Introduction to open cv

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a common infrastructure for computer vision applications and is designed to be efficient and fast. OpenCV is widely used in real-time image processing, robotics, machine learning, and artificial intelligence applications, among others.

1. overview

OpenCV is aimed at real-time computer vision and is capable of processing images and videos to identify objects, faces, or even the handwriting of a human. It has interfaces for C++, Python, Java, and MATLAB/Octave. The library contains over 2500 optimized algorithms, which can be used for a variety of tasks, including:

- Image processing (filtering, transformations, etc.)
- Feature detection (corners, edges, etc.)
- Object detection (using machine learning models)
- Face detection and recognition
- Motion analysis and tracking
- 3D reconstruction
- Image stitching and panorama generation
- Augmented reality

2. History

OpenCV was initially developed by Intel in 1999 to facilitate CPU-intensive applications in computer vision. In 2006, it was made open-source, and since then, a large community has contributed to its development. Over the years, OpenCV has evolved significantly:

- 2000: Initial release of OpenCV (version 1.0).

- 2006: OpenCV is released as open-source software.

- 2012: Release of OpenCV 2.0, which introduced a major restructuring of the library.

- 2016: OpenCV 3.0 introduced the DNN module, which made it easier to deploy deep learning models.

- 2018: OpenCV 4.0 was released, featuring a more user-friendly interface and enhancements for performance and GPU support.

- Today, OpenCV is the most popular computer vision library and is widely used in both industry and academia.

3. installation

OpenCV can be installed on various platforms, including Windows, macOS, and Linux. Below are the basic installation steps for OpenCV using Python, which is one of the most common programming languages used with OpenCV.

Python installation

1. Install Python: If you don't have Python installed, download it from the [official Python website](https://www.python.org/downloads/). Make sure to check the box "Add Python to PATH" during installation.

2. Install OpenCV using pip:

   Open a terminal or command prompt and run the following command:

   `bash

   pip install opencv-python

   `

   Optionally, if you want the additional contrib modules:

   `bash

   pip install opencv-contrib-python

   `

3. Verify the Installation: Open a Python shell or IDE and run:

`python

import cv2

print(cv2.version)

`

This should print the version of OpenCV you installed, confirming that the installation was successful.

Installation on Other Platforms

- Linux: You can install OpenCV using package managers such as apt:

`bash

sudo apt update

sudo apt install python3-opencv

`

- Windows: Besides pip, you may also build OpenCV from source or use pre-built binaries from the [OpenCV releases page](https://github.com/opencv/opencv/releases).

- macOS: OpenCV can also be installed via Homebrew:

`bash

brew install opencv

`

4. Basic Setup

Once OpenCV is installed, you can create a simple script to ensure everything is working properly.

Here’s a basic example of loading and displaying an image using OpenCV:

`python

import cv2

Load an image

image = cv2.imread('pathtoyour_image.jpg') # Replace with your image path

Display the image in a window

cv2.imshow('Loaded Image', image)

Wait for a key press and close the displayed window

cv2.waitKey(0)

cv2.destroyAllWindows()

`

conclusion

OpenCV is a powerful and versatile library that supports a multitude of applications in computer vision. With its rich set of functionalities and ease of use, it has become a preferred choice for both beginners and professionals working in the field of computer vision and machine learning.

Core functionality (core module)

The Core Module of OpenCV forms the backbone of the library, providing essential data structures, functions, and operations for image processing and computer vision. Understanding the core functionalities is critical for anyone looking to leverage OpenCV for image processing tasks. Below, we'll cover the basic data structures, array operations, and utility functions available in the Core module.

1. Basic Data Structures

OpenCV primarily uses the cv::Mat class to represent images and matrices, which forms the foundation for various operations.

cv::Mat

- Definition:- The cv::Mat object represents an image or a matrix. It's a flexible data structure that can represent images in various formats, channels, and depth.

- Constructors:- You can create a cv::Mat object with various constructors, either empty or initialized with specific dimensions and type.

`python

```
import cv2
import numpy as np

# Creating an empty Mat (matrix)
mat1 = np.empty((480, 640, 3), dtype=np.uint8)  # Height, Width, Channels

# Creating a zero-initialized Mat
mat2 = np.zeros((480, 640, 3), dtype=np.uint8)

# Creating a Mat filled with a specific color (blue)
color_image = np.full((480, 640, 3), (255, 0, 0), dtype=np.uint8)  # Blue color
```

`

2. Basic Operations on Arrays

OpenCV provides a variety of operations that can be performed on cv::Mat objects (images), enabling manipulation, filtering, and analysis.

Accessing Pixel Values

You can access individual pixel values using array indexing.

`python

pixelvalue = colorimage[100, 100] # Access pixel at (100, 100)

b, g, r = color_image[100, 100]   # Access Blue, Green, Red channels

`

Modifying Pixel Values

You can modify the pixel values directly:

`python

color_image[100, 100] = [0, 255, 0] # Change pixel to green at (100, 100)

`

Image Operations

OpenCV supports a wide range of operations:

- Arithmetic Operations:

You can perform basic arithmetic operations like addition, subtraction, multiplication, and division.

`python

Adding two images

result_image = cv2.add(mat1, mat2)

Scalar multiplication

scaledimage = cv2.multiply(colorimage, 0.5) # Reduce brightness by half

`

- Image Resizing:

Resize images to different dimensions.

`python

resizedimage = cv2.resize(colorimage, (320, 240)) # Resize to 320x240

`

- Image Cropping:
  You can extract regions of interest (ROI) from an image.

`python
roi = color_image[50:200, 50:200] # Crop a square region
`

- Color Space Conversion:
  Convert between different color spaces (e.g., BGR to Grayscale).

