Practices in visual computing 1

Lab2: Numpy & OpenCV

Simon Fraser University
Fall 2025

Numerical Computing with NumPy

Foundation for scientific Python stack

Powers Machine Learning, Computer Vision, and data analysis workflows

Why NumPy?

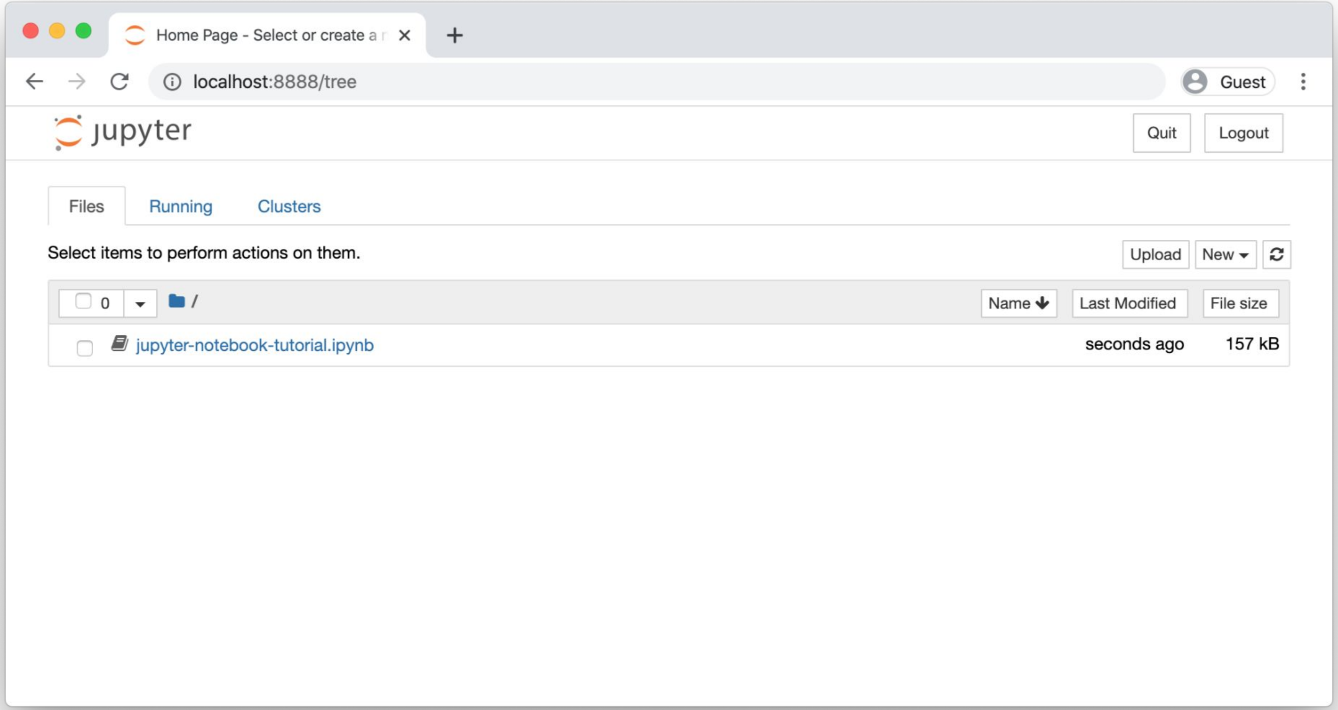
Speed: implemented in C under the hood

Memory efficiency: contiguous storage

Vectorized operations (no explicit loops)

Integration: works seamlessly with OpenCV, SciPy, scikit-learn, etc.

Jupyter and Colab Notebooks



NumPy Arrays vs. Python Lists

Python Lists:

- Store mixed types
- Extra overhead (each element = Python object)
- Scattered in memory → slower math

NumPy Arrays:

- Homogeneous (one data type)
- Contiguous memory → compact & fast
- Backed by efficient C implementation

