Computer Vision

CVI620

Session 7

Overview

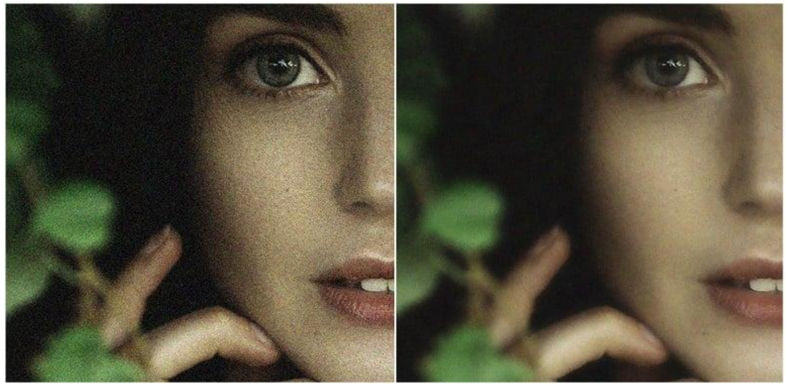
- Geometric Transformations
- Affine
- Scale, rotate, reflection, shear, translation
- warpAffine
- Noise Intro

Agenda

- Noise
- Noise Types
- Denoising Techniques
- Filters
- Convolution
- Mean Denoising

Noise

Definition: Anything that deviates from the ideal image or hinders achieving your imaging goal.



Noise Source

- Environmental factors
- Imaging device limitations
- Electrical interference
- Digitization process
- And more

Noise Characteristics

- Additive and random
- Represented as:

$$P(i, j) = I(i, j) + n(i, j)$$

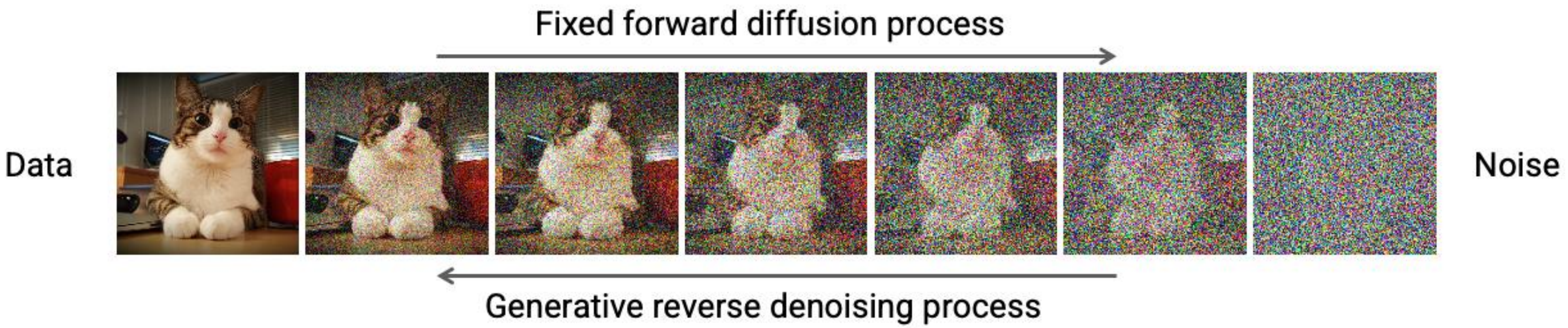
$P(i, j)$: Pixel value in the noisy image

$I(i, j)$: Pixel value in the ideal image

$n(i, j)$: Noise value

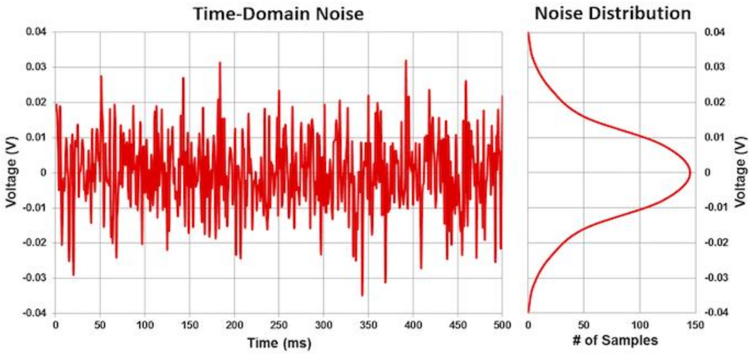
Why we need to study noise?

- Ensures image accuracy for analysis or display.
- Critical for applications in medical imaging, machine vision, and remote sensing.
- Used in state-of-art generative models like DALLE



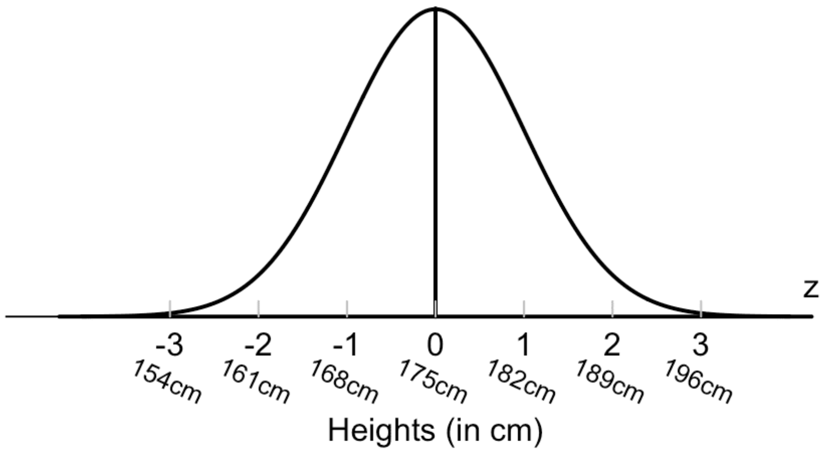
Noise Types

- Gaussian noise
- Salt and Pepper
- Poisson noise
- Speckle noise
- Thermal noise



Gaussian Noise

- Noise that follows a normal (Gaussian) distribution
- Normal distribution and why it is important:
  - Symmetry: Centered around the mean (μ).
  - Mean, Median, and Mode: All are equal and located at the peak.
  - Width determined by standard deviation (σ).



