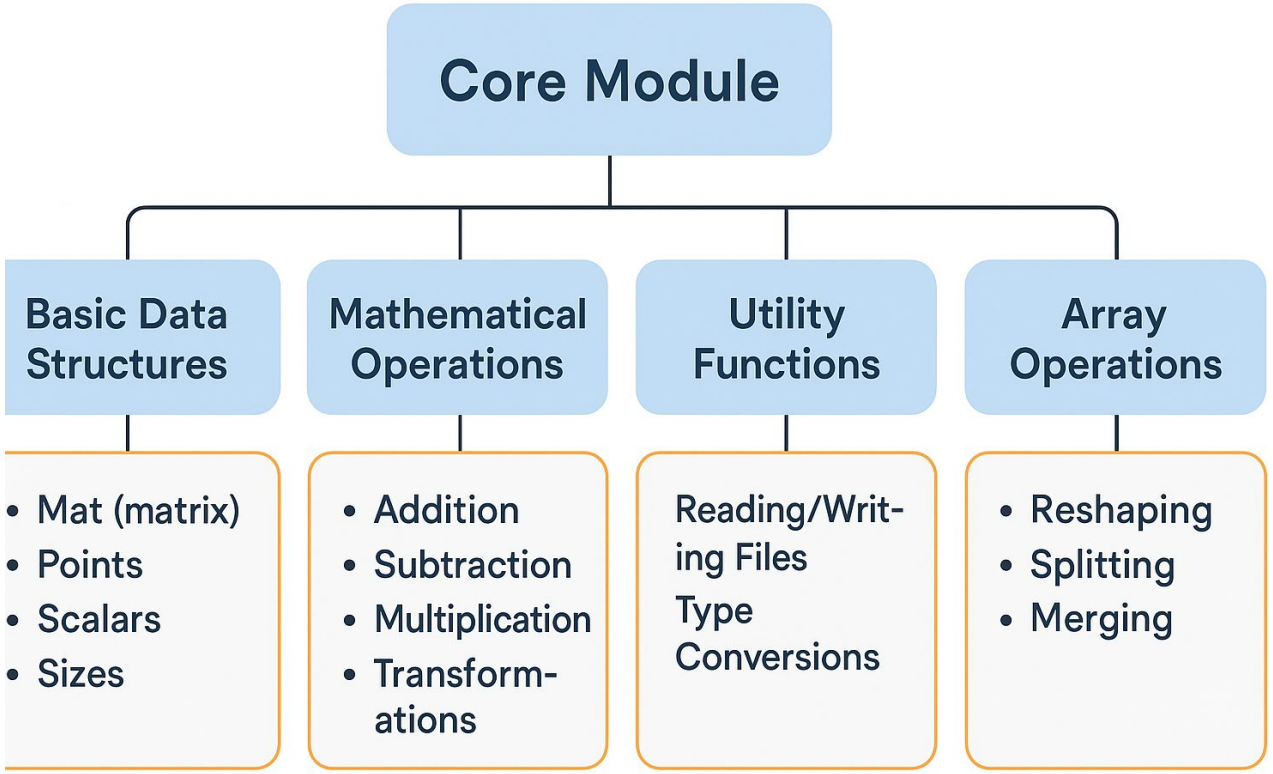
UNIT 5

The Core Module in OpenCV

The **Core (cv2 core)** module is the **foundation of OpenCV**. It provides the **basic building blocks** that are required for all higher-level computer vision and image processing tasks. Without this module, advanced modules like `imgproc`, `ml`, or `video` cannot function. It mainly includes:



1. Basic Data Structures

(a) Mat (Matrix)

- The **Mat object** is the primary data structure for storing images in OpenCV.
- Each image is represented as a **multi-dimensional matrix** (rows × columns × channels).
- Example:
  - A grayscale image → 2D matrix (rows × cols).
  - A color image (BGR) → 3D matrix (rows × cols × 3).

Python Example:

```python
import cv2
import numpy as np

# Create a 3x3 matrix (image)
mat = np.ones((3,3), dtype=np.uint8) * 255
print("Matrix:\n", mat)
```

(b) Points

- Represents 2D or 3D coordinates.
- Used in image transformations, contour detection, shape drawing.
- Example: A point (x, y) = (100, 50) represents a pixel location.

```python
point = (100, 50)    # x=100, y=50
```

(c) Scalars

- Represents a single value or 4-channel value.
- Mostly used for **color representation** in BGR format.

