسل اطراف را محاسبه می‌کنند. این همان فیلتر بلور یا محوکننده است.

```
final_kernel = kernel1-kernel2
```

با کم کردن کرنل بلور از کرنل مرکزی تقویت‌شده، کرنل جدیدی ساخته می‌شود که کارش این است:

- پیکسل اصلی را نگه دارد و تقویت کند.
- اثر همسایه‌ها (یعنی بخش محو) را کم کند.

این دقیقاً همان ایده‌ی **Unsharp Masking** است: *تصویر اصلی - تصویر محو*.

- `sharpened = cv2.filter2D(image, -1, final_kernel)`

با استفاده از کرنل نهایی، عمل **کانولوشن** روی تصویر انجام می‌شود و خروجی یک تصویر شارپ‌شده است.