Topic – Open Computer Vision (CV) Fundamentals – Week1

Created by Kirti Bala

Objective – Installation of open CV by using a programming language called Python along with its libraries like NumPy, such as SciPy and Matplotlib.

Key aspects of Open CV- Python

- ✓ Computer Vision and Image Processing

OpenCV-Python enables a wide range of operations on images and videos, including reading, writing, and displaying images, image transformations (resizing, cropping, rotation), color space conversions, filtering, edge detection, feature extraction, object detection, facial recognition, and many more .

- ✓ Python Bindings

OpenCV's core is written in C++, but it provides Python bindings (accessible via the cv2 module) that allow the developers to hold its capabilities within

Python scripts. This offers the performance benefits of C++ for computationally intensive operations while allowing for rapid development in Python.

Computer Vision

Computer Vision is the field of Computer science, that focuses on creating digital systems that can process, analyse , and make sense of visual data (images or videos) in the same way as humans do.
Computer Vision System Task

- Object Classification
- Object Identification
- Object Tracking
- Image Restoration
- Features Matching
- Video Motion Analysis

✓ Introduction to Open CV

- **Overview**
  Open CV (Open Source Computer Vision Library) which is used for Computer vision, Image processing , and Machine learning.

➢ It is written originally in C/C++, but now it has bindings for Python, Java, and others.

➢ Used in real-time computer vision applications like:
  - Face detection & Recognition
  - Object tracking
  - Augmented reality
  - Image filtering & Enhancement
  - Autonomous driving vision systems

Why Open CV is used ?

- Free to use (BSD License) – even commercially.

- Cross- platform – Works on Windows, Linux, macOS , Android, iOS.
- Highly optimized for speed (uses hardware acceleration like Intel IPP, CUDA for GPUs).

✓ History

- 1999 – Started at Intel Research to accelerate computer vision research.
- 2000 – First release (C interface).
- 2006 – OpenCV 1.0 official release.
- 2009 – OpenCV 2.0 released with new C++ interface.
- 2015 – OpenCV 3.0 released with major optimizations & modular design.
- 2018+ – OpenCV 4.x introduced better deep learning integration (DNN module).

- Today – Maintained by OpenCV.org and supported by a large global community.

Installation Process

A. Install via pip (Python)

```
pip install opencv-python

pip install opencv-python-headless
# For servers without GUI
```

- ✓ Process to done in command prompt

```
Users\KIRTI BALA>pip install opencv-python

Requirement already satisfied: opencv-python in c:\users\kirti bala\appdata\local\programs\python\python312\lib\site-packages (4.12.0.88)
```

Requirement already satisfied: numpy<2.3.0,>=2 in c:\users\kirti bala\appdata\local\programs\python\python312\lib\site-packages (from opencv-python) (2.2.6)

B. From Source (for customization)

- Download from : https://opencv.org/releases/
- Build using CMake (needed if you want GPU support, extra modules ,or optimization).

✓ **Basic Setup**

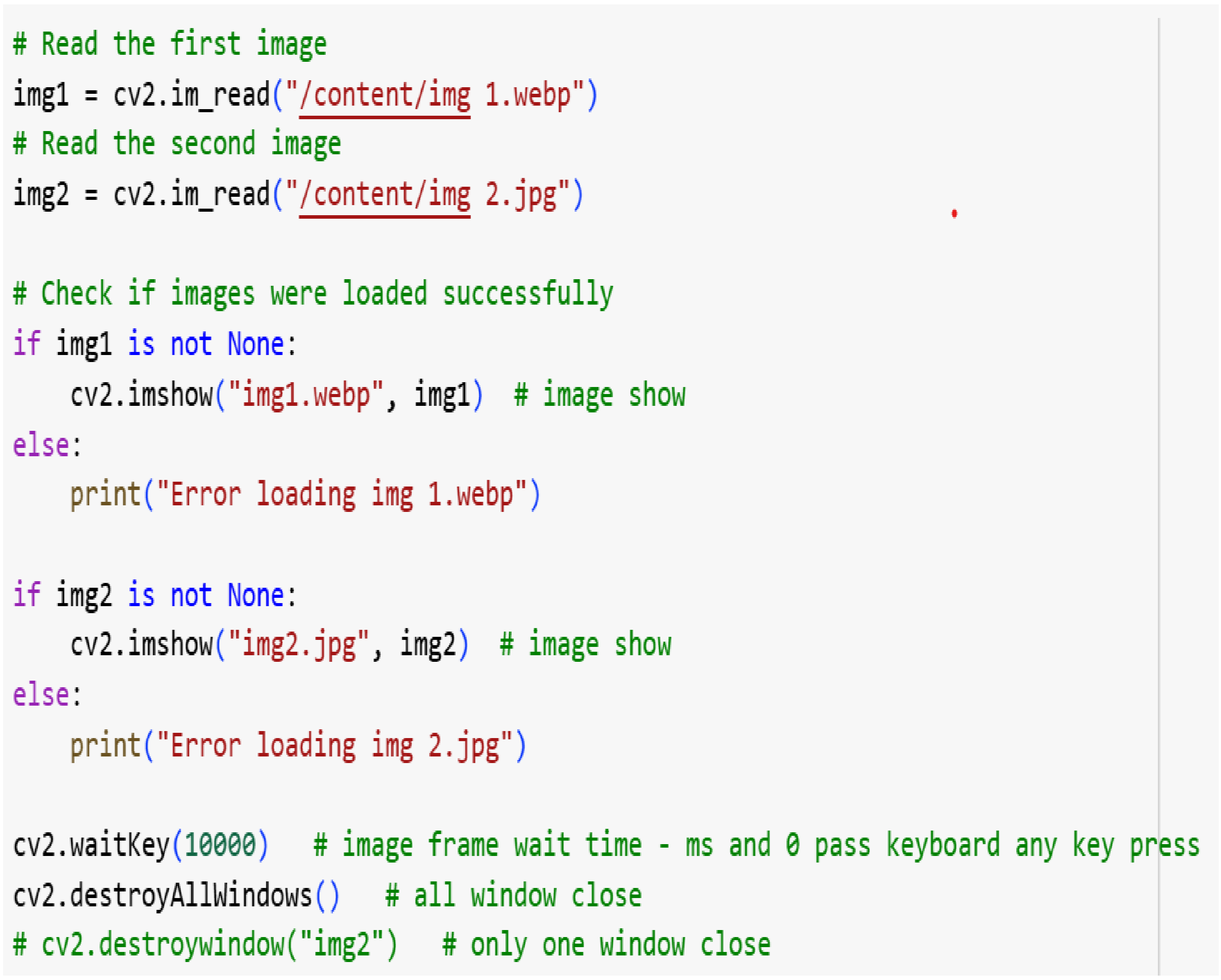
Once installed , verify:

Import cv2

Print(cv2._version_)

Version number should be noted (latest)

Reading & Displaying an image



```
# Read the first image
img1 = cv2.im_read("/content/img 1.webp")
# Read the second image
img2 = cv2.im_read("/content/img 2.jpg")

# Check if images were loaded successfully
if img1 is not None:
    cv2.imshow("img1.webp", img1)  # image show
else:
    print("Error loading img 1.webp")

if img2 is not None:
    cv2.imshow("img2.jpg", img2)  # image show
else:
    print("Error loading img 2.jpg")

cv2.waitKey(10000)   # image frame wait time - ms and 0 pass keyboard any key press
cv2.destroyAllWindows()   # all window close
# cv2.destroywindow("img2")   # only one window close
```

For Images file type supported is JPEG, JPG, PNG, BMP, TIFF.

For Videos file type supported is MP4, AVI, MKV (depends on codecs installed).

Core Functionality (core module)

The core module in Open CV provides :

- Basic data structures for storing images, videos, and other data.
- Matrix operations for numerical computations.
- Utility functions for handling types, drawing, RNG (random numbers), timing, etc.

In Python, the core features are mostly accessed via cv2 — and an OpenCV uses NumPy arrays to store and manipulate image data.

➢ Basic data structure

a. cv::Mat (C++ name) / NumPy ndarray (Python).

Mat is the fundamental data container in OpenCV.

In Python, when you read an image with cv2.imread(), it returns a NumPy array.

Structure:

Rows = image height

Cols = image width

Channels = number of color components per pixel

(3 for BGR, 1 for grayscale)

Example:

```
import cv2
img = cv2.imread('cat.jpg')
print(type(img))        # <class
'numpy.ndarray'>
```

```
print(img.shape)       # (height, width, channels)
print(img.dtype)       # uint8
```

b) Scalar

Represents a 4-element vector (Blue, Green, Red, Alpha).

Used for colors in drawing functions.

```
blue_color = (255, 0, 0)  # BGR format
```

c) Size & Point

Size → Width and height.

Point → x, y coordinates. (Used internally in C++, in Python we usually use tuples.)

<u>Operations on Arrays</u>

Since images are stored as NumPy arrays, you can apply both OpenCV functions and NumPy operations.

a) Access & Modify Pixels

```
px = img[100, 200]      # Pixel at row=100, col=200
print(px)               # [B, G, R]
img[100, 200] = [0, 0, 255]  # Set pixel to red
```

b) Region of Interest (ROI)

```
roi = img[50:150, 100:200]  # Crop region
```

c) Arithmetic Operations

OpenCV provides optimized methods:

```
result = cv2.add(img1, img2)    # Saturated addition
result = cv2.subtract(img1, img2)
```

result = cv2.multiply(img1, img2)

result = cv2.divide(img1, img2)

Or directly use NumPy (img1 + img2), but OpenCV handles overflow differently.

d) Bitwise Operations

Useful for masking:

mask = cv2.bitwise_and(img, img, mask=mask_img)

Utility Functions in Core Module

a) Type & Size Info

print(img.shape)   # (rows, cols, channels)

print(img.size)   # Total number of elements

print(img.dtype)   # Data type (uint8, float32, etc.)

b) Timing

```
t1 = cv2.getTickCount()
#  code ...
t2 = cv2.getTickCount()
time_taken = (t2 - t1) /
cv2.getTickFrequency()
print(f"Time: {time_taken} seconds")
```

c) Random Number Generation

```
rng = cv2.RNG()
rand_val = rng.uniform(0, 255)
```

d) Color Conversions

```
gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
```

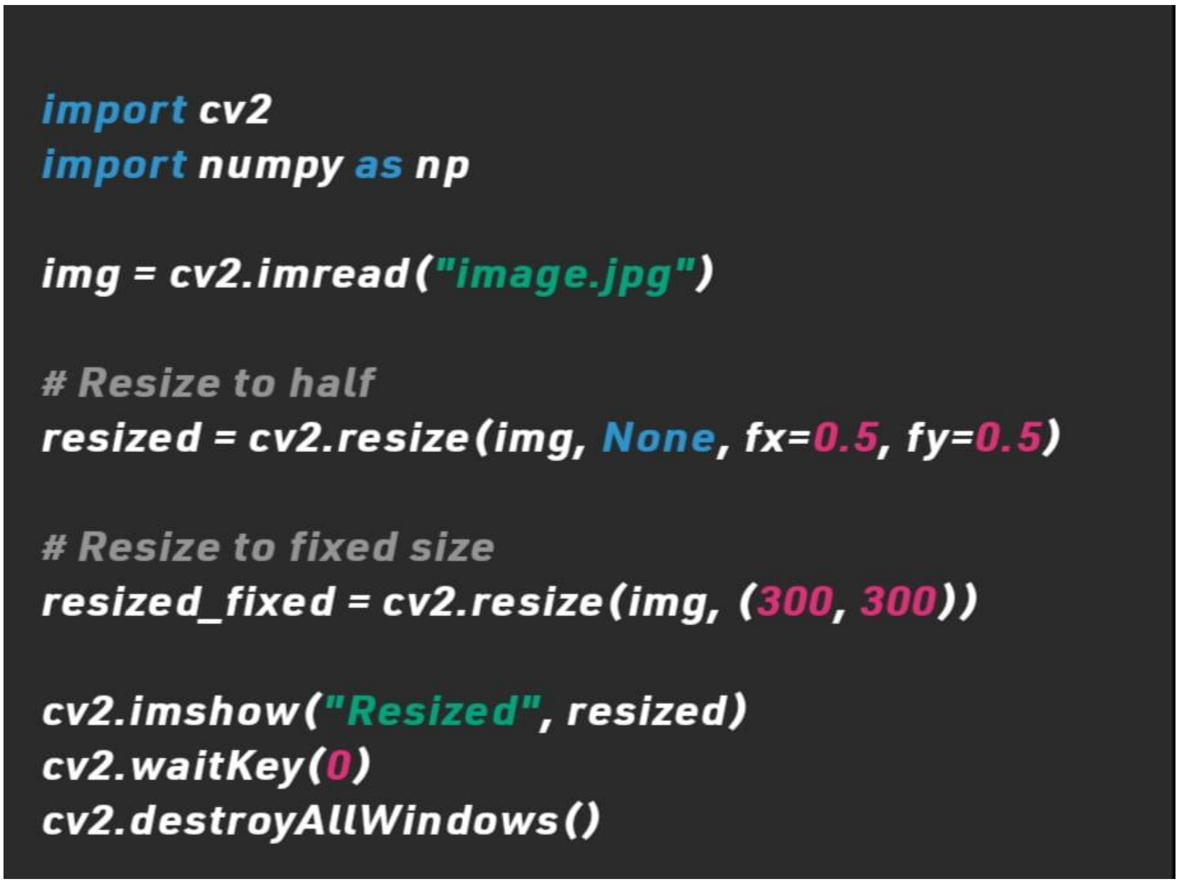
✅ Mat/ NumPy arrays are used to store image data.

- ✓ The core module allows to access, modify, and combine these arrays efficiently.
- ✓ Utility functions handle size, type, colors, random values, and timing.

Image Processing

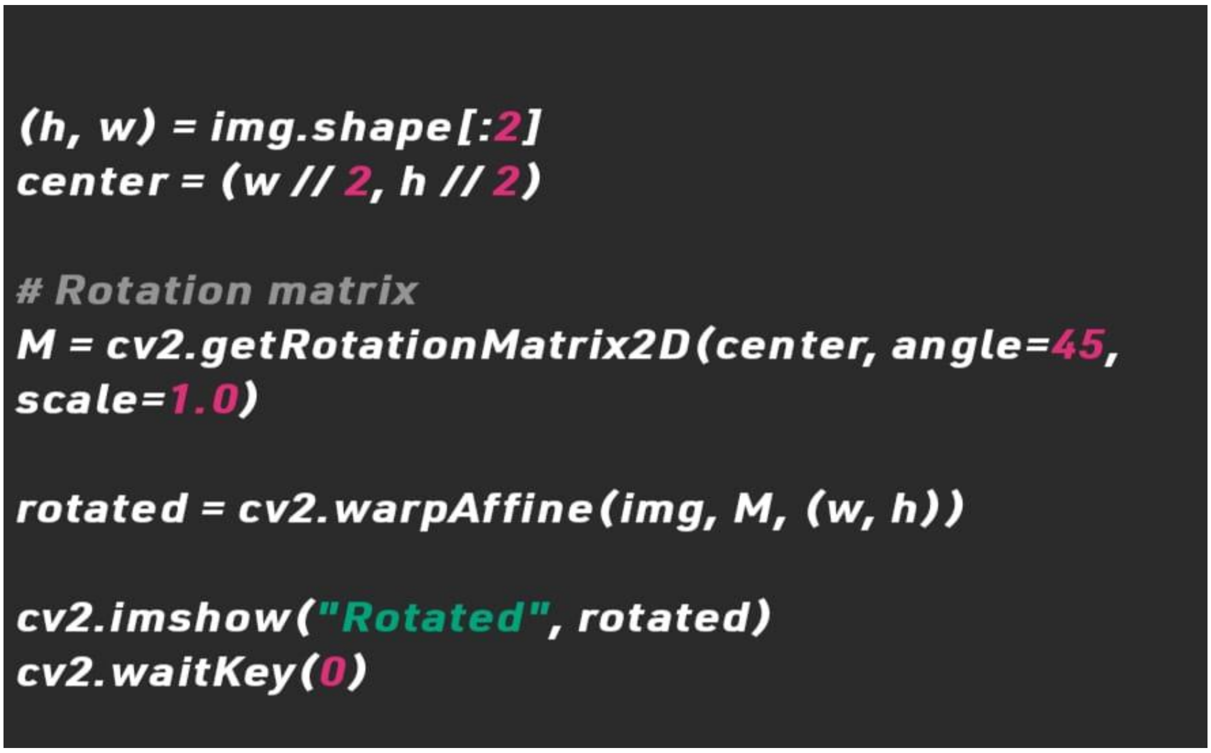
1. *Image Transformation*

It involves the image's geometry – size, orientation, or perspective.