Example:

- `Scalar(255,0,0)` → Blue color.
- `Scalar(0,255,0)` → Green color.

(d) Sizes

- Represents the dimensions of an image or matrix.
- Example: `Size(width, height)`

python

```python
img = cv2.imread("image.jpg")
print("Image Size:", img.shape)  # (rows, cols, channels)
```

2. Mathematical Operations

The Core module supports **basic mathematical operations** on images.

- **Addition:** Increases brightness.

```
brighter = cv2.add(img, 50)
```

- **Subtraction:** Decreases brightness or highlights differences.

```
darker = cv2.subtract(img, 50)
```

- **Multiplication:** Scaling intensity values.

```
scaled = cv2.multiply(img, 1.5)
```

- **Transformation:** Matrix multiplication, dot product, cross product, affine transformations.

👉 These are useful in image enhancement, blending, and transformation tasks.

3. Utility Functions

(a) Reading/Writing Files

- `cv2.imread()` → Load image.
- `cv2.imwrite()` → Save image.

```
img = cv2.imread("input.jpg")
cv2.imwrite("output.jpg", img)
```

(b) Type Conversions

- Images can be converted from one data type to another.
- Example: Converting from `uint8` (0–255) to `float32` for mathematical operations.

```
float_img = img.astype('float32') / 255.0
```

4. Array Operations

(a) Reshaping

- Change the shape of a matrix without changing its data.

```
reshaped = img.reshape((-1,3))  # Flatten image into 2D
array
```

(b) Splitting & Merging

- Splitting: Separate image channels (B, G, R).
- Merging: Combine separate channels into one image.

```
b, g, r = cv2.split(img)       # Split channels
merged = cv2.merge([b, g, r]) # Merge back
```

(c) Sub-matrices & ROI (Region of Interest)

- Extract parts of an image (cropping).

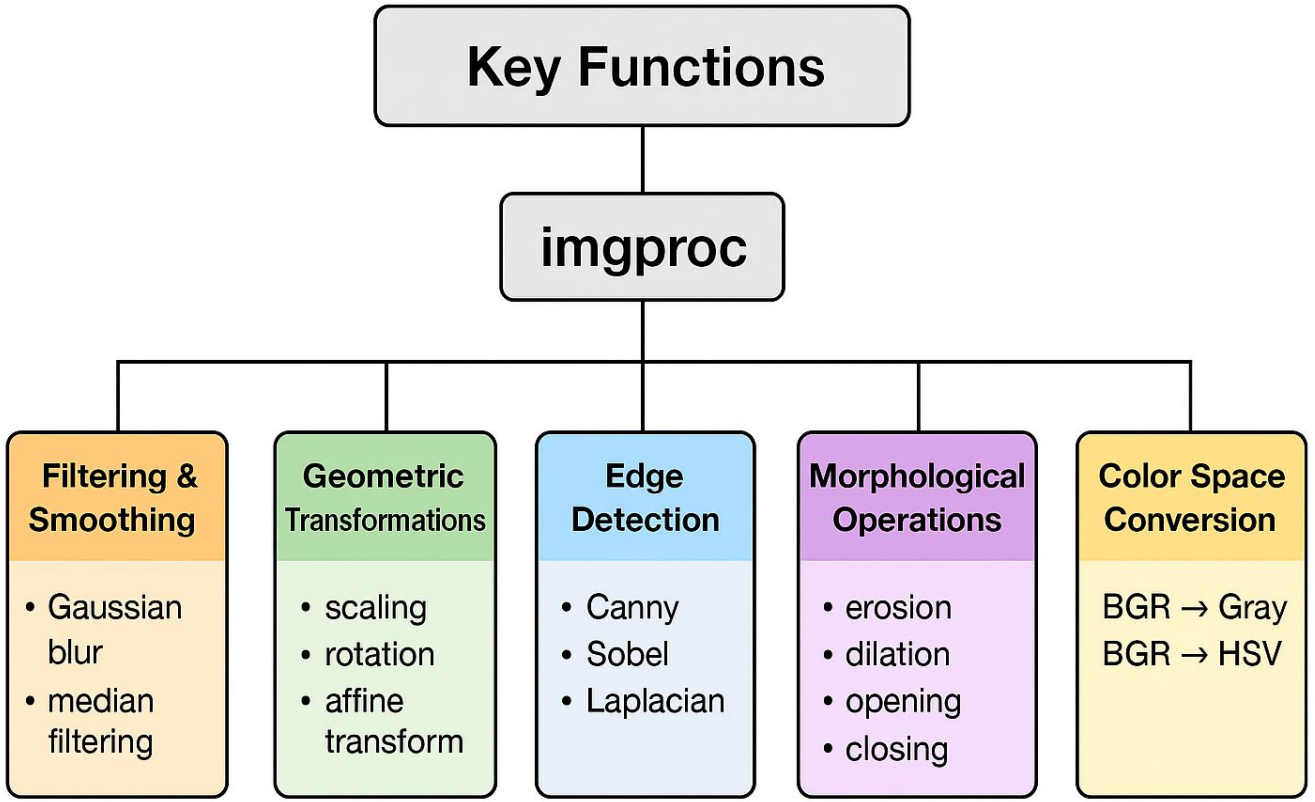
```
roi = img[50:200, 100:300]   # Crop region
```