Gaussian Noise in CV2

- Choose random samples from normal distribution
- Add it to the image

```
image = cv2.imread('Lucy.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

mean = 0
std_dev = 25

gaussian_noise = np.random.normal(mean, std_dev, image.shape).astype('float32')

noisy_image = cv2.add(image.astype('float32'), gaussian_noise)

noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')

plt.figure(figsize= (10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)

plt.subplot(1, 2, 2)
plt.imshow(noisy_image)

plt.show()
```

Impulsive Salt and Pepper Noise

- A type of impulse noise where random pixels in the image are replaced with:
  - White (salt): Maximum intensity (e.g. 255 in an 8-bit image)
  - Black (pepper): Minimum intensity (e.g. 0 in an 8-bit image)
- Appears as random white and black dots in an image

$$P(i,j) = \begin{cases} I(i,j), & \text{with probability } (1-p) \\ 0 \text{ (pepper)}, & \text{with probability } p/2 \\ 255 \text{ (salt)}, & \text{with probability } p/2 \end{cases}$$

Salt and Pepper

```python
import cv2
import numpy as np

image = cv2.imread('Lucy.jpg', cv2.IMREAD_GRAYSCALE)

def add_noise(img):
    row , col = img.shape

    number_of_pixels = np.random.randint(300, 10000)
    for i in range(number_of_pixels):
        y_coord = np.random.randint(0, row - 1)
        x_coord = np.random.randint(0, col - 1)
        img[y_coord, x_coord] = 255


    number_of_pixels = np.random.randint(300, 10000)
    for i in range(number_of_pixels):
        y_coord = np.random.randint(0, row - 1)
        x_coord = np.random.randint(0, col - 1)
        img[y_coord, x_coord] = 0

    return img

noisy_img = add_noise(image)
cv2.imshow('Noisy Image', noisy_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

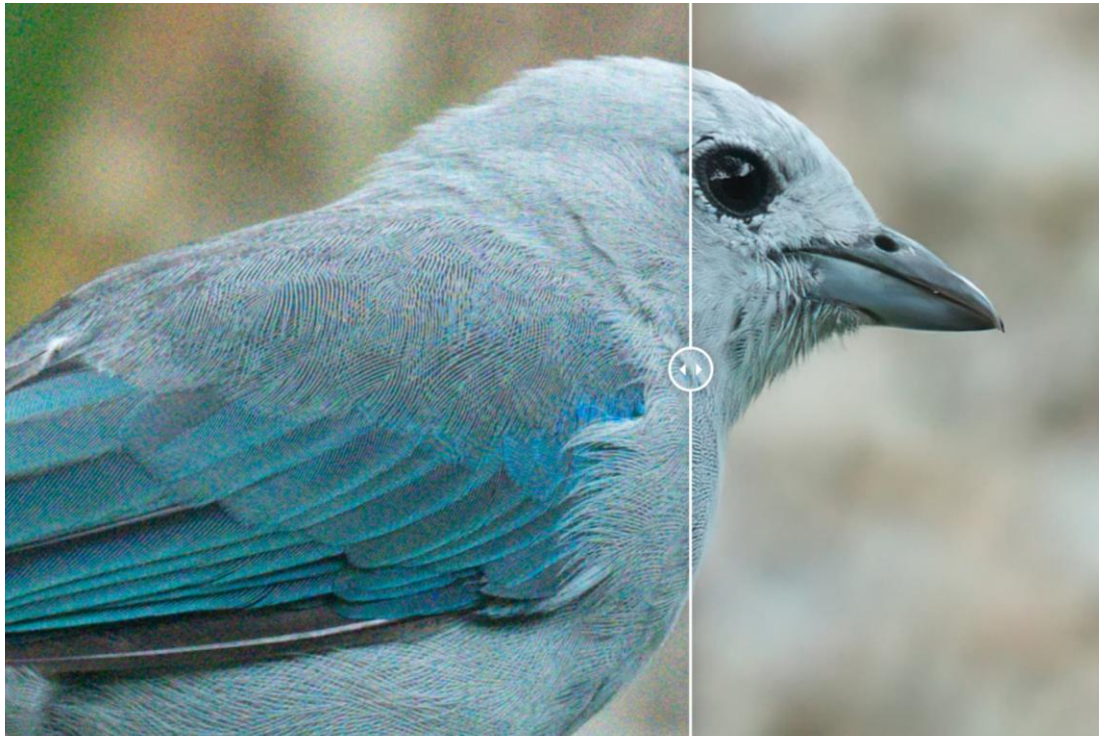
Question?

We have added noise so far.

How can we remove it?

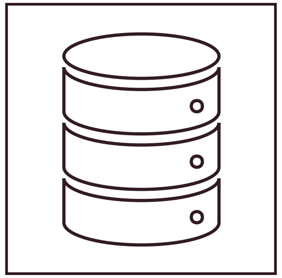
Denoising

- https://www.topazlabs.com/denoise-ai?srsltid=AfmBOopM02_xy6QWHcr_WMSI5iSE0MH4MBimXXkvw7fg80zWAmOHNnUs

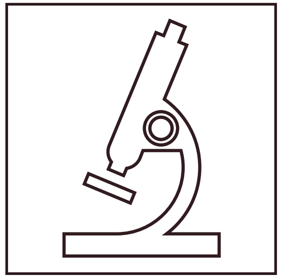


Denoising Techniques

Spatial Filtering



Frequency Domain Filtering



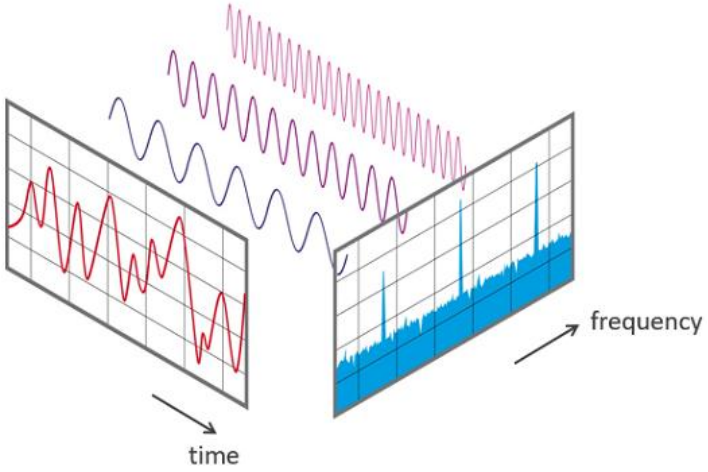