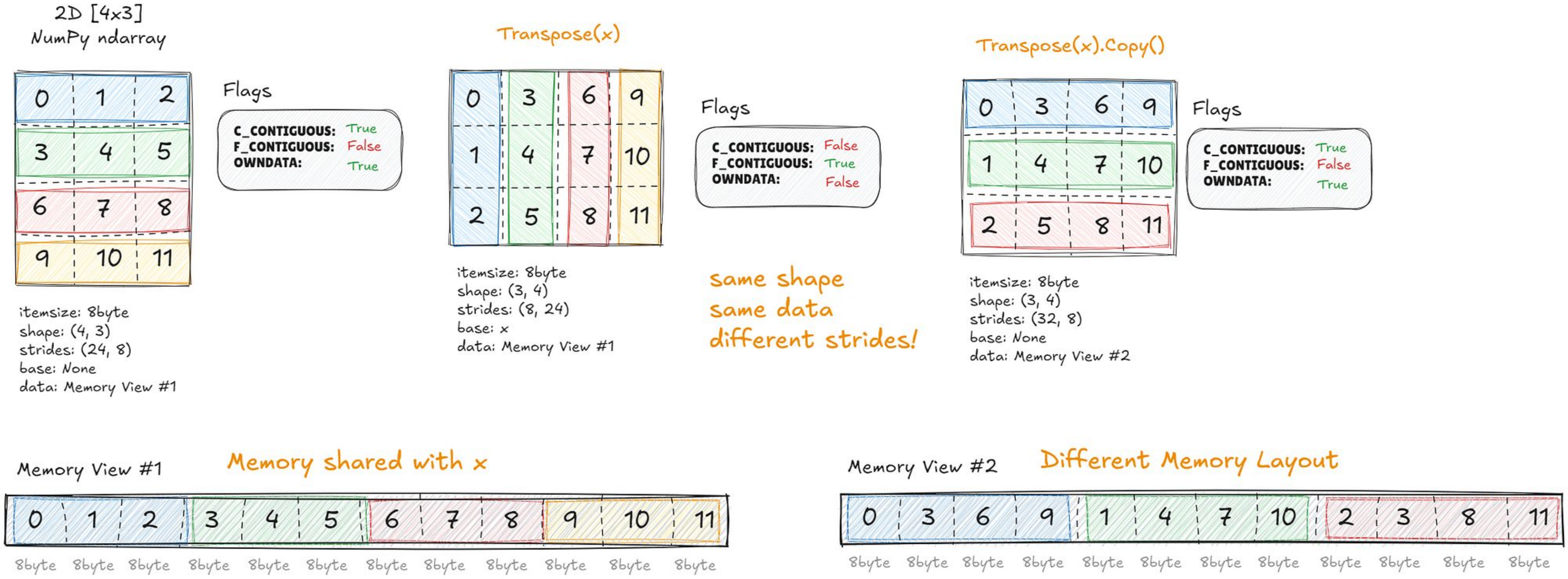
NumPy Views

A view is a new array object that shares the same underlying data.

No extra memory is used → slicing, reshaping, transposing are fast.

Changing data in a view also changes it in the original array.

View vs. Copy


2D [4x3]
NumPy ndarray
0 1 2
3 4 5
6 7 8
9 10 11
Flags
C_CONTIGUOUS: True
F_CONTIGUOUS: False
OWNDATA: True
itemsize: 8byte
shape: (4, 3)
strides: (24, 8)
base: None
data: Memory View #1
Transpose(x)
0 3 6 9
1 4 7 10
2 5 8 11
Flags
C_CONTIGUOUS: False
F_CONTIGUOUS: True
OWNDATA: False
itemsize: 8byte
shape: (3, 4)
strides: (8, 24)
base: x
data: Memory View #1
same shape
same data
different strides!
Transpose(x).Copy()
0 3 6 9
1 4 7 10
2 5 8 11
Flags
C_CONTIGUOUS: True
F_CONTIGUOUS: False
OWNDATA: True
itemsize: 8byte
shape: (3, 4)
strides: (32, 8)
base: None
data: Memory View #2
Memory View #1
Memory shared with x
0 1 2 3 4 5 6 7 8 9 10 11
8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte
Memory View #2
Different Memory Layout
0 3 6 9 1 4 7 10 2 3 8 11
8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte 8byte

Some array creation routines

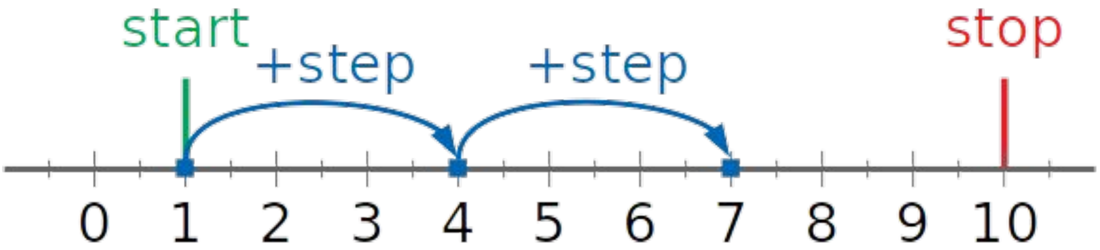
Numerical ranges: arange, linspace, logspace