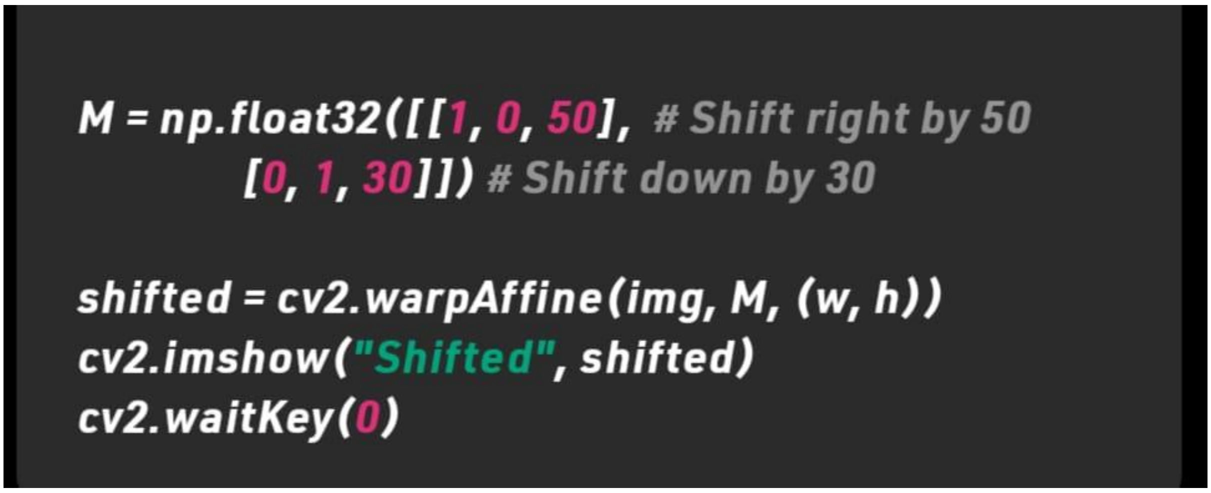
```python
import cv2
import numpy as np

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

# Resize to half
resized = cv2.resize(img, None, fx=0.5, fy=0.5)

# Resize to fixed size
resized_fixed = cv2.resize(img, (300, 300))

cv2.imshow("Resized", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

a. Rotation



```
(h, w) = img.shape[:2]
center = (w // 2, h // 2)

# Rotation matrix
M = cv2.getRotationMatrix2D(center, angle=45,
scale=1.0)

rotated = cv2.warpAffine(img, M, (w, h))

cv2.imshow("Rotated", rotated)
cv2.waitKey(0)
```

b. Translation (shifting)

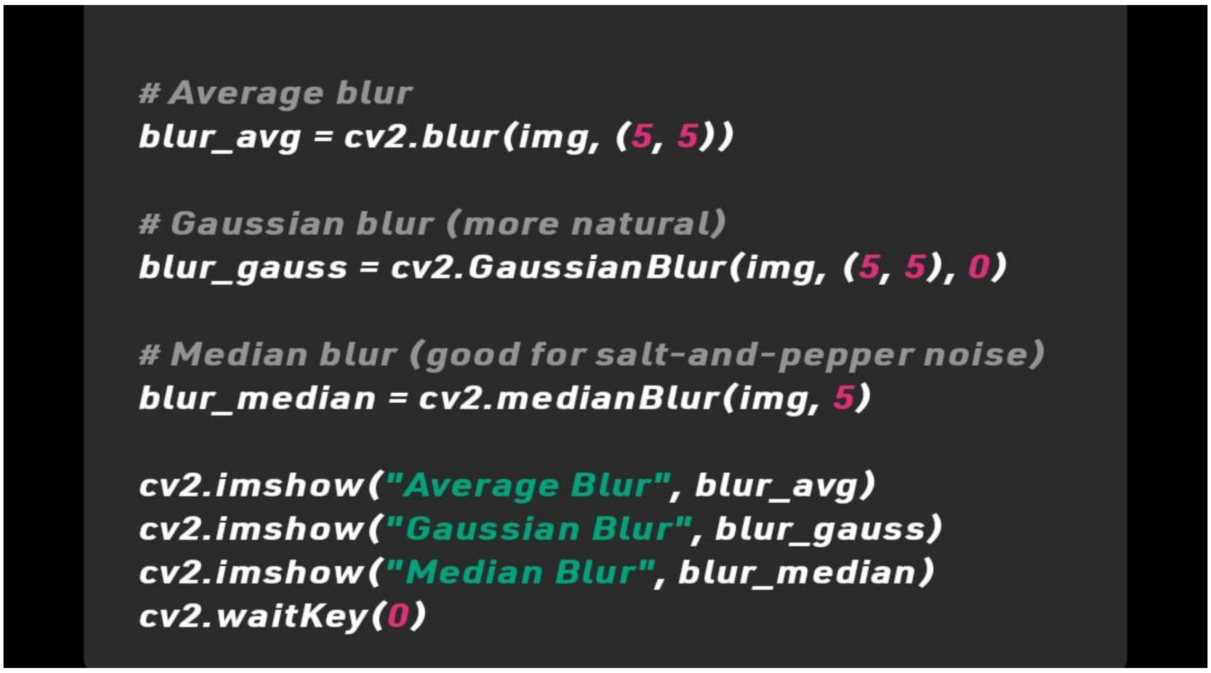


```
M = np.float32([[1, 0, 50],  # Shift right by 50
                [0, 1, 30]]) # Shift down by 30

shifted = cv2.warpAffine(img, M, (w, h))
cv2.imshow("Shifted", shifted)
cv2.waitKey(0)
```

2. *Filtering*

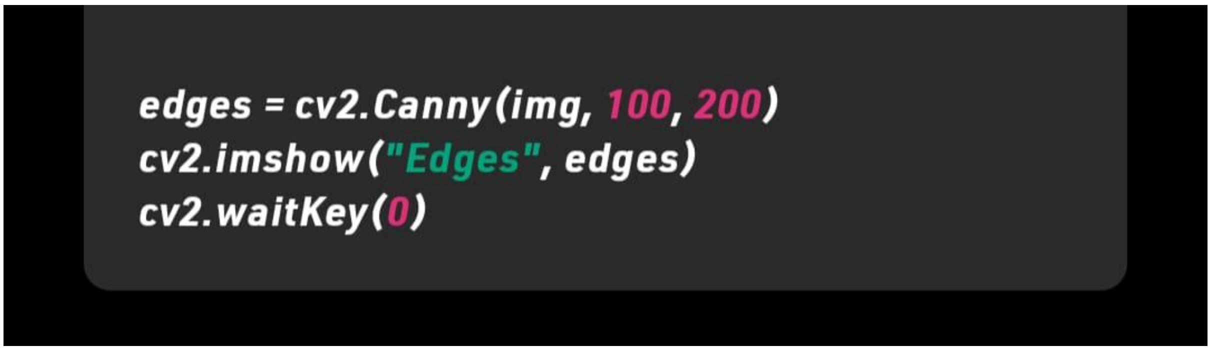
Filtering modifies pixel values to achieve effects like smoothing, sharpening, or edge detection.

a. Blurring (Smoothing)



```
# Average blur
blur_avg = cv2.blur(img, (5, 5))

# Gaussian blur (more natural)
blur_gauss = cv2.GaussianBlur(img, (5, 5), 0)

# Median blur (good for salt-and-pepper noise)
blur_median = cv2.medianBlur(img, 5)

cv2.imshow("Average Blur", blur_avg)
cv2.imshow("Gaussian Blur", blur_gauss)
cv2.imshow("Median Blur", blur_median)
cv2.waitKey(0)
```

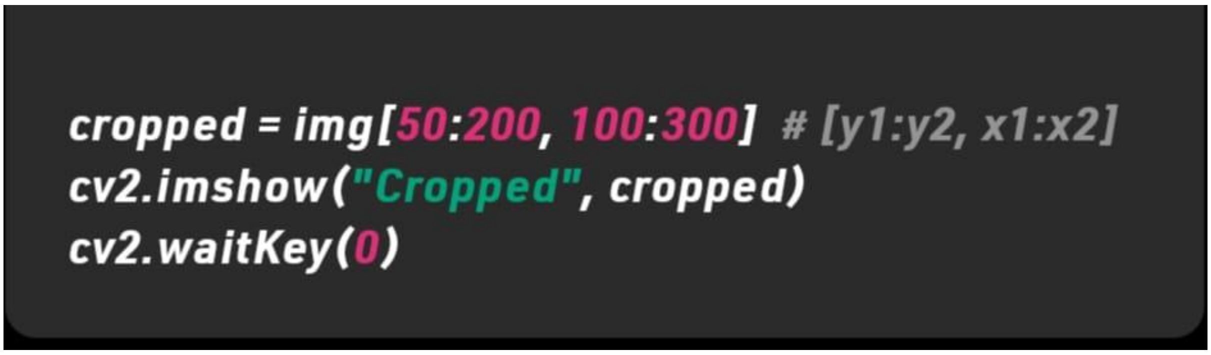
b. Edge Detection



```
edges = cv2.Canny(img, 100, 200)
cv2.imshow("Edges", edges)
cv2.waitKey(0)
```

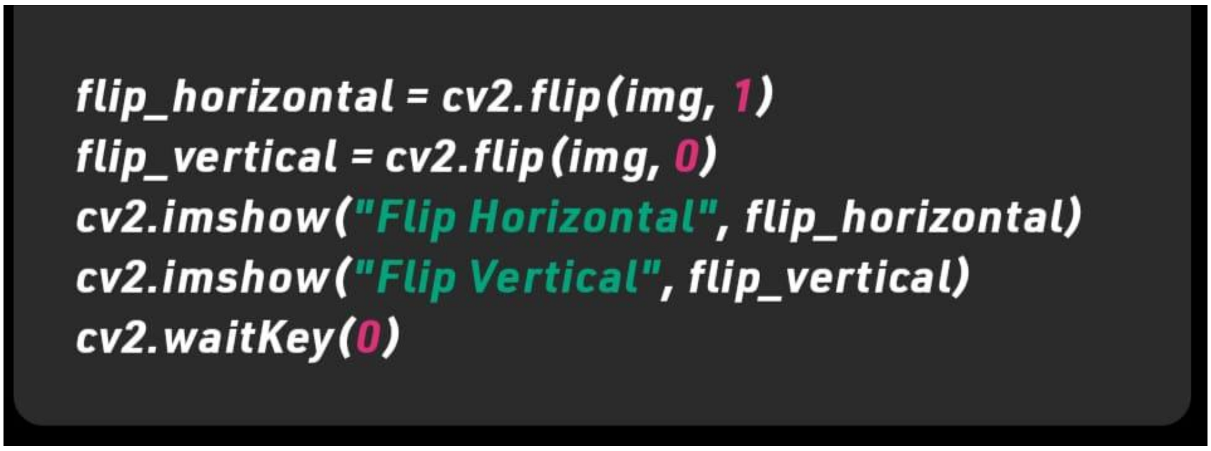
3. *Geometric Operations*

a. Cropping



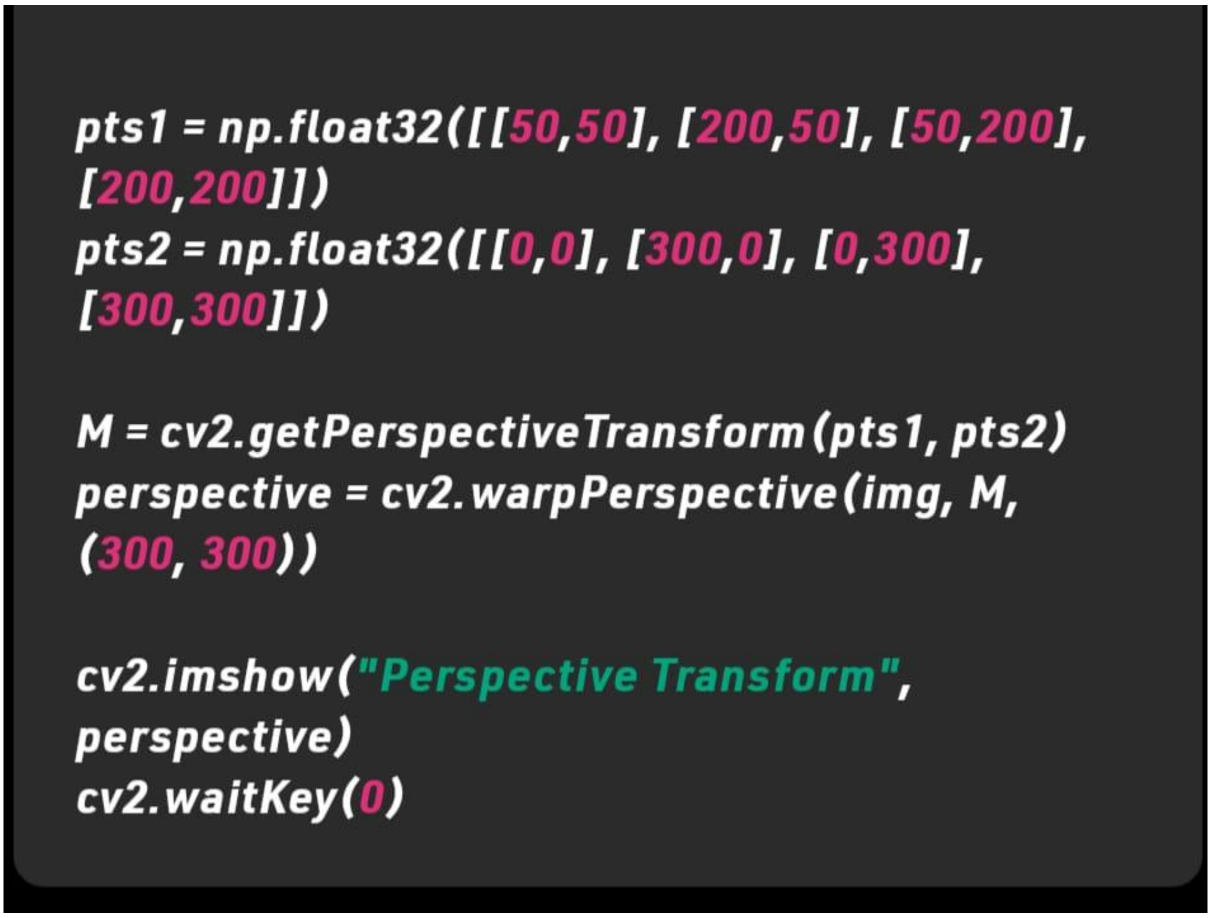
```
cropped = img[50:200, 100:300]  # [y1:y2, x1:x2]
cv2.imshow("Cropped", cropped)
cv2.waitKey(0)
```

b. Flipping



```
flip_horizontal = cv2.flip(img, 1)
flip_vertical = cv2.flip(img, 0)
cv2.imshow("Flip Horizontal", flip_horizontal)
cv2.imshow("Flip Vertical", flip_vertical)
cv2.waitKey(0)
```

c. Perspective Transformation

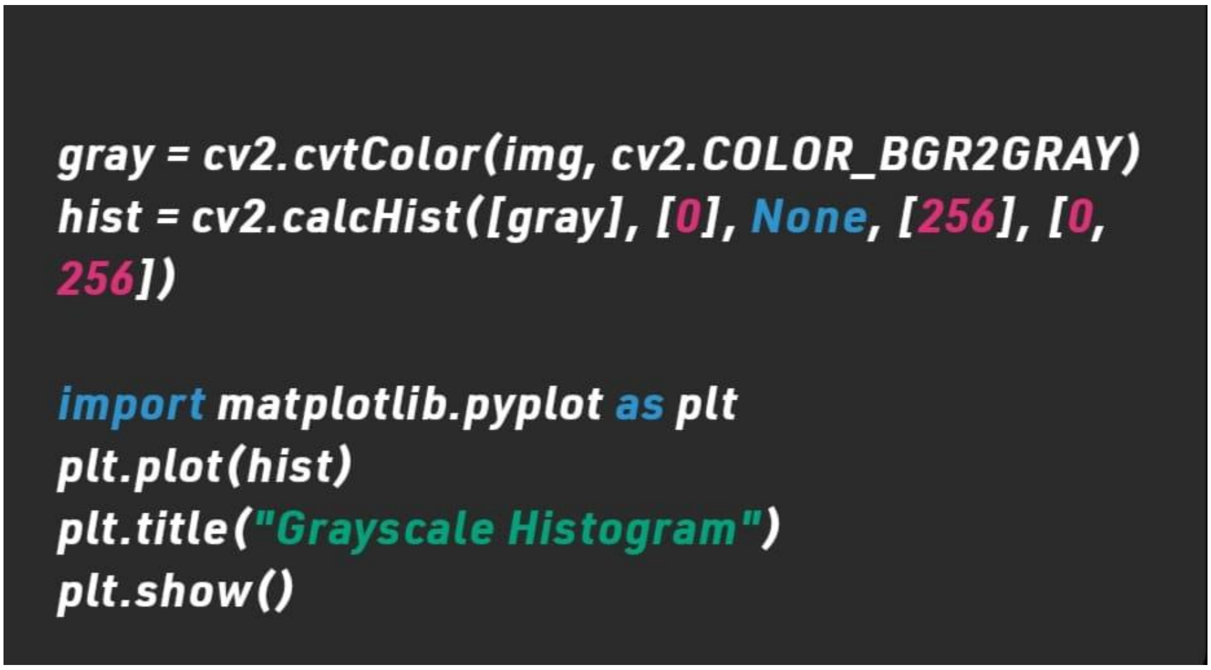


```
pts1 = np.float32([[50,50], [200,50], [50,200],
[200,200]])
pts2 = np.float32([[0,0], [300,0], [0,300],
[300,300]])

M = cv2.getPerspectiveTransform(pts1, pts2)
perspective = cv2.warpPerspective(img, M,
(300, 300))

cv2.imshow("Perspective Transform",
perspective)
cv2.waitKey(0)
```

4. Histograms

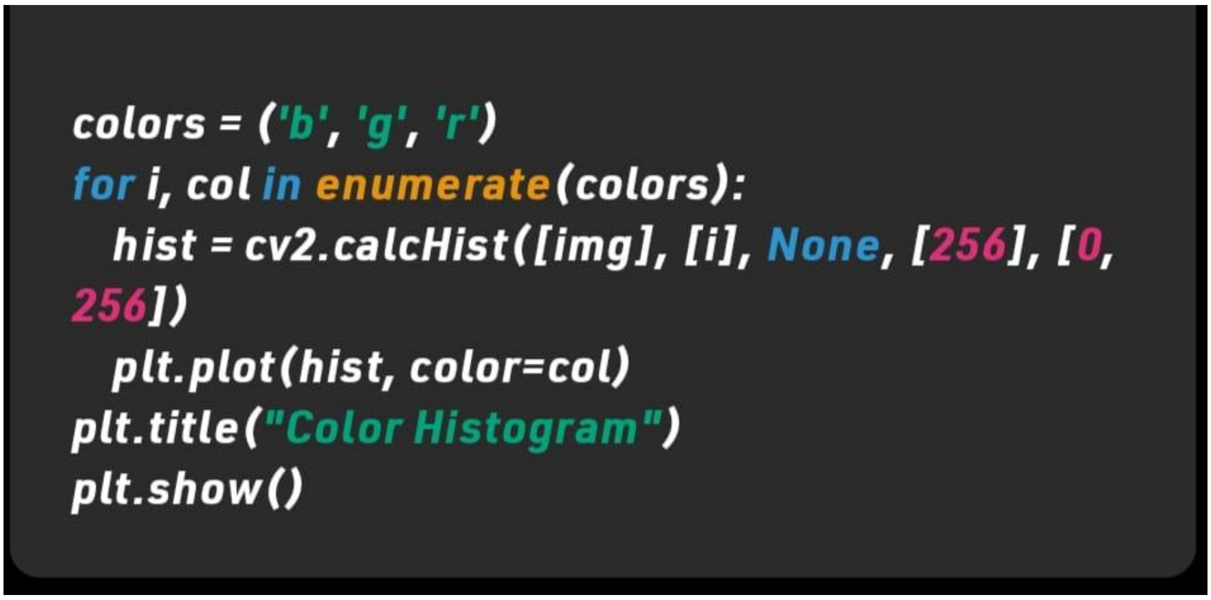
Histograms shows the distribution of pixel intensity values.

a. Grayscale Histogram



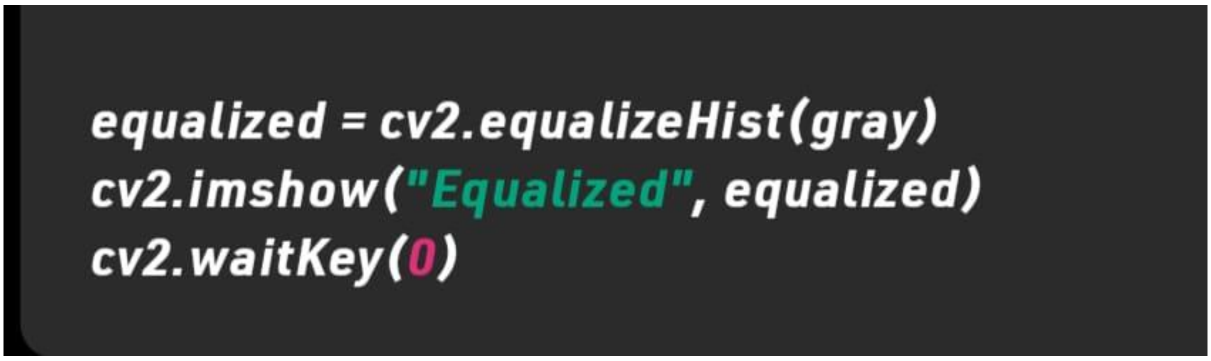
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hist = cv2.calcHist([gray], [0], None, [256], [0,
256])

import matplotlib.pyplot as plt
plt.plot(hist)
plt.title("Grayscale Histogram")
plt.show()
```

b. Color Histogram



```
colors = ('b', 'g', 'r')
for i, col in enumerate(colors):
   hist = cv2.calcHist([img], [i], None, [256], [0,
256])
   plt.plot(hist, color=col)
plt.title("Color Histogram")
plt.show()
```

c. Histogram Equalization (Enhances contrast)



