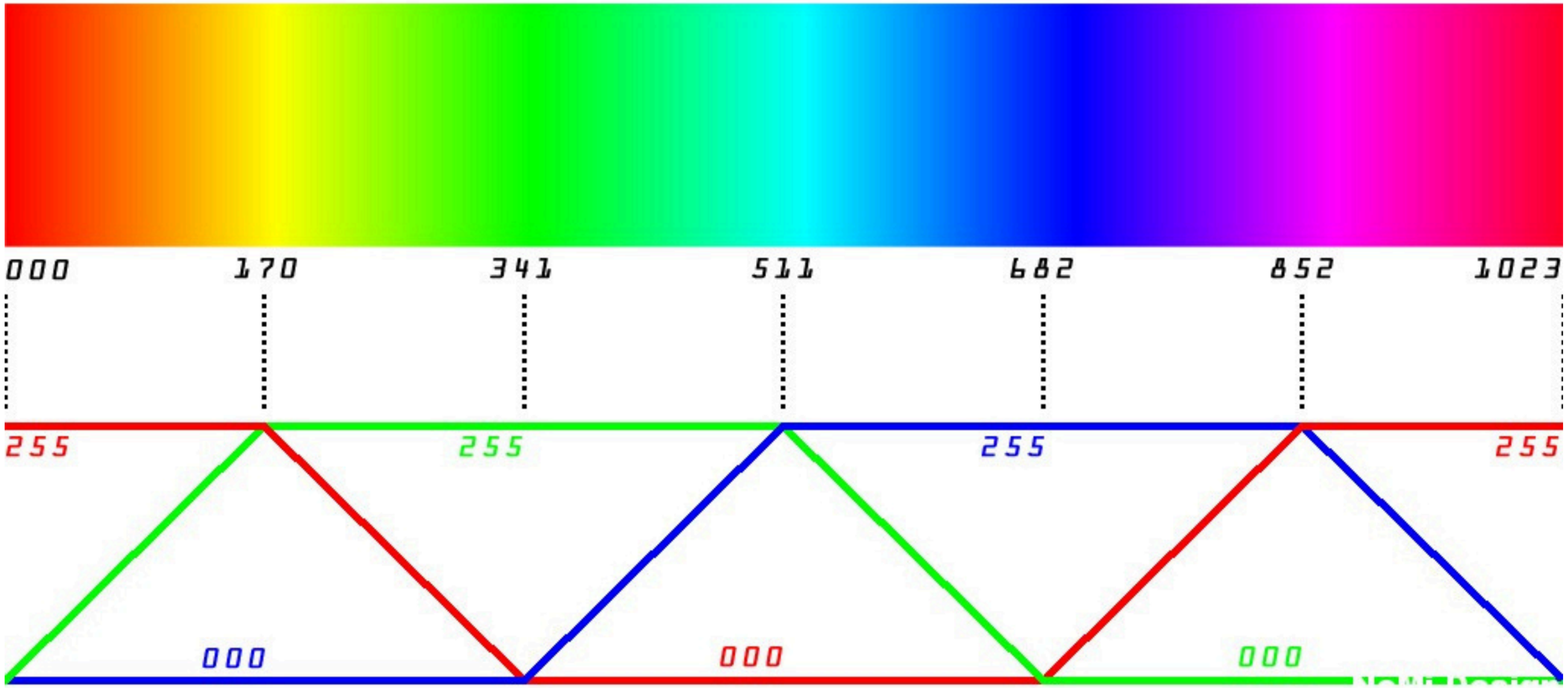
Week 2

DEEP LEARNING FOR COMPUTER VISION

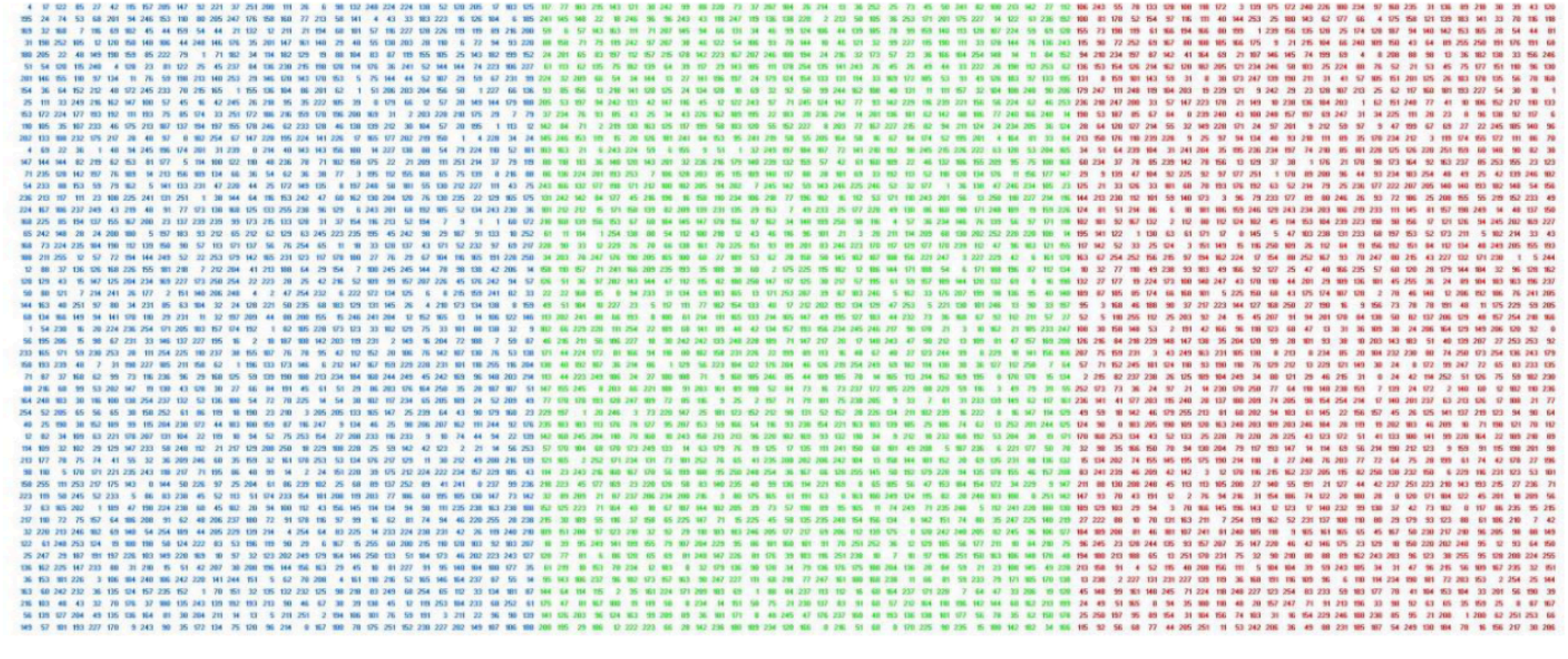
Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**

Understanding Images

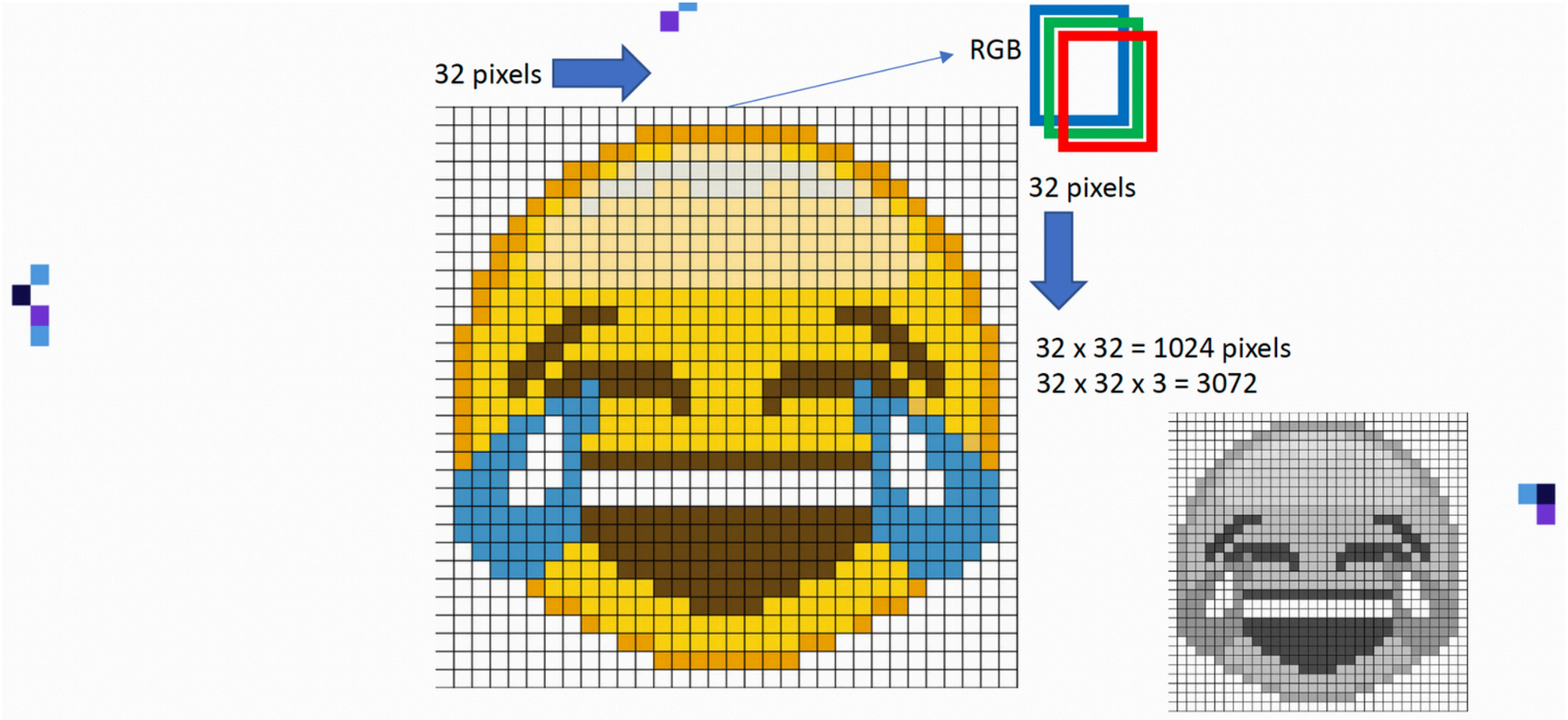
Image Basics



How do Computers ‘see’ Images?