Homogeneous data: zeros, ones

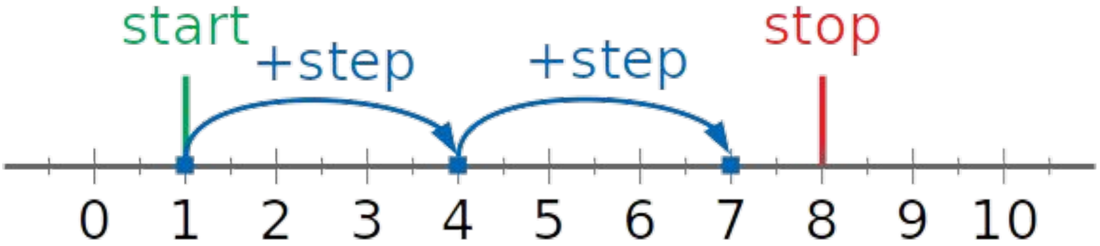
Diagonal elements: diag, eye

Random numbers: rand, randint

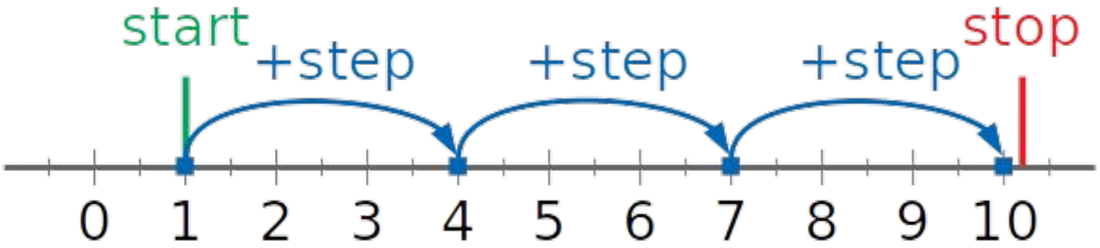
```
>>> np.arange(1, 10, 3)
array([1, 4, 7])
```

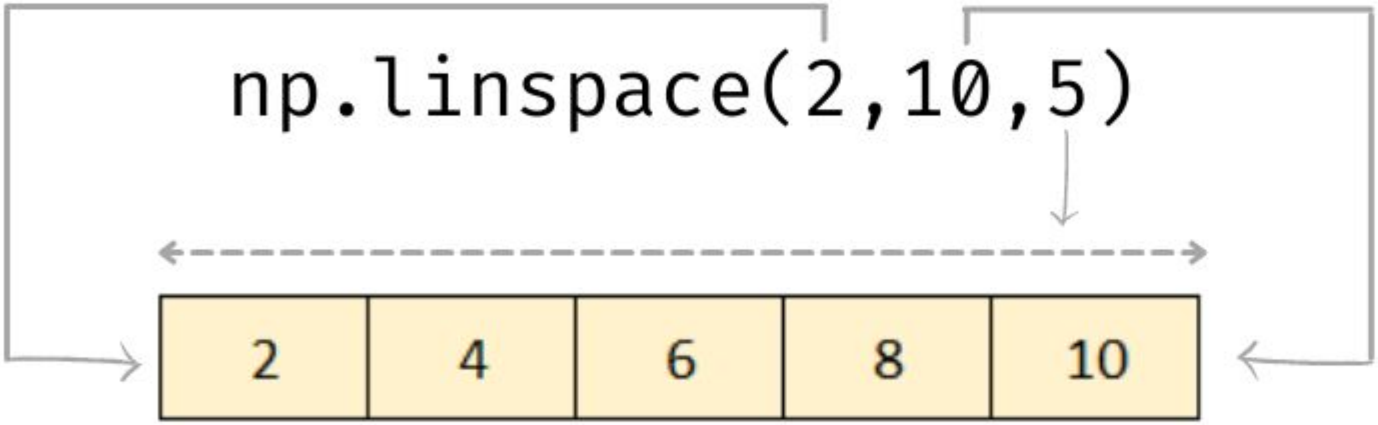



```
>>> np.arange(1, 8, 3)
array([1, 4, 7])
```




```
>>> np.arange(1, 10.1, 3)
array([1., 4., 7., 10.])
```





Equally spaced values

Indexing and slicing in one dimension

1d arrays: indexing and slicing as for lists

- first element has index 0
- negative indices count from the end
- slices: [start:stop:step] without the element indexed by stop
- if values are omitted:
  - start: starting from rst element
  - stop: until (and including) the last element
  - step: all elements between start and stop-1

Indexing and slicing in higher dimensions

a[2, -3]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[:3, :5]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

YOUR TURN

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[-3:, -3:]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

YOUR TURN

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[:, 3]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

YOUR TURN

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[1, 3:6]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

YOUR TURN

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[1::2, ::3]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

YOUR TURN

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Indexing and slicing in higher dimensions

a[a % 3 == 0]

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Fancy indexing – array of integers

```
a[(1, 1, 2, 2, 3, 3), (3, 4, 2, 5, 3, 4)]
```

0	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	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Matrix multiplication

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} \boxed{6} & 7 \\ 26 & 31 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} 6 & \boxed{7} \\ 26 & 31 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ \boxed{26} & 31 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 26 & \boxed{31} \end{pmatrix}$$

try `np.dot(•, •)`

`•.dot(•)`

`• @ •` *)

---

*) Python ≥ 3.5, NumPy ≥ 1.10

Sorting, searching, and counting in NumPy

Sorting:

sort, argsort, ndarray.sort, sort_complex, partition, argpartition

Searching:

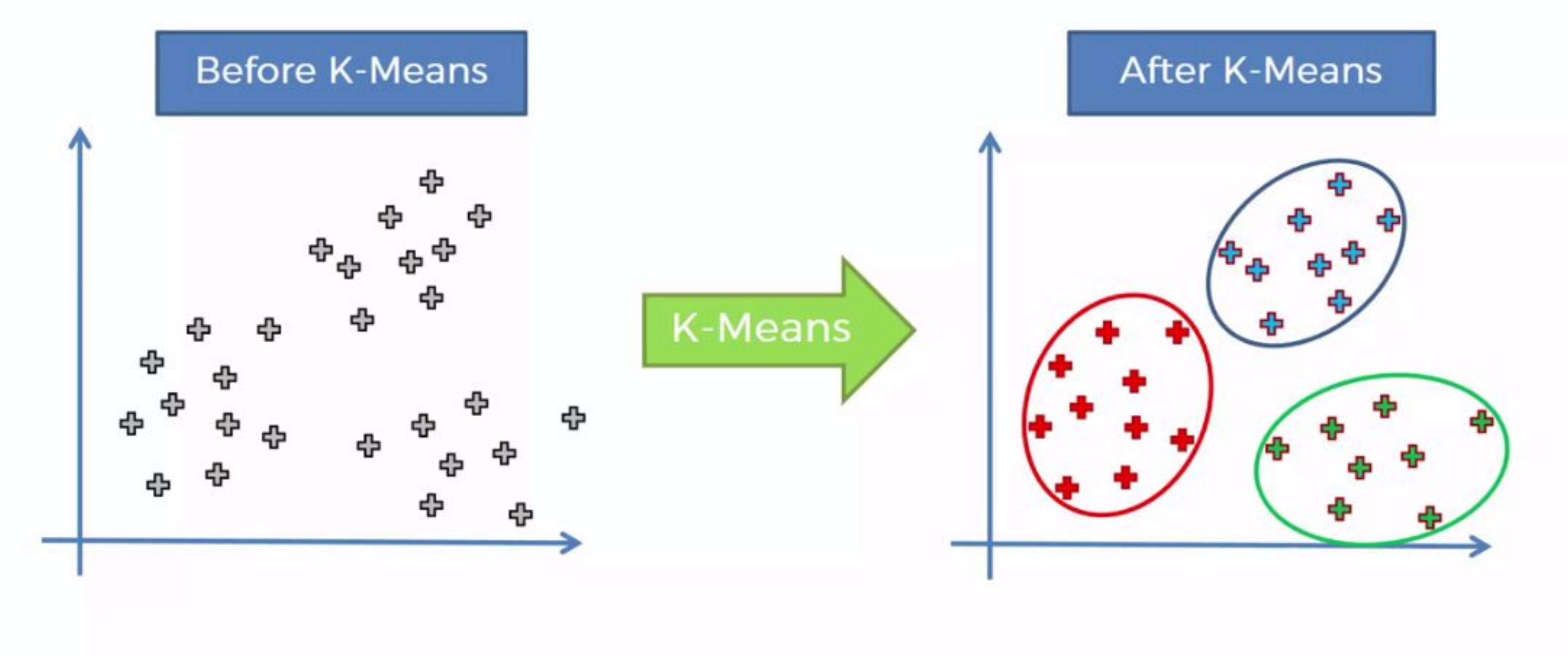
argmax, nanargmax, argmin, nanargmin, argwhere, nonzero, where, searchsorted

Counting:

count_nonzero

K-Means Clustering

Goal: Group data into k clusters by similarity



Algorithm Steps

Initialize:

Choose k cluster centers (random or k-means++).

Assign:

Each point goes to the nearest center.

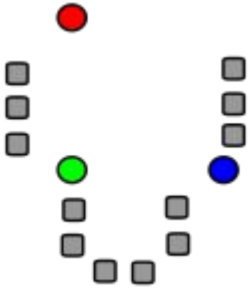
Update:

Recompute centers as the mean of assigned points.

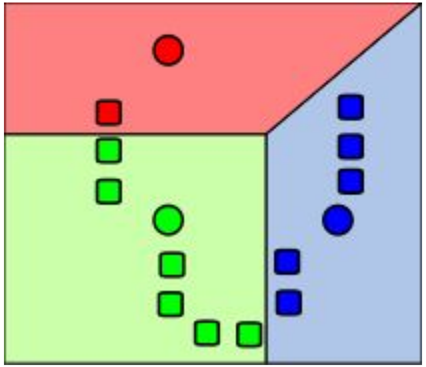
Repeat:

Alternate assign/update until centers stop moving.