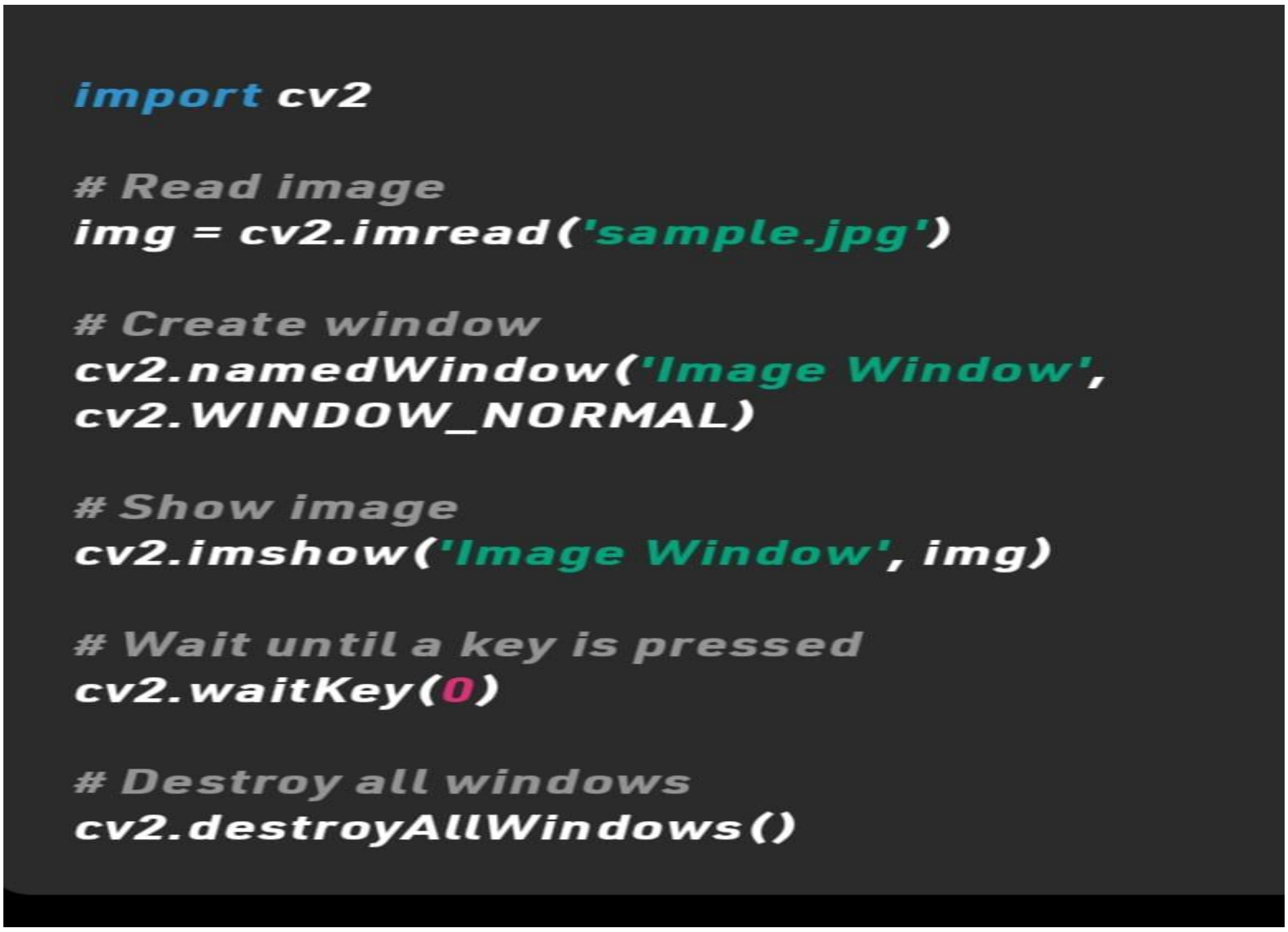
```
equalized = cv2.equalizeHist(gray)
cv2.imshow("Equalized", equalized)
cv2.waitKey(0)
```

1. **Overview of the Modules**
    - highgui – It handles the graphical user interface (GUI) for the images/ video display, mouse events, and trackbars.
    - Imgcodecs – It handles the image file reading and writing (various formats like JPG, PNG, TIFF, JPEG etc).
    - Videoio – As it handles the video capturing from cameras and reading/ writing from/ to video files.
2. **highgui Module (GUI Operations)**

HighGUI provides simple interfaces for:

- Displaying images (cv2.imshow)
- Creating windows (cv2.namedWindow)
- Adding trackbars (cv2.createTrackbar)
- Handling keyboard/mouse events (cv2.waitKey, cv2.setMouseCallback)

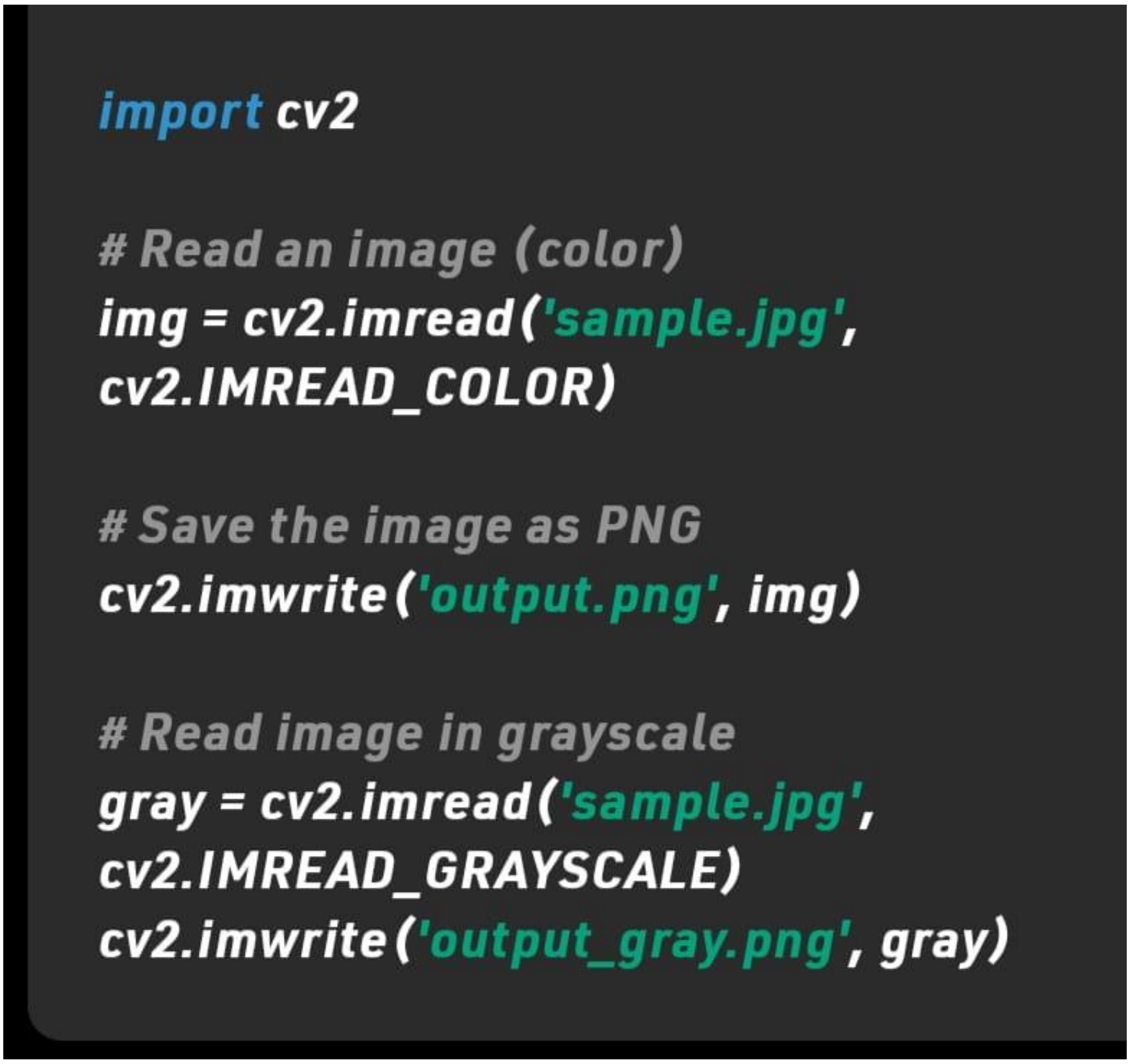
EXAMPLES – Display an image:



```
import cv2

# Read image
img = cv2.imread('sample.jpg')

# Create window
cv2.namedWindow('Image Window',
cv2.WINDOW_NORMAL)

# Show image
cv2.imshow('Image Window', img)

# Wait until a key is pressed
cv2.waitKey(0)

# Destroy all windows
cv2.destroyAllWindows()
```

3. imgcodecs Module (Image Reading / Writing)

It is the module which handles the reading and writing of image files in various formats(JPG , BMP). It provides functions like imread() to load images from disk and imwrite() to save images.

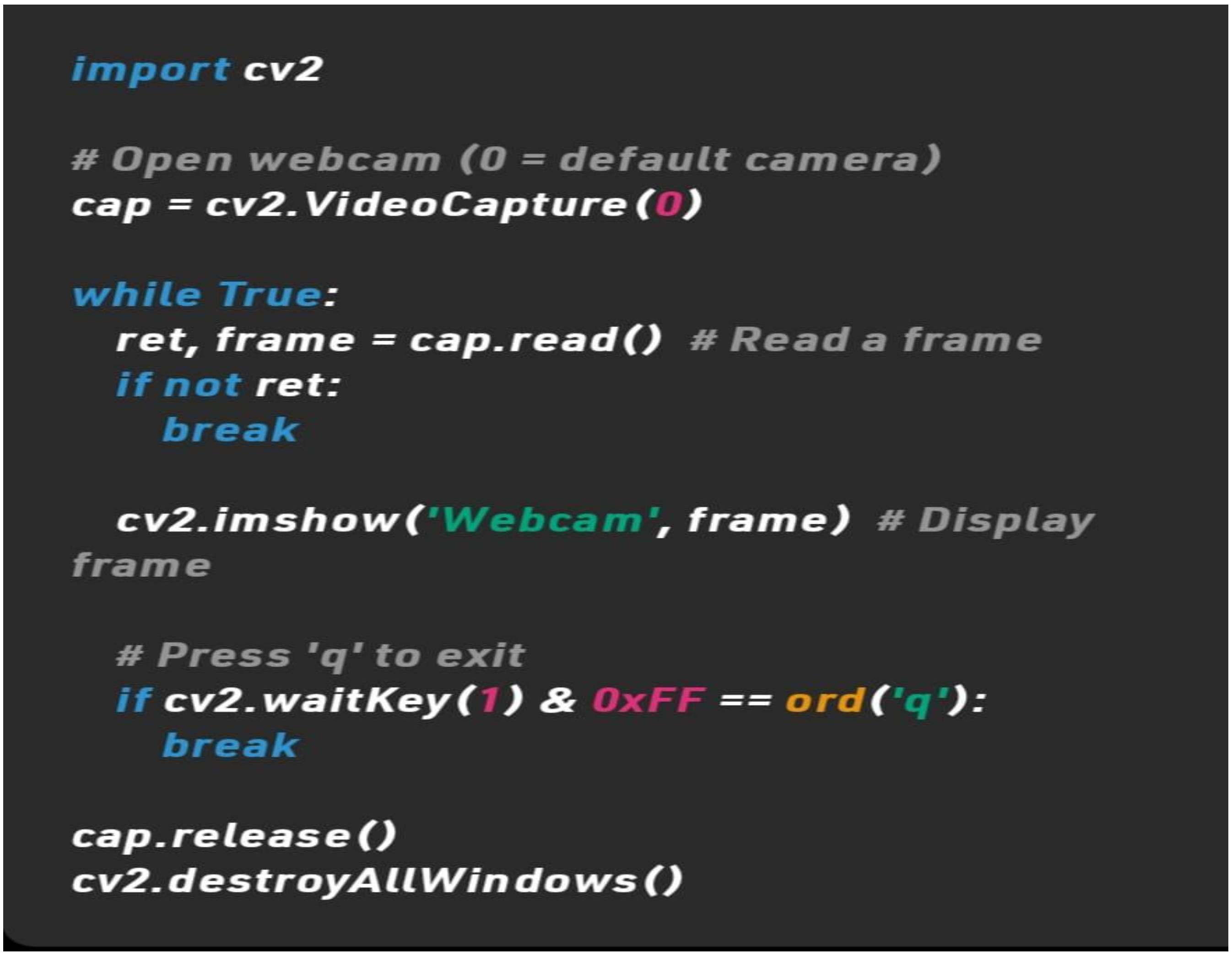


```
import cv2

# Read an image (color)
img = cv2.imread('sample.jpg',
cv2.IMREAD_COLOR)

# Save the image as PNG
cv2.imwrite('output.png', img)

# Read image in grayscale
gray = cv2.imread('sample.jpg',
cv2.IMREAD_GRAYSCALE)
cv2.imwrite('output_gray.png', gray)
```

- **Cv2.imread_color (default)**
- **Cv2.imread_grayscale**
- **Cv2.imread_unchanged (keeps alpha/ transparency)**

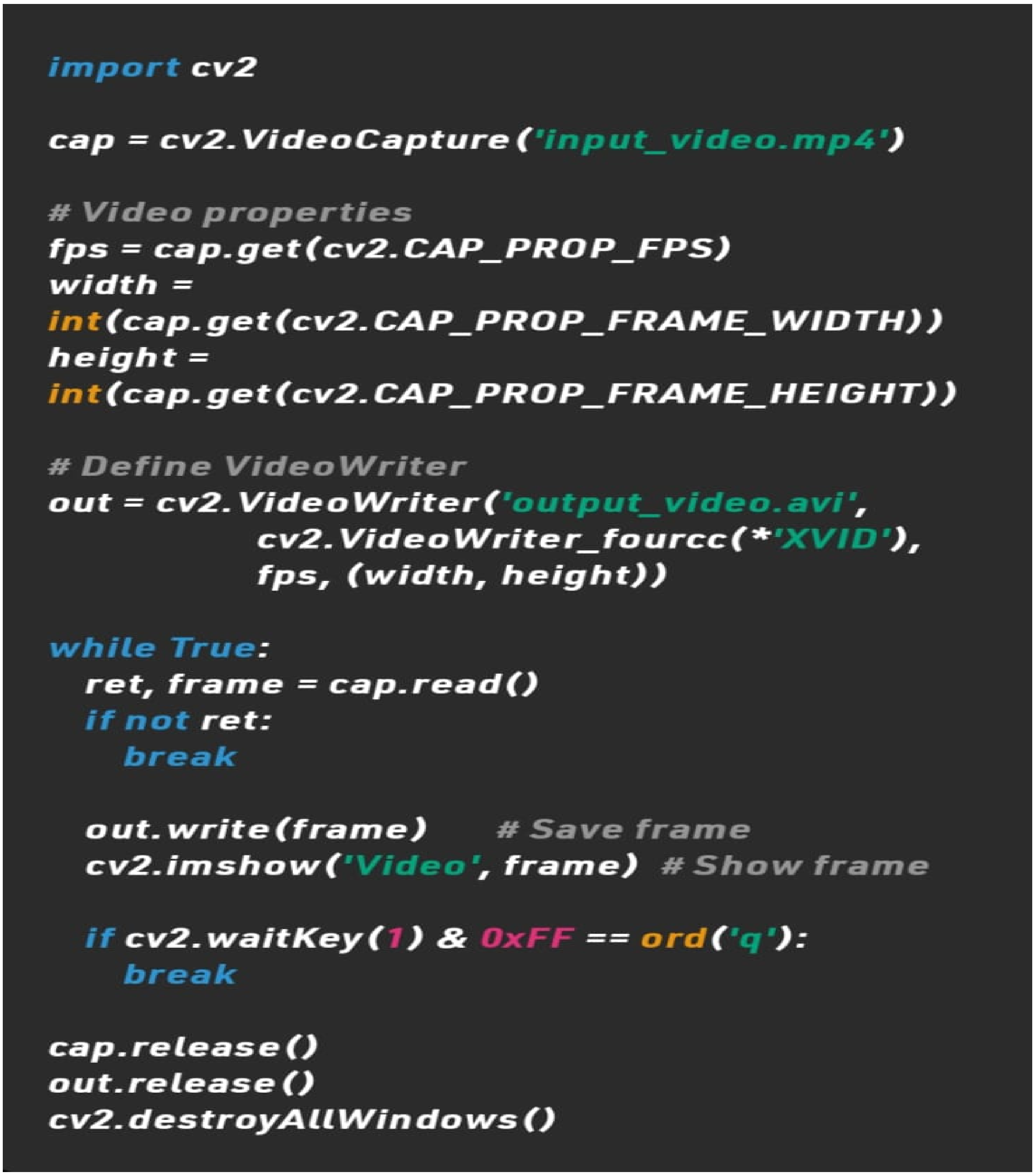
4. videoio Module (Video Capture & Writing)

This module deals with video input/output. It allows reading video streams from files or cameras using VideoCapture() and writing video frames to files using VideoWriter().



```
import cv2

# Open webcam (0 = default camera)
cap = cv2.VideoCapture(0)

while True:
   ret, frame = cap.read()  # Read a frame
   if not ret:
      break

   cv2.imshow('Webcam', frame)  # Display
frame

   # Press 'q' to exit
   if cv2.waitKey(1) & 0xFF == ord('q'):
      break

cap.release()
cv2.destroyAllWindows()
```