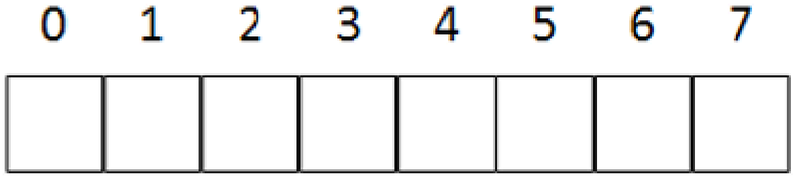
Pixels



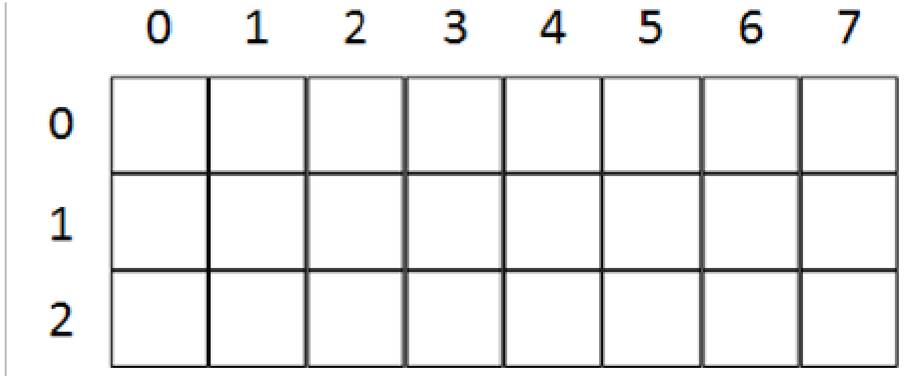
Digital Images Format

Images are stored in Multi-Dimensional Arrays

- A 1-Dimensional array looks like this



- A 2-Dimensional array looks like this



- A **3-Dimensional** array looks like this

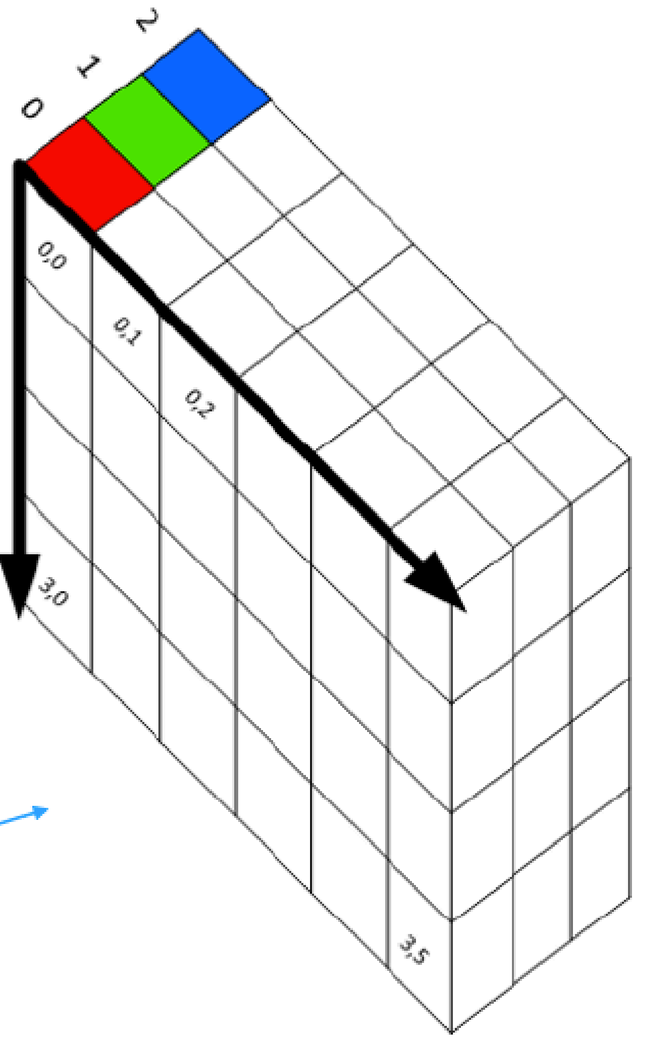
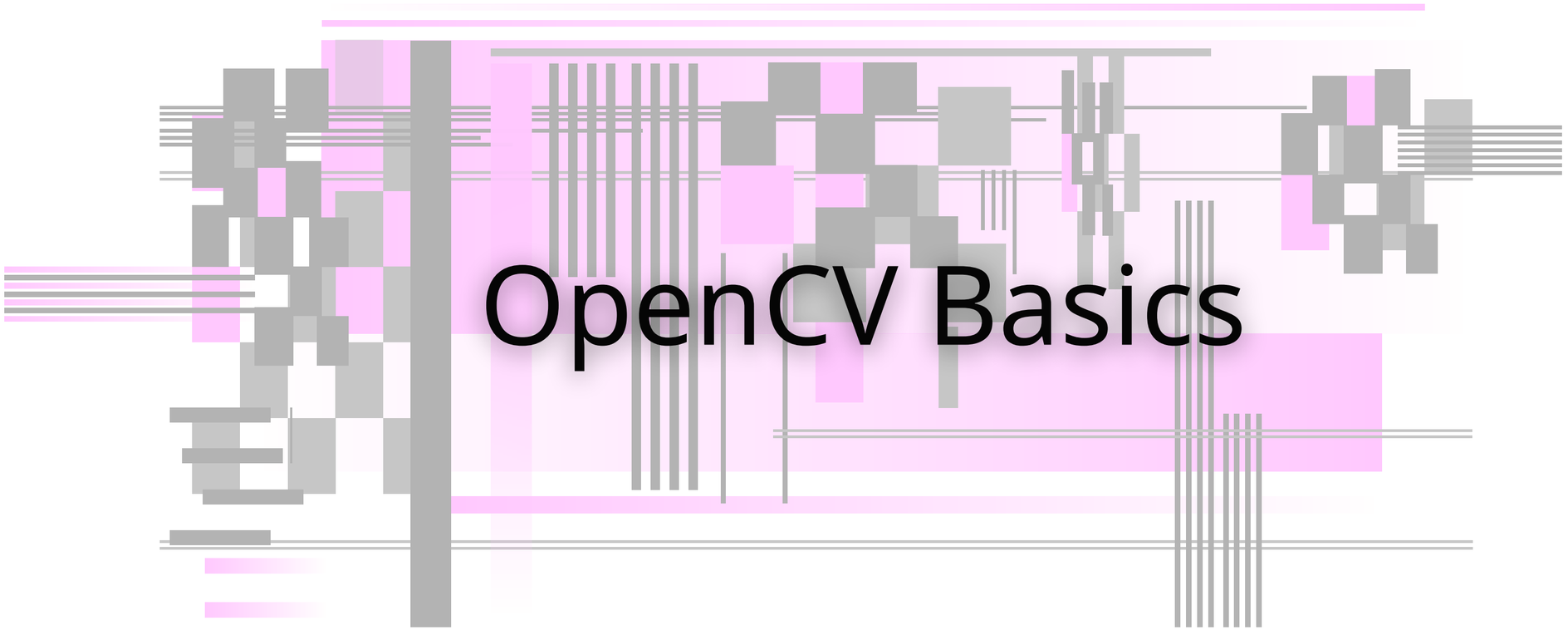


Image File Formats


OpenCV Basics

OpenCV Basics: Reading & Displaying Images

Displaying Images BGR

```
from matplotlib import pyplot as plt

image = cv2.imread('./images/Taj_Mahal.jpg')
plt.imshow(image)
plt.axis('off')
plt.show()
```

✓ 0.1s

```
from matplotlib import pyplot as plt

# Show the image with matplotlib without axis
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

✓ 0.3s

Loading Images

```
# Load an image using 'imread' specifying the path to image
image = cv2.imread('./images/Taj_Mahal.jpg')
image.shape
```

✓ 0.0s

```
(712, 796, 3)
```

Saving Images

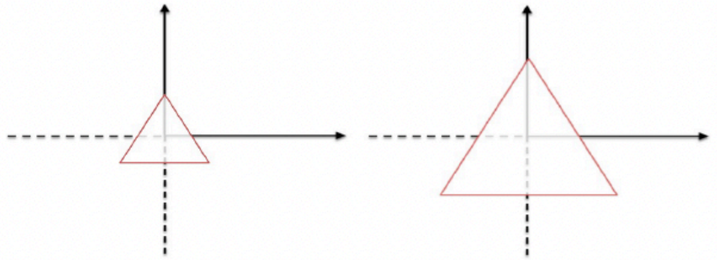
```
cv2.imwrite('output.jpg', image)
```

```
True
```

```
cv2.imwrite('output.png', image)
```

OpenCV — Resize an Image

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

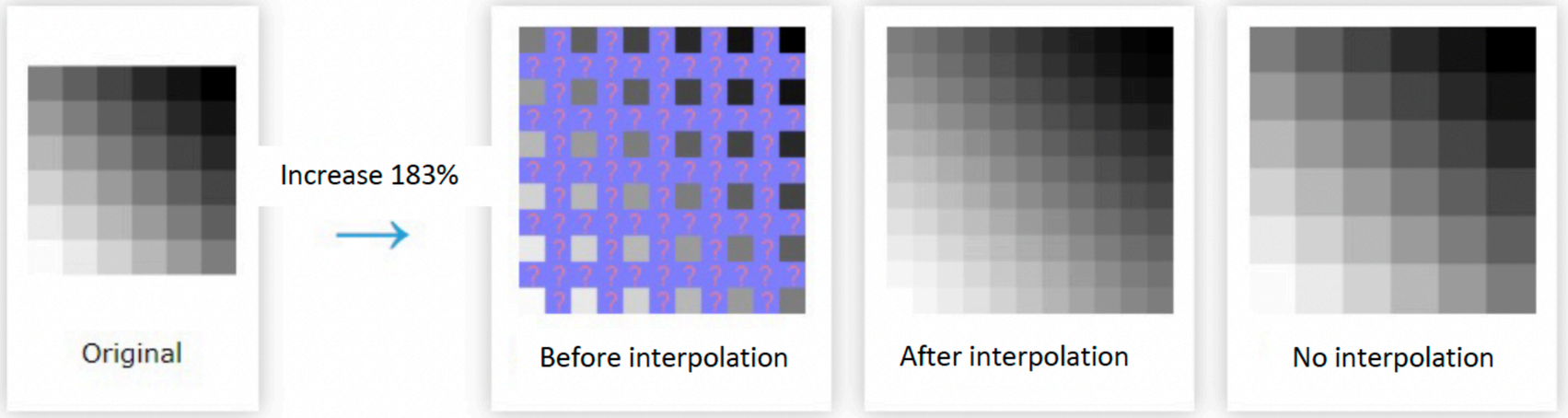


Scale factor:

>1 increase

<1 decrease

sx = sy uniform scale factor (no distortion)



OpenCV — Resize an Image

```
# load our input image
image = cv2.imread('./images/Taj_Mahal.jpg')
print("width : {} pixels, Height : {} pixels".format(image.shape[1], image.shape[0]))
```