Advanced Methods

Spatial Filtering

- Mean Filter: Averages pixel values in a neighborhood
- Median Filter: Replaces pixel value with the median in a local window
- Gaussian Filter: Weighted averaging with a Gaussian kernel

Frequency Domain Filtering

- Fourier Transform-Based Filtering: Suppresses high-frequency noise



Advanced Methods

01

Wavelet Transform Denoising: Removes noise at different scales

02

Non-Local Means (NLM): Uses similar patches across the image

03

Deep Learning-Based Denoising (Denoising Autoencoders, CNNs)

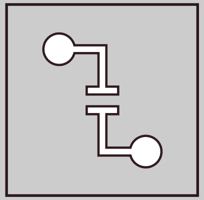
Another Categorization

1. Filtering Based Denoising
2. Transform Based Denoising
3. Statistical & Probabilistic Methods
4. Machine Learning & Deep Learning Approaches
5. Optimization-Based Methods

Filtering Based Denoising

- Mean Filter
- Gaussian Filter
- Median Filter
- Bilateral Filter
- Wavelet Denoising

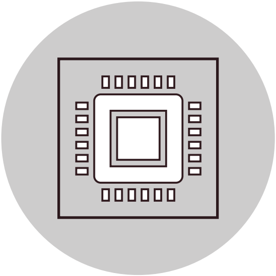
Transformer Based Denoising



Fourier Transform Filtering: Removes noise in the frequency domain.

Wavelet Transform: Decomposes the image into different scales and removes noise selectively.