The **Core module** is the **foundation of OpenCV**, providing:

1. **Data structures** (`Mat`, Points, Scalars, Sizes).
2. **Mathematical operations** (addition, subtraction, multiplication).
3. **Utility functions** (file I/O, type conversions).
4. **Array operations** (reshaping, splitting, merging, ROI).

👉 Without the core module, **higher-level modules like `imgproc`, `ml`, and `video` cannot function**, making it the **backbone of OpenCV**.

Key Functions in OpenCV Image Processing (imgproc Module)



1. Filtering & Smoothing

Filtering is used to **remove noise** and smooth an image for better processing.

- **Gaussian Blur:**
  - Reduces noise and detail using a Gaussian kernel.
  - Useful before edge detection.
- `blurred = cv2.GaussianBlur(img, (5,5), 0)`
- **Median Filtering:**
  - Replaces each pixel with the median of its neighborhood.
  - Very effective in removing **salt-and-pepper noise**.
- `median = cv2.medianBlur(img, 5)`

👉 **Applications:** Noise reduction in medical/scanned images.

2. Geometric Transformations

Transformations change the **position, shape, or orientation** of an image.

- **Scaling (Resizing):**
- `resized = cv2.resize(img, (200,200))`
- **Rotation:**
- `M = cv2.getRotationMatrix2D((cx, cy), 45, 1.0)`
- `rotated = cv2.warpAffine(img, M, (w, h))`
- **Affine Transformation:**
    - Uses a transformation matrix to shift, rotate, or shear the image.
- `pts1 = np.float32([[50,50],[200,50],[50,200]])`
- `pts2 = np.float32([[10,100],[200,50],[100,250]])`
- `M = cv2.getAffineTransform(pts1, pts2)`
- `affine = cv2.warpAffine(img, M, (cols, rows))`

👉 **Applications:** Image registration, alignment, zooming, and rotation.

3. Edge Detection

Detecting boundaries of objects in an image.

- **Canny Edge Detector:**
  - Multi-stage algorithm for detecting sharp edges.
- `edges = cv2.Canny(img, 100, 200)`
- **Sobel Operator:**
  - Finds gradients in **horizontal/vertical** directions.
- `sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)`
- `sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)`
- **Laplacian Operator:**
  - Detects **edges in all directions** (second derivative).
- `laplacian = cv2.Laplacian(img, cv2.CV_64F)`

👉 **Applications:** Object detection, medical imaging (tumor boundary detection).

4.Morphological Operations

These are used for **shape-based processing** of binary images.

- **Erosion:** Removes small white noises, shrinks white regions.
- `kernel = np.ones((5,5), np.uint8)`
- `erosion = cv2.erode(img, kernel, iterations=1)`
- **Dilation:** Expands white regions, fills gaps.
- `dilation = cv2.dilate(img, kernel, iterations=1)`
- **Opening (Erosion → Dilation):** Removes noise but preserves main object.
- `opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`
- **Closing (Dilation → Erosion):** Fills small holes inside the object.
- `closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`

👉 **Applications:** Object extraction, text recognition preprocessing, medical image cleaning.