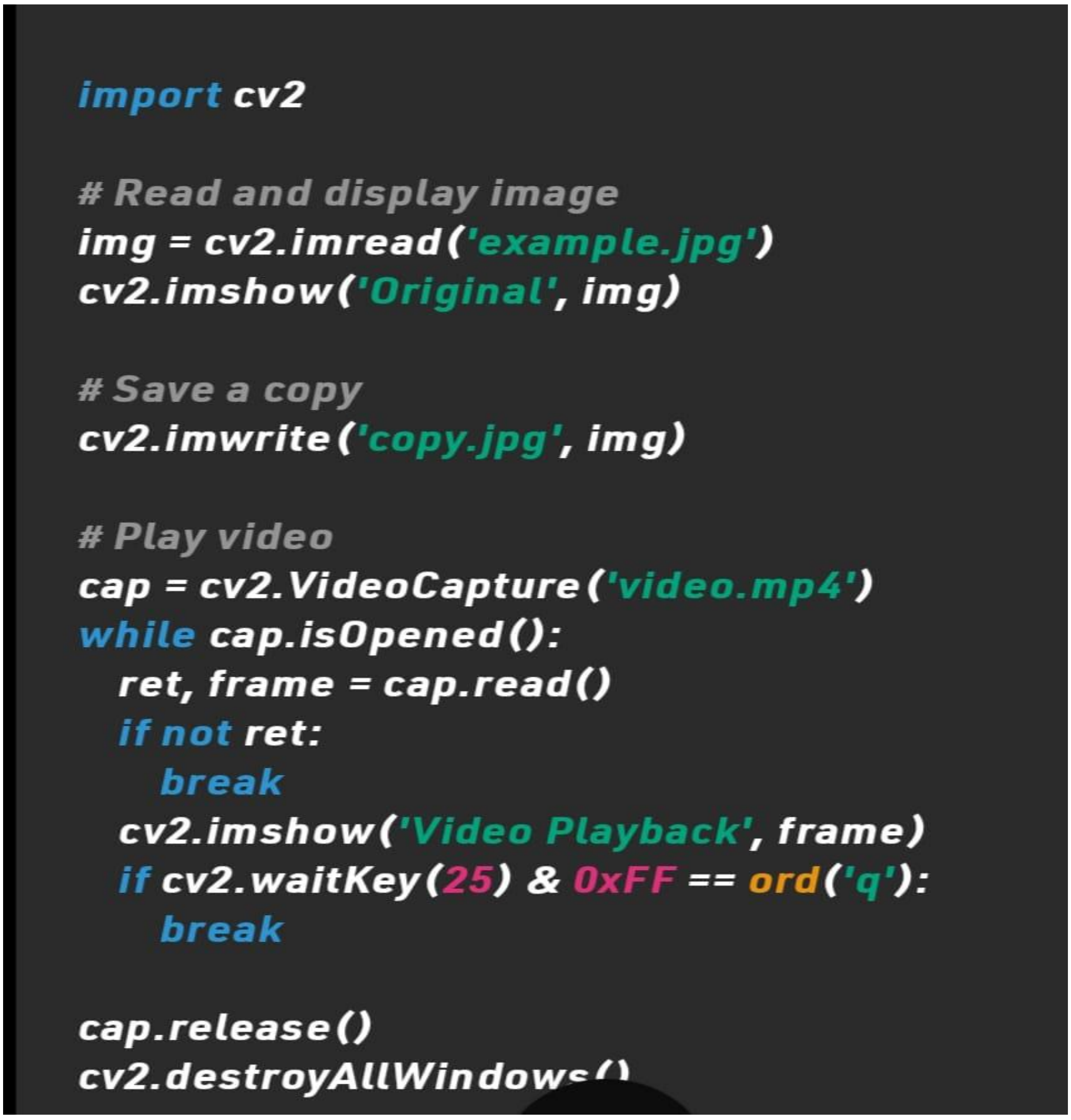
Example – Save Video:



```python
import cv2

cap = cv2.VideoCapture('input_video.mp4')

# Video properties
fps = cap.get(cv2.CAP_PROP_FPS)
width =
int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height =
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Define VideoWriter
out = cv2.VideoWriter('output_video.avi',
            cv2.VideoWriter_fourcc(*'XVID'),
            fps, (width, height))

while True:
  ret, frame = cap.read()
  if not ret:
    break

  out.write(frame)     # Save frame
  cv2.imshow('Video', frame)  # Show frame

  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
out.release()
cv2.destroyAllWindows()
```

If They Work Together

Example: Load an image → Display it → Save → Play video from file



```python
import cv2

# Read and display image
img = cv2.imread('example.jpg')
cv2.imshow('Original', img)

# Save a copy
cv2.imwrite('copy.jpg', img)

# Play video
cap = cv2.VideoCapture('video.mp4')
while cap.isOpened():
   ret, frame = cap.read()
   if not ret:
      break
   cv2.imshow('Video Playback', frame)
   if cv2.waitKey(25) & 0xFF == ord('q'):
      break

cap.release()
cv2.destroyAllWindows()
```

Calibration Process:

It typically involves capturing images of a known calibration pattern (e.g., chessboard, circular grid) from different poses and using OpenCV functions like cv2.calibrateCamera() to compute the camera matrix, distortion coefficients, rotation vectors, and translation vectors.

Camera Calibration & 3D Reconstruction (calib3d module)

The calib3d module handles:

Camera Model – Mathematical representation of how a camera forms images.

Undistortion – Removing lens distortion from captured images.

Stereo Vision – Depth estimation from two camera views.

Pose Estimation – Finding the position & orientation of an object/camera in 3D space.

1. Camera model (pinhole + distortion)

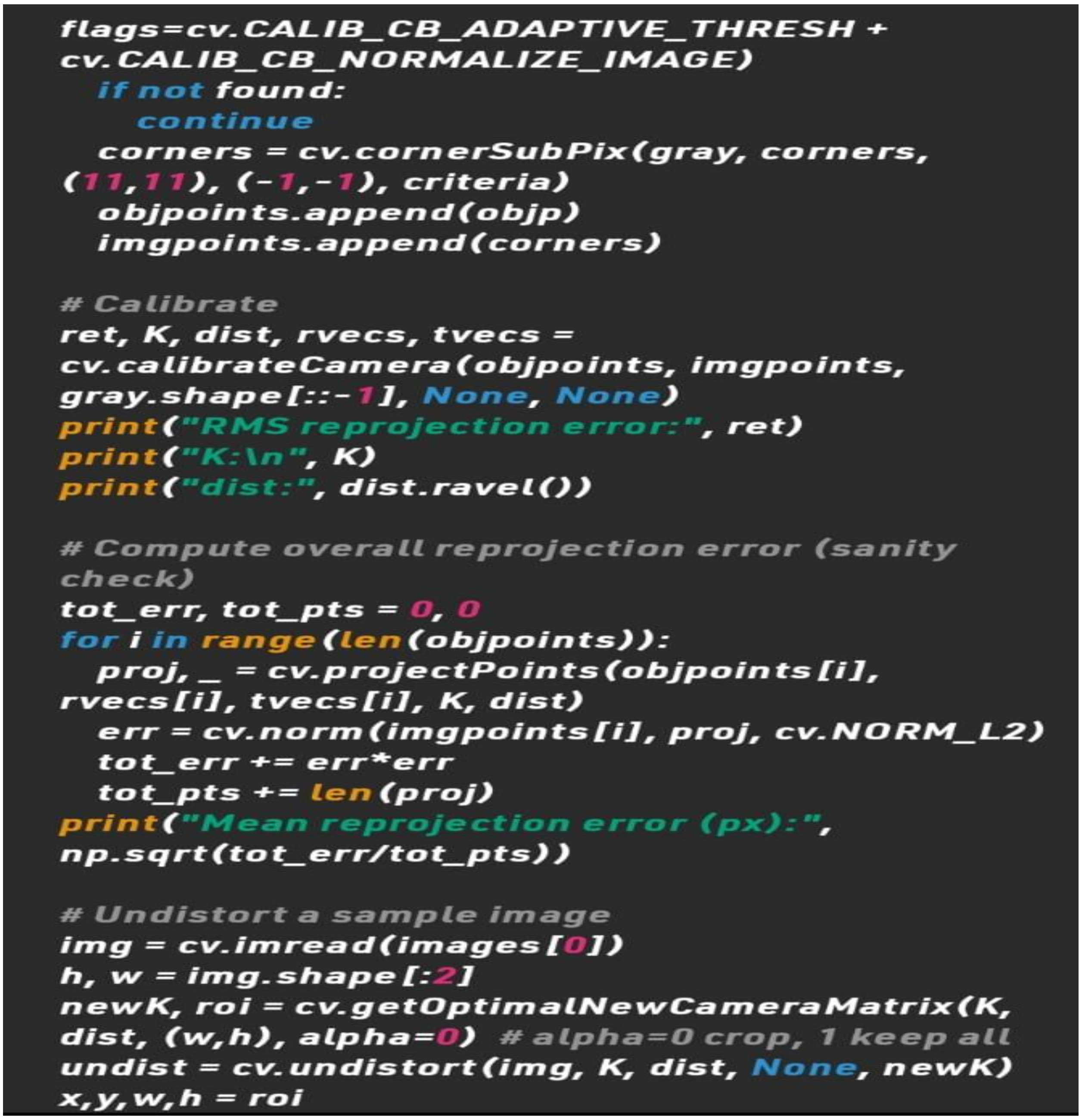
We map a 3D point $\mathbf{X} = [X, Y, Z]^T$ ***to pixels*** $\mathbf{x} = [u, v]^T$***:***

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} [\, R \mid t \,] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

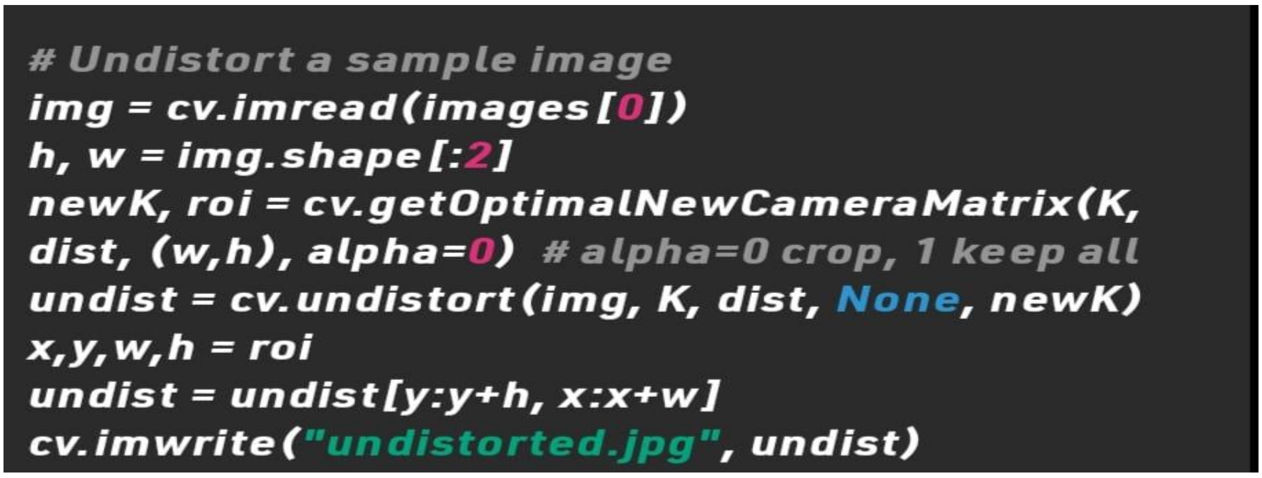
- Image point = (u, v) relates to 3Dpoint Xc = (X,Y)

2. Single- camera calibration (chessboard) + undistortion

```
import cv2 as cv
import numpy as np
import glob

# --- Settings ---
chessboard_size = (9, 6)        # inner corners
(columns, rows)
square_size = 0.024             # meters (use your
real square size)
img_dir = "calib_imgs/*.jpg"      # path to your
calibration images

# --- Prepare object points: (0,0,0) ... (8,5,0)
scaled by square_size ---
objp =
np.zeros((chessboard_size[0]*chessboard_size
[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:chessboard_size[0],
0:chessboard_size[1]].T.reshape(-1, 2)
objp *= square_size

objpoints = []  # 3D points (world)
imgpoints = []  # 2D points (image)

criteria = (cv.TERM_CRITERIA_EPS +
cv.TERM_CRITERIA_MAX_ITER, 30, 1e-6)

images = glob.glob(img_dir)
for fname in images:
   img = cv.imread(fname)
   gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
   found, corners =
cv.findChessboardCorners(gray,
chessboard_size,
```

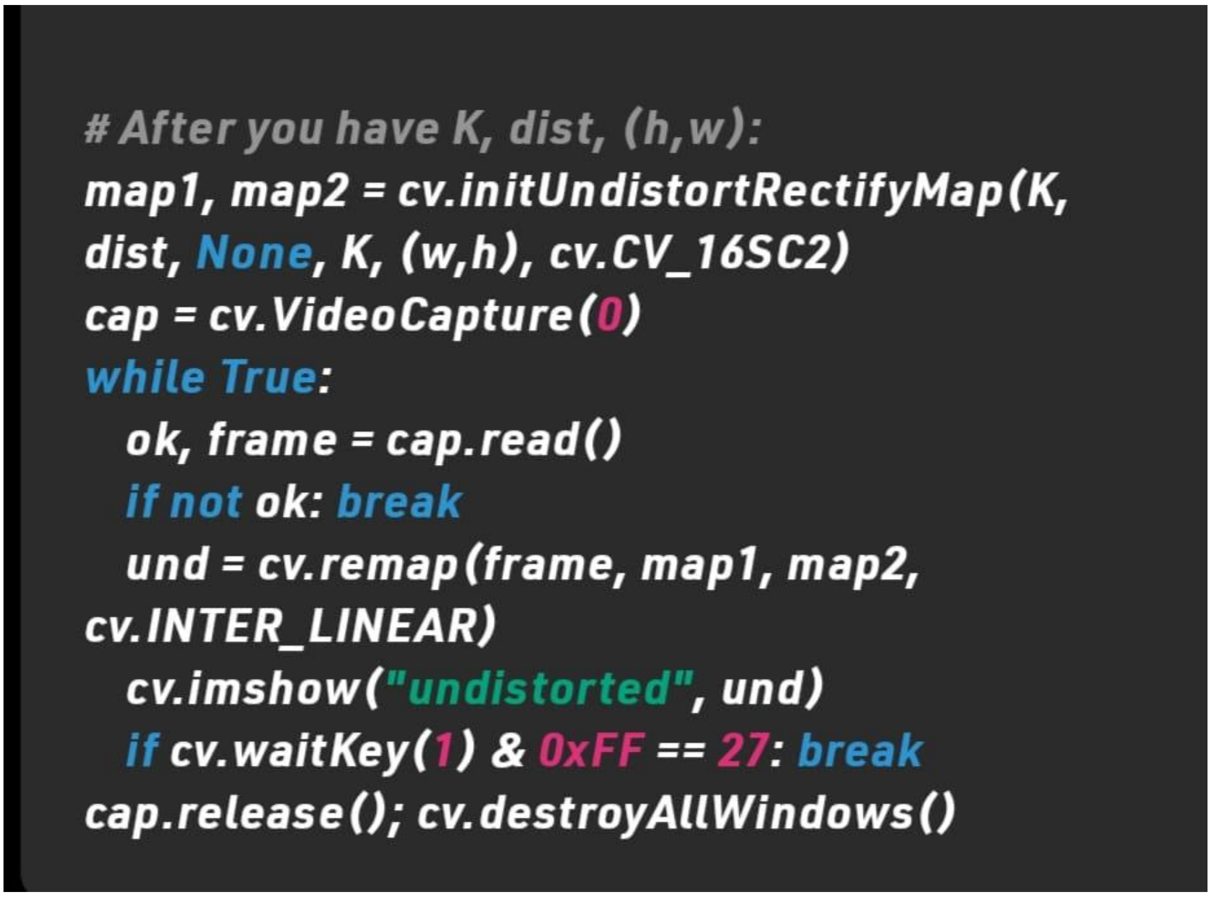


```
flags=cv.CALIB_CB_ADAPTIVE_THRESH +
cv.CALIB_CB_NORMALIZE_IMAGE)
  if not found:
    continue
  corners = cv.cornerSubPix(gray, corners,
(11,11), (-1,-1), criteria)
  objpoints.append(objp)
  imgpoints.append(corners)

# Calibrate
ret, K, dist, rvecs, tvecs =
cv.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
print("RMS reprojection error:", ret)
print("K:\n", K)
print("dist:", dist.ravel())

# Compute overall reprojection error (sanity
check)
tot_err, tot_pts = 0, 0
for i in range(len(objpoints)):
  proj, _ = cv.projectPoints(objpoints[i],
rvecs[i], tvecs[i], K, dist)
  err = cv.norm(imgpoints[i], proj, cv.NORM_L2)
  tot_err += err*err
  tot_pts += len(proj)
print("Mean reprojection error (px):",
np.sqrt(tot_err/tot_pts))

# Undistort a sample image
img = cv.imread(images[0])
h, w = img.shape[:2]
newK, roi = cv.getOptimalNewCameraMatrix(K,
dist, (w,h), alpha=0) # alpha=0 crop, 1 keep all
undist = cv.undistort(img, K, dist, None, newK)
x,y,w,h = roi
```



```
# Undistort a sample image
img = cv.imread(images[0])
h, w = img.shape[:2]
newK, roi = cv.getOptimalNewCameraMatrix(K,
dist, (w,h), alpha=0) # alpha=0 crop, 1 keep all
undist = cv.undistort(img, K, dist, None, newK)
x,y,w,h = roi
undist = undist[y:y+h, x:x+w]
cv.imwrite("undistorted.jpg", undist)
```

2. Real – time undistortion map (fast for video)



```
# After you have K, dist, (h,w):
map1, map2 = cv.initUndistortRectifyMap(K,
dist, None, K, (w,h), cv.CV_16SC2)
cap = cv.VideoCapture(0)
while True:
   ok, frame = cap.read()
   if not ok: break
   und = cv.remap(frame, map1, map2,
cv.INTER_LINEAR)
   cv.imshow("undistorted", und)
   if cv.waitKey(1) & 0xFF == 27: break
cap.release(); cv.destroyAllWindows()
```