Statistical Denoising



Non-Local Means (NLM): Uses patches from the image to reduce noise.

Bayesian Denoising: Uses probabilistic models to infer the denoised image.

ML and DL Denoising

- Denoising Autoencoders: Neural networks trained to remove noise from images.
- Convolutional Neural Networks (CNNs): Trained models that learn patterns to reconstruct noise-free images.
- Generative Models (GANs, Diffusion Models): Learn distributions of clean images to remove noise effectively.

Optimization Based Denoising

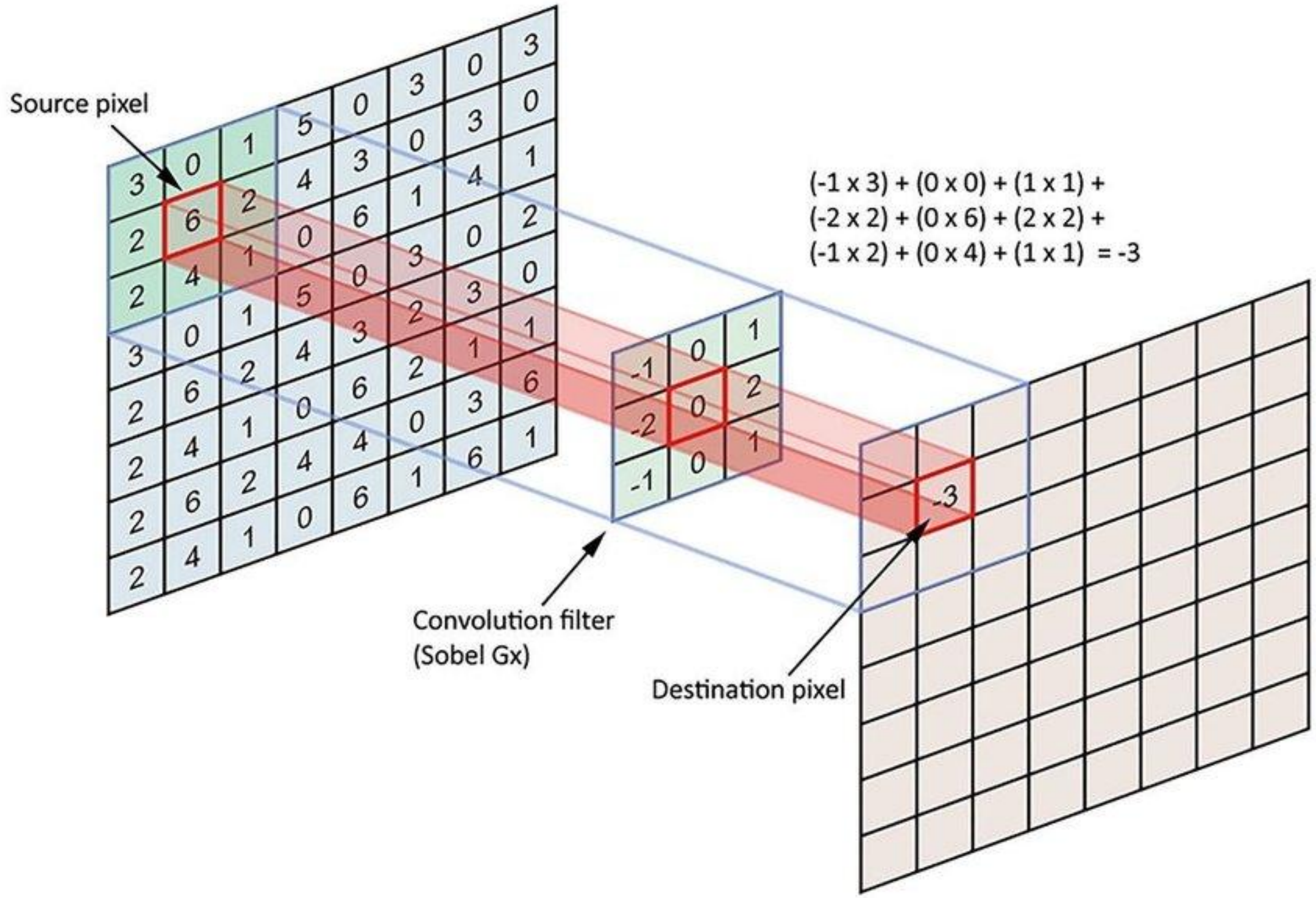
- Total Variation (TV) Denoising: Minimizes variations while preserving edges.
- Sparse Coding: Represents images in a sparse domain and removes noise adaptively.

noise -> filter -> convolution -> denoise

Image Filtering

Filter

- The core idea is to manipulate image features like noise, edges, and textures based on specific objectives.
- Adding, changing, detecting features or filters in a picture is better to be applied step by step and region by region

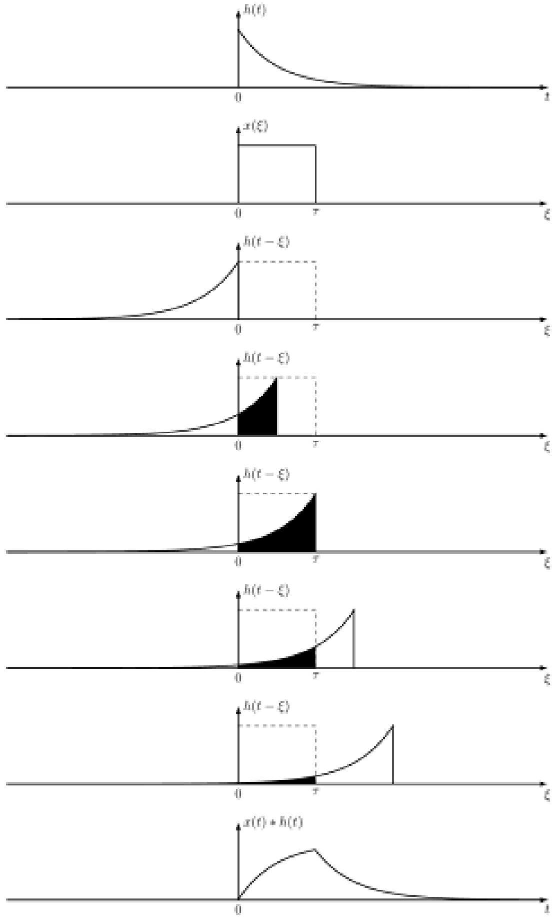