```
width : 796 pixels, Height : 712 pixels
```

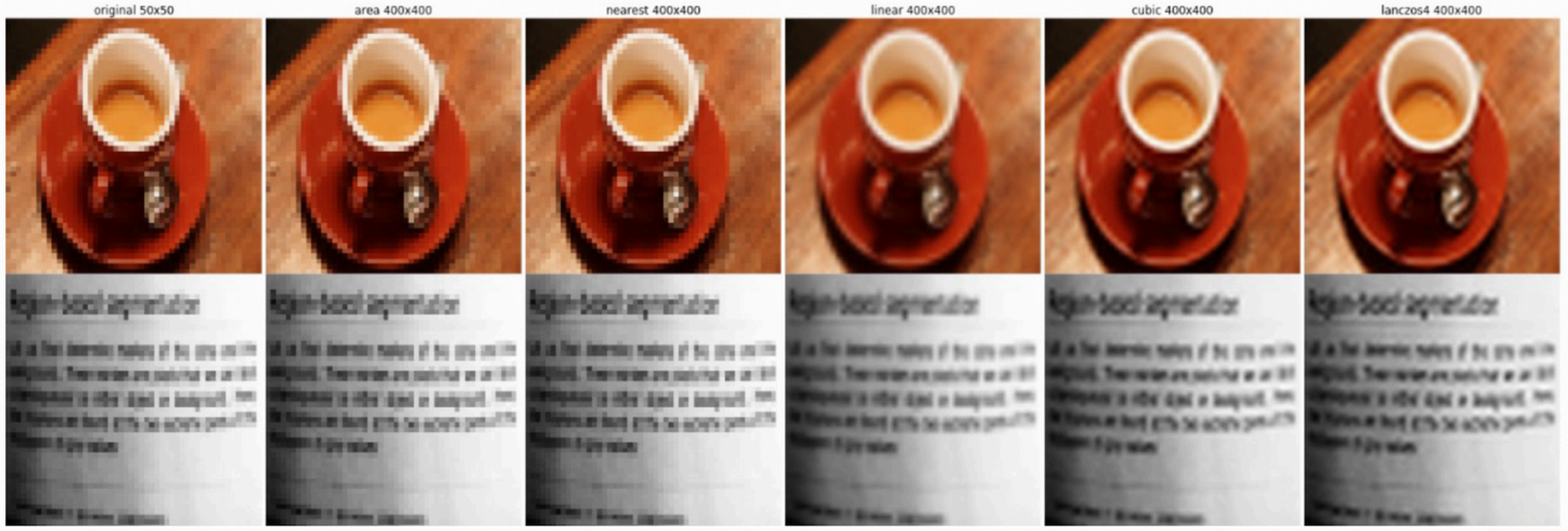
```
# Let's make our image 3/4 of its original size
image_scaled = cv2.resize(image, None, fx=0.75, fy=0.75)
imshow("0.75x Scaling - Linear Interpolation", image_scaled)

# Let's double the size of our image
img_scaled2 = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
imshow("2x Scaling - Inter Cubic", img_scaled2)

# Let's double the size of our image using inter_nearest interpolation
img_scaled3 = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_NEAREST)
imshow("2x Scaling - Inter Nearest", img_scaled3)

# Let's skew the re-sizing by setting exact dimensions
img_scaled4 = cv2.resize(image, (900, 400), interpolation=cv2.INTER_AREA)
imshow("Scaling - Inter Area", img_scaled4)
```

OpenCV options

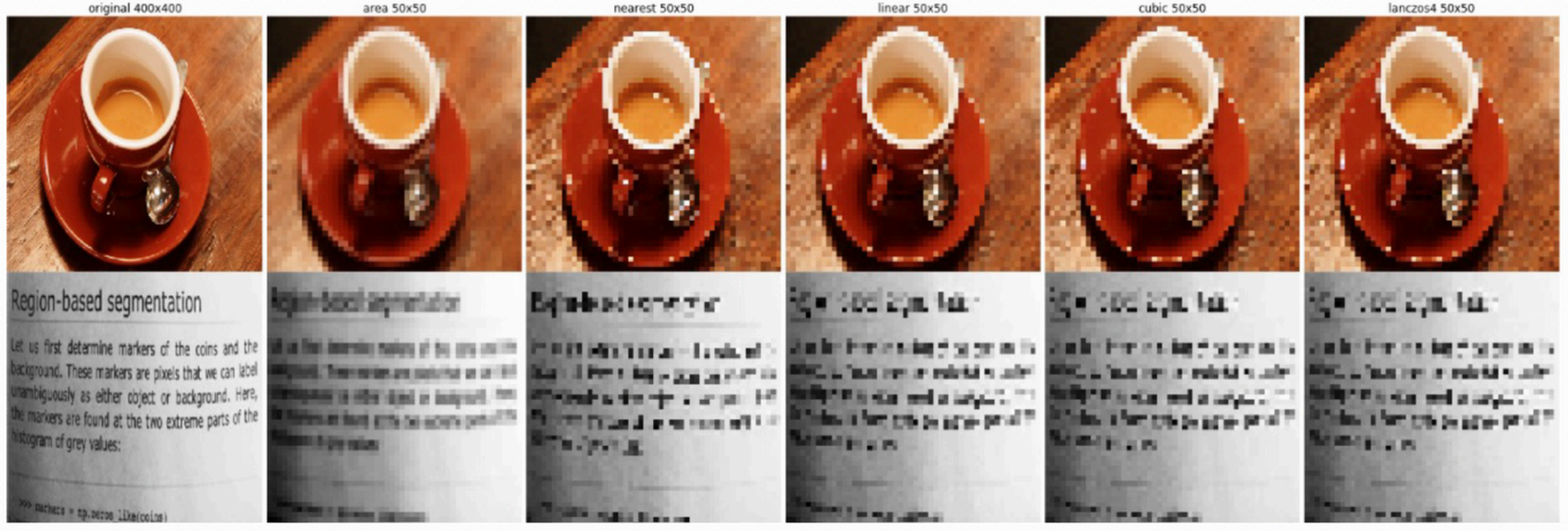
- **INTER_NEAREST** - a nearest neighbor interpolation. It is widely used because it is the fastest
- **INTER_LINEAR** - a bilinear interpolation (it's used by default), generally good for zooming in and out of images
- **INTER_AREA** - uses the pixel area ratio. May be a preferred method for image reduction as it provides good results
- **INTER_CUBIC** - bicubic (4x4 neighboring pixels). It has better results
- **INTER_LANCZOS4** - Lanczos interpolation (8x8 neighboring pixels). Among these algorithms, it is the one with the best quality results

Comparison between interpolation algorithms – increasing the size

original (50x50) area (400x400) nearest (400x400) linear (400x400) cubic (400x400) lanczos4 (400x400)



original (400x400) area (50x50) nearest (50x50) linear (50x50) cubic (50x50) lanczos4 (50x50)



Interpolation of OpenCV

1. INTER_NEAREST (Nearest Neighbor Interpolation)

- **หลักการทำงาน:** วิธีนี้จะใช้ค่า pixel ที่ใกล้เคียงที่สุดโดยตรงในการคำนวณ ไม่มีการเฉลี่ยหรือคำนวณค่าใหม่
- **จุดเด่น:**
  - **ความเร็วสูง:** เป็นวิธีที่เร็วที่สุด เพราะไม่มีการคำนวณซับซ้อน
  - **เหมาะสำหรับงานที่ไม่ต้องการคุณภาพสูง** เช่น งานที่เน้นความเร็วมากกว่าคุณภาพของภาพ
- **จุดด้อย:**
  - **คุณภาพต่ำ:** เมื่อนำมาใช้กับการขยายภาพ อาจทำให้ภาพมีลักษณะเป็นบล็อกหรือหยาบ
  - **ไม่มีความนุ่มนวล:** เมื่อเทียบกับวิธีอื่น ภาพจะดูแข็งและไม่มีการเปลี่ยนแปลงสีที่ราบเรียบ

2. INTER_LINEAR (Bilinear Interpolation)

- **หลักการทำงาน:** วิธีนี้จะเฉลี่ยค่าน้ำหนักจาก 4 pixel ที่ใกล้เคียงที่สุด ซึ่งทำให้การเปลี่ยนแปลงมีความนุ่มนวลกว่าวิธี Nearest Neighbor
- **จุดเด่น:**
  - **เหมาะสำหรับการซูมเข้าและออก:** ให้ผลลัพธ์ที่มีความนุ่มนวลพอสมควร เหมาะสำหรับการซูมเข้าและออกภาพ
  - **สมดุลระหว่างความเร็วและคุณภาพ:** เร็วกว่าวิธีที่ซับซ้อนกว่า เช่น Bicubic หรือ Lanczos4
- **จุดด้อย:**
  - **เสียรายละเอียดบางส่วน:** อาจทำให้รายละเอียดในภาพหายไปบ้างเล็กน้อย
  - **ไม่เหมาะสำหรับการขยายภาพขนาดใหญ่:** การขยายภาพมาก ๆ อาจเกิดการเบลอ

Interpolation of OpenCV

3. **INTER_AREA (Area-Based Interpolation)**
   - **หลักการทำงาน:** ใช้หลักการของพื้นที่ในภาพเพื่อนำมาคำนวณค่าพิกเซลใหม่ วิธีนี้เหมาะกับการลดขนาดภาพ
   - **จุดเด่น:**
     - **ให้คุณภาพสูงในการย่อภาพ:** เมื่อใช้ลดขนาดภาพ จะให้ผลลัพธ์ที่ดีเยี่ยมและลดการเกิดอาร์ติแฟกต์
     - **รักษารายละเอียดได้ดี:** เนื่องจากการคำนวณครอบคลุมพื้นที่ภาพ ทำให้ได้รายละเอียดที่ชัดเจนเมื่อย่อภาพ
   - **จุดด้อย:**
     - **ช้า:** มีความซับซ้อนสูง ทำให้ใช้เวลานานกว่า
     - **ไม่เหมาะสำหรับการขยายภาพ:** ไม่ให้ผลลัพธ์ที่ดีเมื่อใช้ในการขยาย

4. **INTER_CUBIC (Bicubic Interpolation)**
   - **หลักการทำงาน:** วิธีนี้จะใช้ค่าเฉลี่ยของ pixel รอบข้าง 16 จุด (4x4) เพื่อคำนวณค่า pixel ใหม่ โดยใช้การคำนวณที่ซับซ้อนกว่า Bilinear
   - **จุดเด่น:**
     - **ให้คุณภาพดีขึ้น:** ภาพที่ได้จะมีความนุ่มนวลและคมชัด เหมาะสำหรับการขยายภาพ
     - **เหมาะสำหรับการขยายภาพ:** สามารถรักษารายละเอียดของภาพได้ดีกว่าวิธี Bilinear
   - **จุดด้อย:**
     - **ช้ากว่า Bilinear:** ใช้เวลามากกว่าเพราะการคำนวณที่ซับซ้อนกว่า
     - **อาจเกิดอาร์ติแฟกต์ได้เล็กน้อย:** เมื่อขยายภาพอาจมีขอบเส้นที่ดูคล้าย halo เล็กน้อย

5. **INTER_LANCZOS4 (Lanczos Interpolation with 8x8 Neighbors)**
   - **หลักการทำงาน:** ใช้การคำนวณจาก Sinc function โดยใช้ pixel รอบข้าง 8x8 จุด (รวม 64 pixel) วิธีนี้ให้คุณภาพที่ดีที่สุดใน OpenCV แต่ต้องใช้เวลานาน
   - **จุดเด่น:**
     - **คุณภาพสูงสุด:** เป็นวิธีที่ดีที่สุดสำหรับการขยายและย่อภาพ โดยรักษารายละเอียดและลดการเกิดอาร์ติแฟกต์ได้ดี
     - **การเปลี่ยนแปลงที่นุ่มนวล:** ให้ความนุ่มนวลที่มากที่สุด ทำให้ภาพดูเป็นธรรมชาติ
   - **จุดด้อย:**
     - **ช้ามาก:** เป็นวิธีที่ใช้เวลามากที่สุด เหมาะกับงานที่ไม่ต้องการ real-time
     - **อาจเกิดอาร์ติแฟกต์ประเภท ringing:** เมื่อใช้งานกับภาพที่มีความคมสูง อาจเกิดริ้วรอบๆ ขอบของภาพ

สรุปการเลือกใช้งาน

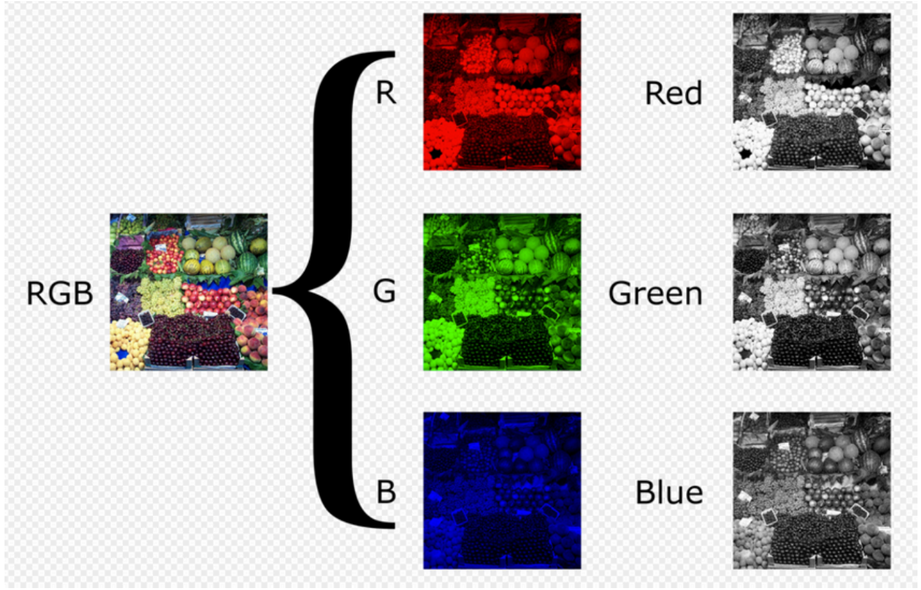
วิธี	เหมาะสำหรับ	จุดเด่น	จุดด้อย
INTER_NEAREST	งานที่เน้นความเร็วมากกว่าคุณภาพ	เร็วที่สุด ใช้ทรัพยากรต่ำ	ภาพแตกเมื่อขยาย
INTER_LINEAR	การซูมเข้า-ออกทั่วไป	คุณภาพพอใช้และเร็ว	อาจทำให้ภาพเบลอเล็กน้อย
INTER_AREA	การย่อภาพ	คุณภาพสูงเมื่อย่อภาพ	ช้า และไม่เหมาะกับการขยาย
INTER_CUBIC	การขยายภาพคุณภาพสูง	ภาพคมชัด เหมาะกับการขยาย	ช้ากว่า Bilinear และอาจมี halo
INTER_LANCZOS4	งานที่ต้องการคุณภาพสูงสุด	ภาพคมและเปลี่ยนแปลงนุ่มนวล	ช้าที่สุด อาจเกิด ringing รอบขอบภาพ

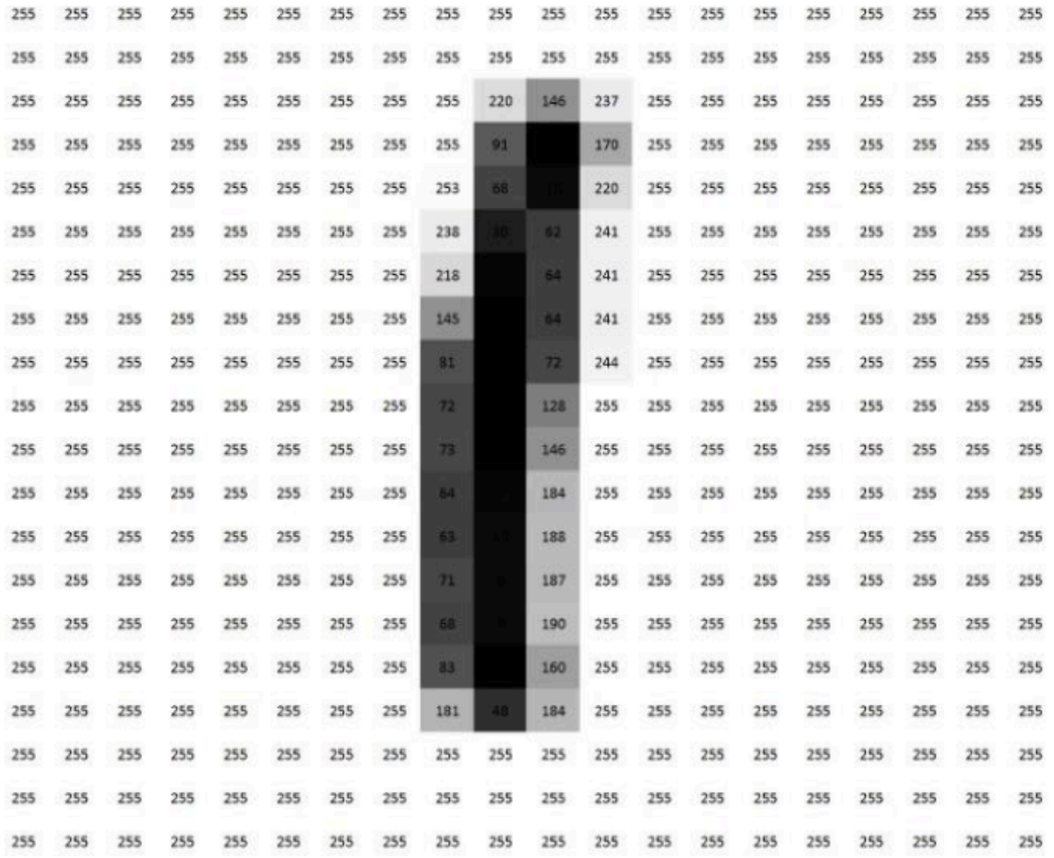
การเลือกใช้แต่ละวิธีควรพิจารณาจากความต้องการของงาน หากเน้นความเร็ว INTER_NEAREST หรือ INTER_LINEAR จะเป็นตัวเลือกที่ดี แต่หากต้องการคุณภาพสูง ควรเลือก INTER_CUBIC หรือ INTER_LANCZOS4 ตามลำดับ

"ฮาโล" (halo) หมายถึงปรากฏการณ์ที่เกิดขอบสว่างหรือมืดรอบๆ ขอบของวัตถุในภาพ ซึ่งเป็นผลมาจากการคำนวณค่าเฉลี่ยของพิกเซลรอบข้างที่อาจทำให้เกิดการเน้นขอบเกินจริง ส่งผลให้ภาพมีลักษณะขอบที่ไม่เป็นธรรมชาติ

Grayscale Image

Sometimes referred to as black and white image





Luma coding in video systems [edit]

Main article: luma (video)

For images in color spaces such as Y'UV and its relatives, which are used in standard color TV and video systems such as PAL, SECAM, and NTSC, a nonlinear luma component (Y') is calculated directly from gamma-compressed primary intensities as a weighted sum, which, although not a perfect representation of the colorimetric luminance, can be calculated more quickly without the gamma expansion and compression used in photometric/colorimetric calculations. In the Y'UV and Y'IQ models used by PAL and NTSC, the rec601 luma (Y') component is computed as

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

where we use the prime to distinguish these nonlinear values from the sRGB nonlinear values (discussed above) which use a somewhat different gamma compression formula, and from the linear RGB components. The ITU-R BT.709 standard used for HDTV developed by the ATSC uses different color coefficients, computing the luma component as

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'.$$

Although these are numerically the same coefficients used in sRGB above, the effect is different because here they are being applied directly to gamma-compressed values rather than to the linearized values. The ITU-R BT.2100 standard for HDR television uses yet different coefficients, computing the luma component as

$$Y' = 0.2627R' + 0.6780G' + 0.0593B'.$$

The Formula