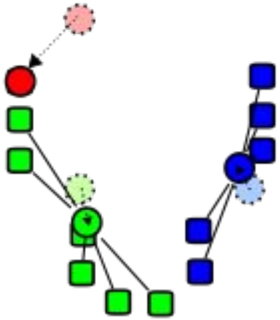
Algorithm Steps



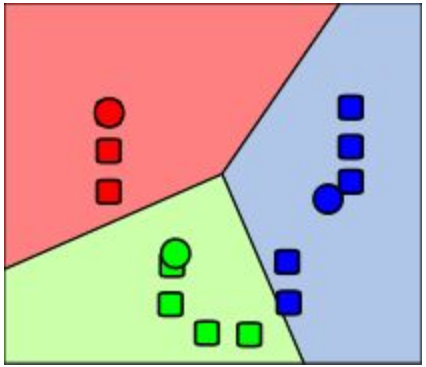
1- Initialize



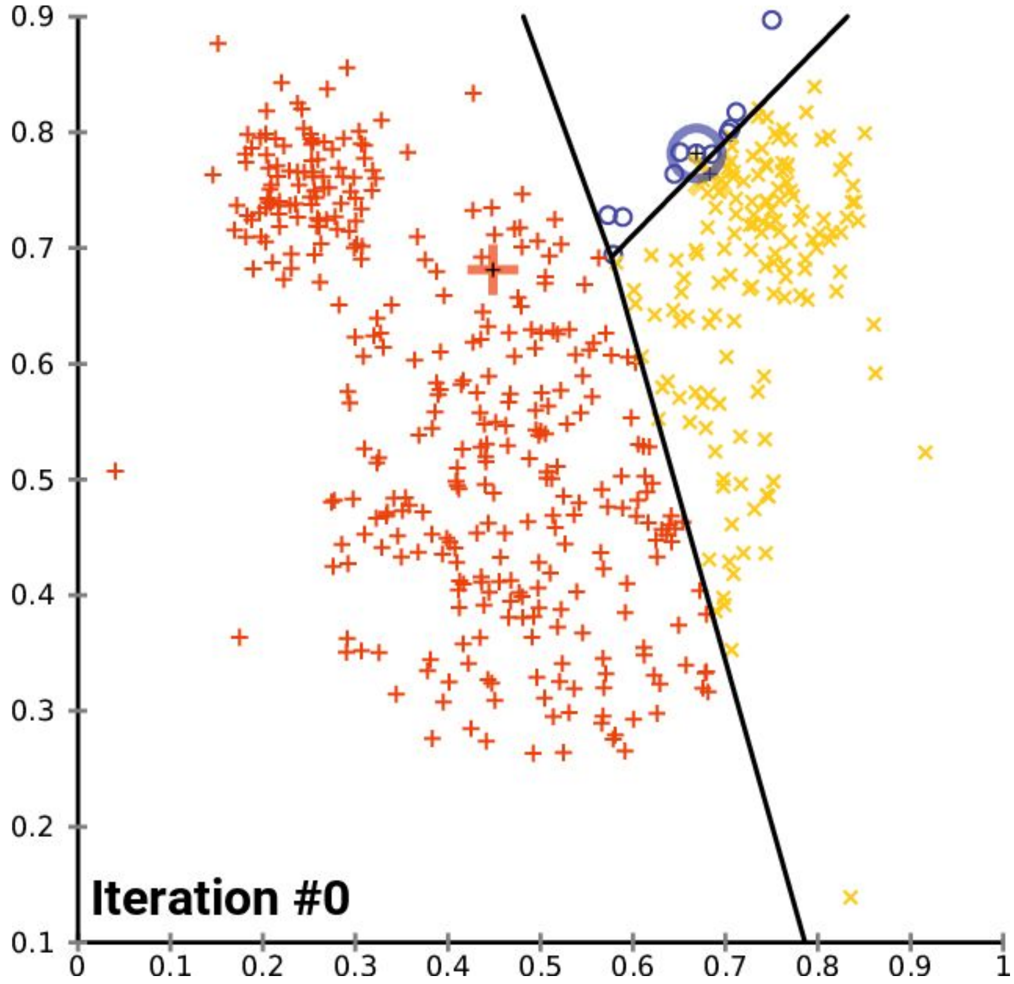
2- Assign



3- Update



4- Repeat 2 & 3



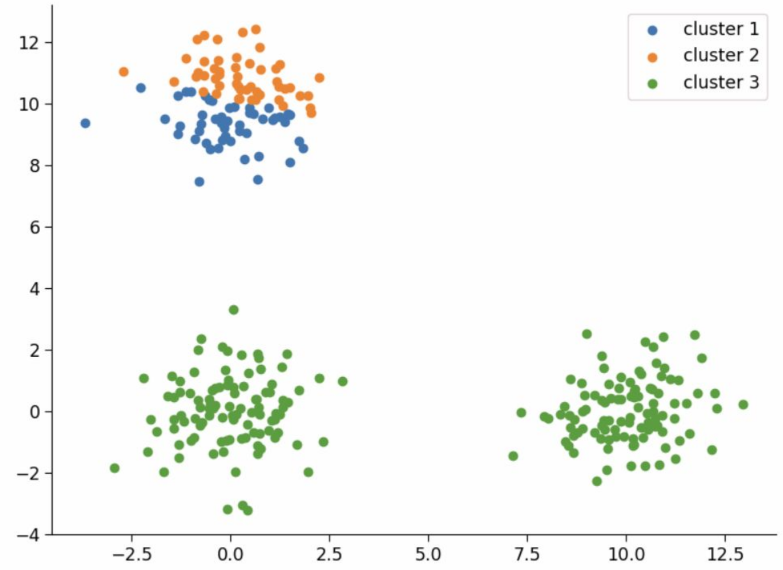
Iteration #0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Weakness of naive initialization

In basic K-Means, the centroids are initialized randomly.