3. Stereo calibration, Rectification, disparity → depth

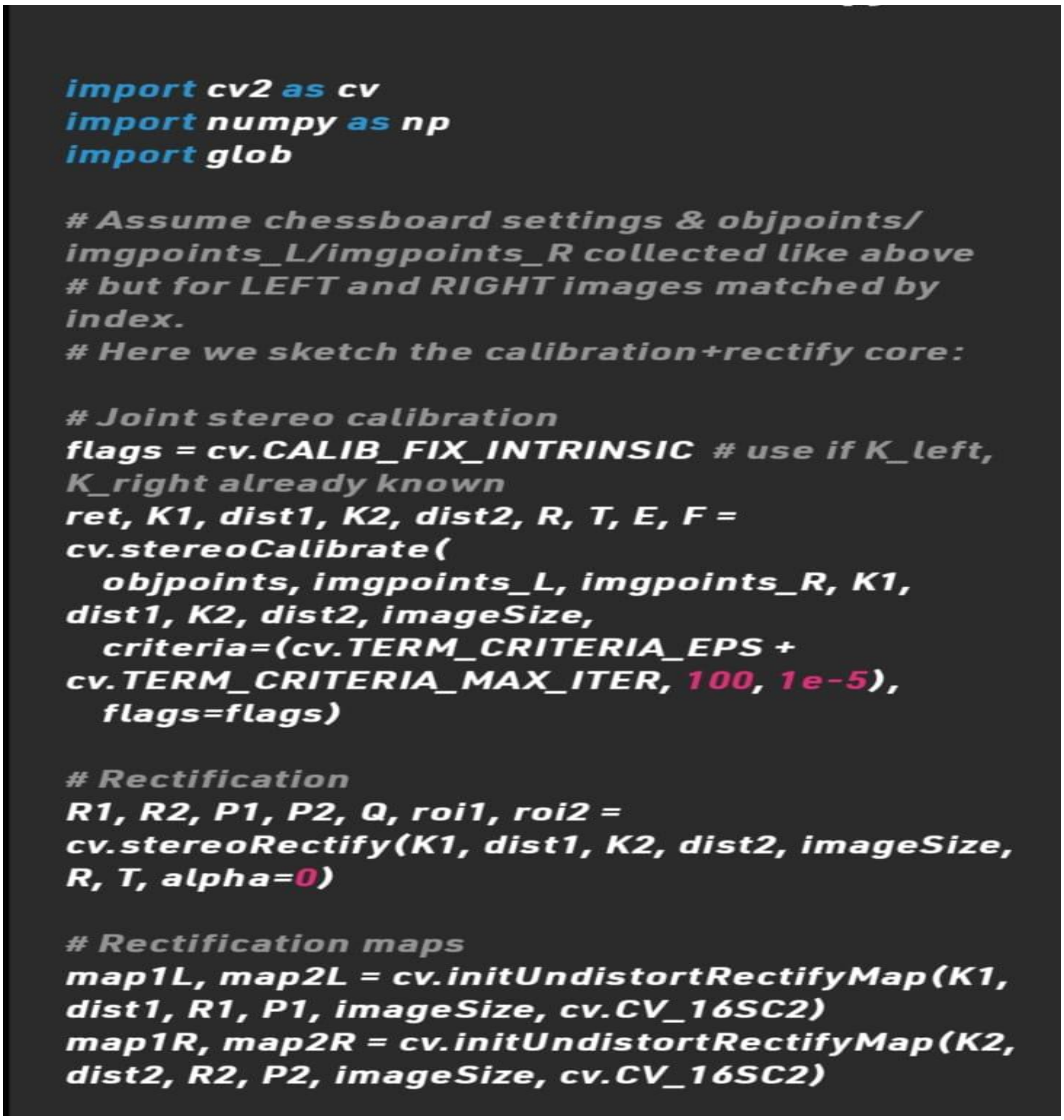
- **Pipeline**

1. Calibrate each camera (or do it jointly).

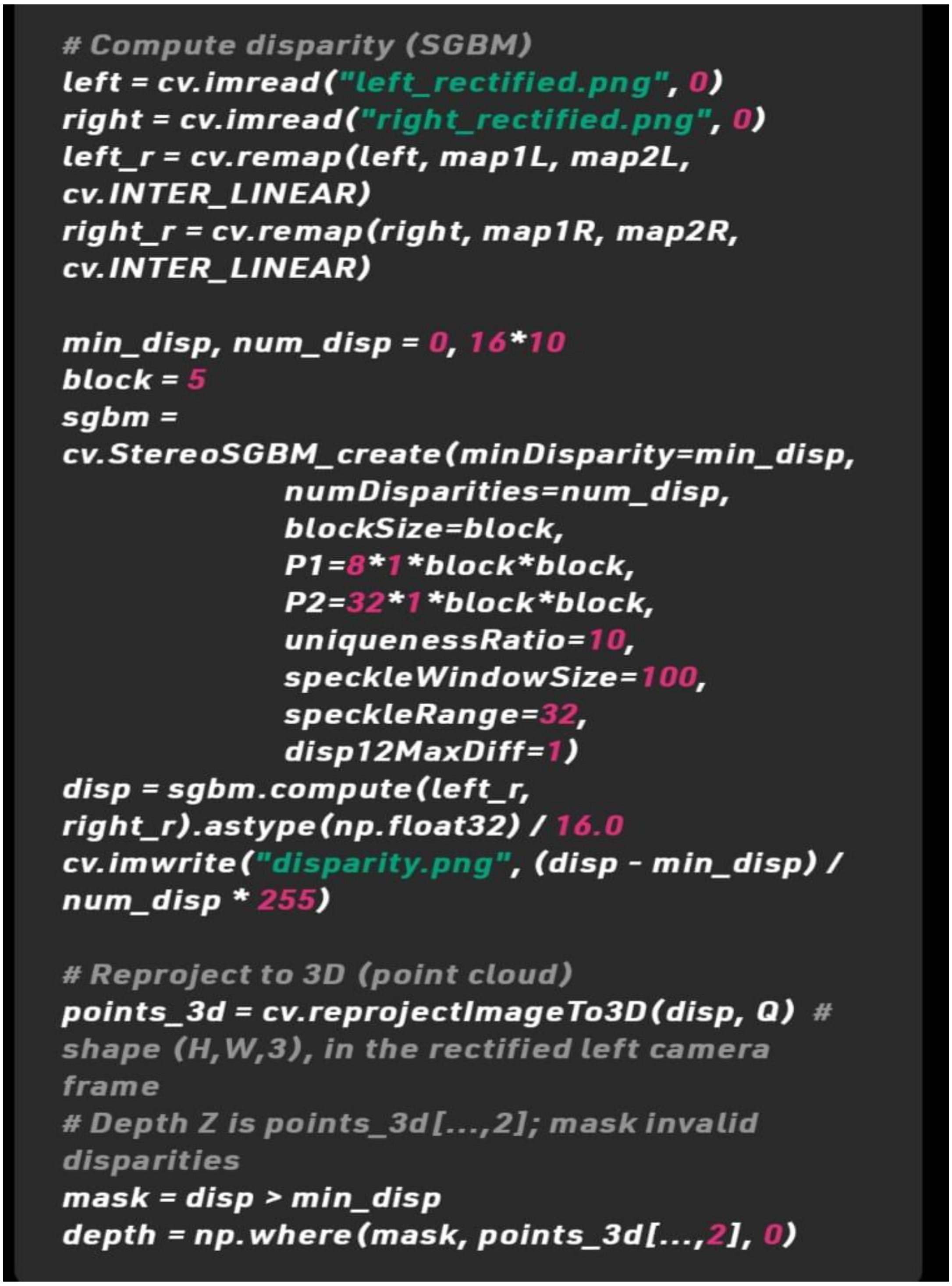
2. stereoCalibrate → relative rotation R and translation T.

3. stereoRectify → rectification transforms & Q (reprojection matrix).

4. Compute disparity (e.g., Stereo SGBM), then reproject to 3D with Q.



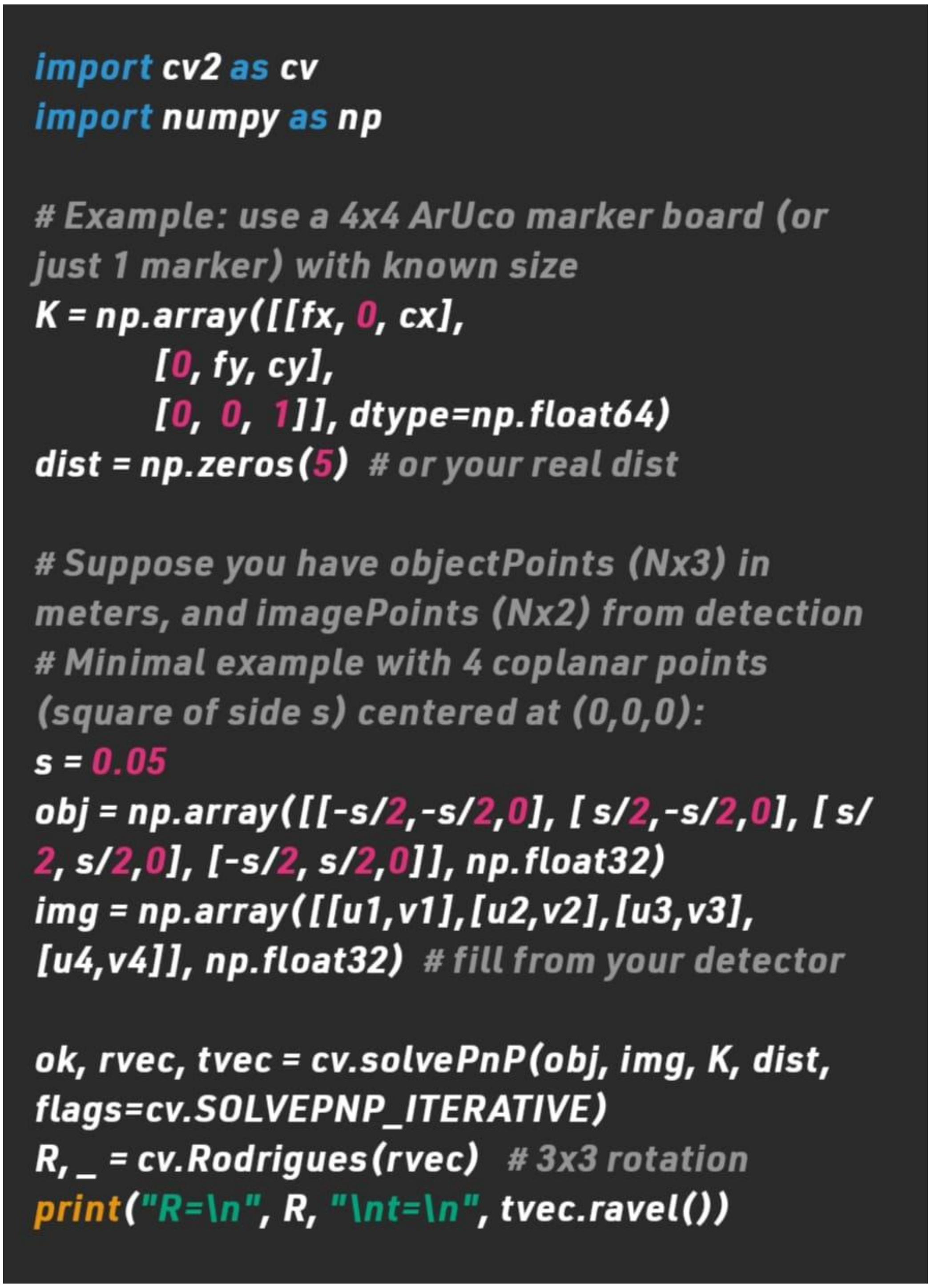
```
import cv2 as cv
import numpy as np
import glob

# Assume chessboard settings & objpoints/
imgpoints_L/imgpoints_R collected like above
# but for LEFT and RIGHT images matched by
index.
# Here we sketch the calibration+rectify core:

# Joint stereo calibration
flags = cv.CALIB_FIX_INTRINSIC  # use if K_left,
K_right already known
ret, K1, dist1, K2, dist2, R, T, E, F =
cv.stereoCalibrate(
   objpoints, imgpoints_L, imgpoints_R, K1,
dist1, K2, dist2, imageSize,
   criteria=(cv.TERM_CRITERIA_EPS +
cv.TERM_CRITERIA_MAX_ITER, 100, 1e-5),
   flags=flags)

# Rectification
R1, R2, P1, P2, Q, roi1, roi2 =
cv.stereoRectify(K1, dist1, K2, dist2, imageSize,
R, T, alpha=0)

# Rectification maps
map1L, map2L = cv.initUndistortRectifyMap(K1,
dist1, R1, P1, imageSize, cv.CV_16SC2)
map1R, map2R = cv.initUndistortRectifyMap(K2,
dist2, R2, P2, imageSize, cv.CV_16SC2)
```



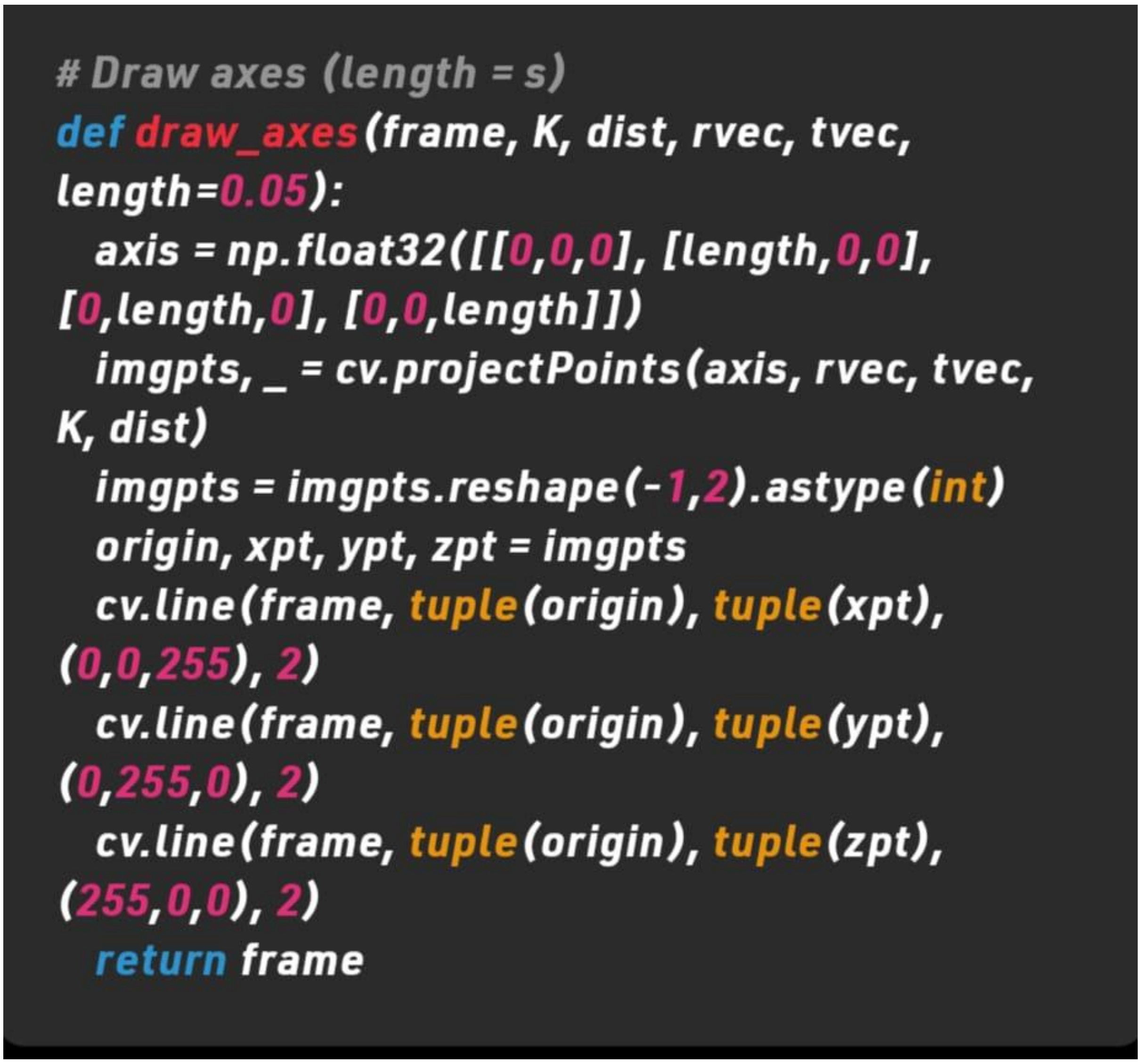
```
# Compute disparity (SGBM)
left = cv.imread("left_rectified.png", 0)
right = cv.imread("right_rectified.png", 0)
left_r = cv.remap(left, map1L, map2L,
cv.INTER_LINEAR)
right_r = cv.remap(right, map1R, map2R,
cv.INTER_LINEAR)

min_disp, num_disp = 0, 16*10
block = 5
sgbm =
cv.StereoSGBM_create(minDisparity=min_disp,
                     numDisparities=num_disp,
                     blockSize=block,
                     P1=8*1*block*block,
                     P2=32*1*block*block,
                     uniquenessRatio=10,
                     speckleWindowSize=100,
                     speckleRange=32,
                     disp12MaxDiff=1)
disp = sgbm.compute(left_r,
right_r).astype(np.float32) / 16.0
cv.imwrite("disparity.png", (disp - min_disp) /
num_disp * 255)

# Reproject to 3D (point cloud)
points_3d = cv.reprojectImageTo3D(disp, Q)  #
shape (H,W,3), in the rectified left camera
frame
# Depth Z is points_3d[...,2]; mask invalid
disparities
mask = disp > min_disp
depth = np.where(mask, points_3d[...,2], 0)
```

Depth scale: Z is in the same units as your baseline/ square_size (typically meters) if you calibrated with metric units.