`python
grayimage = cv2.cvtColor(colorimage, cv2.COLOR_BGR2GRAY) # Convert to grayscale
`

3. Utility Functions

The Core Module also provides utility functions that can prove useful in various tasks.

- Image Initialization: Create images with certain specifications.

`python
all_zeros = np.zeros((480, 640), dtype=np.uint8) # 1-channel grayscale image
all_ones = np.ones((480, 640, 3), dtype=np.uint8) * 255 # 3-channel white image
`

- Image Saving and Loading:
  You can save and load images using OpenCV functions.

`python

```
cv2.imwrite('outputimage.jpg', colorimage)  # Save the image
loadedimage = cv2.imread('outputimage.jpg')  # Load the image
`
```

- Drawing Functions: Basic drawing functions to overlay shapes on images.

```
`python
cv2.rectangle(color_image, (50, 50), (200, 200), (0, 255, 0), 2)  # Draw a green rectangle
cv2.circle(color_image, (250, 250), 50, (255, 0, 0), -1)          # Draw a filled blue circle
`
```

conclusion

The Core module of OpenCV is fundamental to image processing and manipulation, providing essential data structures (like cv::Mat), array operations, and utility functions that enable developers to create powerful computer vision applications. With a solid understanding of these core functionalities, you can effectively utilize OpenCV to handle various image tasks, ranging from basic loading and saving to sophisticated image manipulation and analysis.

Image processing

The imgproc module of OpenCV provides a wide range of functions for image processing tasks, including transformations, filtering, geometric operations, and histograms. This module is essential for manipulating images and preparing them for further analysis or machine learning applications. Below is an overview of these functionalities, along with examples.

1. transformation

Transformations modify the pixel values of an image according to certain rules or algorithms. Common transformations include resizing, rotation, and affine transformations.

Image Resizing-

Resizing changes the dimensions of an image.

`python

import cv2

Load the original image

image = cv2.imread('example.jpg')

Resize to 480x360

resized_image = cv2.resize(image, (480, 360))

Display the resized image

cv2.imshow('Resized Image', resized_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Image Rotation

You can rotate an image by a certain angle:

`python

Get the image dimensions

(h, w) = image.shape[:2]

Calculate the center of the image

center = (w // 2, h // 2)

Create a rotation matrix

angle = 45 # degrees

scale = 1.0 # scale factor

rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

Perform the rotation

rotatedimage = cv2.warpAffine(image, rotationmatrix, (w, h))

Display the rotated image

cv2.imshow('Rotated Image', rotated_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Affine Transformation

Affine transformations preserve lines and parallelism (e.g., translations, rotation, scaling).

`python

Define 3 points for the transformation

pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

Get the transformation matrix

matrix = cv2.getAffineTransform(pts1, pts2)

Apply the affine transformation

affinetransformedimage = cv2.warpAffine(image, matrix, (w, h))

Display the affine transformed image

cv2.imshow('Affine Transformed Image', affinetransformedimage) cv2.waitKey(0)

cv2.destroyAllWindows()

`

2. filtering

Filtering is used to enhance or suppress certain features in an image, such as noise reduction or edge detection.

Gaussian Blur

A Gaussian blur is a common way to smooth images and reduce noise.

`python

Apply Gaussian Blur

blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

Display the blurred image

cv2.imshow('Gaussian Blurred Image', blurred_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Median Blur

Median filtering is effective for removing noise while preserving edges.

`python

Apply Median Blur

medianblurredimage = cv2.medianBlur(image, 5)

Display the median blurred image

cv2.imshow('Median Blurred Image', medianblurredimage)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Edge Detection

Canny edge detection is widely used in image processing.

`python

Perform Canny edge detection

edges = cv2.Canny(image, 100, 200)

Display the edges

cv2.imshow('Edges', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

3. Geometric Operations

Geometric transformations and operations modify the spatial relationships in the image.

Image Flipping

You can flip images both horizontally and vertically.

`python

Flip the image vertically

flipped_image = cv2.flip(image, 0) # 0 for vertical, 1 for horizontal, -1 for both

Display the flipped image

cv2.imshow('Flipped Image', flipped_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Image Cropping

You can extract specific regions of an image:

`python

Crop the image to a specified region

cropped_image = image[100:300, 100:400]

Display the cropped image

cv2.imshow('Cropped Image', cropped_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

4. Histograms

Histograms represent the frequency distribution of pixel intensities in an image. They are useful for analyzing and manipulating images based on their intensity levels.

Calculating Histograms

You can compute the histogram of an image using OpenCV:

`python

Convert to grayscale for histogram calculation

grayimage = cv2.cvtColor(image, cv2.COLORBGR2GRAY)

Calculate histogram

histogram = cv2.calcHist([gray_image], [0], None, [256], [0, 256])

Plot histogram (using matplotlib for visualization)

import matplotlib.pyplot as plt

plt.figure()

plt.title("Grayscale Histogram")

plt.xlabel("Bins")

plt.ylabel("Number of Pixels")

plt.plot(histogram)

plt.xlim([0, 256])

plt.show()

`

Histogram Equalization

Histogram equalization improves the contrast of an image by effectively spreading out the most frequent intensity values.

`python

Equalize the histogram of the grayscale image

equalizedimage = cv2.equalizeHist(grayimage)

Display equalized image

cv2.imshow('Equalized Image', equalized_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

`

Conclusion

The imgproc module in OpenCV offers a plethora of functionalities central to image processing tasks. From geometric transformations and filtering techniques to histogram analysis, these tools empower you to manipulate and analyze images effectively. Mastering these operations forms a critical part of building robust computer vision applications and enables you to prepare images for subsequent processes, such as feature extraction, classification, or machine learning tasks.