Bad initialization can cause:

- Slow convergence (more iterations).
- Poor clustering (algorithm stuck in bad local minima).



K-Means++ Initialization Strategy

Instead of picking all centroids at random, K-Means++ spreads them out intelligently:

1. Pick the first centroid randomly from the data.
2. For each remaining point, compute its distance to the nearest chosen centroid.
3. Pick the next centroid with probability proportional to the squared distance.

    Farther points are more likely to be picked.

4. Repeat until k centroids are chosen.
5. Run normal K-Means as usual.

Introduction to OpenCV

Open Source Computer Vision Library

Focus: image processing, computer vision, machine learning

Widely used in academia & industry

Images in OpenCV

Images are NumPy arrays (H × W × C)

Color channels: BGR (not RGB)

Loading:

```python
# Importing the OpenCV library
import cv2
# Reading the image using imread() function
image = cv2.imread('image.jpg')

# Extracting the height and width of an image
h, w = image.shape[:2]
# Displaying the height and width
print("Height = {}, Width = {}".format(h, w))
```

Basic Operations

Resize: cv2.resize()

Crop: NumPy slicing

Rotate & flip: cv2.rotate(), cv2.flip()

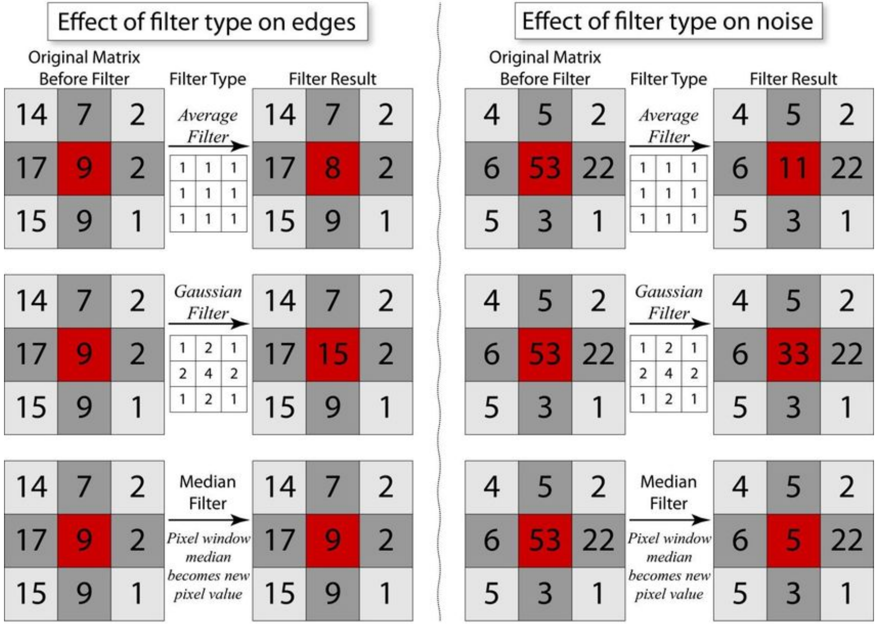
Convert color: cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