5. Color Space Conversion

Used for **changing image color models** depending on the application.

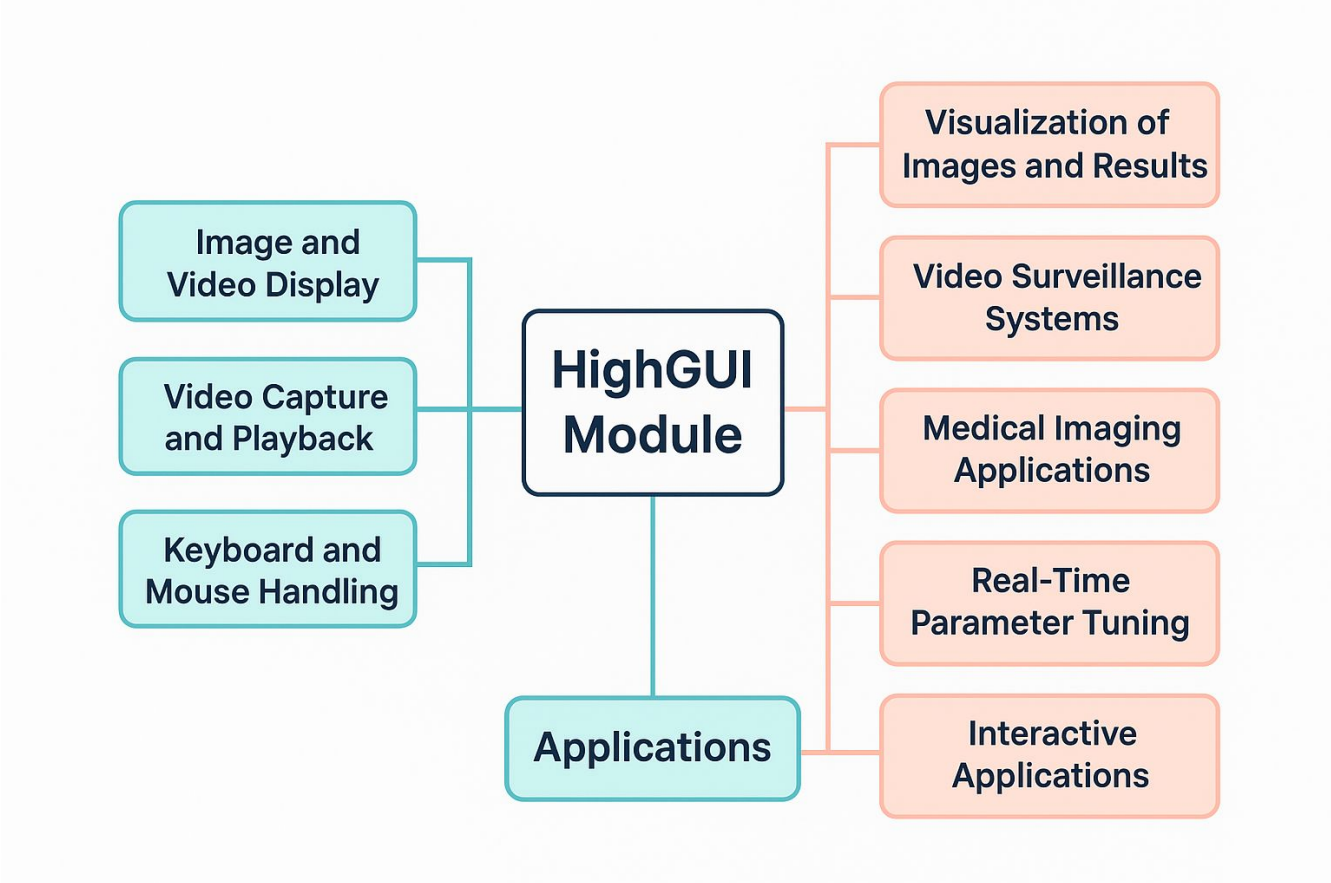
- **BGR → Grayscale:**
- `gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
- **BGR → HSV (Hue, Saturation, Value):**
- `hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)`
- **BGR → LAB / YCrCb:** For better illumination correction.

👉 **Applications:**

- Grayscale → used in edge detection, thresholding.
- HSV → useful for color-based object detection (e.g., tracking a red ball).