Application utils

📷 highgui — GUI for Image and Video Display

This module handles **Graphical User Interface** operations such as:

- **Displaying images and videos** (cv::imshow, cv::waitKey, etc.)
  - **Creating and managing windows** (cv::namedWindow, cv::destroyAllWindows)
  - **Basic GUI elements** like trackbars and mouse callbacks

🔷 **Key Functions:**

- cv::imshow("Window Name", image) — Show image in window
- cv::waitKey(0) — Wait for key press (in milliseconds)
- cv::destroyAllWindows() — Close all OpenCV windows

🖼 imgcodecs — Image Reading and Writing

- Handles **loading and saving images** in various formats.

🔷 **Key Functions:**

- cv::imread("path/to/image.jpg", flags) — Load image
- cv::imwrite("output.jpg", image) — Save image
- cv::imdecode() cv::imencode() — Work with images in memory (e.g., from byte streams)

📁 **Supported Formats:** PNG, JPEG, BMP, TIFF, WebP, etc.

📼 videoio — Video and Camera Access

Responsible for reading/writing video files and accessing camera streams.

🔷 **Key Functions:**

- cv::VideoCapture — Open video file or camera
- cv::VideoWriter — Save video to file
- cap.read(frame) — Capture frame from video/camera
- cap.open(index) — Open default or external webcam

🎥 **Supported Codecs & Devices:** Depends on backend (e.g., FFmpeg, DirectShow, V4L2)

🔁 Example (Python):

```
import cv2

# Read and display an image
img = cv2.imread('example.jpg')
cv2.imshow('Image Window', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Open webcam and show feed
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow('Camera Feed', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Camera calibration and 3d Reconstruction

🔧 Key Features of the calib3d Module

1. Camera Calibration

Used to find the camera's intrinsic (focal length, optical center) and extrinsic (position and orientation) parameters, along with distortion coefficients.

- Why it's needed: Real-world lenses cause distortion. Calibration helps correct that.

🔹 **Functions:**

- cv::calibrateCamera() — Calibrates a single camera using images of a known pattern (e.g., a chessboard).
- cv::getOptimalNewCameraMatrix() — Computes a new camera matrix to minimize unwanted pixels after undistortion.

Outputs:

- **Camera Matrix** (intrinsic parameters)
- **Distortion Coefficients** (radial and tangential)
- **Rotation and Translation Vectors** (extrinsic parameters)

2. Undistortion

Removes lens distortion using calibration parameters.

🔹 **Functions:**

- cv::undistort() — Straightens a distorted image
- cv::initUndistortRectifyMap() + cv::remap() — More control over undistortion and rectification

3. Stereo Vision (Two Cameras)

Used to recover 3D depth from two 2D images taken from slightly different viewpoints.

🔹 **Functions:**

- cv::stereoCalibrate() — Calibrate two cameras together
- cv::stereoRectify() — Rectify images so that epipolar lines are aligned (makes depth estimation easier)
- cv::computeCorrespondEpilines() — Find corresponding epilines
- cv::StereoBM, cv::StereoSGBM — Create disparity (depth) maps

Disparity Map: Difference in pixel locations between two views → used to compute depth.

4. Pose Estimation

Estimates the **3D position and orientation** of a known object or camera from 2D image points.

🔷 **Functions:**

- cv::solvePnP() — Estimates pose from 2D-3D point correspondences
- cv::Rodrigues() — Converts between rotation vectors and matrices
- cv::projectPoints() — Projects 3D points onto the image plane using given camera parameters

Used for:

- Augmented Reality
- Robotics
- Object tracking and navigation

📌 Common Calibration Workflow

1. Collect multiple images of a known calibration pattern (like a chessboard)
2. Detect corners using cv::findChessboardCorners()
3. Calibrate with cv::calibrateCamera()
4. Use cv::undistort() to remove lens distortion

🧠 Real-World Applications

- AR (projecting 3D objects in the real world)
- SLAM (Simultaneous Localization and Mapping)

- Robot navigation
- Depth perception for drones and autonomous vehicles
- Industrial inspection (accurate object measurements)

Object detection

The objdetect **module** in OpenCV is designed for **object detection**, especially using **pre-trained model** like **cascade classifiers**. It's most famously used for **face detection**, but can also detect eyes, smiles, bodies, license plates, etc.

🎯 Main Capabilities of the objdetect Module

1. **Cascade Classifiers (Haar and LBP)**

These are **machine learning-based** object detectors that use a cascade of increasingly complex classifiers to detect objects efficiently.

- **Haar cascades**: More accurate, but slower.
- **LBP cascades**: Faster, less accurate.

🔷 Function:

cv2.CascadeClassifier.detectMultiScale()

- Pre-trained Models:

OpenCV provides XML files trained on:

- Faces
- Eyes
- Smiles
- Full bodies
- Upper bodies
- License plates

➡️ You can find them in the OpenCV GitHub repo: opencv/data/haarcascades/

2. **Face Detection with Haar Cascade — Example (Python)**

```
import cv2

# Load the cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

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

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

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

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

🔍 3. detectMultiScale() **Parameters Explained**

Parameter	**Description**
scaleFactor	How much the image size is reduced at each scale (e.g., 1.1 = 10% smaller)
minNeighbors	How many neighbors each rectangle should have to be retained (higher = fewer detections)
minSize/maxSize	Size limits of the object to be detected

4. **Other Detection Options**

While objdetect is centered on classic detection (Haar/LBP), OpenCV also integrates with modern **deep learning models** via:

- **DNN module** (for YOLO, SSD, etc.)
- **Face recognition libraries** like dlib or mediapipe

-But objdetect is still popular for:

- Lightweight real-time detection
- Projects on limited hardware (e.g., Raspberry Pi)

- Quick prototyping

Use Cases

- Face detection for authentication
- People counting
- Object tracking (when combined with `cv2.Tracker`)
- Smile detection in camera apps
- License plate recognition systems

2D features framework

The `feature2d` **module** in OpenCV is the core of the **2D feature detection, description, and matching** framework — the building block for many computer vision tasks like object recognition, image stitching, tracking, and 3D reconstruction.