```
Grayscale = 0.299R + 0.587G + 0.114B
```

Where:

- R = Red channel value (0-255)
- G = Green channel value (0-255)
- B = Blue channel value (0-255)

Why These Weights?

• Green (0.587): Human eyes are most sensitive to green

• Red (0.299): Second most impactful color

• Blue (0.114): Least impact on perceived brightness

Example Calculation

```
For a pink pixel:
R = 255
G = 192
B = 203
```

Calculation:

(0.299 × 255) +

(0.587 × 192) +

(0.114 × 203)

= 76.245 + 112.704 + 23.142

= 212 (rounded)

Key Points

• Result is rounded to nearest integer

• Output range is 0 (black) to 255 (white)

• Same value applied to all RGB channels

Grayscaling Images

```
import cv2
from matplotlib import pyplot as plt
```

✓ 0.4s

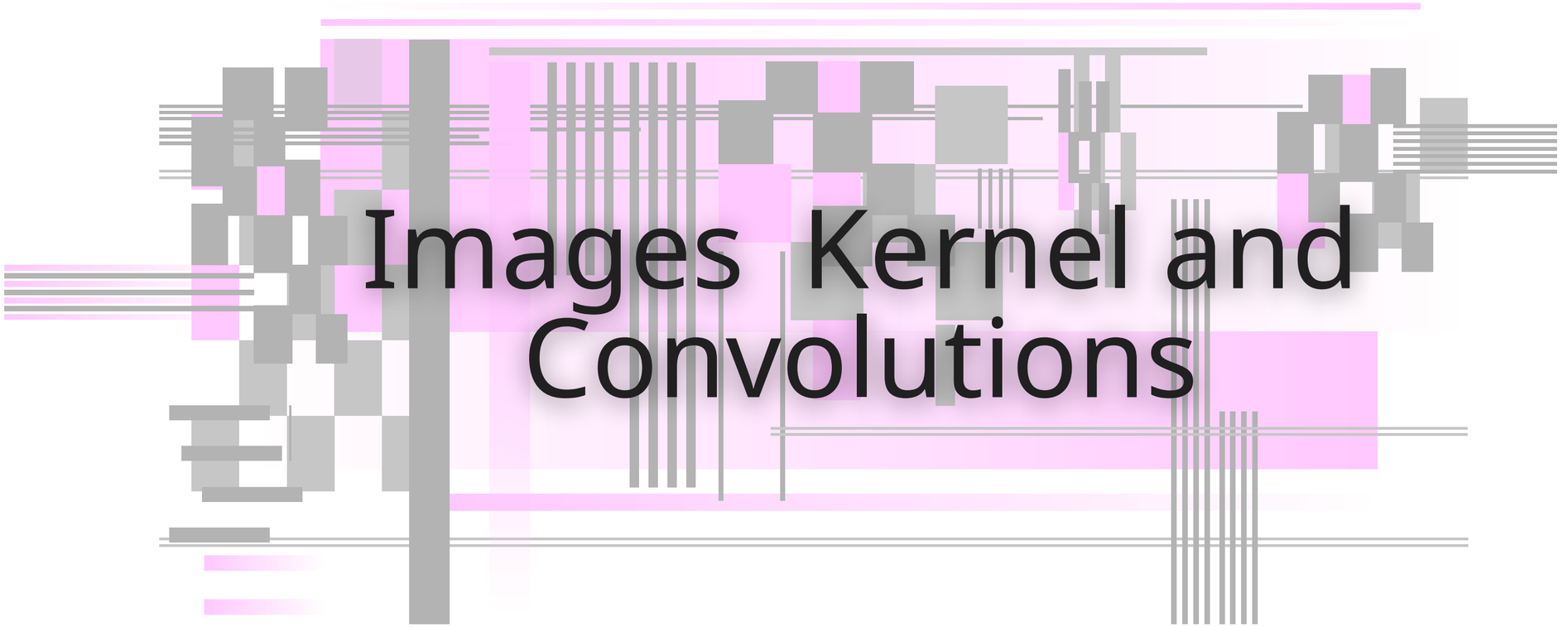
```
# Load our input image
image = cv2.imread('./images/castara.jpeg')
image.shape
```

✓ 0.0s