Convolution

Filtering is commonly implemented using **convolution**, especially in spatial domain filtering.

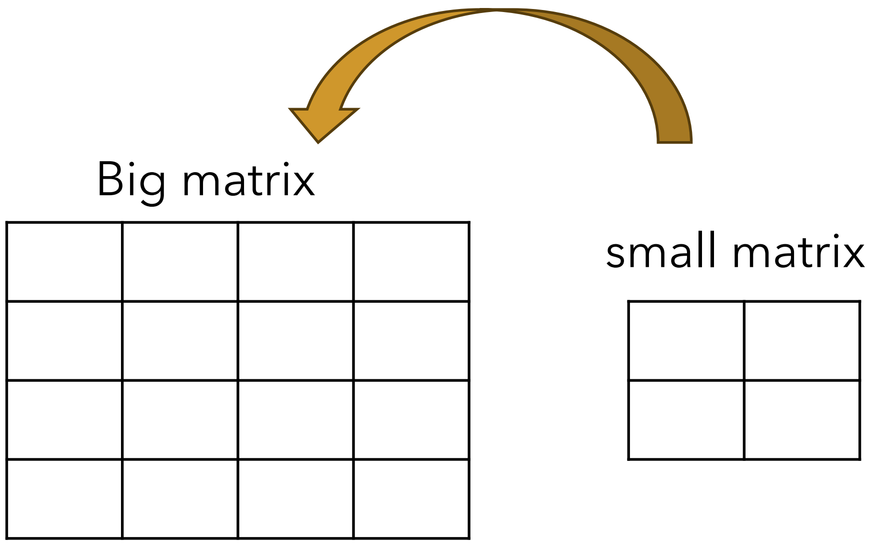
In frequency domain filtering, convolution is performed using Fourier Transform methods.

Convolution in Signal and Systems



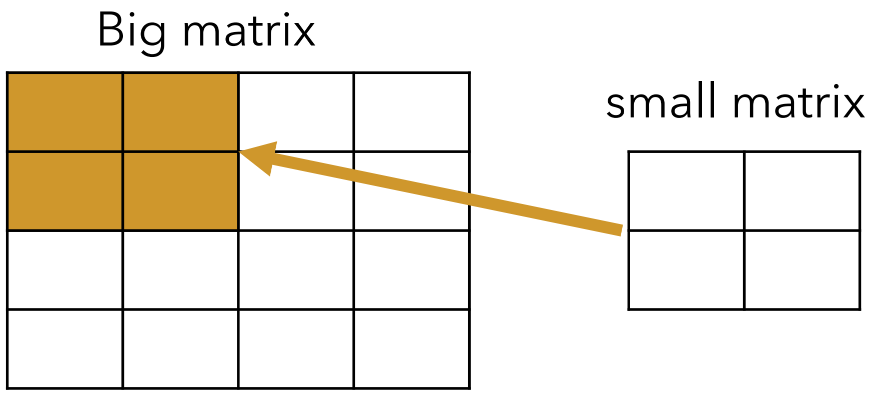
Convolution

- Sum of pixel multiplications in a region



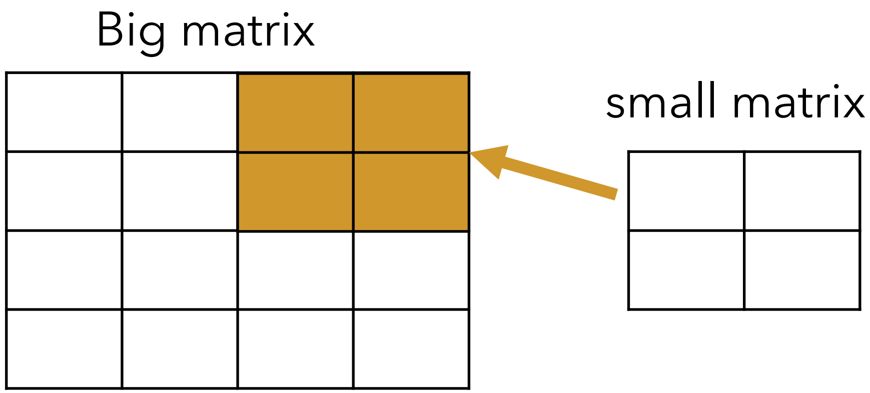
Convolution

- Align from (0,0)



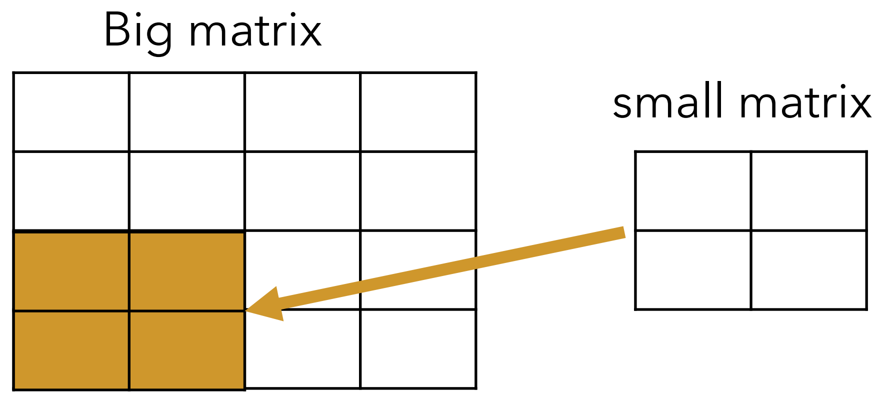
Convolution

- Move left to right with a step size

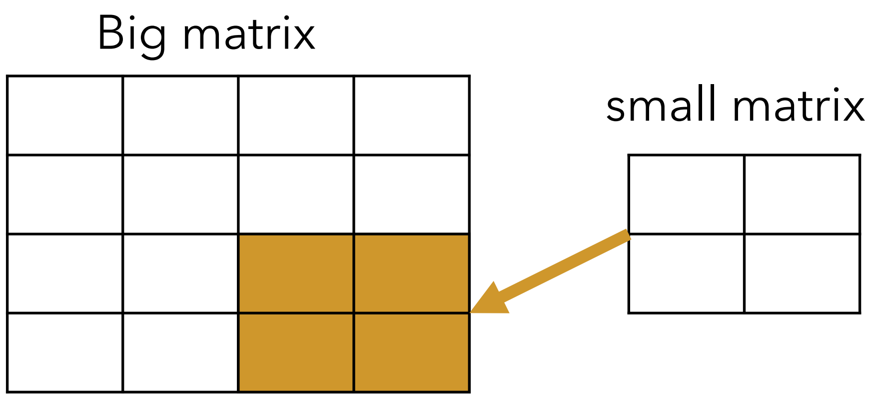


Convolution

- Move top to bottom with a step size

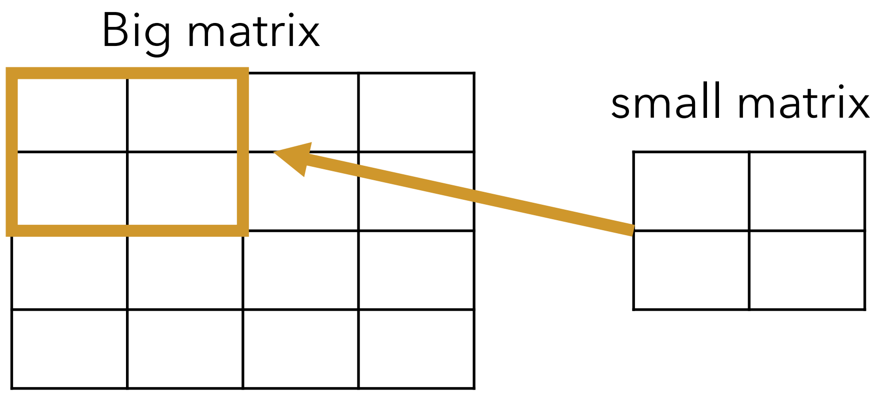


Convolution



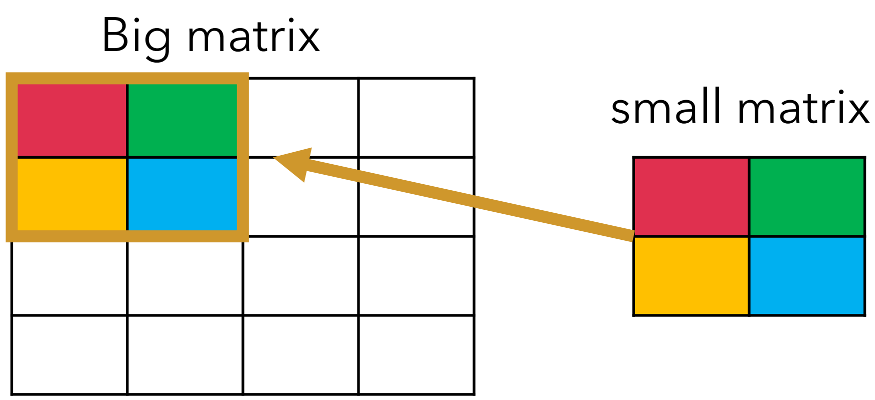
Convolution

- Mathematical operation



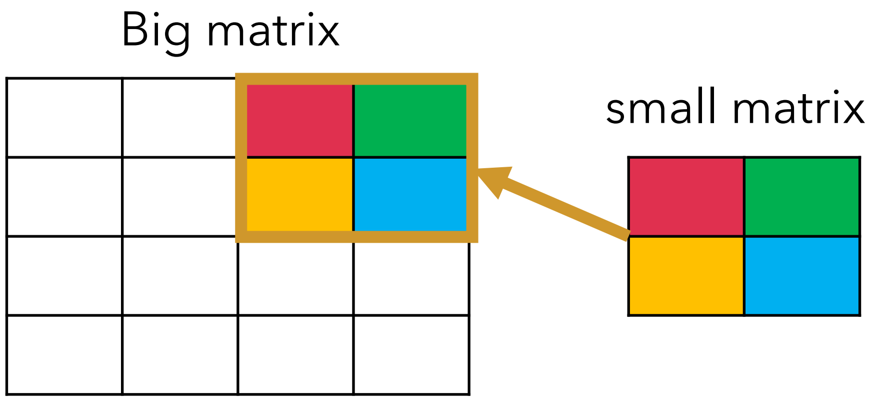
Convolution

- Sum of pixel multiplications in a region
- Multiply aligned pixels and add them together



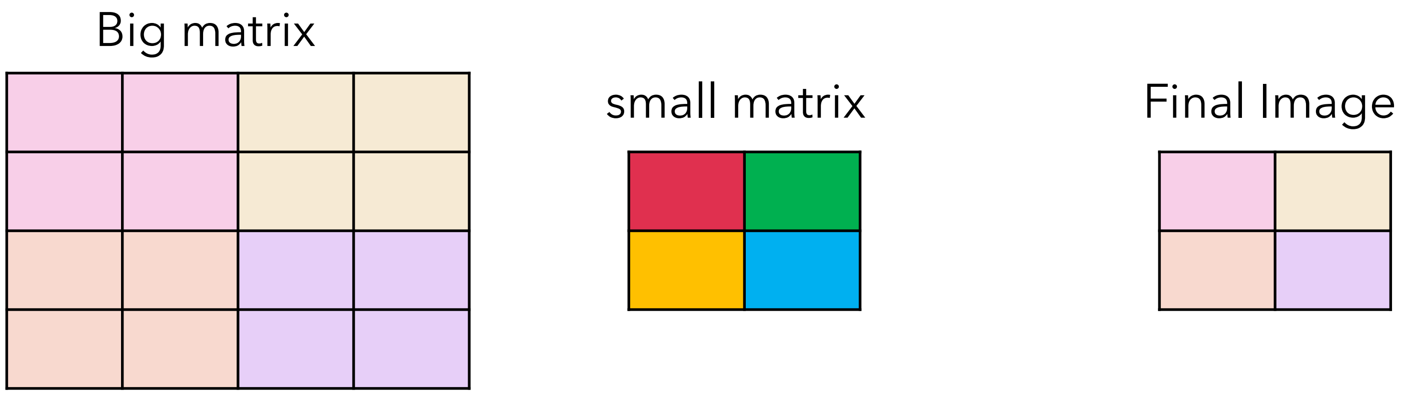
Convolution

- Sum of pixel multiplications in a region
- Multiply aligned pixels and add them together