HighGUI Module in OpenCV

The **HighGUI module** in OpenCV provides a **simple graphical user interface (GUI)** for displaying images, handling video streams, and capturing user interactions. It acts as a **bridge between OpenCV algorithms and users** by allowing visualization and interaction.



Features of HighGUI Module

1.Image and Video Display

1. Display images using `cv2.imshow()`
2. Supports windows for showing real-time video streams

2.`cv2.imshow("Image Window", img)`

2.

3.Video Capture and Playback

2. Read video from files (`.mp4`, `.avi`) or webcams.
3. Functions: `cv2.VideoCapture()`, `cv2.VideoWriter()`

4.`cap = cv2.VideoCapture(0)  # Open webcam`

3.Keyboard and Mouse Handling

3. `cv2.waitKey()` → waits for a key press.
4. `cv2.setMouseCallback()` → detects mouse clicks/positions on an image window.

4.

5.GUI Elements

3. Trackbars (sliders) for adjusting parameters dynamically.
4. Used for real-time tuning of algorithms (e.g., thresholds in Canny).

6.`cv2.createTrackbar("Threshold", "Window", 0, 255, callback)`

5.

6.Image I/O (Input/Output)

5. `cv2.imread()` → load image.
6. `cv2.imwrite()` → save image in multiple formats (PNG, JPG, BMP, TIFF).

6.Window Management

6. `cv2.namedWindow()`,`cv2.destroyWindow()` for creating and managing windows.
7. Different window modes: autosize, resizable.

💡 Applications of HighGUI Module

1.Visualization of Images and Results

1. Displaying intermediate outputs of image processing (edges, filtered images).

2.Video Surveillance Systems

1. Display live webcam feeds with motion detection.

3.Medical Imaging Applications

1. Viewing CT, MRI, and X-ray images with zoom and brightness adjustments.

4.Real-Time Parameter Tuning

1. Using trackbars to fine-tune threshold values, filter sizes, or color ranges.

5.Interactive Applications

1. Drawing shapes, annotating objects, and cropping using mouse interactions.

6.Testing & Debugging Computer Vision Algorithms

1. Quickly visualize steps in image enhancement, segmentation, or object detection.

ML Module in OpenCV

The **ml module** in OpenCV provides tools and algorithms for **supervised and unsupervised machine learning**. It is used for **training, testing, and applying models** to solve classification, regression, and clustering problems in computer vision.

✨ Key Features of ML Module

1. **Classification algorithms**: k-Nearest Neighbors (kNN), Decision Trees, Random Forest, SVM.
2. **Regression algorithms**: Linear Regression, Logistic Regression.
3. **Clustering**: k-Means clustering.
4. **Model handling**: Training, prediction, saving, and loading models.
5. **Integration**: Works directly with OpenCV's data structures (`cv::Mat`) for easy use in vision tasks.

Example: Support Vector Machine (SVM) for Image Classification

Steps:

1. **Data Preparation** – Collect training images (e.g., digits dataset, face dataset).
2. **Feature Extraction** – Extract features such as pixel values, HOG (Histogram of Oriented Gradients), or SIFT.
3. **Training** – Train the SVM model using labeled data.
4. **Prediction** – Use the trained model to classify new/unseen images.
5. **Evaluation** – Measure accuracy, precision, recall.

Python Example Code (Digit Classification with SVM)

```python
import cv2
import numpy as np

# Load dataset (digits in OpenCV)
digits = cv2.imread('digits.png', cv2.IMREAD_GRAYSCALE)
cells = [np.hsplit(row,100) for row in np.vsplit(digits,50)]
x = np.array(cells)

# Prepare training data
train = x[:,:50].reshape(-1,400).astype(np.float32)  # 50 samples
test  = x[:,50:100].reshape(-1,400).astype(np.float32)

# Labels
k = np.arange(10)
train_labels = np.repeat(k,250)[:,np.newaxis]
test_labels = train_labels.copy()

# Train SVM
svm = cv2.ml.SVM_create()
svm.setKernel(cv2.ml.SVM_LINEAR)
svm.train(train, cv2.ml.ROW_SAMPLE, train_labels)

# Predict
result = svm.predict(test)[1]
accuracy = (result == test_labels).mean()
print("SVM Accuracy:", accuracy)
```