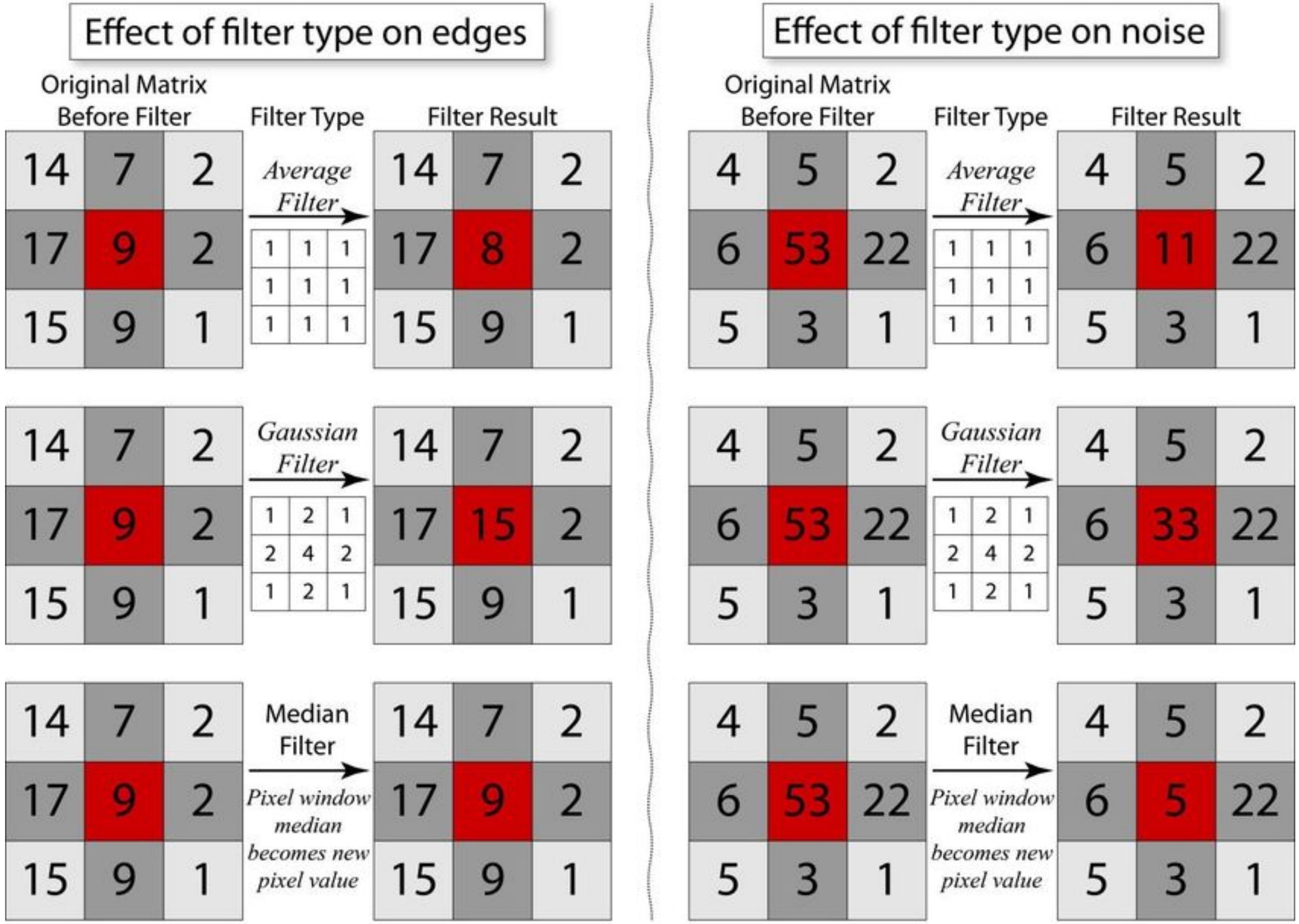
Filtering & Blurring

Smoothing reduces noise

Common filters:

- Average blur → cv2.blur()
- Gaussian blur → cv2.GaussianBlur()
- Median blur → cv2.medianBlur()




Effect of filter type on edges
Original Matrix Before Filter
Filter Type
Filter Result
14 7 2
17 9 2
15 9 1
Average Filter
1 1 1
1 1 1
1 1 1
14 7 2
17 8 2
15 9 1
Gaussian Filter
1 2 1
2 4 2
1 2 1
14 7 2
17 15 2
15 9 1
Median Filter
Pixel window median becomes new pixel value
14 7 2
17 9 2
15 9 1
Effect of filter type on noise
Original Matrix Before Filter
Filter Type
Filter Result
4 5 2
6 53 22
5 3 1
Average Filter
4 5 2
6 11 22
5 3 1
Gaussian Filter
4 5 2
6 33 22
5 3 1
Median Filter
Pixel window median becomes new pixel value
4 5 2
6 5 22
5 3 1

Edge Detection

Edges = sudden changes in intensity (where brightness or color shifts quickly).

Important for:

1. Object boundaries
2. Feature extraction
3. Shape analysis

Sobel Operator

Approximates the image derivative → highlights intensity changes (edges).

Uses 3×3 kernels separately for x and y directions.

X-gradient kernel: negatives on the left, positives on the right → detects vertical edges.

Y-gradient kernel: negatives on the bottom, positives on the top → detects horizontal edges.

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

Gx

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

Gy

Sobel Operator

Place a gradient kernel (3×3) over each pixel of the image.

Compute difference in intensity → approximates the derivative.

Produces two output images:

- X-Direction (G_x): vertical edges
- Y-Direction (G_y): horizontal edges

Using kernel convolution, sharp changes appear as bright lines (edges).

Sobel Operator

100	100	200	200
100	100	200	200
100	100	200	200
100	100	200	200

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

-100
-200
-100
200
400
+200
=400

Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

Sobel Operator

Place a gradient kernel (3×3) over each pixel of the image.

Compute difference in intensity → approximates the derivative.