```
(1280, 960, 3)
```

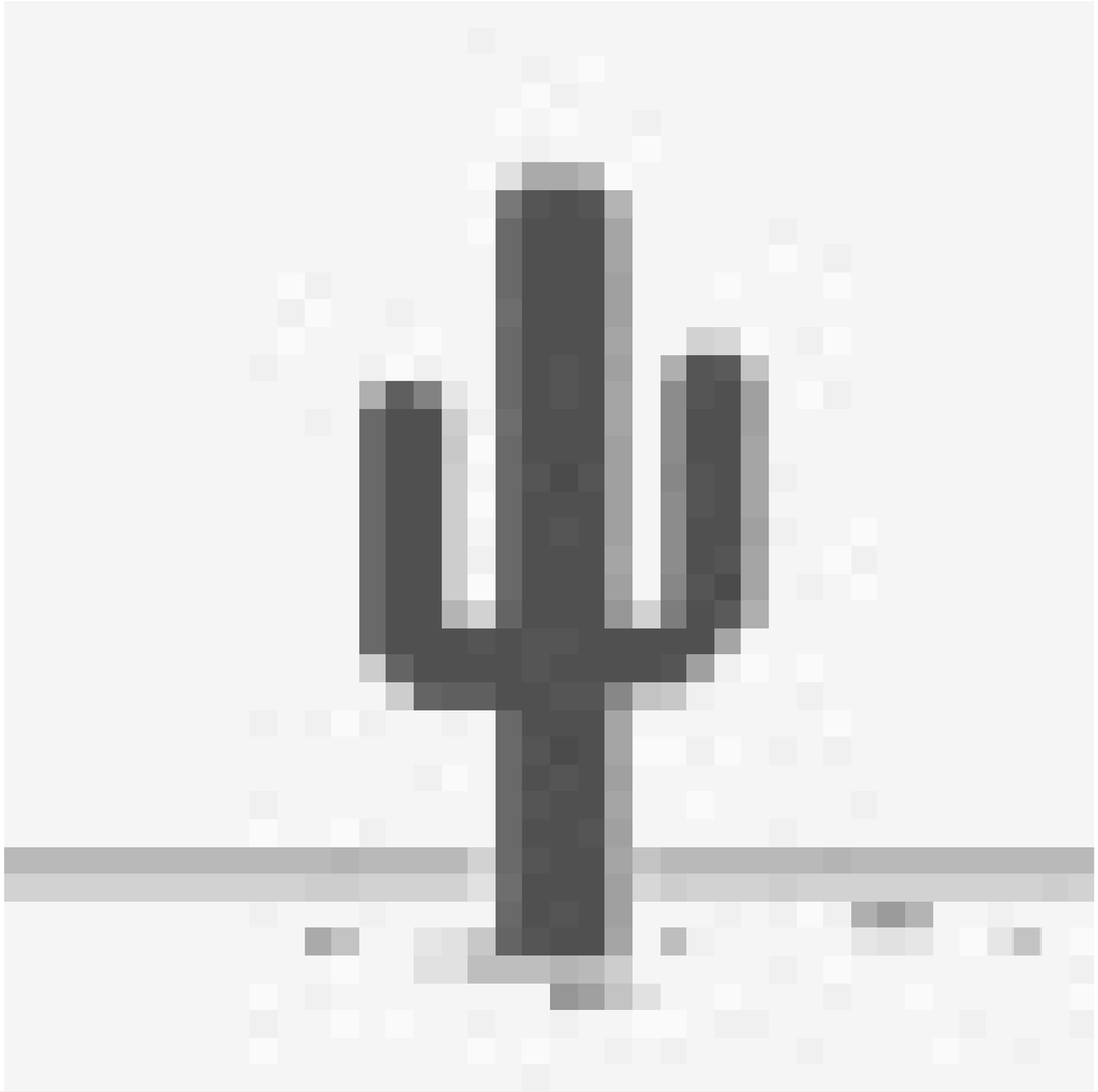
```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray_image, cv2.COLOR_GRAY2RGB))
plt.axis('off')
plt.title('Grayscale Image')
plt.show()
```

✓ 0.1s



Images Kernel and Convolutions

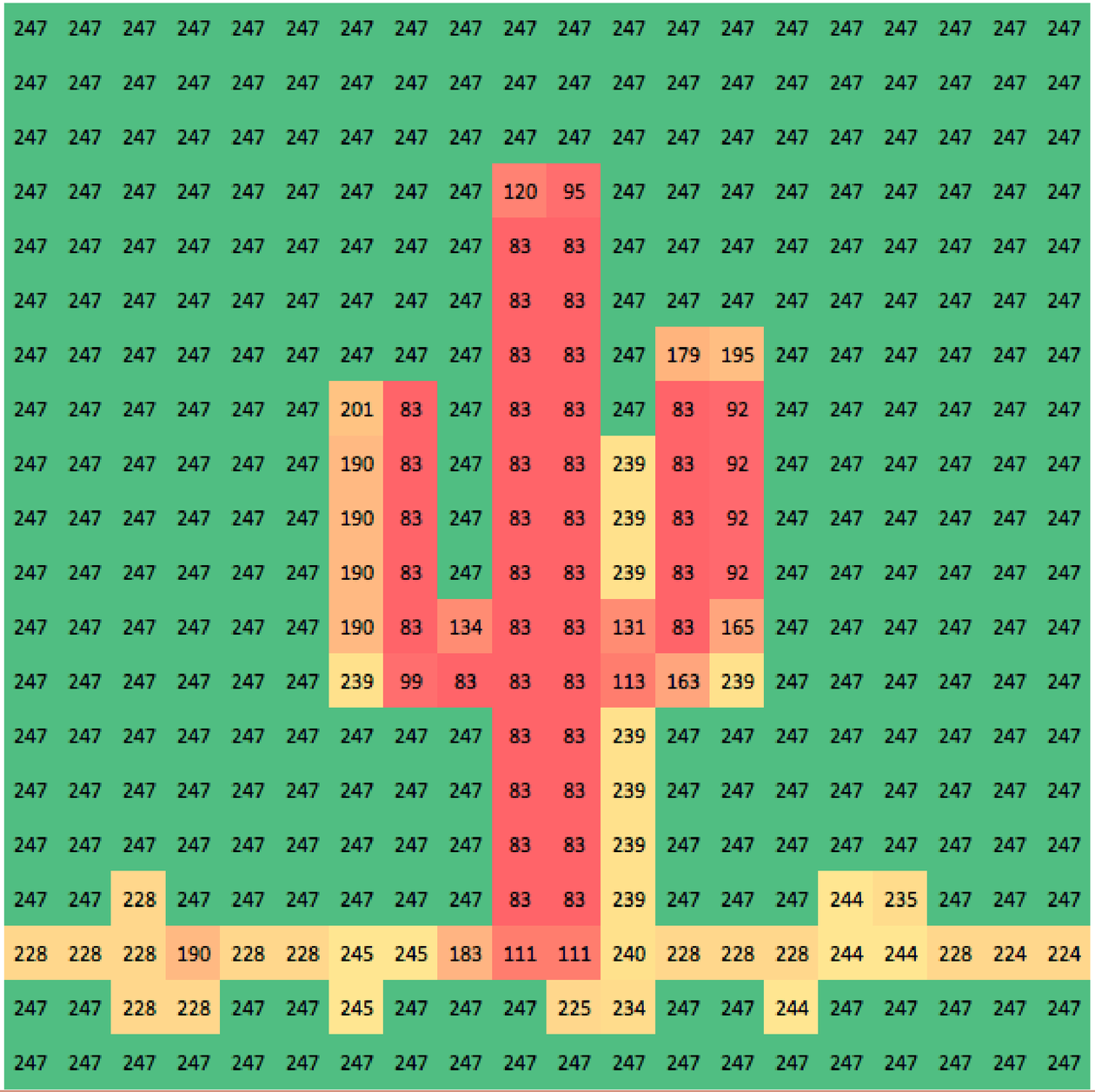
Images & Kernel Convolutions



Images & Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Images & Kernel Convolutions



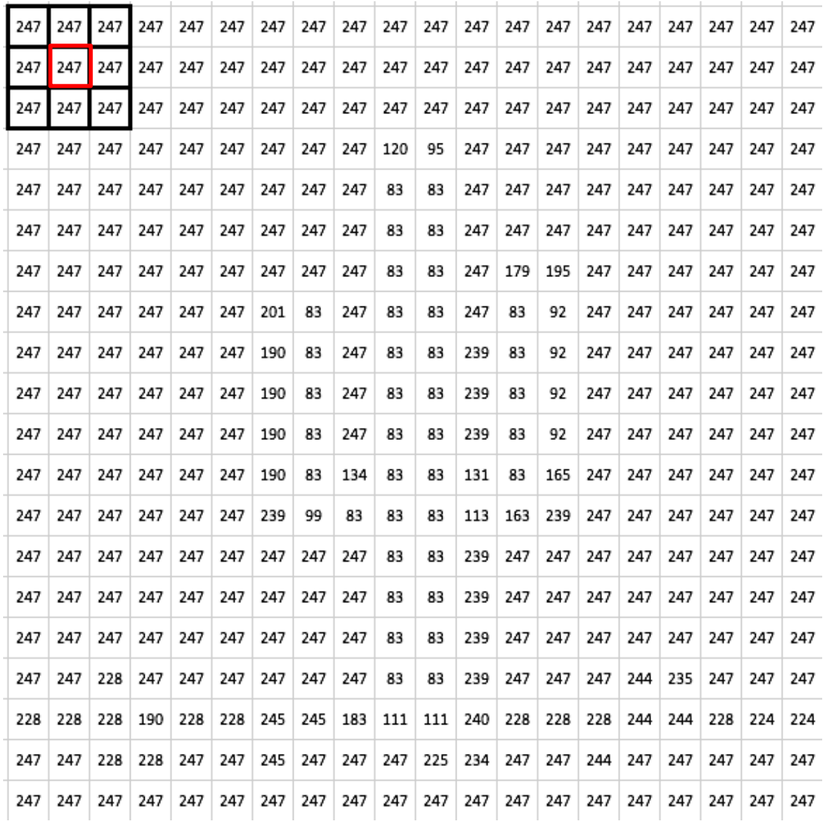
Kernel Convolutions

Kernel is an MxN matrix

3x3 Kernel Example

1	1	1
1	1	1
1	1	1

Kernel Convolutions



Window

247	247	247
247	247	247
247	247	247

x

Kernel

1	1	1
1	1	1
1	1	1

Sum = 9

=

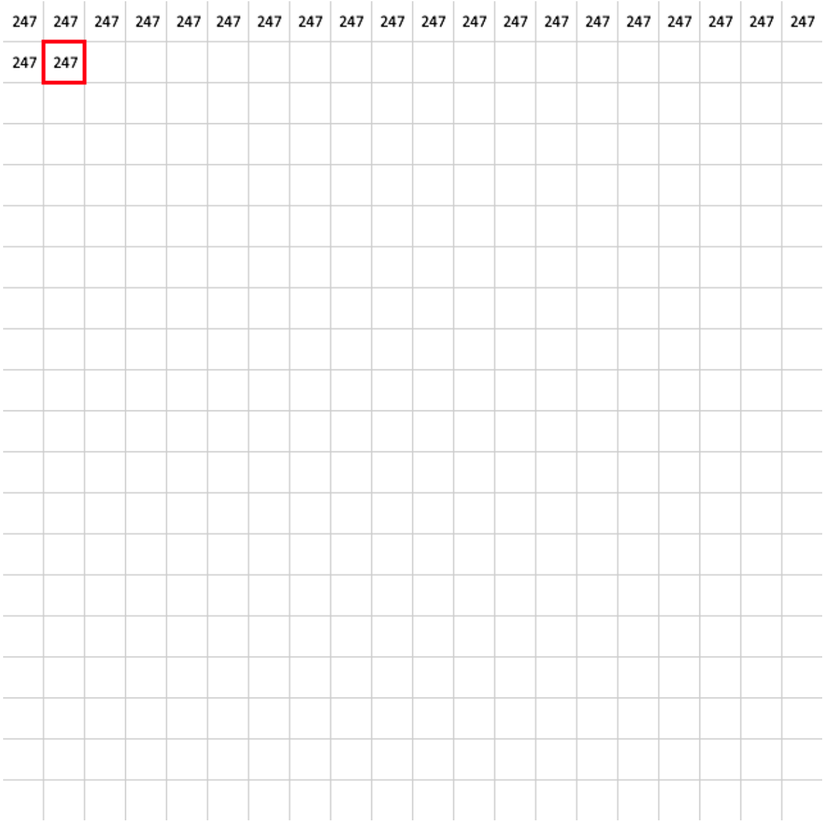
Multiplied

247	247	247
247	247	247
247	247	247

Sum = 2223

New Pixel Value

2223 / 9 = **247**



Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Window

247	247	247
247	247	247
247	247	247

x

Kernel

1	1	1
1	1	1
1	1	1

Sum = 9

=

Multiplied

247	247	247
247	247	247
247	247	247

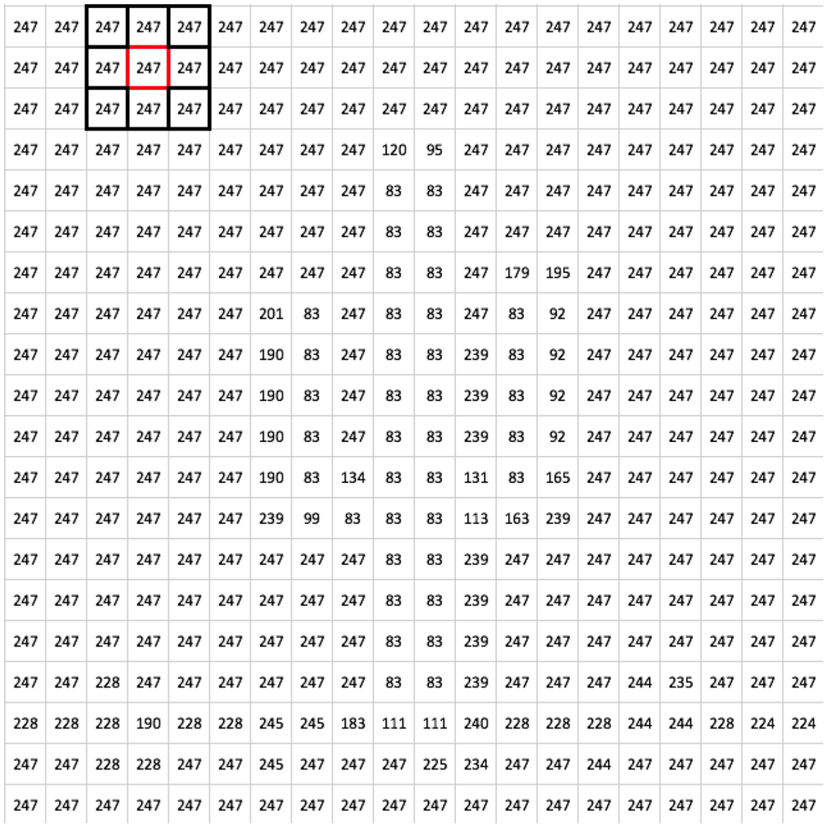
Sum = 2223

New Pixel Value

2223 / 9 = **247**

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247																

Kernel Convolutions



Window

247	247	247
247	247	247
247	247	247

x

Kernel

1	1	1
1	1	1
1	1	1

Sum = 9

=

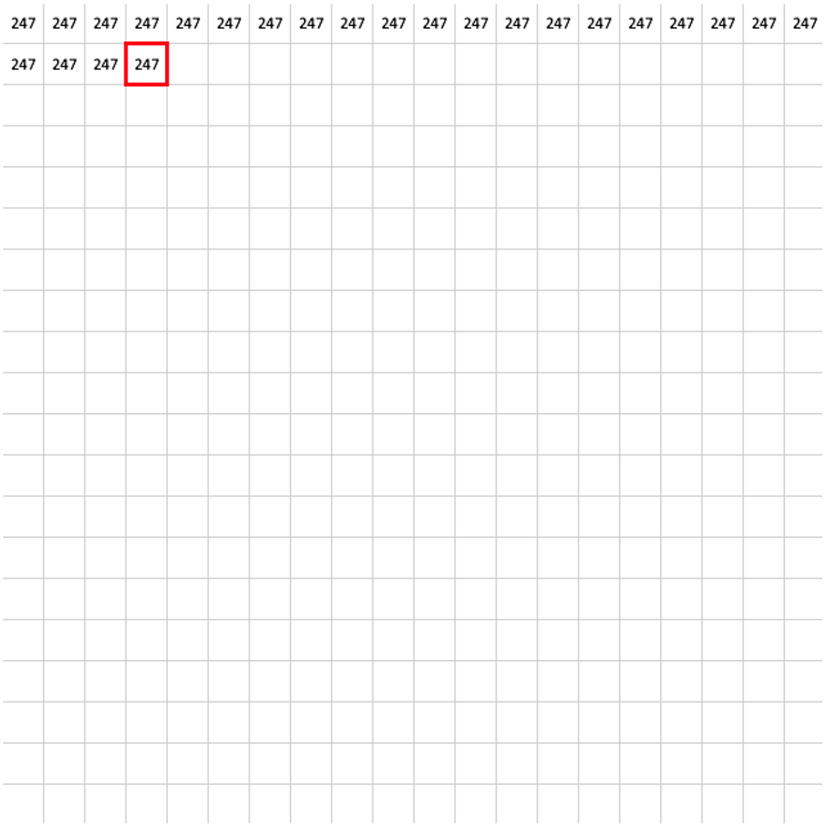
Multiplied

247	247	247
247	247	247
247	247	247

Sum = 2223

New Pixel Value

2223 / 9 = **247**



Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Window

247	247	120
247	247	83
247	247	83

x

Kernel

1	1	1
1	1	1
1	1	1

Sum = 9

=

Multiplied

247	247	120
247	247	83
247	247	83

Sum = 1768

New Pixel Value

1768 / 9 = 196.4

196

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	230	216	216	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	214	180	180	218	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	196										

Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Window

247	120	95
247	83	83
247	83	83

x

Kernel

1	1	1
1	1	1
1	1	1

Sum = 9

=

Multiplied

247	120	95
247	83	83
247	83	83

Sum = 1288

New Pixel Value

1288 / 9 = 143.1

143

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	230	216	216	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	214	180	180	218	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	196	143									

Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Window

247	247	
247	247	

x

Kernel

1	1	
1	1	

Sum = 4

=

Multiplied

247	247	
247	247	

Sum = 988

New Pixel Value

998 / 4 = 247

247

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	233	216	216	230	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	215	180	178	211	246	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	196	143	141	192	245	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	182	231	234	241	247	247	247	247	247
247	247	247	247	247	242	224	224	174	138	135	164	196	198	224	247	247	247	247	247
247	247	247	247	247	236	199	199	156	138	135	146	160	163	207	247	247	247	247	247
247	247	247	247	247	229	175	175	138	138	135	135	138	141	195	247	247	247	247	247
247	247	247	247	247	228	173	173	138	138	135	135	138	141	195	247	247	247	247	247
247	247	247	247	247	228	173	161	125	125	123	123	134	149	203	247	247	247	247	247
247	247	247	247	247	234	181	151	109	107	109	118	146	175	221	247	247	247	247	247
247	247	247	247	247	241	206	175	127	107	109	136	182	210	238	247	247	247	247	247
247	247	247	247	247	247	231	212	158	119	121	166	221	238	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	245	245	246	247	247
247	247	247	247	246	246	246	240	188	137	141	191	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	207	171	175	206	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	225	207	212	225	243	245	247	247	247	244	244	242
247	247	247	247	247	247	247	247	247	242	239	239	244	247	247	247	247	247	247	247

Kernel Convolutions

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

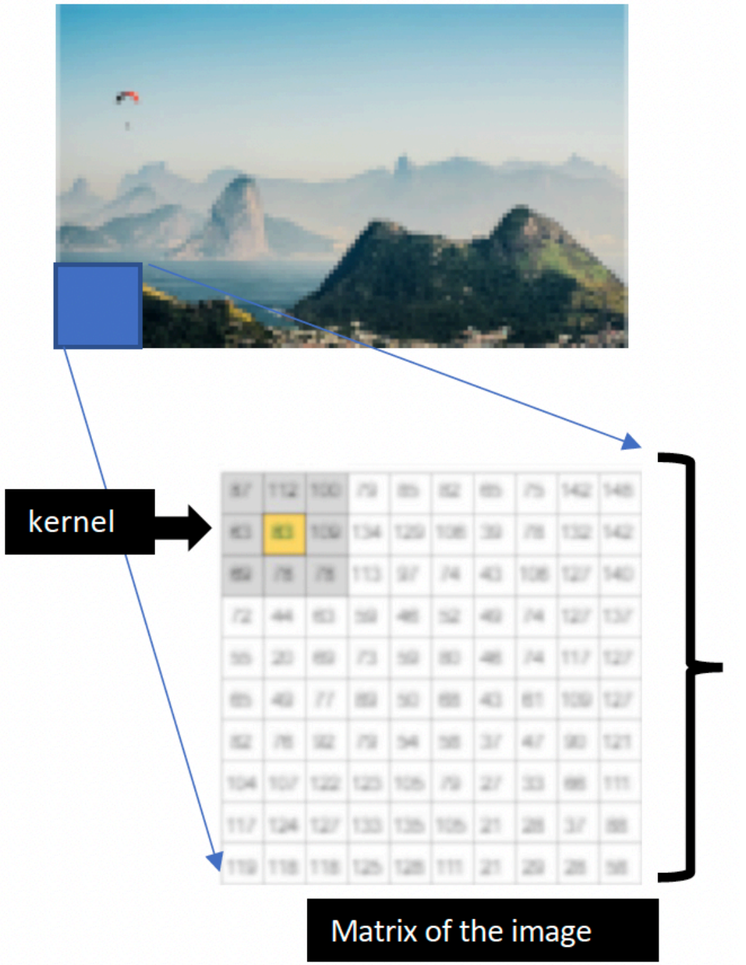
Kernel

1	1	1
1	1	1
1	1	1

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	233	216	216	230	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	215	180	178	211	246	247	247	247	247	247	247
247	247	247	247	247	247	247	247	196	143	141	192	245	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	182	231	234	241	247	247	247	247
247	247	247	247	247	242	224	224	174	138	135	164	196	198	224	247	247	247	247
247	247	247	247	247	236	199	199	156	138	135	146	160	163	207	247	247	247	247
247	247	247	247	247	229	175	175	138	138	135	135	138	141	195	247	247	247	247
247	247	247	247	247	228	173	173	138	138	135	135	138	141	195	247	247	247	247
247	247	247	247	247	228	173	161	125	125	123	123	134	149	203	247	247	247	247
247	247	247	247	247	234	181	151	109	107	109	118	146	175	221	247	247	247	247
247	247	247	247	247	241	206	175	127	107	109	136	182	210	238	247	247	247	247
247	247	247	247	247	247	231	212	158	119	121	166	221	238	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	245	245	246	247	247
247	247	247	247	246	246	246	240	188	137	141	191	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	207	171	175	206	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	225	207	212	225	243	245	247	247	247	244	244	242
247	247	247	247	247	247	247	247	247	242	239	239	244	247	247	247	247	247	247	247

CONVOLUTION

- Convolution is a mathematical operation performed on two matrices, which produces a third matrix that is the result of the operation
- The primary matrix is the image to be treated, and the treatment of this matrix of the original image is done by another matrix called “kernel”, or mask
- Depending on the kernel values it is possible to obtain filters of different types, such as blur and sharpen



Kernel Convolutions

Mean Blur

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	120	95	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	247	179	195	247	247	247	247	247
247	247	247	247	247	247	201	83	247	83	83	247	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	247	83	83	239	83	92	247	247	247	247	247
247	247	247	247	247	247	190	83	134	83	83	131	83	165	247	247	247	247	247
247	247	247	247	247	247	239	99	83	83	83	113	163	239	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	83	83	239	247	247	247	247	247	247	247
247	247	228	247	247	247	247	247	247	83	83	239	247	247	247	244	235	247	247	247
228	228	228	190	228	228	245	245	183	111	111	240	228	228	228	244	244	228	224	224
247	247	228	228	247	247	245	247	247	247	225	234	247	247	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247

Kernel

1	1	1
1	1	1
1	1	1

247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	233	216	216	230	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	215	180	178	211	246	247	247	247	247	247	247
247	247	247	247	247	247	247	247	196	143	141	192	245	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	182	231	234	241	247	247	247	247
247	247	247	247	247	242	224	224	174	138	135	164	196	198	224	247	247	247	247
247	247	247	247	247	236	199	199	156	138	135	146	160	163	207	247	247	247	247
247	247	247	247	247	229	175	175	138	138	135	135	138	141	195	247	247	247	247
247	247	247	247	247	228	173	173	138	138	135	135	138	141	195	247	247	247	247
247	247	247	247	247	228	173	161	125	125	123	123	134	149	203	247	247	247	247
247	247	247	247	247	234	181	151	109	107	109	118	146	175	221	247	247	247	247
247	247	247	247	247	241	206	175	127	107	109	136	182	210	238	247	247	247	247
247	247	247	247	247	247	231	212	158	119	121	166	221	238	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	245	245	246	247	247
247	247	247	247	246	246	246	240	188	137	141	191	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	207	171	175	206	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	225	207	212	225	243	245	247	247	247	244	244	242
247	247	247	247	247	247	247	247	247	242	239	239	244	247	247	247	247	247	247	247

Kernel Convolutions

Blur

Kernel

1	1	1
1	1	1
1	1	1

Kernel Convolutions

Sharpen

Kernel

0	-1	0
-1	5	-1
0	-1	0

Kernel Convolutions

Edge
Highlight

Kernel

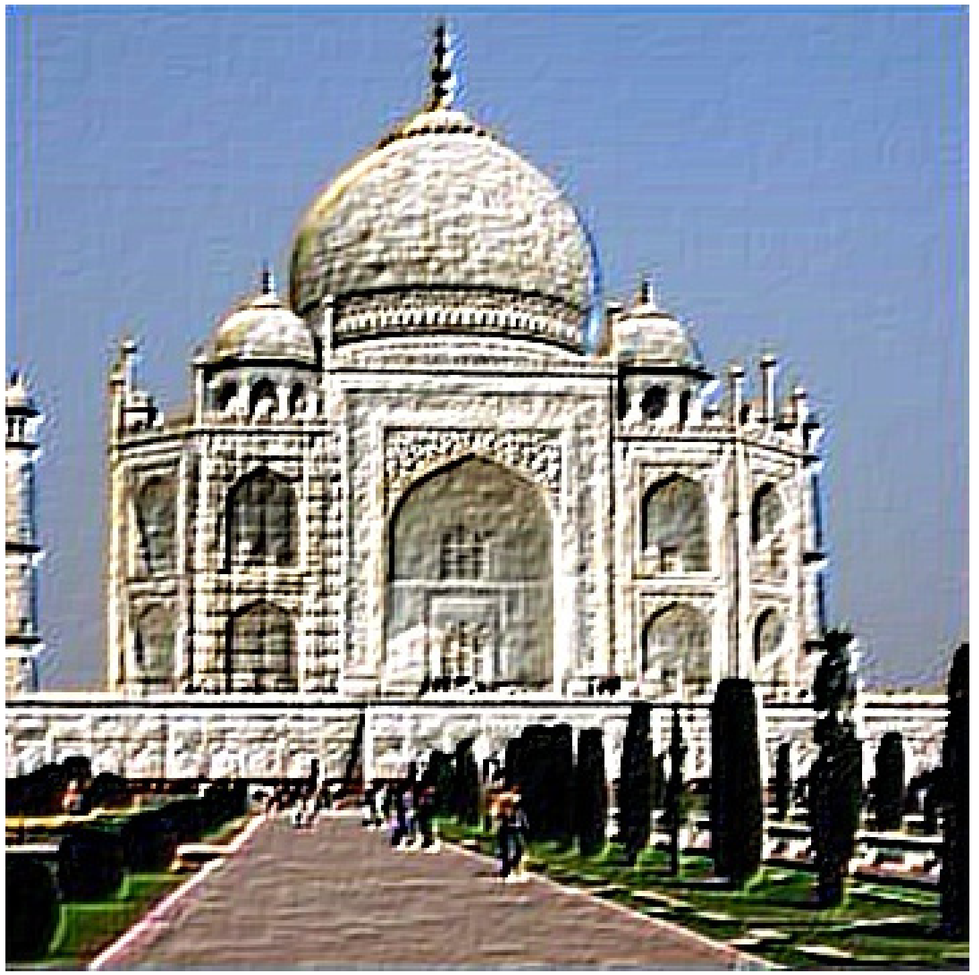
0	1	0
1	-4	1
0	1	0

Kernel Convolutions

Emboss

Kernel

-2	-1	0
-1	1	1
0	1	2



Kernel Convolutions

Horizontal “Derivative”

Kernel

1	0	-1
2	0	-2
1	0	-1

Kernel Convolutions

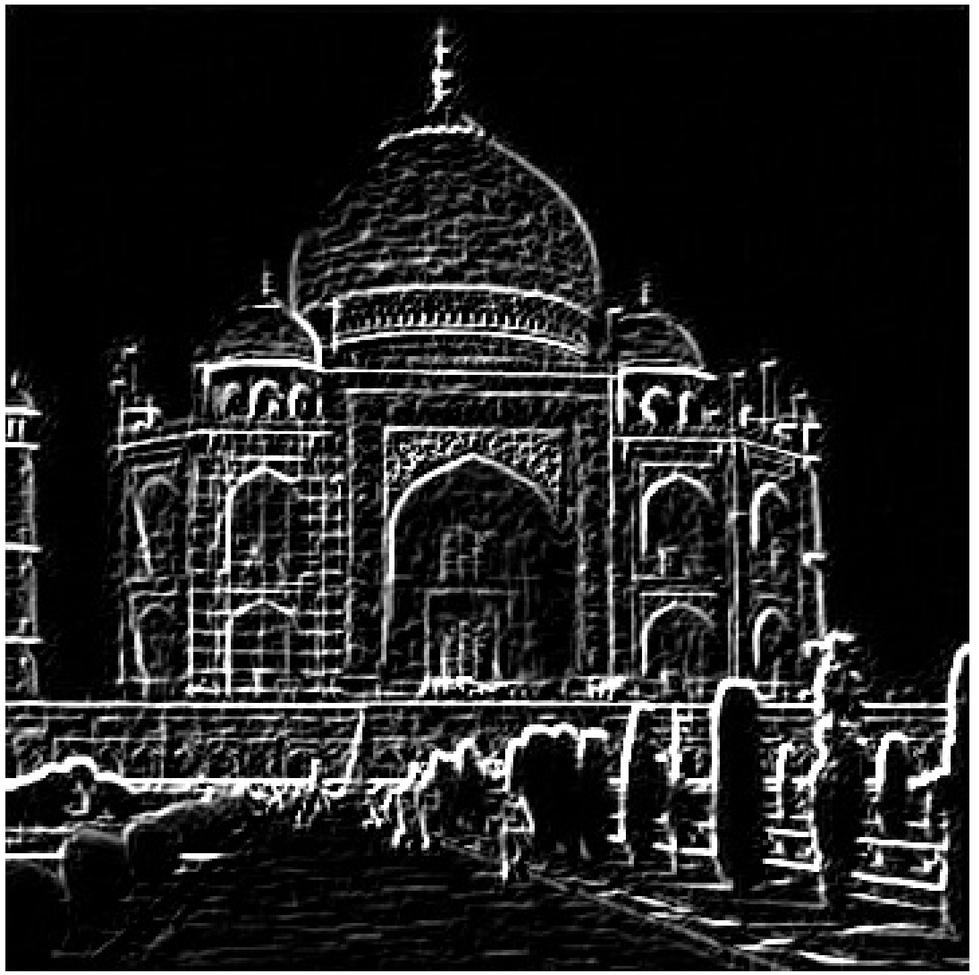
Vertical “Derivative”

Kernel

1	2	1
0	0	0
-1	-2	-1

Kernel Convolutions

Overlay them to get **Sobel Operator**



Gaussian filter

It is another linear spatial filter used for smoothing, noise reduction or blurring . Gaussian filtering is more effective at smoothing images. It works similar to human visual perception when processing on images. The kernel coefficients diminish with increasing distance from the kernel's center. Normally, the kernel matrix are considered symmetric and size of the filter is odd. The values inside the kernel are computed by the Gaussian function, which is as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Two dimensional gaussian function

In gaussian kernel, central pixels have a higher weighting than those on the periphery. The kernel for gaussian filter can be represented as

$$\begin{bmatrix} c & a & c \\ a & b & a \\ c & a & c \end{bmatrix}$$

General form of gaussian kernel (c<a<b)

A 3×3 Gaussian Kernel with standard deviation = 1,is as shown below.

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

source: geeksforgeeks

Laplacian of Gaussian Filter

A Laplacian filter is one of edge detectors used to compute the second spatial derivatives of an image. It measures the rate at which the first derivatives changes. In other words, Laplacian filter highlights the regions where the pixel intensities dramatically change. Due to this characteristic of Laplacian filter, it is often used to detect edges in an image. We will see how the filter find edges with visual illustration later.

The mathematical definition of Laplacian is as follows.

$$\begin{aligned} \textit{Laplacian } \nabla^2 f &= \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \\ &= f(x+1,y) + f(x-1,y) - 2f(x,y) + f(x,y+1) + f(x,y-1) - 2f(x,y) \\ &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \end{aligned}$$

Given the definition, discretized *3 x 3* Laplacian filter (because we are dealing with images and they are discrete) for an image f is defined as an array below:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 8. 3 × 3 Laplacian filter

Note that, the defined kernel above, which is not identical to the mathematical definition of Laplacian due to the opossite signs, uses a negative peak because it is more commonly used and straightforward. **However, it is still valid.**

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

# Load an image
image = cv2.imread('./images/Taj_Mahal.jpg', cv2.IMREAD_GRAYSCALE)

# Define different kernels
kernels = {
    "Blur": np.ones((5, 5), np.float32) / 25,
    "Sharpen": np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]),
    "Edge Detection": np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]),
    "Emboss": np.array([[-2, -1, 0], [-1, 1, 1], [0, 1, 2]]),
    "Horizontal Derivative (Sobel X)": np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]),
    "Vertical Derivative (Sobel Y)": np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]),
    "Gaussian Blur": np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]], np.float32) / 16,
    "Laplacian Edge Detection": np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
}

# Calculate the number of subplots needed
num_kernels = len(kernels) + 1  # +1 for the original image
num_cols = 3  # Define number of columns you want in the grid
num_rows = math.ceil(num_kernels / num_cols)  # Calculate rows required

# Apply each kernel to the image
filtered_images = {name: cv2.filter2D(image, -1, kernel) for name, kernel in kernels.items()}

# Plot original and filtered images
plt.figure(figsize=(12, 10))
plt.subplot(num_rows, num_cols, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis("off")

for i, (name, filtered_img) in enumerate(filtered_images.items(), 2):
    plt.subplot(num_rows, num_cols, i)
    plt.imshow(filtered_img, cmap='gray')
    plt.title(name)
    plt.axis("off")

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

kernels in image processing

Noise Removal Techniques using OpenCV

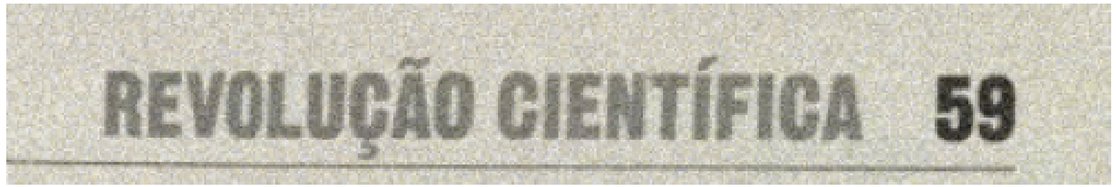
Spatial filters are implemented through kernels (masks/arrays), with odd dimensions. Filters can be::

- Low-pass – used for blur
- High-pass – used to sharpen

Filter with Code

Filter Name	OpenCV Code
Average Blur	`cv2.blur(image, (k, k))`
Gaussian Blur	`cv2.GaussianBlur(image, (k, k), sigmaX)`
Median Blur	`cv2.medianBlur(image, k)`
Bilateral Filter	`cv2.bilateralFilter(image, d, sigmaColor, sigmaSpace)`

Original Image with Noise



Average Blur

3.1 Average Blur

- Average blur calculates the mean of the pixel values in a kernel (sliding window) around each pixel.
- Formula:

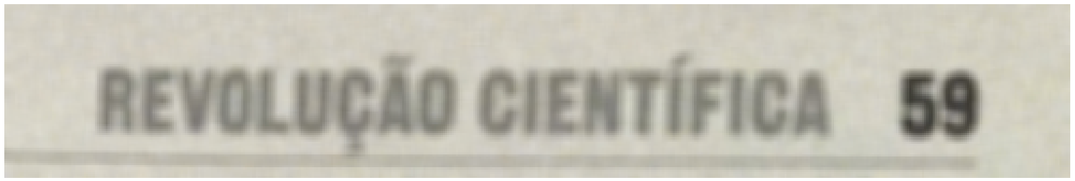
$$I_{\text{blurred}}(x, y) = \frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} I(x + i, y + j)$$

- **Effect**: Simple but tends to blur edges.