📊 Applications

- Handwritten digit/character recognition
- Face recognition
- Object classification (cars, humans, animals)
- Medical image classification (tumor vs. non-tumor)
- Gesture recognition

📌 Flowchart: SVM with ML Module

Image Input → Feature Extraction → Train SVM Model → Save/Load Model → Prediction on New Data → Classification Result

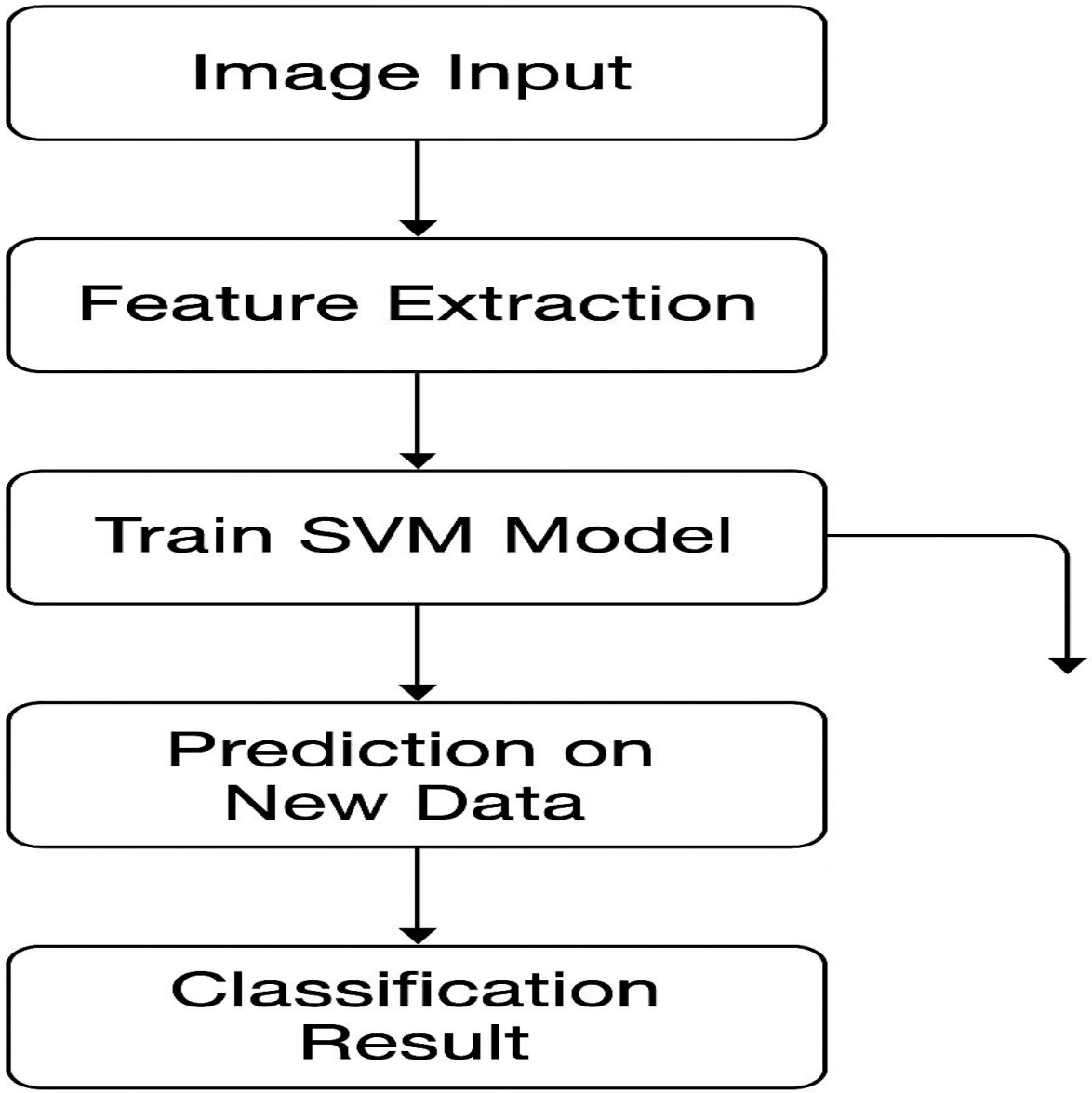
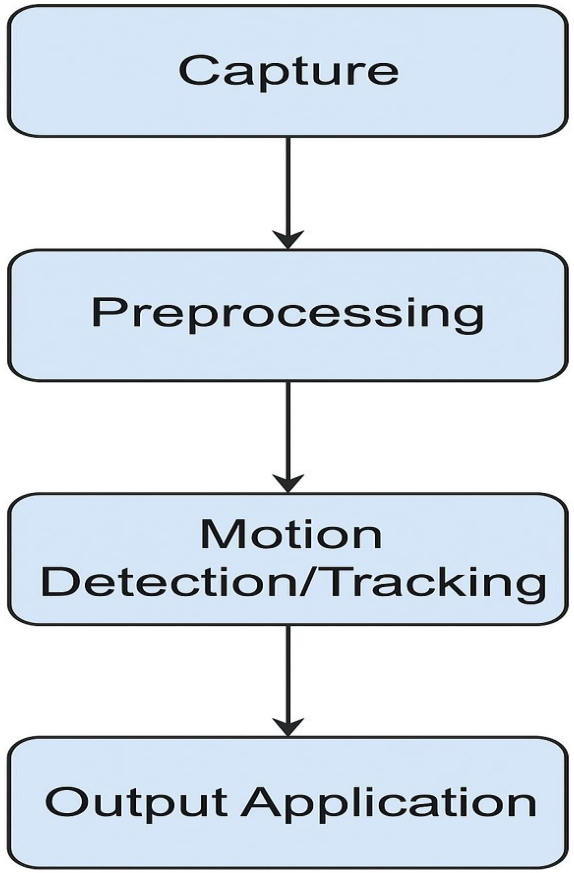