4. Pose estimation (Solve PnP) from known 3D objects



```
import cv2 as cv
import numpy as np

# Example: use a 4x4 ArUco marker board (or
just 1 marker) with known size
K = np.array([[fx, 0, cx],
              [0, fy, cy],
              [0,  0,  1]], dtype=np.float64)
dist = np.zeros(5)  # or your real dist

# Suppose you have objectPoints (Nx3) in
meters, and imagePoints (Nx2) from detection
# Minimal example with 4 coplanar points
(square of side s) centered at (0,0,0):
s = 0.05
obj = np.array([[-s/2,-s/2,0], [ s/2,-s/2,0], [ s/
2, s/2,0], [-s/2, s/2,0]], np.float32)
img = np.array([[u1,v1],[u2,v2],[u3,v3],
[u4,v4]], np.float32)  # fill from your detector

ok, rvec, tvec = cv.solvePnP(obj, img, K, dist,
flags=cv.SOLVEPNP_ITERATIVE)
R, _ = cv.Rodrigues(rvec)   # 3x3 rotation
print("R=\n", R, "\nt=\n", tvec.ravel())
```



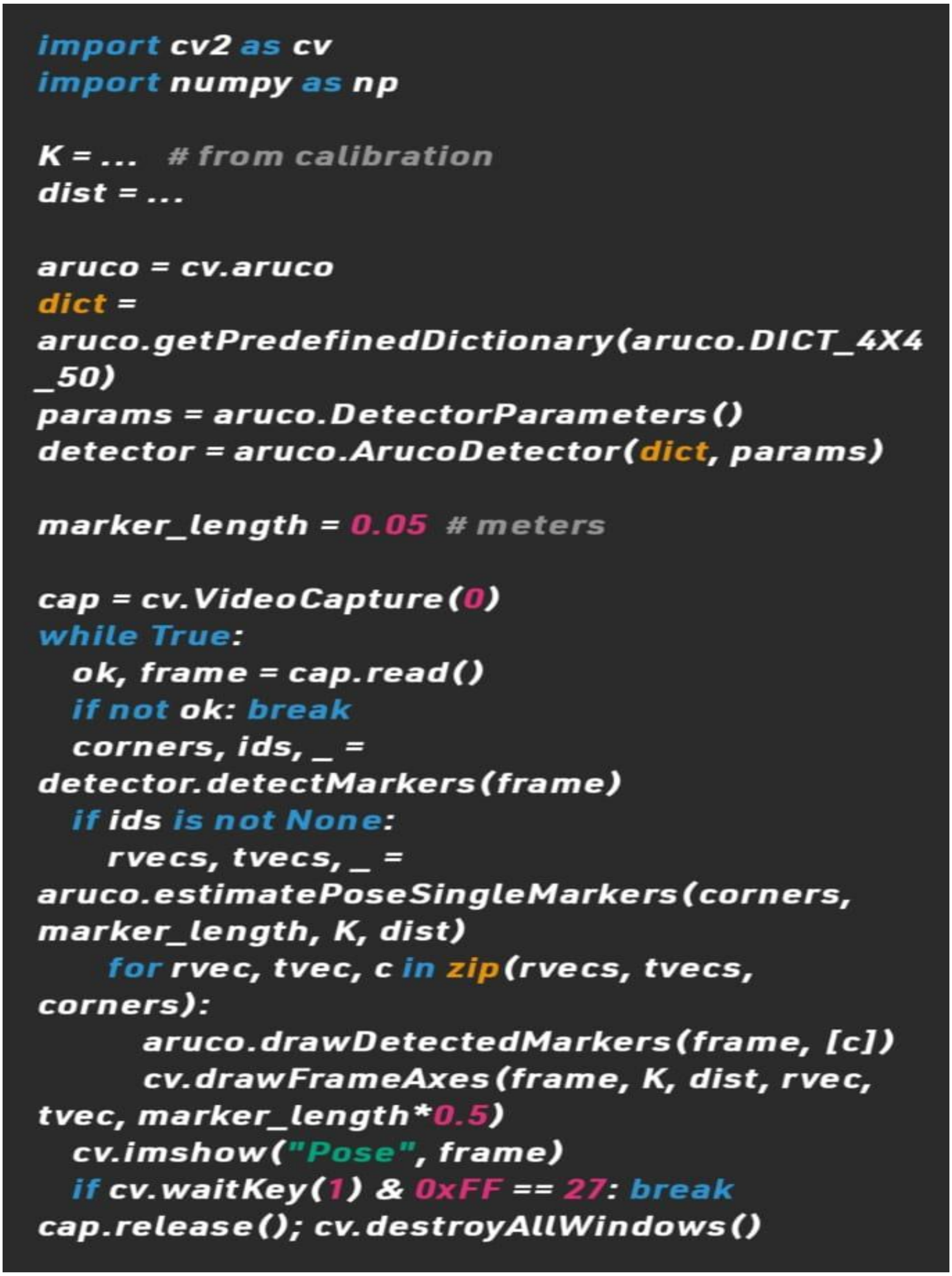
```
# Draw axes (length = s)
def draw_axes(frame, K, dist, rvec, tvec,
length=0.05):
  axis = np.float32([[0,0,0], [length,0,0],
[0,length,0], [0,0,length]])
  imgpts, _ = cv.projectPoints(axis, rvec, tvec,
K, dist)
  imgpts = imgpts.reshape(-1,2).astype(int)
  origin, xpt, ypt, zpt = imgpts
  cv.line(frame, tuple(origin), tuple(xpt),
(0,0,255), 2)
  cv.line(frame, tuple(origin), tuple(ypt),
(0,255,0), 2)
  cv.line(frame, tuple(origin), tuple(zpt),
(255,0,0), 2)
  return frame
```

Pose with ArUco (Robust & convenient)

```
import cv2 as cv
import numpy as np

K = ...   # from calibration
dist = ...

aruco = cv.aruco
dict =
aruco.getPredefinedDictionary(aruco.DICT_4X4
_50)
params = aruco.DetectorParameters()
detector = aruco.ArucoDetector(dict, params)

marker_length = 0.05  # meters

cap = cv.VideoCapture(0)
while True:
  ok, frame = cap.read()
  if not ok: break
  corners, ids, _ =
detector.detectMarkers(frame)
  if ids is not None:
    rvecs, tvecs, _ =
aruco.estimatePoseSingleMarkers(corners,
marker_length, K, dist)
    for rvec, tvec, c in zip(rvecs, tvecs,
corners):
      aruco.drawDetectedMarkers(frame, [c])
      cv.drawFrameAxes(frame, K, dist, rvec,
tvec, marker_length*0.5)
  cv.imshow("Pose", frame)
  if cv.waitKey(1) & 0xFF == 27: break
cap.release(); cv.destroyAllWindows()
```

Tips & troubleshooting

- Corner quality: Use cornerSubPix and good lighting; blur hurts calibration.
- Diverse views: Tilt/rotate the board; fill the image; avoid all shots from the same plane.
- Metric scale: Depth/poses are metric only if your square/marker sizes are in meters.
- Check reprojection error: < 0.5–1.0 px is generally good for typical webcams.
- Save/load params: Use cv.FileStorage (YAML/JSON) to reuse K, dist, R, T, Q.



```python
import cv2 as cv
import numpy as np

K = ...   # from calibration
dist = ...

aruco = cv.aruco
dict =
aruco.getPredefinedDictionary(aruco.DICT_4X4
_50)
params = aruco.DetectorParameters()
detector = aruco.ArucoDetector(dict, params)

marker_length = 0.05  # meters

cap = cv.VideoCapture(0)
while True:
   ok, frame = cap.read()
   if not ok: break
   corners, ids, _ =
detector.detectMarkers(frame)
   if ids is not None:
      rvecs, tvecs, _ =
aruco.estimatePoseSingleMarkers(corners,
marker_length, K, dist)
      for rvec, tvec, c in zip(rvecs, tvecs,
corners):
         aruco.drawDetectedMarkers(frame, [c])
         cv.drawFrameAxes(frame, K, dist, rvec,
tvec, marker_length*0.5)
   cv.imshow("Pose", frame)
   if cv.waitKey(1) & 0xFF == 27: break
cap.release(); cv.destroyAllWindows()
```

Step – by – step process of using the Object detection in Open CV by (objdetect) module, which focuses on Cascade Classifiers, Face Detection, and other detectors.

objdetect Module in OpenCV

The objdetect module provides algorithms for **object detection** — finding instances of objects (faces, eyes, cars, smile, etc.) in images or videos.

The most well-known feature here is the **Haar Cascade Classifier**, a machine-learning based approach where:

- Features (edges, lines, textures) are extracted from images.
- A cascade of classifiers is trained to detect objects step by step, rejecting non-object regions early for speed.

- Features in `objdetect`

Functionality	Purpose
CascadeClassifier	Loads a pre-trained object detection model (XML).
detectMultiScale()	Detects objects at multiple scales in an image.
Pretrained XML files	Available for faces, eyes, cars, license plates, etc.

Cascade Classifier Basics

How it Works

1. **Training phase** – Haar or LBP features are trained on thousands of positive and negative samples.
2. **Detection phase** – The trained XML model is applied to the image in a sliding-window fashion.

3. **Multi-scale search** – Detect objects of different sizes in the same image.

OpenCV comes with pre

Examples include:

- haarcascade_frontalface_default.xml – front-facing faces
- haarcascade_eye.xml – eyes
- haarcascade_fullbody.xml – whole body
- haarcascade_car.xml – cars

---

4. Example: Face Detection using Haar Cascade

python

Copy code

import cv2

```
# Load pre-trained Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Read image
img = cv2.imread('people.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

```
# Draw rectangles around faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Show the result
cv2.imshow('Face Detection', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Parameters explained:

- scaleFactor=1.1 → Image is reduced by 10% at each scale for multi-scale detection.
- minNeighbors=5 → Higher values reduce false positives.

- minSize=(30,30) → Smallest size of detected object.

---

5. Detecting Multiple Objects (Face + Eyes)

python

Copy code

```
import cv2


# Load cascades
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
```

```
# Read image
img = cv2.imread('face.jpg')
gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)

# Detect faces
faces =
face_cascade.detectMultiScale(gray,
1.1, 5)

for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x+w, y+h),
(255, 0, 0), 2)
   roi_gray = gray[y:y+h, x:x+w]
   roi_color = img[y:y+h, x:x+w]

   # Detect eyes within the face region
```

```
    eyes =
eye_cascade.detectMultiScale(roi_gray)
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey),
(ex+ew, ey+eh), (0, 255, 0), 2)

cv2.imshow('Face and Eye Detection',
img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

6. Real-Time Face Detection from Webcam

python

Copy code

```
import cv2
```

```
# Load the face cascade
face_cascade =
cv2.CascadeClassifier(cv2.data.haarca
scades +
'haarcascade_frontalface_default.xml')

# Open webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)

    # Detect faces
    faces =
face_cascade.detectMultiScale(gray,
1.1, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w,
y+h), (0, 255, 0), 2)

    cv2.imshow('Webcam Face
Detection', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

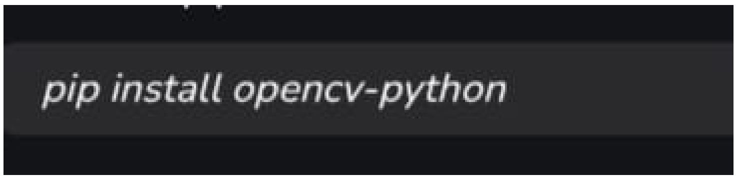
---

7. Vehicle – detection using Open CV in Python

Vehicle detection is an important application of computer vision that helps in monitoring traffic, automating parking systems and surveillance systems. In this article, we'll implement a simple vehicle detection system using Python and OpenCV using a pre-trained Haar Cascade classifier and we will get a video in which vehicles will be detected and it will be represented by a rectangular frame around it.

Step 1: Installing OpenCV Library

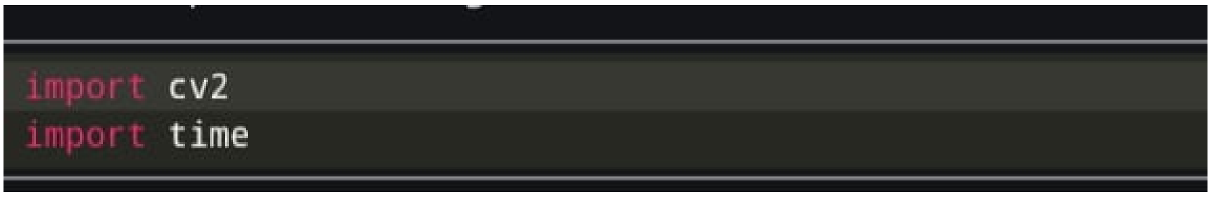
We need to install the OpenCV library or its implementation. To install it in our system we can use the below pip command:



```
pip install opencv-python
```

Step 2: Importing Required Libraries

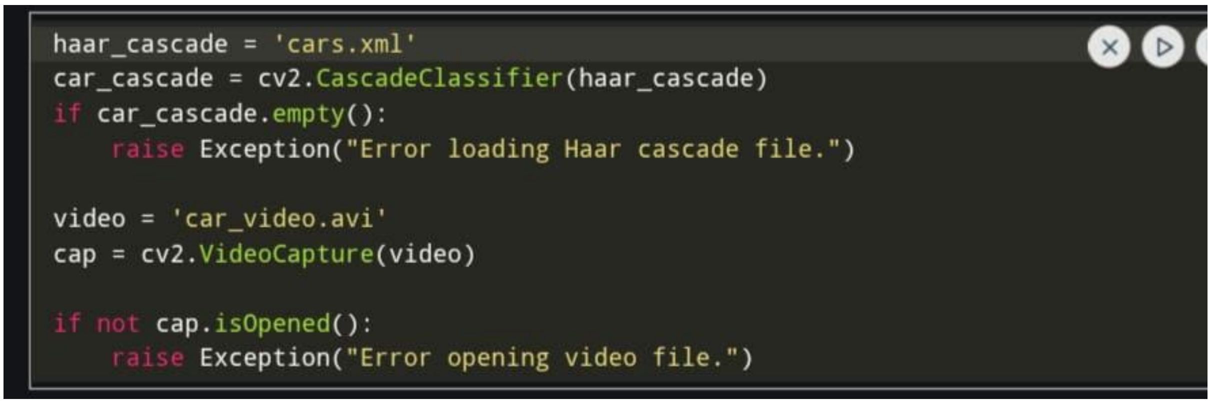
Using an OpenCV for vehicle detection and handling video streams. Also we need time module to add delays after detecting a vehicle.



```python
import cv2
import time
```

Step 3: Loading Haar Cascade Classifier and Video

So here ,we need to downlaod the Haar Cascade files (download it from here) and load the Haar Cascade classifier that’s trained to detect vehicles. Also we have to load the video file (download it from here) from which vehicles will be detected. Now to capture a video we need to create a VideoCapture() object.

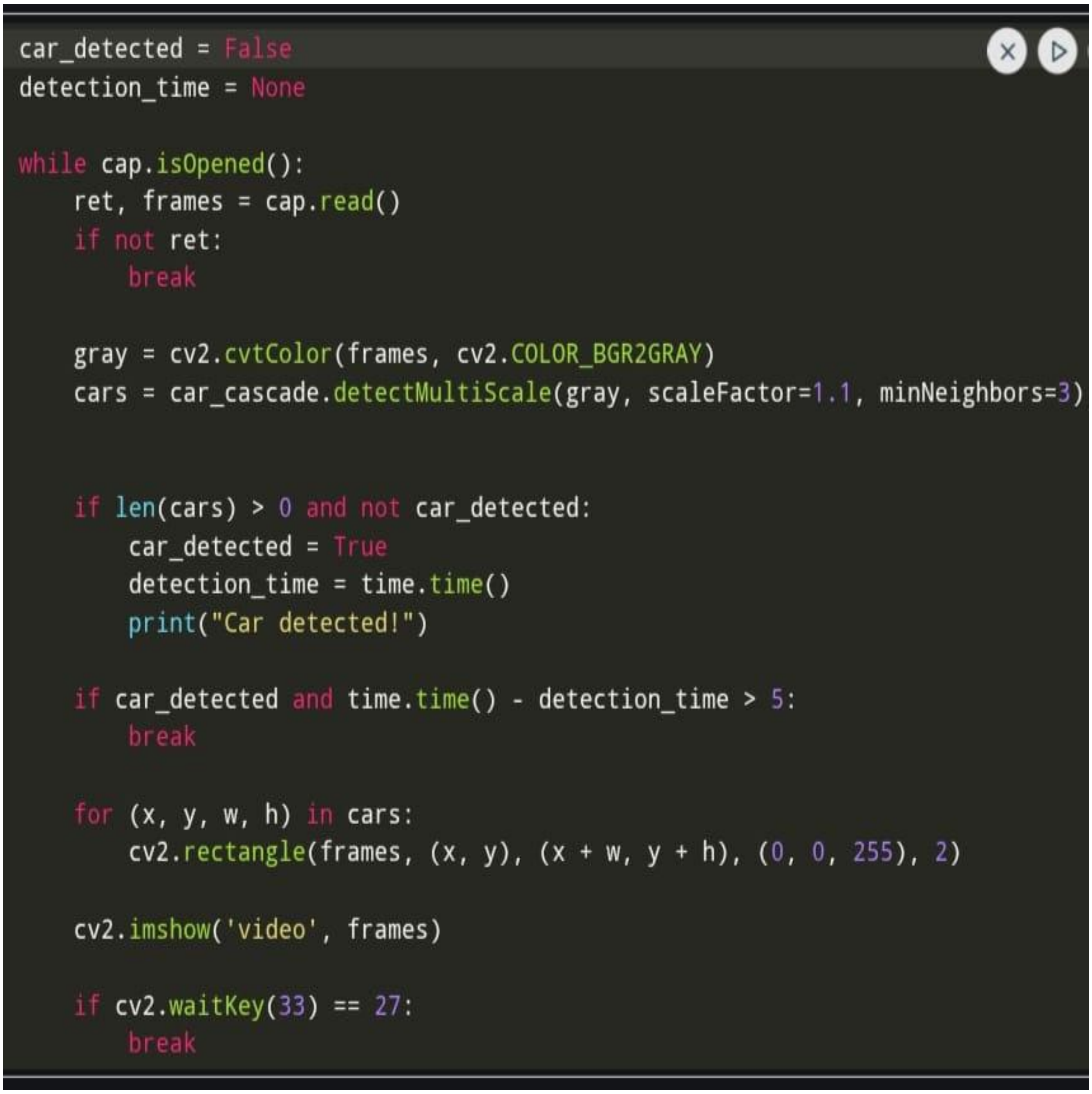


```python
haar_cascade = 'cars.xml'
car_cascade = cv2.CascadeClassifier(haar_cascade)
if car_cascade.empty():
    raise Exception("Error loading Haar cascade file.")

video = 'car_video.avi'
cap = cv2.VideoCapture(video)

if not cap.isOpened():
    raise Exception("Error opening video file.")
```

Step 4: Detecting Vehicles in Video Frames

We read each frame of the video, convert it to grayscale and use the Haar Cascade classifier to detect vehicles. Once a vehicle is detected we draw rectangles around it.

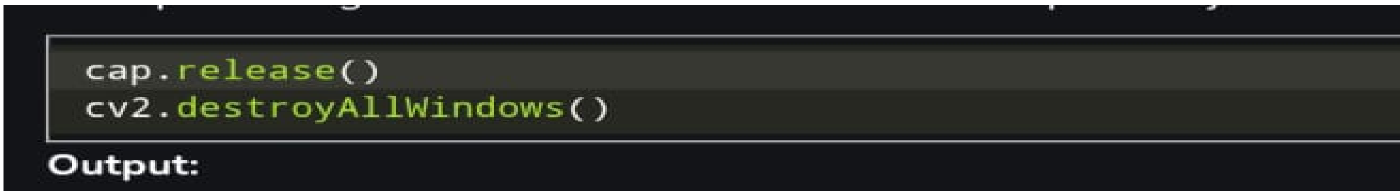
cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3): Detects vehicles in the grayscale image using the Haar Cascade classifier.



```python
car_detected = False
detection_time = None

while cap.isOpened():
    ret, frames = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3)


    if len(cars) > 0 and not car_detected:
        car_detected = True
        detection_time = time.time()
        print("Car detected!")

    if car_detected and time.time() - detection_time > 5:
        break

    for (x, y, w, h) in cars:
        cv2.rectangle(frames, (x, y), (x + w, y + h), (0, 0, 255), 2)

    cv2.imshow('video', frames)

    if cv2.waitKey(33) == 27:
        break
```

Step 5: Release Video and Close Windows

After processing the frames now, we can release the video capture object and close any Open CV windows.



```
cap.release()
cv2.destroyAllWindows()
```

Output:

Output:

8. Other Object Detection with Cascades

You can replace the XML file with:

python

Copy code

```python
car_cascade = cv2.CascadeClassifier('haarcascade_car.xml')
fullbody_cascade = cv2.CascadeClassifier('haarcascade_fullbody.xml')
```

The rest of the code stays the same — just change which classifier you call detectMultiScale on.

---

9. Limitations & Modern Alternatives

- Haar cascades are **fast but less accurate** compared to modern deep learning models.
- They struggle with:

    - Large variation in pose, lighting, occlusion.
    - Very small objects.
- Modern alternatives:
    - **DNN module in OpenCV** with pretrained models like SSD, YOLO, Faster-R-CNN.
    - cv2.dnn.readNetFromDarknet() or cv2.dnn.readNetFromCaffe().

Overview: 2D Features Framework

- The cv2.features2d module in OpenCV is used for:
- Feature Detection → Finding key points (corners, blobs, edges).