Produces two output images:

- X-Direction (G_x): vertical edges
- Y-Direction (G_y): horizontal edges

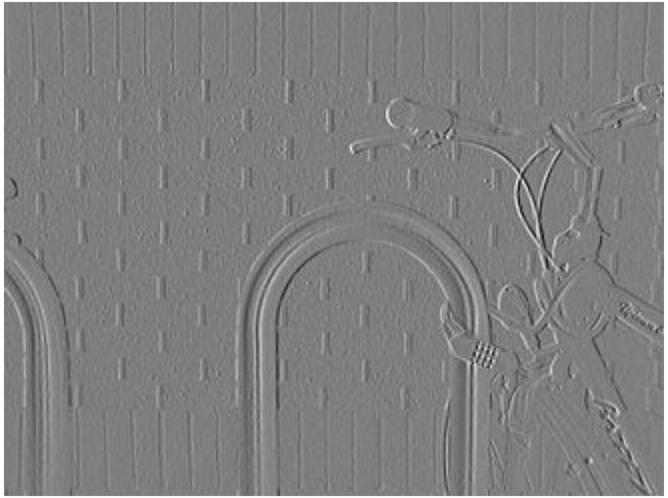
Using kernel convolution, sharp changes appear as bright lines (edges).

Sobel Operator

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

Gx

Normalized x-gradient from Sobel operator

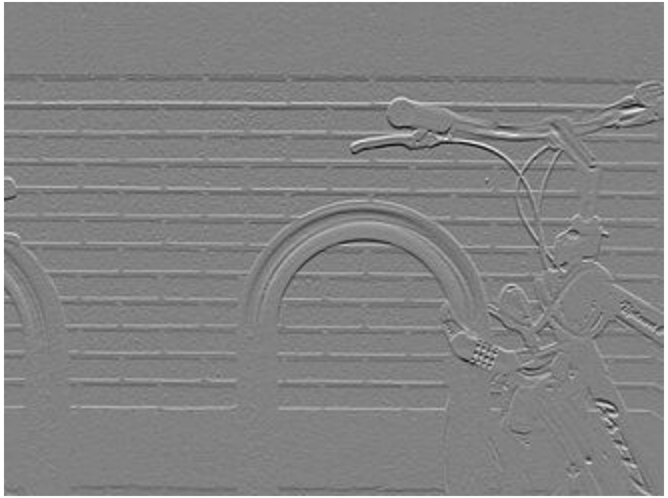


Sobel Operator

Normalized y-gradient from Sobel operator

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

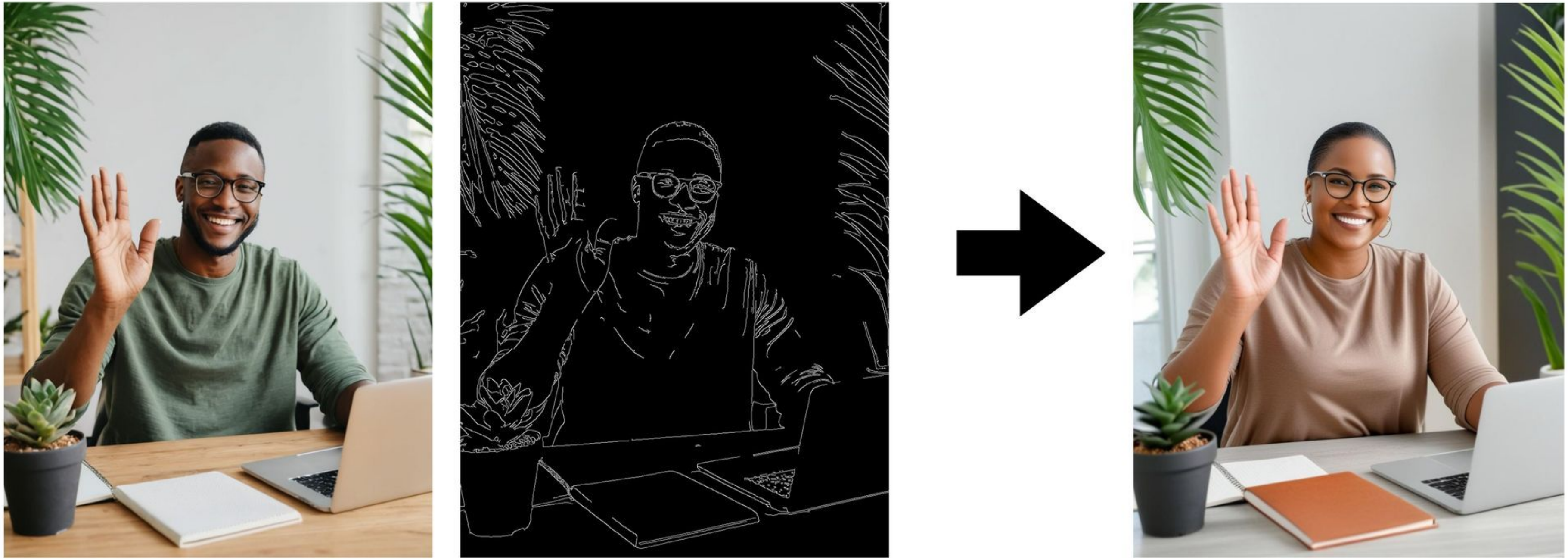
Gy



Edge Detection

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Applications



Applications

MIT License