Image Input
Feature Extraction
Train SVM Model
Prediction on New Data
Classification Result

Video Module in OpenCV

The **video module** in OpenCV provides functions for **video analysis, motion tracking, background subtraction, and object tracking**. It is widely used in real-time computer vision applications.



Key Features of Video Module

1.Background Subtraction

1. Removes static background and detects moving objects.
2. Algorithms:
    1. `cv2.createBackgroundSubtractorMOG2()`
    2. `cv2.createBackgroundSubtractorKNN()`

2.`fgbg = cv2.createBackgroundSubtractorMOG2()`

3.`fgmask = fgbg.apply(frame)`

2.Optical Flow Estimation

Tracks motion between two frames by calculating pixel displacement.

Algorithms:

2. Dense Optical Flow (`cv2.calcOpticalFlowFarneback()`)
3. Sparse Optical Flow (Lucas-Kanade, `cv2.calcOpticalFlowPyrLK()`)

3.
```
p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, new_gray,
 p0, None, **lk_params)
```

3.Object Tracking

* Follows a specific object in a video sequence.
* Algorithms: KCF, CSRT, MIL, GOTURN, MOSSE trackers.

```
tracker = cv2.TrackerKCF_create()
tracker.init(frame, bbox)
```

4.Video I/O Functions

Read and write video streams.

`cv2.VideoCapture()` → load video file/webcam.

`cv2.VideoWriter()` → save processed video.

```
cap = cv2.VideoCapture(0)    # Open webcam
out = cv2.VideoWriter('output.avi', fourcc, 20.0,
(640,480))
```

5.Motion Detection

* Detects moving objects using frame differencing or background subtraction.
* Useful in surveillance systems.

6.Camera Calibration (in video streams)

* Corrects distortion and aligns video feeds for precise analysis.

eal-Time Use Cases

1.Surveillance and Security Systems

1. Motion detection, intruder detection, people counting.
2. Background subtraction to highlight moving objects.

2.Self-Driving Cars

1. Object tracking for vehicles, pedestrians, and traffic signs.
2. Optical flow for estimating vehicle movement and lane detection.

3.Sports Analytics

1. Tracking ball, players, and analyzing motion paths.

4.Gesture Recognition

1. Track hand/face movements in real time using optical flow or trackers.

5.Medical Applications

1. Motion analysis in X-ray/ultrasound video sequences.
2. Tracking movement of organs, heartbeat, or surgical instruments.

6.Human-Computer Interaction (HCI)

1. Real-time webcam-based gesture or eye-tracking.

7.Drone and Robotics Vision

1. Object tracking and obstacle avoidance using real-time video feeds.