- Feature Description → Representing the detected points as numerical vectors.
- Feature Matching → Comparing descriptors between images to find similarities.
- These features are scale- and rotation-invariant (depending on the algorithm), making them useful for:
    - Image matching
    - Object recognition
    - Image stitching (panoramas)
    - Tracking

2. Core Steps

Step 1: Feature Detection

- Algorithms find keypoints — points in the image that have strong contrast in multiple directions (like corners, blobs).

Common detectors:

- SIFT (Scale-Invariant Feature Transform) → Accurate, robust (patented before OpenCV 4.4, now free).
- ORB (Oriented FAST and Rotated BRIEF) → Faster, open-source, less computational cost.
- SURF (Speeded-Up Robust Features) → Similar to SIFT, faster, patented.

Step 2: Feature Description

- Once keypoints are detected, each one gets a descriptor: a vector summarizing the local neighbourhood.

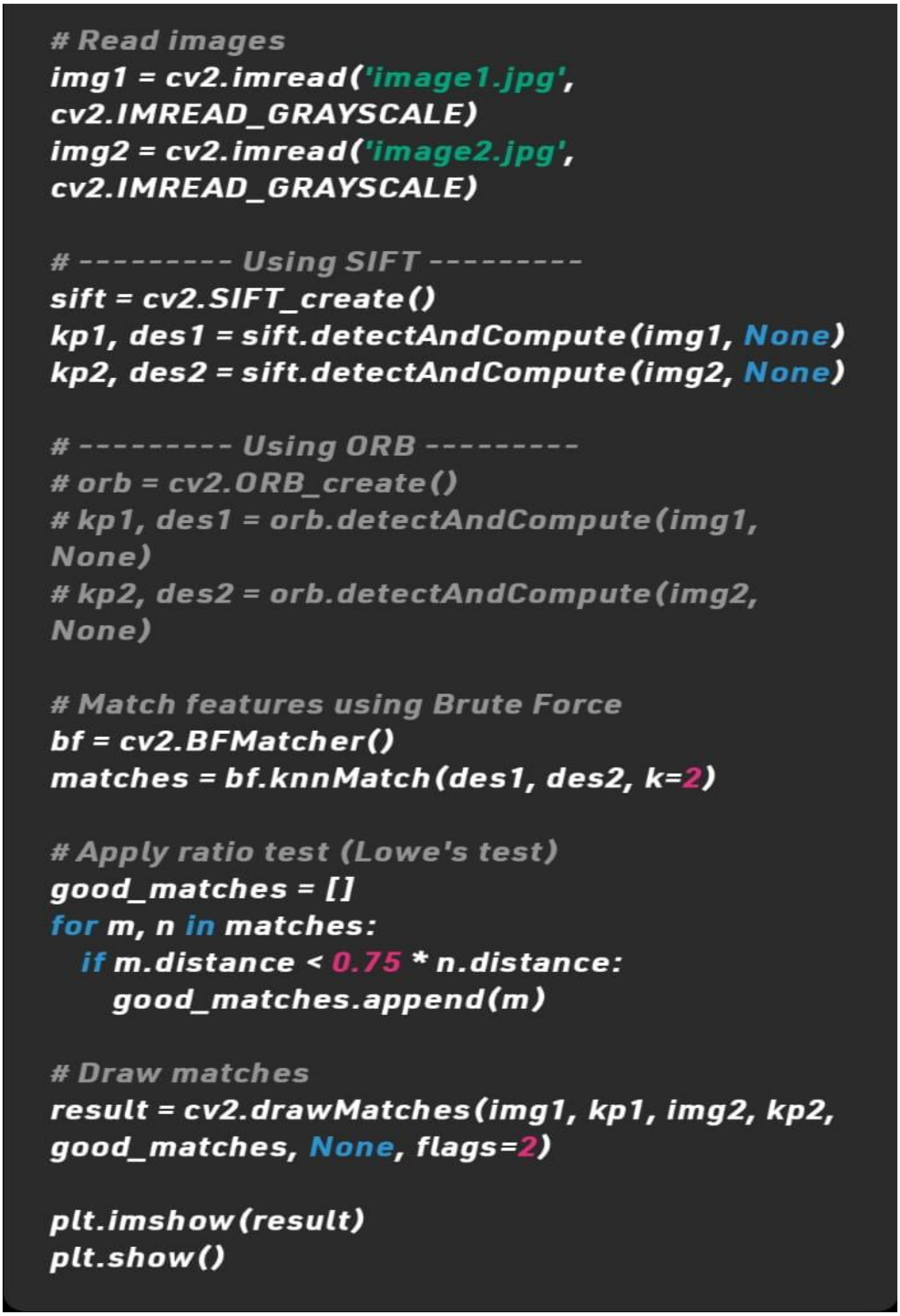
    - Example:

- SIFT → 128-dimensional descriptor
- ORB → 32-dimensional binary descriptor

Step 3: Feature Matching

- We match descriptors between two images using:
- Brute-Force Matcher (cv2.BFMatcher) → Compares every descriptor with every other.
- FLANN (Fast Library for Approximate Nearest Neighbors) → Faster for large datasets.

Example: SIFT & ORB in Open CV



```
# Read images
img1 = cv2.imread('image1.jpg',
cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('image2.jpg',
cv2.IMREAD_GRAYSCALE)

# --------- Using SIFT ---------
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# --------- Using ORB ---------
# orb = cv2.ORB_create()
# kp1, des1 = orb.detectAndCompute(img1,
None)
# kp2, des2 = orb.detectAndCompute(img2,
None)

# Match features using Brute Force
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)

# Apply ratio test (Lowe's test)
good_matches = []
for m, n in matches:
  if m.distance < 0.75 * n.distance:
    good_matches.append(m)

# Draw matches
result = cv2.drawMatches(img1, kp1, img2, kp2,
good_matches, None, flags=2)

plt.imshow(result)
plt.show()
```

DNN module in OpenCV

The cv2.dnn module allows you to:

- Load pre-trained deep learning models from popular frameworks (TensorFlow, Caffe, Darknet, ONNX, Torch).
- Perform inference (forward pass) efficiently without needing the original deep learning framework installed.
- Run models on CPU or GPU (if OpenCV is built with CUDA support).

It’s inference-only, meaning you can’t train models here — just load and run them.

Steps to Load and Run a Model

General workflow:

1. Load model architecture & weights (readNetFrom...()).
    - cv2.dnn.readNetFromCaffe()
    - cv2.dnn.readNetFromTensorflow()
    - cv2.dnn.readNetFromDarknet()
    - cv2.dnn.readNetFromONNX()
2. Prepare the image (resize, normalize, blob creation).
    - cv2.dnn.blobFromImage()
3. Set the input to the network (net.setInput()).
4. Run forward pass (net.forward()).
5. Process output (interpret predictions).
    - Draw boxes, labels, or handle classification output.

Example - Object Detection with MobileNet-SSD (Caffe Model)

We'll detect objects in an image using a pre-trained Caffe MobileNetSSD model.

Model Files

Download:

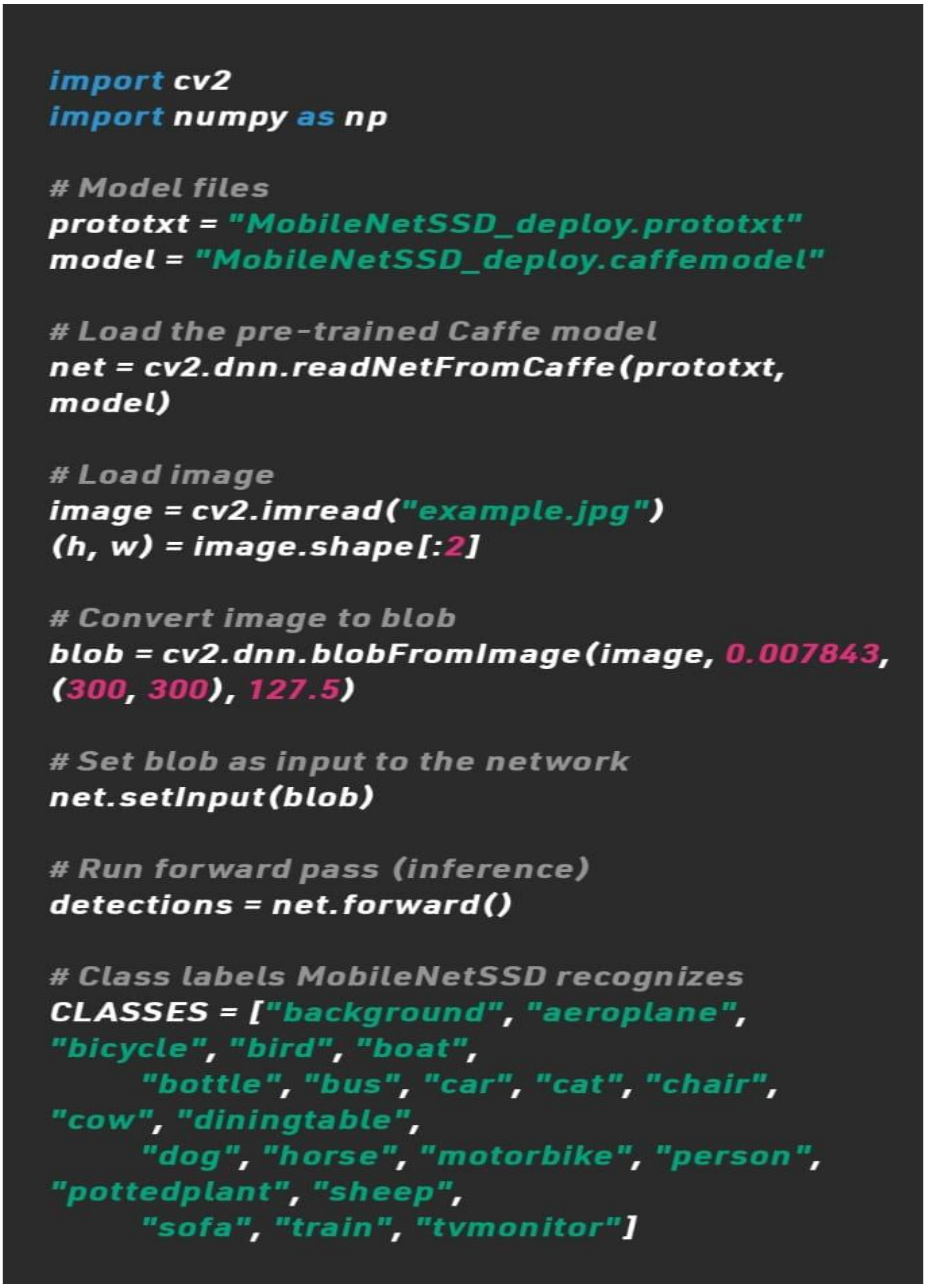
Architecture:

MobileNetSSD_deploy.prototxt

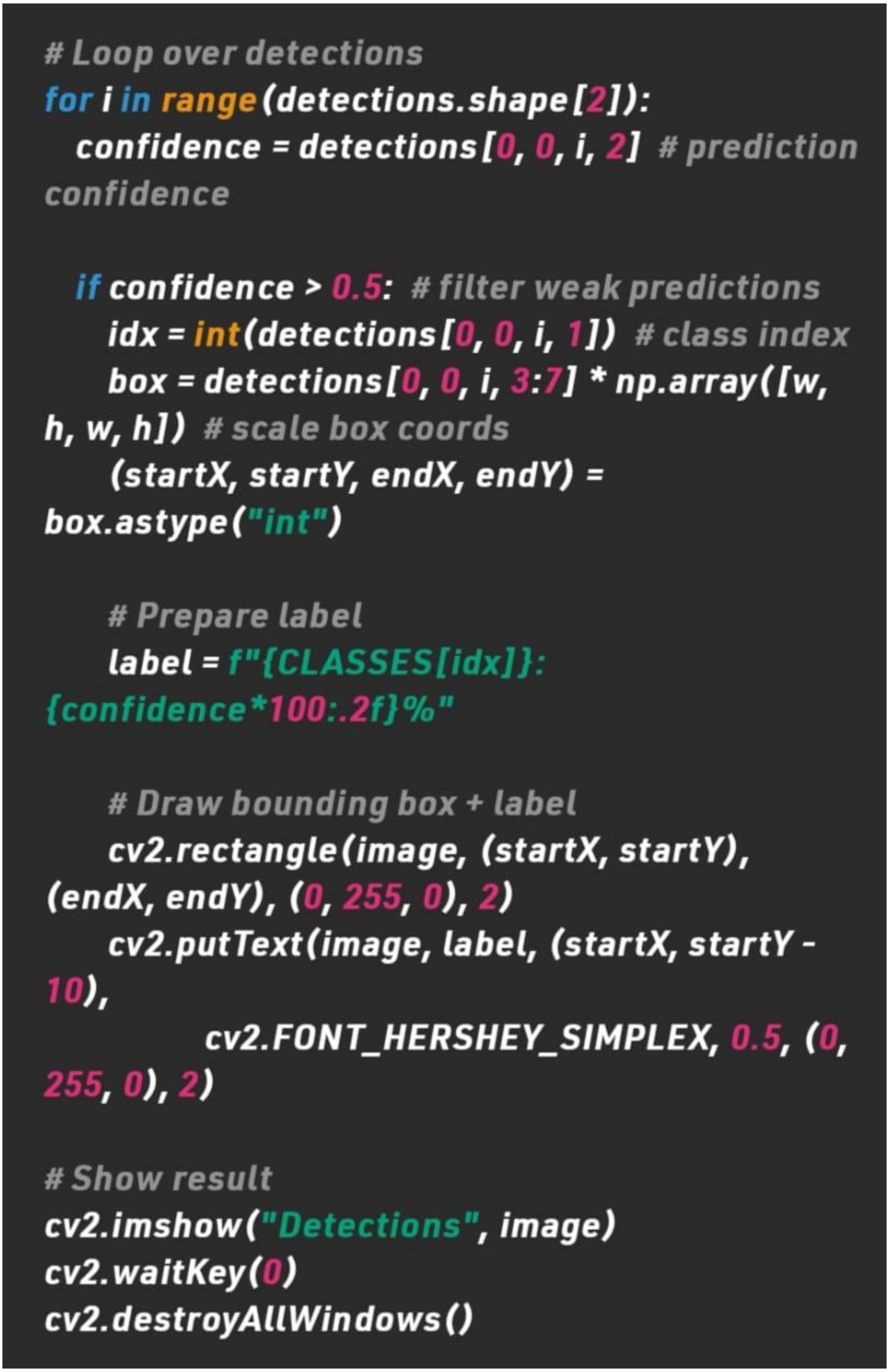
Weights:

MobileNetSSD_deploy.caffemodel

(MobileNetSSD is trained on 20 common object categories like person, car, dog, etc.)



```
import cv2
import numpy as np

# Model files
prototxt = "MobileNetSSD_deploy.prototxt"
model = "MobileNetSSD_deploy.caffemodel"

# Load the pre-trained Caffe model
net = cv2.dnn.readNetFromCaffe(prototxt,
model)

# Load image
image = cv2.imread("example.jpg")
(h, w) = image.shape[:2]

# Convert image to blob
blob = cv2.dnn.blobFromImage(image, 0.007843,
(300, 300), 127.5)

# Set blob as input to the network
net.setInput(blob)

# Run forward pass (inference)
detections = net.forward()

# Class labels MobileNetSSD recognizes
CLASSES = ["background", "aeroplane",
"bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair",
"cow", "diningtable",
        "dog", "horse", "motorbike", "person",
"pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]
```



```
# Loop over detections
for i in range(detections.shape[2]):
  confidence = detections[0, 0, i, 2]  # prediction
confidence

  if confidence > 0.5:  # filter weak predictions
    idx = int(detections[0, 0, i, 1])  # class index
    box = detections[0, 0, i, 3:7] * np.array([w,
h, w, h])  # scale box coords
    (startX, startY, endX, endY) =
box.astype("int")

    # Prepare label
    label = f"{CLASSES[idx]}:
{confidence*100:.2f}%"

    # Draw bounding box + label
    cv2.rectangle(image, (startX, startY),
(endX, endY), (0, 255, 0), 2)
    cv2.putText(image, label, (startX, startY -
10),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)

# Show result
cv2.imshow("Detections", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Code Works

cv2.dnn.readNetFromCaffe() → loads Caffe model.

cv2.dnn.blobFromImage() → resizes, normalizes, and prepares image.

- net.setInput() → feeds blob into the network.
- net.forward() → runs inference and returns results.
- Output contains [batch_id, class_id, confidence, x1, y1, x2, y2].

Graph API (gapi module)

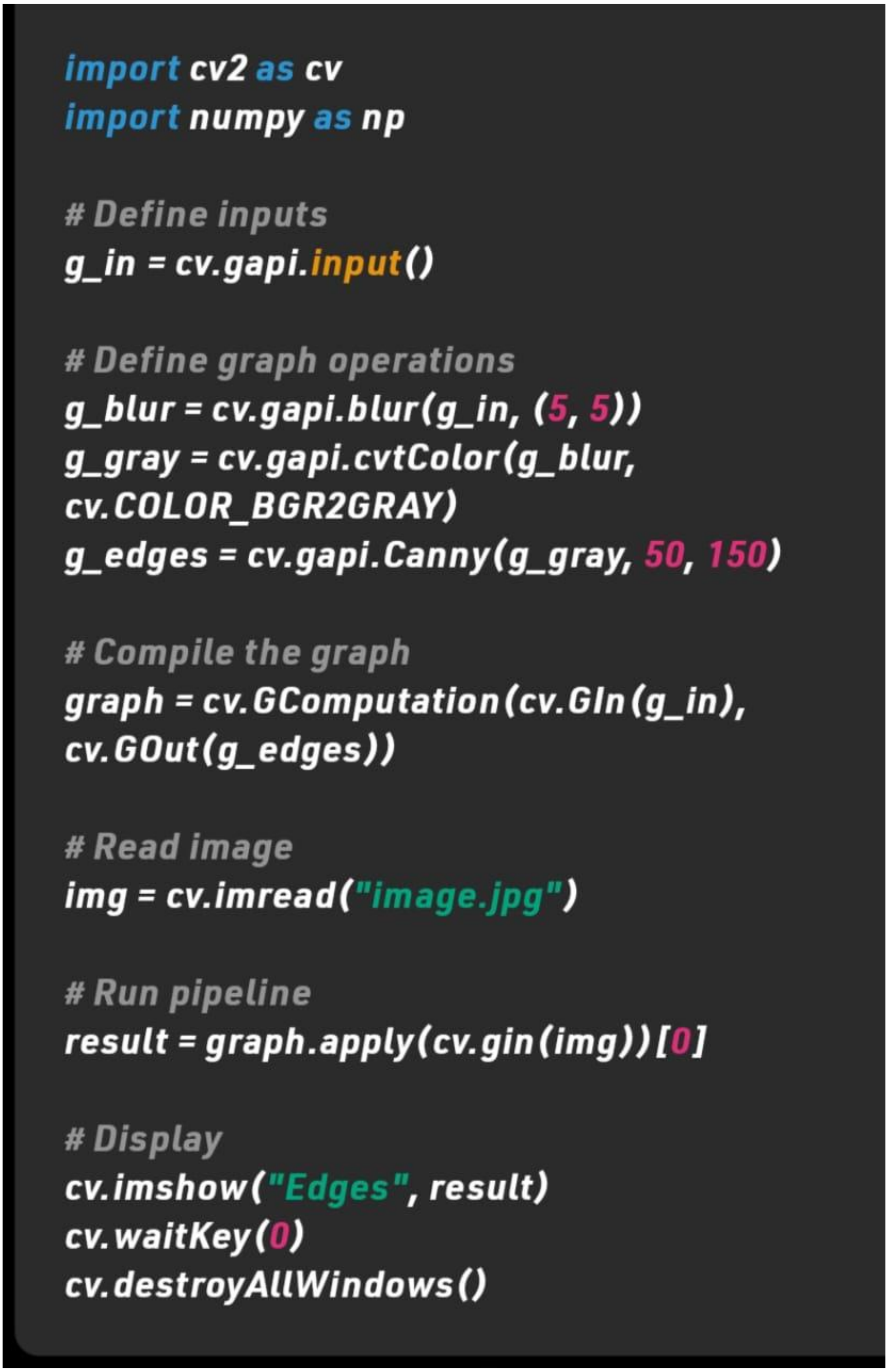
OpenCV’s G-API (Graph API) is designed for pipeline-based image processing where operations are predefined in a graph and executed efficiently.

Instead of calling image-processing functions one-by-one, you define a computation graph (nodes = operations, edges = data flow).

Benefits:

- Optimized execution (can run on CPU, GPU, or special backends).
- Better performance in complex pipelines.
- Reusability of computation graphs.

Example: Simple G-API Pipeline