🔍 What is a Feature?

A **feature** is a keypoint or region in an image that is **distinctive**, **repeatable**, and **invariant** to changes in scale, rotation, or lighting. Think of corners, edges, or blobs that "stand out" in an image.

- Key Concepts in `feature2d`

1. **Feature Detection**

Find **keypoints** — distinctive image points (like corners or blobs).

✅ Examples:

- **SIFT** (Scale-Invariant Feature Transform)
- **SURF** (Speeded-Up Robust Features — now proprietary)
- **ORB** (Oriented FAST and Rotated BRIEF — fast & open-source)
- **FAST**, **Harris**, **AGAST**

2. Feature Description

Convert keypoints into **numeric descriptors** that capture local appearance.

✅ Descriptors:

- **SIFT** — 128D float vector
- **ORB** — 256-bit binary string
- **BRIEF**, **FREAK** — binary descriptors

3. 🔗 Feature Matching

Match descriptors across images using distance metrics.

✅ Matchers:

- **Brute-Force Matcher** (e.g., Hamming, L2)
- **FLANN** (Fast Library for Approximate Nearest Neighbors) — faster for large datasets

- Popular Detectors & Descriptors in OpenCV

Feature	Type	Open Source	Scale/Rotation Invariant	Speed
SIFT	Float	✅	✅	🐢 Slower
ORB	Binary	✅	✅	Fast
FAST	Detector only	✅	❌	⚡ Very Fast
BRIEF	Descriptor only	✅	❌	Fast
AKAZE	Binary	✅	✅	Medium
KAZE	Float	✅	✅	Slower

- Example: Feature Detection + Matching (SIFT + BFMatcher)

```
import cv2

# Load images
img1 = cv2.imread('img1.jpg', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('img2.jpg', cv2.IMREAD_GRAYSCALE)

# Create SIFT detector
sift = cv2.SIFT_create()
```

```
# Detect and compute features
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# Brute-force matcher with L2 norm
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)

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

# Draw matches
result = cv2.drawMatches(img1, kp1, img2, kp2, good, None, flags=2)
cv2.imshow("Matches", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Other Useful Functions in feature2d:

- cv::drawKeypoints() / cv::drawMatches() — visualize features/matches
- cv::BFMatcher / cv::FlannBasedMatcher — for matching descriptors
- cv::KeyPoint — structure that stores keypoint info

Applications

- Image matching & alignment
- Panorama stitching
- Object recognition/tracking
- Structure-from-motion (SfM)
- Augmented reality

Deep neural networks

The **Deep Neural Network (DNN) module** in OpenCV (cv::dnn) provides tools to **load, configure, and run pre-trained deep learning models** for inference. This module is designed to be lightweight and efficient, enabling the integration of popular models trained in frameworks like TensorFlow, PyTorch (via ONNX), Caffe, and others.

🔧 1. **Supported Model Formats**

The cv::dnn module supports models trained in:

- **Caffe** (.prototxt + .caffemodel)
  - **TensorFlow** (.pb, .pbtxt)
  - **Torch** (.t7, .net)
  - **ONNX** (.onnx) — common for PyTorch models
  - **Darknet** (.cfg, .weights)
  - **TensorFlow Lite** (.tflite)

2. **Basic Workflow**

✅ Step 1: Load the model

```
cv::dnn::Net net = cv::dnn::readNetFromONNX("model.onnx");
// or from TensorFlow
// cv::dnn::Net net = cv::dnn::readNetFromTensorflow("model.pb");
```

✅ Step 2: Prepare the input

```
cv::Mat inputBlob = cv::dnn::blobFromImage(
    image,         // input image
    1.0,           // scale factor
    cv::Size(224, 224), // size expected by the model
    cv::Scalar(0, 0, 0), // mean subtraction values
    true,          // swapRB
    false          // crop
);
```

✅ Step 3: Set input and run forward pass

```
net.setInput(inputBlob);
cv::Mat output = net.forward();
```

✅ Step 4: Process output

The output format depends on the model. For classification models, it might be a 1D vector of class scores; for detection models, a matrix of bounding boxes, etc.

3. **Example: Image Classification (e.g., ResNet)**

```
cv::Mat image = cv::imread("image.jpg");

// Load pre-trained model
cv::dnn::Net net = cv::dnn::readNetFromONNX("resnet50.onnx");

// Prepare input blob
cv::Mat blob = cv::dnn::blobFromImage(image, 1.0/255.0, cv::Size(224, 224),
                                      cv::Scalar(0, 0, 0), true, false);

// Set the input and run inference
net.setInput(blob);
cv::Mat prob = net.forward();

// Get class with highest score
cv::Point classIdPoint;
double confidence;
cv::minMaxLoc(prob.reshape(1, 1), 0, &confidence, 0, &classIdPoint);
int classId = classIdPoint.x;

std::cout << "Predicted class ID: " << classId << " with confidence " << confidence <<
std::endl;
```

4. **Optional: Set Backend and Target**