```
# Apply Average Blur
average_blur = cv2.blur(image_rgb, (5, 5))
```

✓ 0.0s

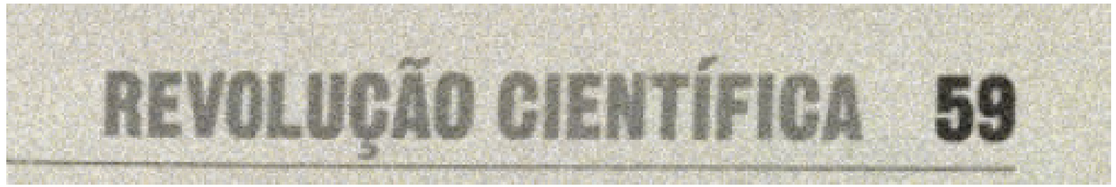
Average Blur



Filter with Code

Filter Name	OpenCV Code
Average Blur	`cv2.blur(image, (k, k))`
Gaussian Blur	`cv2.GaussianBlur(image, (k, k), sigmaX)`
Median Blur	`cv2.medianBlur(image, k)`
Bilateral Filter	`cv2.bilateralFilter(image, d, sigmaColor, sigmaSpace)`

Original Image with Noise



Gaussian Blur

3.2 Gaussian Blur

- Gaussian Blur applies a Gaussian kernel to smooth the image.
- Formula:

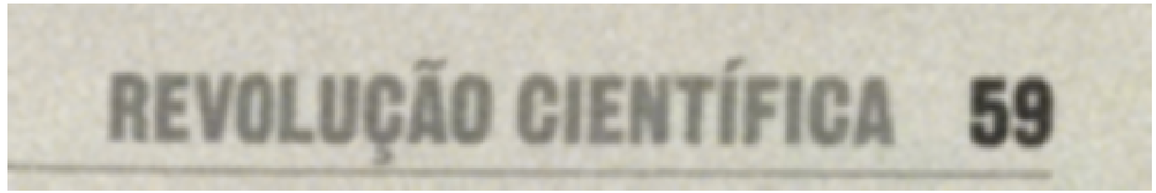
$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

  where σ is the standard deviation.
- **Effect**: Reduces noise with minimal edge blurring.