```python
import cv2 as cv
import numpy as np

# Define inputs
g_in = cv.gapi.input()

# Define graph operations
g_blur = cv.gapi.blur(g_in, (5, 5))
g_gray = cv.gapi.cvtColor(g_blur,
cv.COLOR_BGR2GRAY)
g_edges = cv.gapi.Canny(g_gray, 50, 150)

# Compile the graph
graph = cv.GComputation(cv.GIn(g_in),
cv.GOut(g_edges))

# Read image
img = cv.imread("image.jpg")

# Run pipeline
result = graph.apply(cv.gin(img))[0]

# Display
cv.imshow("Edges", result)
cv.waitKey(0)
cv.destroyAllWindows()
```

Steps of Image processing

1. Pixels

The smallest addressable element in an image.

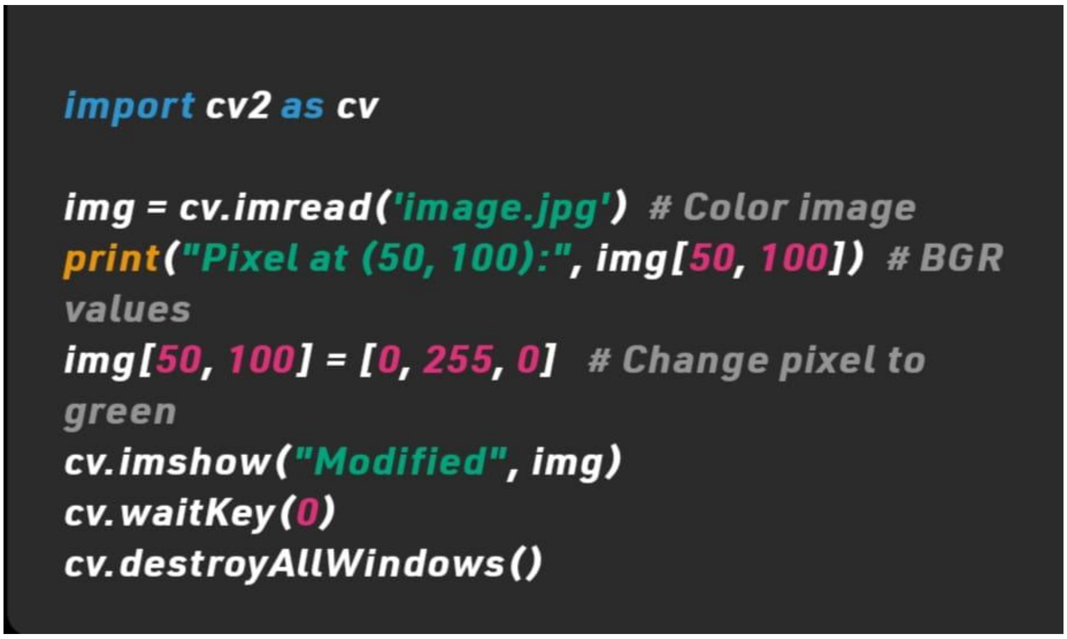
Each pixel stores intensity (grayscale) or color values (e.g., RGB or BGR).

In OpenCV:

Grayscale → 1 value per pixel (0–255).

Color → 3 values per pixel (e.g., (B, G, R) in OpenCV).

Example : Access and Modify Pixels



```
import cv2 as cv

img = cv.imread('image.jpg') # Color image
print("Pixel at (50, 100):", img[50, 100]) # BGR
values
img[50, 100] = [0, 255, 0]  # Change pixel to
green
cv.imshow("Modified", img)
cv.waitKey(0)
cv.destroyAllWindows()
```

2. Color Spaces

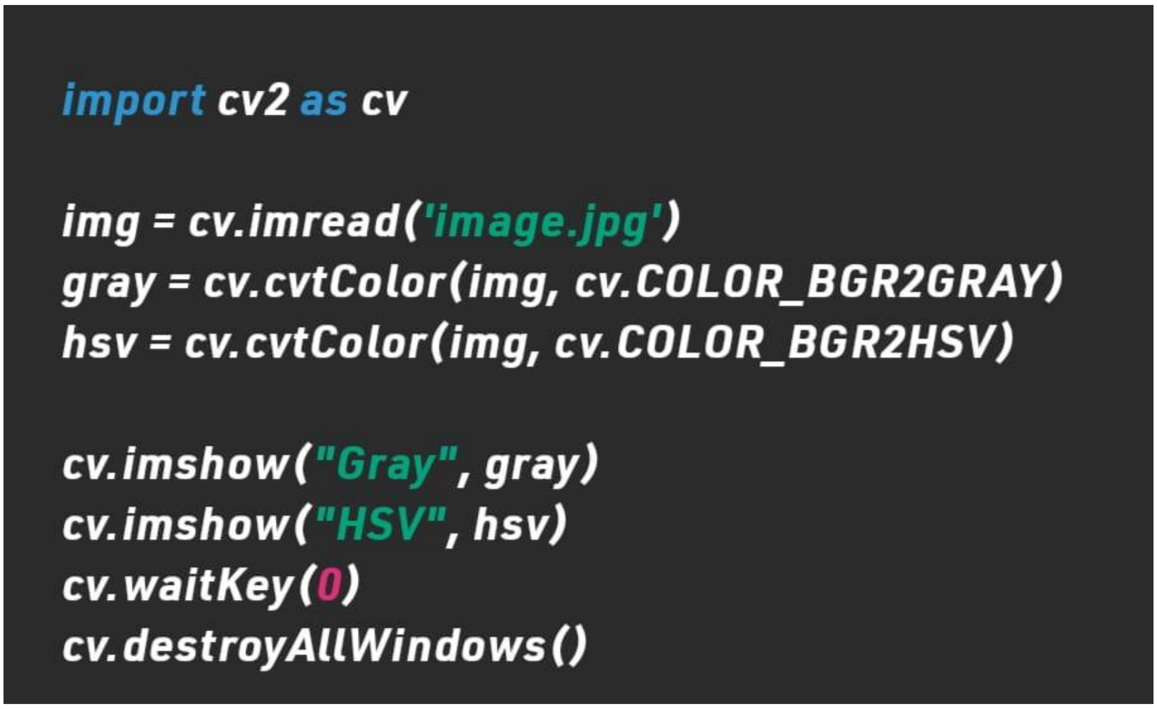
BGR / RGB: Basic color channels.

HSV: Hue, Saturation, Value (better for color-based segmentation).

Grayscale: Only intensity.

YCrCb, LAB: Other formats used in compression and lighting-invariant tasks.

Example: Convert Between Color Spaces



```
import cv2 as cv

img = cv.imread('image.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

cv.imshow("Gray", gray)
cv.imshow("HSV", hsv)
cv.waitKey(0)
cv.destroyAllWindows()
```

3. Convolution

Convolution is applying a kernel (matrix) over an image to transform it.

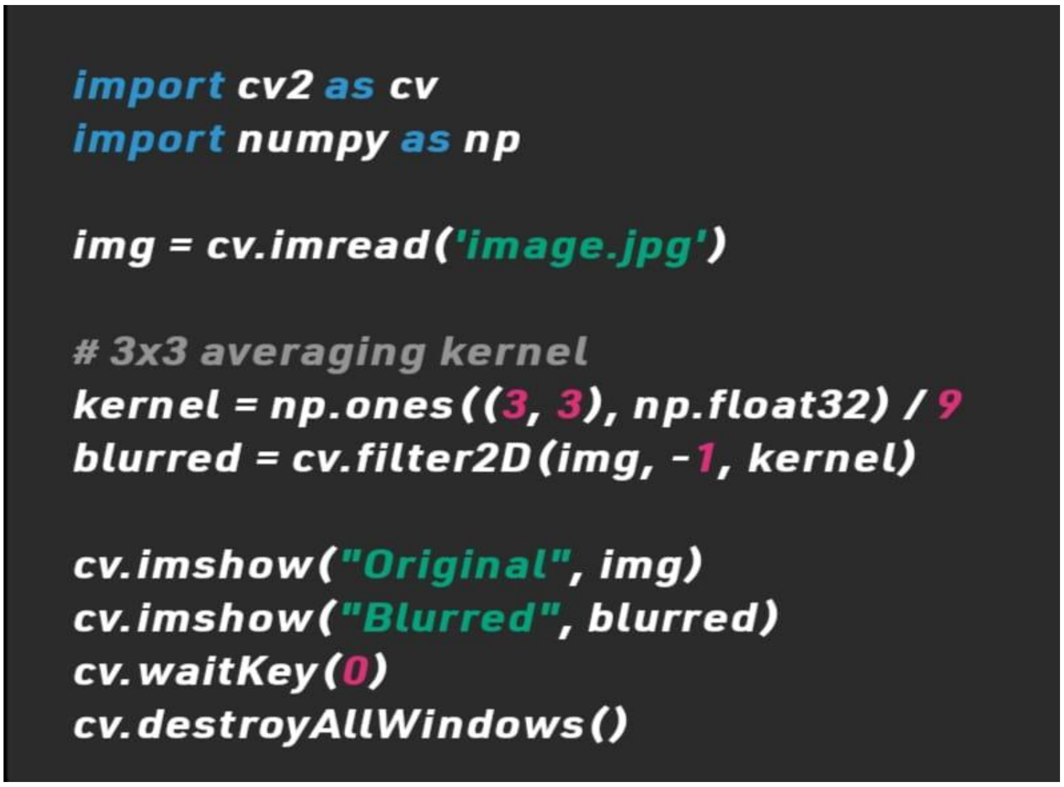
Common uses:

Blurring → averaging pixel values.

Sharpening → highlighting edges.

Formula: New pixel value = sum of (kernel × neighborhood pixels).

Example: Custom Kernel Filtering

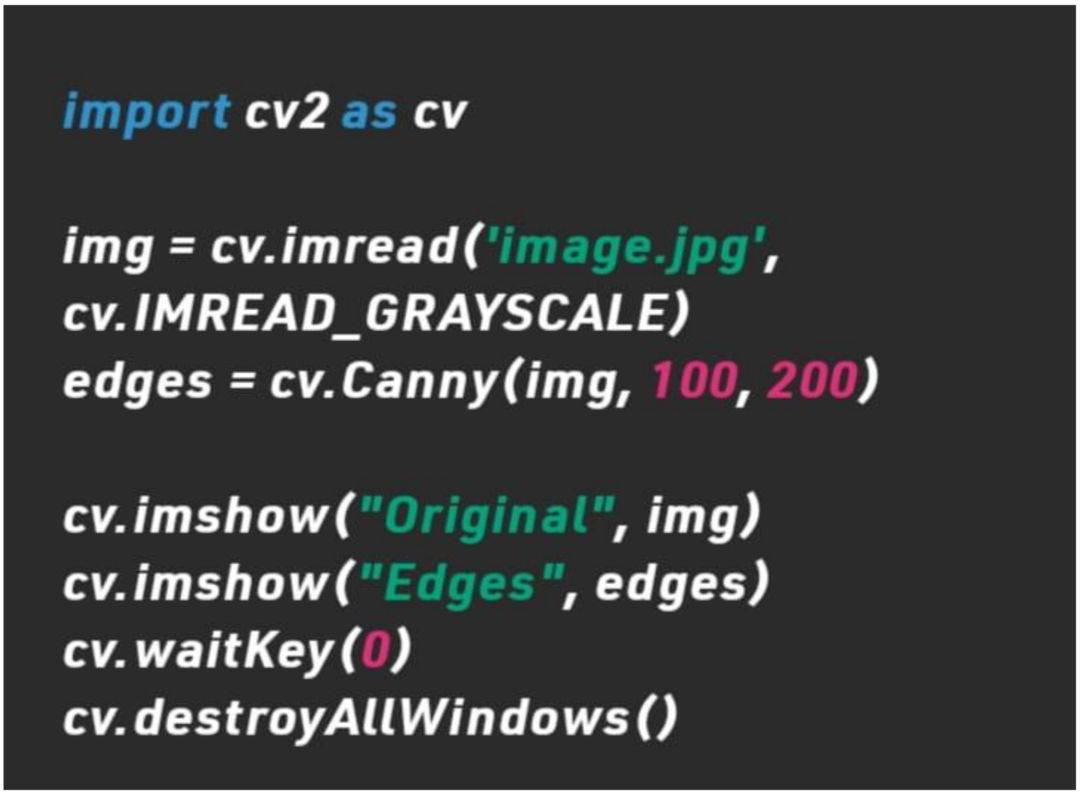


```python
import cv2 as cv
import numpy as np

img = cv.imread('image.jpg')

# 3x3 averaging kernel
kernel = np.ones((3, 3), np.float32) / 9
blurred = cv.filter2D(img, -1, kernel)

cv.imshow("Original", img)
cv.imshow("Blurred", blurred)
cv.waitKey(0)
cv.destroyAllWindows()
```

4. Edge Detection

- Goal: Detect sharp intensity changes (object boundaries).
- Common methods:
- Sobel (gradients in X/Y direction).
- Laplacian (second derivative).
- Canny (multi-stage robust detector).

Example: Canny Edge Detection

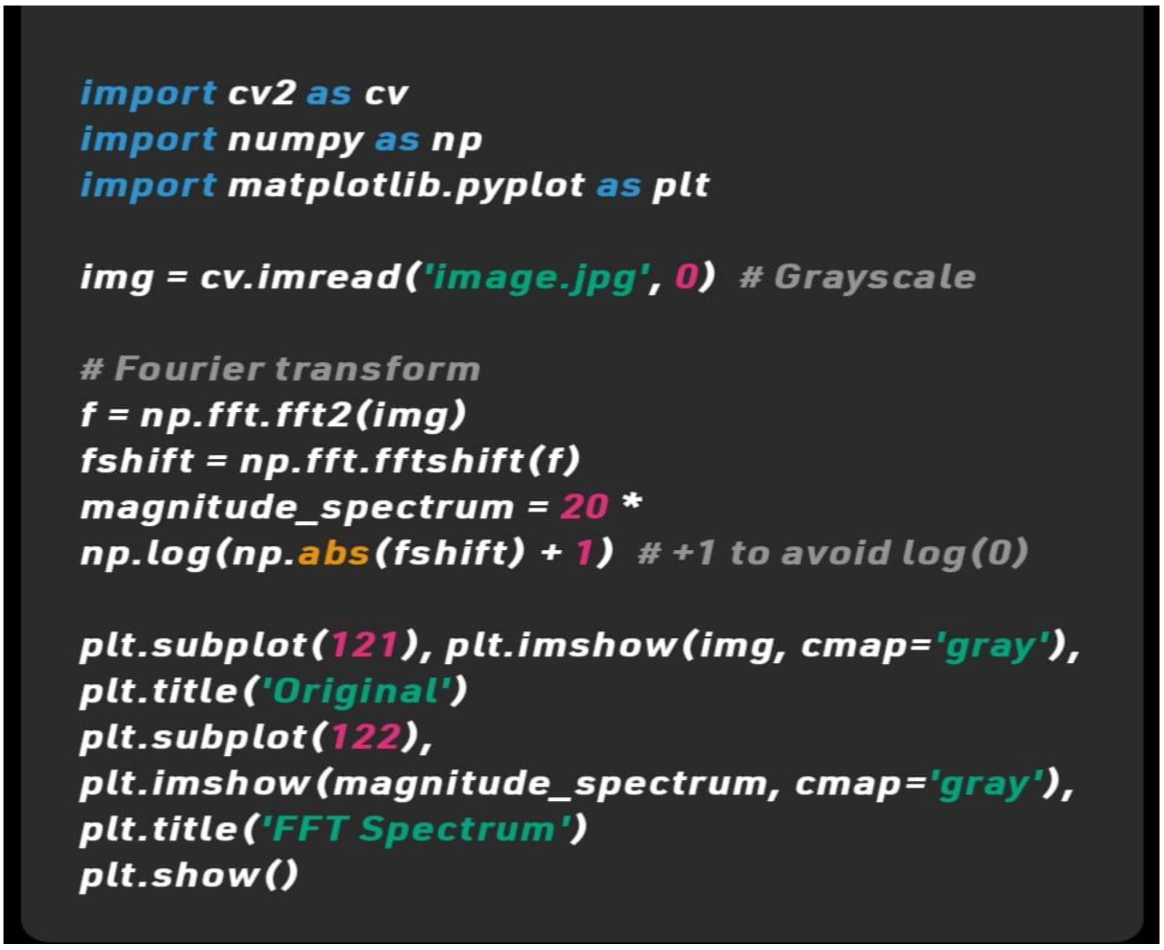


```
import cv2 as cv

img = cv.imread('image.jpg',
cv.IMREAD_GRAYSCALE)
edges = cv.Canny(img, 100, 200)

cv.imshow("Original", img)
cv.imshow("Edges", edges)
cv.waitKey(0)
cv.destroyAllWindows()
```

5. Fourier Transform

- Converts image from spatial domain (pixels) to frequency domain.
- Low frequencies → smooth areas.
- High frequencies → edges and details.
- Used for frequency filtering, image compression, and noise removal.

Example: Magnitude Spectrum



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

img = cv.imread('image.jpg', 0)  # Grayscale

# Fourier transform
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20 *
np.log(np.abs(fshift) + 1)  # +1 to avoid log(0)

plt.subplot(121), plt.imshow(img, cmap='gray'),
plt.title('Original')
plt.subplot(122),
plt.imshow(magnitude_spectrum, cmap='gray'),
plt.title('FFT Spectrum')
plt.show()
```