```
net.setPreferableBackend(cv::dnn::DNN_BACKEND_OPENCV); // or
DNN_BACKEND_CUDA
net.setPreferableTarget(cv::dnn::DNN_TARGET_CPU);      // or DNN_TARGET_CUDA
```

📘 Notes:

- Use blobFromImage() to preprocess images to match your model's input size, normalization, and channel ordering.
  - Check the model documentation or source (e.g., ONNX model zoo) for input/output formats.
  - Use cv::dnn::NMSBoxes() for filtering overlapping boxes in object detection.

Graph API

The **Graph API (G-API)** in OpenCV (gapi module) is a powerful framework designed for building **pipeline-based, highly optimized computer vision applications**. Unlike traditional imperative OpenCV code, G-API enables you to **describe a computation as a graph**, which can then be **compiled and executed efficiently** across various backends (CPU, OpenCL, OpenVINO, etc.).

- Core Concepts of G-API

Concept	Description
Graph	A dataflow graph where nodes are operations and edges are data.
Compile-time separation	You describe what you want to compute, then compile it for execution.
Backends	You can plug in different execution backends for acceleration (e.g., OpenCL, OpenVINO).
Pipeline	A chain of operations optimized together (e.g., resize → filter → inference).

1. **Basic G-API Pipeline Example**

✅ Step 1: Define Inputs and Computation Graph

```
#include <opencv2/gapi.hpp>
#include <opencv2/gapi/core.hpp>
#include <opencv2/gapi/imgproc.hpp>

cv::GMat in;                        // Input node
cv::GMat blurred = cv::gapi::blur(in, cv::Size(5, 5));   // Blur operation
cv::GMat edges = cv::gapi::Canny(blurred, 100, 200);    // Canny edge detection
```

```
cv::GComputation graph(cv::GIn(in), cv::GOut(edges));    // Wrap into a graph
```

✅ Step 2: Compile the Graph

```
cv::GCompiled compiledGraph = graph.compile(cv::descr_of(inputImage));
```

✅ Step 3: Execute the Graph

```
cv::Mat outputEdges;
compiledGraph(inputImage, outputEdges);
```

2. Why Use G-API?

Advantage	Benefit
Performance	Combines operations into fewer passes, reducing memory overhead and CPU/GPU cycles.
Backend Flexibility	Can run on CPU, OpenCL, Intel OpenVINO, and more.
Clean Abstraction	Separates algorithm definition from execution details.
Inference Integration	Integrates well with cv::dnn and OpenVINO for efficient DNN inference.

3. DNN + Preprocessing with G-API

You can build a pipeline like:

```
Input → Resize → Mean Subtract → DNN Inference → Postprocessing
```

With G-API:

```
cv::GMat in;
cv::GMat resized = cv::gapi::resize(in, cv::Size(224, 224));
cv::GMat floatImg = cv::gapi::convertTo(resized, CV_32F);
cv::GMat normImg = cv::gapi::subC(floatImg, cv::Scalar(123.68, 116.78, 103.94));

// Use GNet to do inference
cv::GMat dnnOut = cv::gapi::infer<SomeNet>(normImg);

cv::GComputation comp(cv::GIn(in), cv::GOut(dnnOut));
```

Then compile with a specific backend like OpenVINO:

```
cv::GNetPackage networks = cv::gapi::networks(cv::gapi::ade::make<SomeNet>("model.xml", "model.bin"));
```

```
auto compiled = comp.compile(cv::descr_of(image), cv::compile_args(cv::gapi::ie::params(SomeNet::tag)
   .model_path("model.xml")
   .weights_path("model.bin")
   .backend("IE")
   .device("CPU")));
```

4.Use Cases for G-API

- **Real-time video processing pipelines**
- **Pre- and post-processing around DNN models**
- **Edge deployment with OpenVINO**
- **Optimized embedded vision apps**

- G-API vs Regular OpenCV

Feature	OpenCV (Classic)	G-API
Execution Model	Imperative	Declarative
Optimization	Manual	Automatic
Backend Flexibility	Limited	High (CPU, OpenCL, OpenVINO)
Pipeline Fusion	❌	✅
DNN Integration	Yes	Yes (via infer<>)

Other tutorials

1. ml — Machine Learning Module

The ml module in OpenCV provides classical machine learning algorithms (not deep learning), fully integrated with OpenCV data structures (cv::Mat).

🔷 Supported Algorithms:

- **SVM (Support Vector Machine)**
- **KNN (k-Nearest Neighbors)**
- **DTrees (Decision Trees)**
- **RTrees (Random Forest)**

- **Boosting, NormalBayesClassifier**
- **ANN_MLP (Multi-layer Perceptron)**

✅ Example: Train SVM

```
cv::Ptr<cv::ml::SVM> svm = cv::ml::SVM::create();
svm->setType(cv::ml::SVM::C_SVC);
svm->setKernel(cv::ml::SVM::LINEAR);
svm->train(trainingData, cv::ml::ROW_SAMPLE, labels);
svm->save("svm_model.xml");
```

2. objdetect — **Object Detection Module**

This module provides **traditional object detection** tools like Haar cascades and HOG descriptors.

✅ Example: Face Detection with Haar Cascades

```
cv::CascadeClassifier faceCascade;
faceCascade.load("haarcascade_frontalface_default.xml");

std::vector<cv::Rect> faces;
faceCascade.detectMultiScale(image, faces);

for (auto& face : faces)
    cv::rectangle(image, face, cv::Scalar(255, 0, 0), 2);
```

Also includes:

- HOG + SVM for pedestrian detection
- QR code detection
- ArUco marker detection (in `aruco` submodule)

3. photo — **Photo Enhancement Module**

Used for tasks like **image denoising**, **seamless cloning**, **inpainting**, **HDR**, and **style transfer**.

✅ Example: Seamless Cloning