```
# Apply Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image_rgb, (5, 5), 0)
```

✓ 0.0s

Gaussian Blur



Medium Blur

What is Median Blur?

- Median blur is like "smoothing" an image but in a smart way.
- It works by looking at a small square (called a kernel) around each pixel in the image.
- Instead of averaging the pixel values, it replaces the center pixel with the **middle value** (the median) of all the pixel values in that square.

How does it work?

1. Pick a **kernel size** (e.g., 3x3, 5x5). This is the area it will look at around each pixel.
2. For each pixel:
   - Take the pixel values inside the square.
   - Arrange those values from smallest to largest.
   - Choose the **middle value** (median).
   - Replace the pixel with this median value.

Why use Median Blur?

- It removes noise (like random white and black dots called "salt-and-pepper noise") but keeps the edges sharp.
- Unlike normal blurring (like averaging), it doesn't make edges look blurry.

Example (3x3 Kernel):

Imagine a small grid of pixel values:

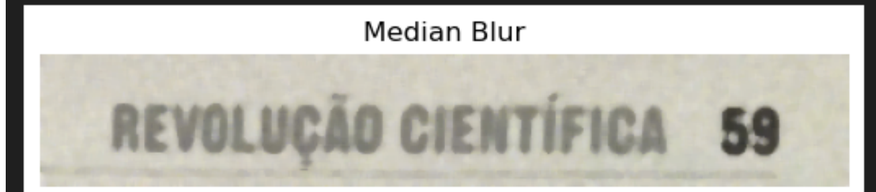
$$\begin{bmatrix} 15 & 18 & 22 \\ 20 & 25 & 30 \\ 10 & 12 & 17 \end{bmatrix}$$

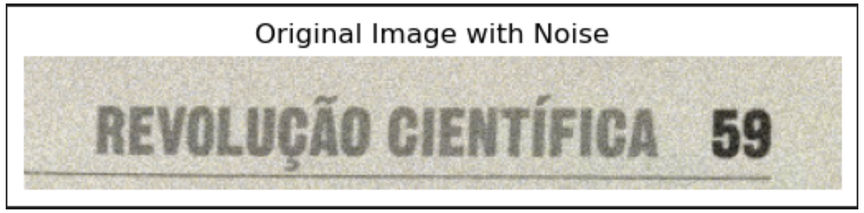
1. Look at all 9 values in this grid: $[15, 18, 22, 20, 25, 30, 10, 12, 17]$.
2. Sort them: $[10, 12, 15, 17, 18, 20, 22, 25, 30]$.
3. Pick the middle value: **18**.
4. Replace the center pixel (25) with **18**.

That's it! Repeat this for every pixel in the image. 😊