Convolution

- Sum of pixel multiplications in a region
- Multiply aligned pixels and add them together
- **Make a new image**



Example

Image

0	1	2
3	4	5
6	7	8

Kernel

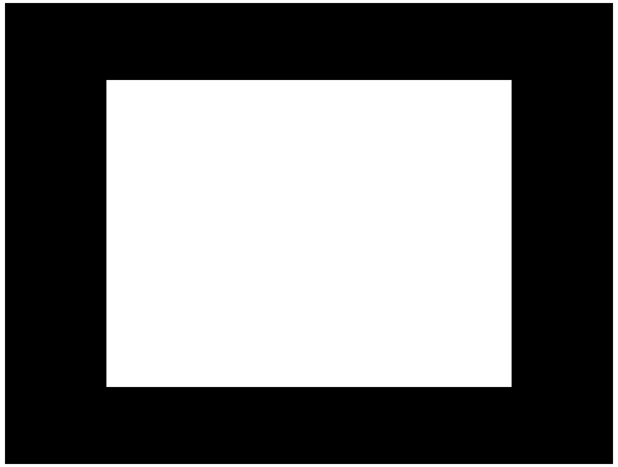
0	1
2	3

19	25
37	43

0*0 + 1*1 + 3*2 + 3*4 = 19
1*0 + 2*1 + 4*2 + 5*3 = 25
…

Real Example

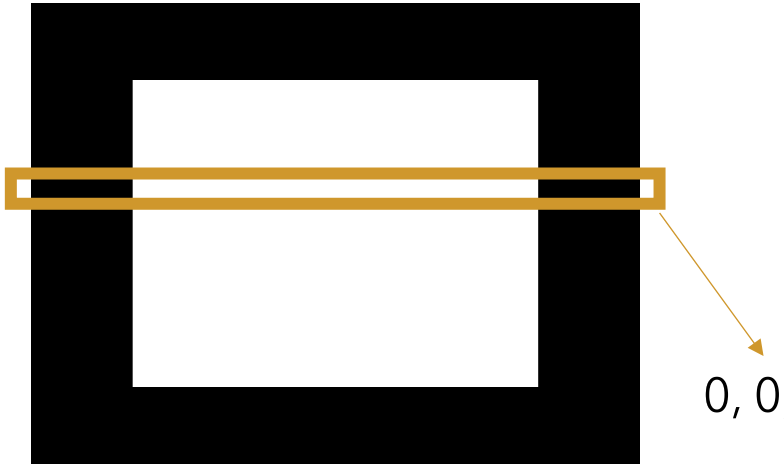
Image



Kernel
[-1,1]

Real Example

Image

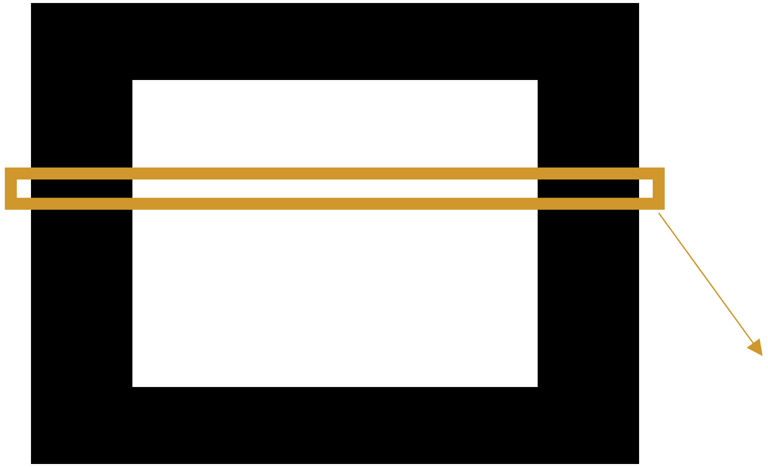


Kernel
[-1,1]

0, 0, 0, …., 255, 255, 255, …., 0, 0, …

Real Example

Image



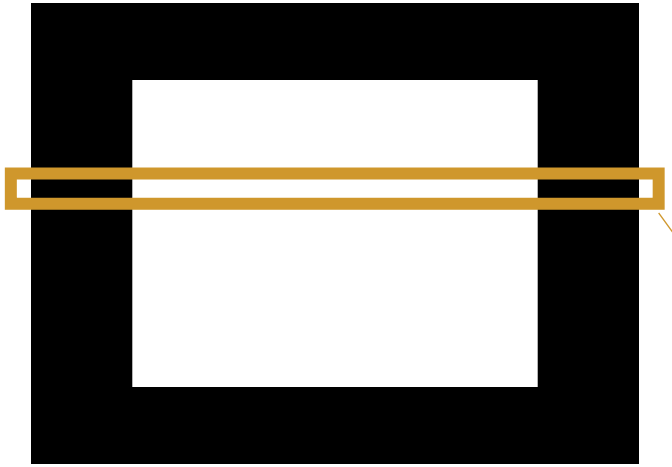
Kernel
[-1,1]

0, 0, 0, ….,0, 255, 255, 255, …., 0, 0, …

0*-1 + 0*1 = 0
0*1 + 255*1 = 255
255*-1 + 255*1 = 0
255*-1 + 255*1 = 0
255*-1 + 255*1 = 0
255*-1 + 0*-1 = -255 → clip(-255) =0

Real Example

Image



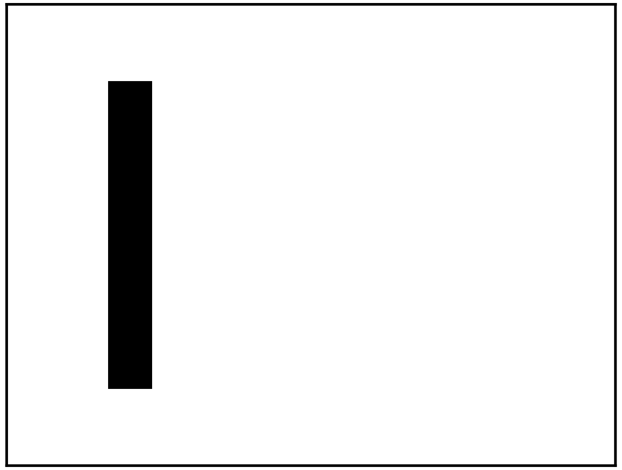
Kernel
[-1,1]

0, 0, 0, ….,0, 255, 255, 255, …., 0, 0, …

0, 0, 0, …, 0, 255, 0, 0, 0, …

Real Example

0, 0, 0, ..., 0, 255, 0, 0, 0, ...



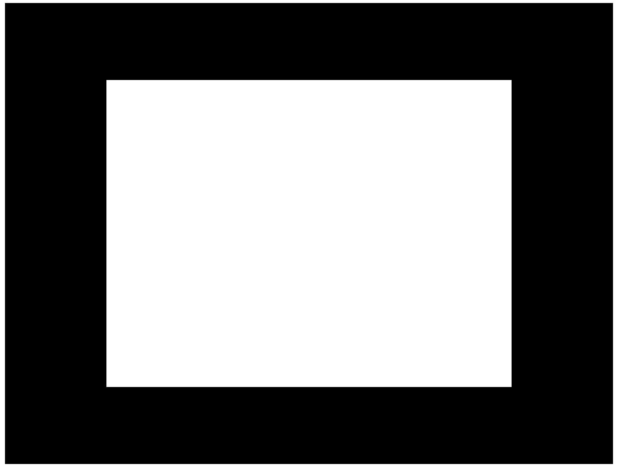
Edge!!

Let's Code

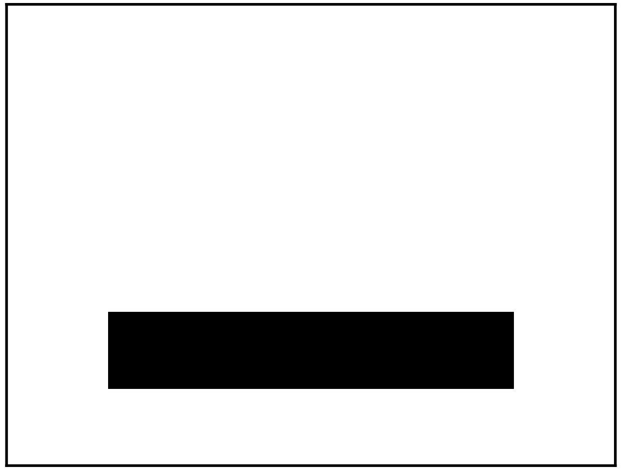
```python
import cv2
import numpy as np