```
cv::Mat result;
cv::seamlessClone(src, dst, mask, center, result, cv::NORMAL_CLONE);
```

Other highlights:

- cv::fastNlMeansDenoising()
- cv::inpaint() (removes objects from images)
- cv::detailEnhance(), cv::stylization() (artistic effects)

4. stitching — **Image Stitching Module**

Used to **automatically create panoramas** from multiple overlapping images.

☑ Basic Stitching Pipeline

```
std::vector<cv::Mat> images = {img1, img2, img3};
cv::Mat pano;

cv::Ptr<cv::Stitcher> stitcher = cv::Stitcher::create(cv::Stitcher::PANORAMA);
cv::Stitcher::Status status = stitcher->stitch(images, pano);

if (status == cv::Stitcher::OK)
   cv::imwrite("panorama.jpg", pano);
```

- Uses:

- Feature detection (ORB, SURF, etc.)
- Homography estimation
- Warping and blending

🎥 5. video — **Video Analysis Module**

Provides algorithms for **motion analysis**, **tracking**, **optical flow**, and **background subtraction**.

☑ Optical Flow (Farneback)

```
cv::Mat flow;
cv::calcOpticalFlowFarneback(prevGray, gray, flow,
                        0.5, 3, 15, 3, 5, 1.2, 0);
```

☑ Background Subtraction

```
cv::Ptr<cv::BackgroundSubtractor> bgSub = cv::createBackgroundSubtractorMOG2();
cv::Mat fgMask;
```

```
bgSub->apply(frame, fgMask);
```

☑ Tracking APIs (e.g., KCF, CSRT)

```
cv::Ptr<cv::Tracker> tracker = cv::TrackerKCF::create();
tracker->init(frame, bbox);  // bbox: initial object location
tracker->update(frame, bbox);  // bbox updated
```

- Summary Table

Module	Key Features
ml	Classical ML (SVM, KNN, DTrees, ANN)
objdetect	Face, object, QR code detection
photo	Inpainting, denoising, HDR, cloning
stitching	Panorama stitching from multiple images
video	Optical flow, motion tracking, background subtraction

Theory of image processing

1. **Pixels and Images**

☑ What is an image?

- An image is a **2D grid of pixels** (picture elements).
- Each pixel contains **intensity** or **color** information.

☑ Types of images:

Type	Description
Grayscale	1 channel (0–255)
RGB	3 channels (Red, Green, Blue)
RGBA	RGB + Alpha (transparency)
Binary	Black and white only (0 or 255)

🌈 2. **Color Spaces**

Color spaces define how colors are represented in an image.

✅ Common color spaces:

Color Space	Use Case
RGB	Standard display
HSV	Color-based segmentation (Hue = color)
Grayscale	Simpler processing
Lab	Perceptually uniform (used in image enhancement)
YCrCb	Used in compression (JPEG, video)

- Color conversion:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

3. **Convolution and Filters**

✅ What is convolution?

Convolution is applying a **kernel (small matrix)** to an image to produce a filtered version.
It's a core operation in image processing.

✅ Example: 3x3 kernel

```
[[-1, -1, -1],
 [-1,  8, -1],
 [-1, -1, -1]]
```

This kernel enhances edges by highlighting regions with rapid intensity change.

✅ Types of filters:

Filter Type	Purpose
Blur / Smoothing	Noise reduction
Sharpening	Edge emphasis
Edge Detection	Find object boundaries
Gaussian Filter	Smooths while preserving structure

4. **Edge Detection**

✅ Why edge detection?

Edges mark significant transitions (object boundaries, features).

✅ Common edge detectors:

Method	Description
Sobel	Finds edges in X/Y direction using gradients
Laplacian	Detects edges using second derivative
Canny	Most popular, uses gradient + thresholding + NMS

- Canny Edge Example:

```
edges = cv2.Canny(image, threshold1=100, threshold2=200)
```

5. **Fourier Transform (FFT)**

✅ What is it?

The **Fourier Transform** converts an image from **spatial domain** (pixels) to **frequency domain**.

✅ Why is it useful?

- High frequencies: sharp edges, noise
- Low frequencies: smooth regions, lighting

✅ Use cases:

- Image compression (e.g., JPEG)
- Filtering (removing noise or patterns)
- Texture analysis

- OpenCV Fourier transform:

```
f = np.fft.fft2(gray)
fshift = np.fft.fftshift(f)
magnitude = 20 * np.log(np.abs(fshift))
```

- Summary Table

Concept	Key Idea
Pixels	Building blocks of images
Color Spaces	Ways to interpret pixel color
Convolution	Applying filters to transform images
Edge Detection	Find boundaries/features
Fourier Transform	Analyze frequencies in an image

Bonus: Image Representation in Computers

- Images are stored as arrays: height x width x channels
- Example: a 640x480 RGB image is a 3D array of shape (480, 640, 3)
- Libraries like OpenCV, PIL, NumPy allow access and manipulation

Plan to guide open cv

Here's a structured plan to guide you through all those tasks — from theory to practice — and help you set up your opencv-week1 **GitHub repo** with confidence.

📘 PART 1: **Understand Core Concepts**

- Image Filtering

- **Definition**: Applying a kernel to an image to enhance or extract features.
- **Types**:
  - **Smoothing**: Gaussian, median, bilateral
  - **Sharpening**: Laplacian or custom kernels
  - **Use Case**: Reduce noise in medical X-rays, enhance edges in satellite imagery

-Thresholding

- Converts grayscale to binary image (simplifies segmentation).
- Methods:

        - cv2.threshold() (global)
        - cv2.adaptiveThreshold() (local)
        - **Otsu's method**: auto-thresholding based on image histogram
    - **Use Case**: License plate extraction, document binarization

- Geometric Transformations

    - **Translation, rotation, scaling**: Move, rotate, or resize images.
    - **Affine and Perspective** transforms: Maintain straight lines; useful for correcting image distortions.
    - **Use Case**: Camera calibration, stitching panoramas

PART 2: **Research OpenCV Applications**

Domain	Application Description
1. Robotics	Object tracking, SLAM (Simultaneous Localization and Mapping), navigation using vision
2. Medical Imaging	Tumor detection, organ segmentation, real-time surgical guidance
3. Autonomous Vehicles	Lane detection, pedestrian recognition, traffic sign detection
4. Surveillance	Motion detection, face recognition, anomaly detection
5. Industrial Automation	Defect detection on production lines, OCR for labeling
6. Augmented Reality	Marker-based AR using ArUco, real-time overlays
7. Agriculture	Plant disease detection, fruit counting, yield estimation
8. Retail & Inventory	Shelf monitoring, barcode recognition
9. Sports Analytics	Player tracking, ball trajectory estimation
10. Drone Vision	Aerial mapping, object following, obstacle avoidance

PART 3: **OpenCV Window Differences (Linux vs Windows)**

✅ Event Loop and GUI Threads:

OS	Behavior
Windows	cv2.imshow() blocks the main thread and handles GUI in the same thread.
Linux (X11)	OpenCV GUI must run in the **main thread** due to how X11 event loops work.

✅ Issues:

- **Multiple** imshow **calls in threads** → segmentation fault on Linux
- **Fix**: Always call imshow() and waitKey() from the **main thread**

✅ Workarounds:

- Use cv2.namedWindow() with flags (cv2.WINDOW_GUI_EXPANDED)
- Avoid GUI code in threads; use queues to pass frames to a main-thread viewer

PART 4: **Solve Exercises - (YouTube Playlist)**

- Go through the exercises from your **YouTube playlist** (likely related to P. J. Reddie or CV fundamentals)
- Use:
  - GitHub repo: https://github.com/pjreddie
  - Tools: **Python + OpenCV + NumPy + Matplotlib**
- Adapt C++ solutions to Python
- Save each solution in exercises/ folder, e.g.:

```
exercises/
├── 01_blur.py
├── 02_threshold.py
├── 03_edge_detection.py
```

PART 5: **Share Your Practice on GitHub**

✅ Repo Structure: opencv-week1

```
opencv-week1/
├── README.md
├── notes/
|   └── theory.md
├── exercises/
|   ├── 01_blur.py
|   ├── 02_threshold.py
|   └── ...
├── research/
```

```
|   ├── opencv_apps.md
|   └── window_handling.md
└── screenshots/
```

- - - - - README.md Template:

```
# OpenCV Week 1

This repository contains my progress through the first week of OpenCV learning and practice. It includes:

## 📚 Theory Notes
- Image filtering
- Thresholding
- Geometric transformations
- Edge detection & frequency domain

##   Exercises
Solved using Python and OpenCV.
Adapted from YouTube playlist and P.J. Reddie's GitHub.

##   Research
- OpenCV applications across industries
- Linux vs Windows window handling in OpenCV

##   Structure
- `notes/`: Written theory summaries
- `exercises/`: Code for hands-on problems
- `research/`: Summarized findings
- `screenshots/`: Visual outputs
```

- Next Steps

1. **Create the repo**: opencv-week1 on GitHub
2. **Clone it locally and organize folders**
3. **Start committing your notes and code**
4. **Push regularly**: Add screenshots and markdown summaries

Report

Here's a detailed guide to help **complete and structure deliverables** for the `opencv-week1` project on GitHub. This includes the **Comprehensive Report (PDF)** and **source code + practice**.

1. **Comprehensive Report — PDF Documentation**

📄 Suggested Filename:

`opencv-week1-report.pdf`

🗀 Location in repo:

Place it in the root of your `opencv-week1/` directory.

📄 Structure of the Report

1. Cover Page

- Project Title: *OpenCV Week 1 Report*
- Name
- Date
- Repo URL

2. Table of Contents

(Include page numbers and section headers)

3. Introduction

- Brief overview of the project
- What this week covered (theory + hands-on practice)
- Tools used (Python, OpenCV, NumPy, Matplotlib, etc.)

4. Theory Summaries

Structure this by topic. Example:

- Color Spaces

- Source: OpenCV Docs, YouTube: Sentdex OpenCV playlist

- RGB vs HSV vs Lab
- Diagram: HSV color wheel
- Use case: Skin color segmentation, lighting-robust detection

- Convolution and Filtering

- Source: PyImageSearch articles, OpenCV docs
- Kernel operations: mean, Gaussian, Laplacian
- Screenshot: Original vs Blurred image
- Insight: Custom filters allow creation of edge detectors or emboss effects

- Edge Detection

- Used Canny, Sobel, Laplacian
- Challenges with noisy input
- Screenshot: Side-by-side comparison

- Fourier Transform

- Realized frequency components are useful for pattern analysis
- Screenshot: Magnitude spectrum

5. Module Overviews

Summarize each OpenCV module covered:

Module	Key Functions Explored	Application
cv.dnn	Loading ONNX models	Object classification
cv.ml	SVM, KNN	Digit classification
cv.photo	Seamless cloning, denoising	Image editing
cv.video	Background subtraction	Motion detection

6. Application Research Summary

(From earlier task, briefly summarize the use cases)

7. Linux vs Windows GUI Behavior in OpenCV

- Source: OpenCV forums, GitHub issues, personal tests

- Diagram: How imshow() behaves on each OS
- Conclusion: Avoid imshow() inside threads on Linux

8. Practical Challenges & Insights

- Adapting C++ code from PJ Reddie's GitHub to Python
- File path handling (relative vs absolute)
- Interoperability of OpenCV with NumPy
- Managing window events in imshow() and waitKey()

9. Screenshots and Diagrams

Include:

- Before/after images for filters and edge detection
- Output from stitching or DNN models
- Annotated diagrams (color space conversions, kernel behavior, etc.)

10. References

Example format:

- OpenCV Documentation: https://docs.opencv.org/
- PyImageSearch: https://pyimagesearch.com/
- P. J. Reddie GitHub: https://github.com/pjreddie
- Sentdex OpenCV Playlist: https://www.youtube.com/playlist?list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v
- ArXiv papers on image filtering and edge detection

2.Source Code and Practice Upload

🗁 Suggested Folder Structure