```
# Apply Median Blur
median_blur = cv2.medianBlur(image_rgb, 5)
```

✓ 0.0s





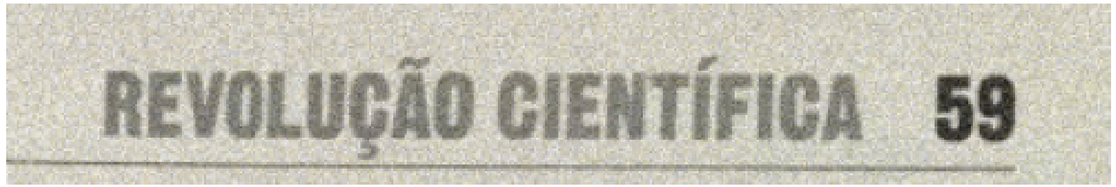
Bilateral Filter

- Bilateral filter considers both spatial and intensity differences for smoothing.
- Formula:

$$I_{\text{filtered}}(p) = \frac{1}{W_p} \sum_q G_{\text{spatial}}(||p - q||) \cdot G_{\text{intensity}}(||I_p - I_q||) \cdot I(q)$$

  where W_p is a normalization factor.
- **Effect:** Smooths noise while preserving edges.

Original Image with Noise




```
# Apply Bilateral Filter
bilateral_filter = cv2.bilateralFilter(image_rgb, 9, 75, 75)
```

✓ 0.0s

Bilateral Filter

Original

Average Blur

Gaussian Blur

Median Blur

Bilateral Filter

Filter Name	Code (OpenCV Function)	Conclusion
Average Blur	cv2.blur(image, (5, 5))	Simple blur that reduces noise but significantly blurs edges.
Gaussian Blur	cv2.GaussianBlur(image, (5, 5), 0)	Smooths noise while retaining more details and edges compared to Average Blur.
Median Blur	cv2.medianBlur(image, 5)	Effectively removes salt-and-pepper noise and preserves edges better.
Bilateral Filter	cv2.bilateralFilter(image, 9, 75, 75)	Reduces noise while preserving edges, making it ideal for complex images.

main

DL-FOR-COMPUTER-VISION / week02 / Homework /

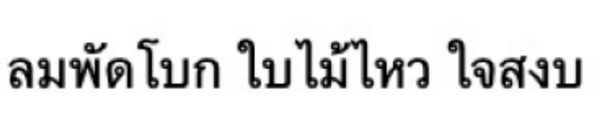
Go to file t

Add file

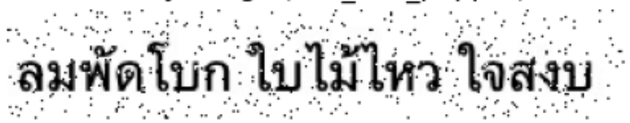
Tuchsanai 33 bf2916f · 2 minutes ago

Name	Last commit message	Last commit date
..		
Work1.ipynb	dd	3 days ago
Work2.ipynb	33	2 minutes ago

Original Image



Noisy Image (salt_and_pepper)



Blurred Image