img = cv2.imread("square.png")
kernel = np.array([[-1, 1]])

out_image = cv2.filter2D(img, cv2.CV_8U, kernel)

cv2.imshow("left edge", out_image)
cv2.waitKey(0)
```

Let’s Convolve Vertically

Image



Kernel
[[-1],
[1]]



Example

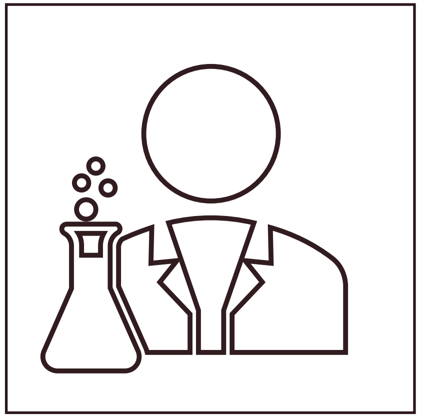
```
import cv2
import numpy as np


img = cv2.imread("Lenna.png")
kernel1 = np.array([[-1, 1]])
kernel2 = np.array([[-1],
                    [1]])

out_image1 = cv2.filter2D(img, cv2.CV_8U, kernel1)
out_image2 = cv2.filter2D(img, cv2.CV_8U, kernel2)

cv2.imshow("left edge", out_image1)
cv2.imshow("bottom edge", out_image2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

More Complex Convolutions!

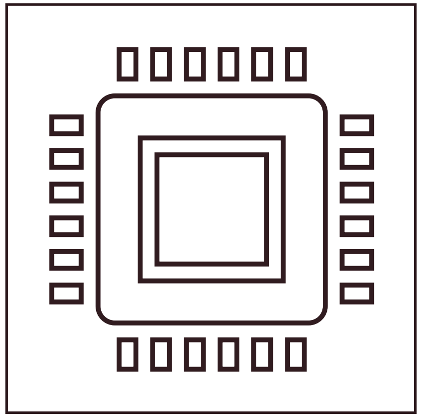
-1	0	1
-2	0	2
-1	0	1

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

How to achieve these kernels?



Experiments!



ML to learn kernels!

Now let's get back to denoising!

Mean Denoising

- Replaces each pixel value with the average of its neighboring pixels.
- Smooths the image by averaging pixel intensities.
- Reduces random noise while preserving structure.

Mean Denoising

The filter slides over the image, replacing each pixel with the mean of its neighbors.

Uses a convolution operation with a kernel

Mean Denoising

- Applying a 3×3 Mean Filter
- Kernel Example:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * 1/9$$

- Each pixel value is replaced by the average of surrounding pixels.
- Result: A smoother image with reduced noise.
- Also called blurring!

Example

```python
import cv2
import numpy as np

image = cv2.imread("Lucy.jpg")

if image is None:
    print("ERROR! Image not available...!")

# Gaussian Noise
noise = np.random.normal(0, 25, image.shape).astype('float32')

noisy_image = image + noise
noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')

kernel = np.ones((3,3), dtype= np.float32) / 9
denoised_image = cv2.filter2D(noisy_image, -1, kernel)

cv2.imshow("left edge", denoised_image)
cv2.imshow('frame', noisy_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```