```
opencv-week1/
├── README.md
├── opencv-week1-report.pdf
├── notes/
|   └── theory.md
├── exercises/
|   ├── 01_blur.py
|   ├── 02_thresholding.py
```

```
|   ├── 03_edge_detection.py
|   ├── 04_fourier.py
|   ├── ...
├── research/
|   ├── opencv_applications.md
|   └── gui_threading.md
├── images/
|   ├── original.jpg
|   ├── canny_edges.jpg
|   ├── fourier_spectrum.png
|   └── ...
├── screenshots/
|   └── annotated_filters.png
```

README.md Must-Haves

- Overview of what's in the repo
- How to run the exercises
- Dependencies (OpenCV, NumPy, Matplotlib)
- Link to your report PDF

Python Code Style Guide

- Add **inline comments** for key logic
- Include **docstrings** at the top of each script

```
"""
Script: 01_blur.py
Applies a Gaussian blur to an image and compares results with median filter.
"""

# Read image
img = cv2.imread('images/original.jpg')

# Apply Gaussian blur
blurred = cv2.GaussianBlur(img, (5, 5), 0)

# Save output
cv2.imwrite('images/blurred.jpg', blurred)
```

Submission Checklist

Task	Status
[ ] All code organized and commented	
[ ] PDF report written and exported	
[ ] Screenshots/diagrams included	
[ ] GitHub repo created and updated	
[ ] README with overview and instructions	
[ ] References listed in report	

Complete structure

Project Structure (GitHub Repo: opencv-week1/)

```
opencv-week1/
├── README.md
├── requirements.txt
├── opencv-week1-report.pdf
├── notes/
|   └── theory.md
├── exercises/
|   ├── 01_blur.py
|   ├── 02_thresholding.py
|   ├── 03_edge_detection.py
|   ├── 04_fourier_transform.py
|   ├── 05_geometric_transformations.py
|   └── ...
├── research/
|   ├── opencv_applications.md
|   └── gui_threading_linux_vs_windows.md
├── images/
|   ├── input/
|   |   ├── lena.png
|   |   └── example.jpg
|   └── output/
|       ├── canny_edges.png
|       └── blurred.png
└── screenshots/
    ├── result_comparisons.png
    └── hsv_segmentation_diagram.png
```

README.md Template

OpenCV Week 1

This repository contains the notes, practice code, research findings, and documentation for Week 1 of OpenCV learning.

Structure

Folder	Contents
`notes/`	Theoretical notes on image processing
`exercises/`	Python code for image filtering, thresholding, transformations, etc.
`research/`	Exploratory research (OpenCV applications, GUI threading)
`images/`	Input/output images used in exercises
`screenshots/`	Diagrams and screenshots used in the report
`opencv-week1-report.pdf`	Final compiled documentation

Requirements

Install dependencies using:

```bash
pip install -r requirements.txt
```

- Topics Covered

- Pixels, color spaces (RGB, HSV, Lab)
- Image filtering (blur, sharpen, edge detection)
- Thresholding (global, adaptive, Otsu)
- Geometric transformations (resize, rotate, affine)
- Fourier transforms
- Real-world applications of OpenCV
- Differences in OpenCV GUI behavior on Linux vs Windows

- References

See opencv-week1-report.pdf for full list of references and screenshots.

📦 `requirements.txt`

```text
opencv-python
numpy
matplotlib
```

opencv-week1-report.pdf Highlights

- Save this in the **root directory**
- Include:
  - Theory summaries (with diagrams)
  - Research findings
  - Screenshots of image outputs
  - List of all reference sources
  - Key takeaways (what you learned, what was challenging)

Example Exercise File: 01_blur.py

```python
"""
01_blur.py
Applies Gaussian and median blur to an input image and saves results.
"""

import cv2
import numpy as np

# Load image
img = cv2.imread("images/input/lena.png")

# Apply Gaussian Blur
gaussian = cv2.GaussianBlur(img, (5, 5), 0)

# Apply Median Blur
median = cv2.medianBlur(img, 5)

# Save outputs
cv2.imwrite("images/output/gaussian_blur.png", gaussian)
cv2.imwrite("images/output/median_blur.png", median)
```

- Comment your code for readability
- Keep inputs/outputs in proper folders

-- Next Steps

1. Create your repo on GitHub: `opencv-week1`
2. Clone it locally and set up the folder structure
3. Write notes and convert the PDF report
4. Push your commits regularly with messages like `Added edge detection example`
5. Final check: open the repo in browser — does it look clean and professional?