```python
# Exno.1 â Implementation of Various Filter Techniques
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read and prepare image
img_bgr = cv2.imread(r"data\peacock.jpeg")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

# Kernel setup
kernel_size = 7
morph_kernel = np.ones((5, 5), np.uint8)

# Apply filters
avg_blur = cv2.blur(img_rgb, (kernel_size, kernel_size))
gaussian_blur = cv2.GaussianBlur(img_rgb, (kernel_size, kernel_size), 0)
median_blur = cv2.medianBlur(img_rgb, kernel_size)
bilateral_filter = cv2.bilateralFilter(img_rgb, d=9, sigmaColor=75, sigmaSpace=75)

sobel_x = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, ksize=5)
sobel_edges = np.sqrt(sobel_x**2 + sobel_y**2)
sobel_edges = cv2.normalize(sobel_edges, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

canny_edges = cv2.Canny(img_gray, 100, 200)
dilated_img = cv2.dilate(img_gray, morph_kernel, iterations=1)
eroded_img = cv2.erode(img_gray, morph_kernel, iterations=1)

# Display results
titles = [
    'Original', 'Averaging', 'Gaussian', 'Median',
    'Bilateral', 'Sobel', 'Canny', 'Dilation', 'Erosion'
]
images = [
    img_rgb, avg_blur, gaussian_blur, median_blur,
    bilateral_filter, sobel_edges, canny_edges, dilated_img, eroded_img
]

plt.figure(figsize=(15, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(images[i], cmap='gray' if len(images[i].shape) == 2 else None)
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```python
# Exno.2 â Implementation of Histogram
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read image
img = cv2.imread(r'data/peacock.jpeg')
if img is None:
    print("Image not found!")
    exit()

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 1. Grayscale Histogram
hist_gray = cv2.calcHist([gray], [0], None, [256], [0, 256])
plt.figure()
plt.title('Grayscale Histogram')
plt.xlabel('Pixel Intensity')
plt.ylabel('Number of Pixels')
plt.plot(hist_gray, color='k')
plt.xlim([0, 256])
plt.show()

# 2. Color Histogram (B, G, R)
colors = ('b', 'g', 'r')
plt.figure()
plt.title('Color Histogram')
plt.xlabel('Pixel Intensity')
plt.ylabel('Number of Pixels')
for i, col in enumerate(colors):
    hist = cv2.calcHist([img], [i], None, [256], [0, 256])
    plt.plot(hist, color=col)
plt.xlim([0, 256])
plt.show()

# 3. Histogram Equalization (Gray Image)
eq_gray = cv2.equalizeHist(gray)
hist_eq = cv2.calcHist([eq_gray], [0], None, [256], [0, 256])

cv2.imshow('Original Grayscale', gray)
cv2.imshow('Equalized Grayscale', eq_gray)

plt.figure()
plt.title('Equalized Grayscale Histogram')
plt.xlabel('Pixel Intensity')
plt.ylabel('Number of Pixels')
plt.plot(hist_eq, color='k')
plt.xlim([0, 256])
plt.show()

# 4. 2D Histogram (Hue vs Saturation)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hist_2d = cv2.calcHist([hsv], [0, 1], None, [30, 32], [0, 180, 0, 256])

plt.figure()
plt.title('2D Hue-Saturation Histogram')
plt.xlabel('Hue')
plt.ylabel('Saturation')
plt.imshow(hist_2d, interpolation='nearest')
plt.colorbar()
plt.show()

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# Exno.3 - Implementation of Various Segmentation Algorithms
import cv2
import numpy as np
import matplotlib.pyplot as plt

# --- Load Image ---
img = cv2.imread(r'data/peacock.jpeg')
if img is None:
    print("Image not found!")
    exit()

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# --- 1. Simple Thresholding ---
_, thresh_simple = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# --- 2. Adaptive Thresholding ---
thresh_adapt = cv2.adaptiveThreshold(gray, 255,
                                     cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                     cv2.THRESH_BINARY, 11, 2)

# --- 3. Otsu’s Thresholding ---
_, thresh_otsu = cv2.threshold(gray, 0, 255,
                               cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# --- 4. Watershed Segmentation ---
_, thresh_w = cv2.threshold(gray, 0, 255,
                            cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh_w, cv2.MORPH_OPEN, kernel, iterations=2)
sure_bg = cv2.dilate(opening, kernel, iterations=3)
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)
_, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0
markers = cv2.watershed(img, markers)
img_ws = img.copy()
img_ws[markers == -1] = [255, 0, 0]

# --- 5. K-means Clustering ---
Z = img.reshape((-1, 3))
Z = np.float32(Z)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 4
_, label, center = cv2.kmeans(Z, K, None, criteria, 10,
                              cv2.KMEANS_RANDOM_CENTERS)
center = np.uint8(center)
res = center[label.flatten()]
img_kmeans = res.reshape((img.shape))

# --- 6. GrabCut Segmentation ---
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
h, w = img.shape[:2]
rect = (int(w * 0.1), int(h * 0.1), int(w * 0.8), int(h * 0.8))
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
img_grabcut = img * mask2[:, :, np.newaxis]

# --- Display Results ---
titles = [
    'Original', 'Simple Threshold', 'Adaptive Threshold',
    'Otsu Threshold', 'Watershed', 'K-means', 'GrabCut'
]
images = [
    img_rgb, thresh_simple, thresh_adapt, thresh_otsu,
    cv2.cvtColor(img_ws, cv2.COLOR_BGR2RGB),
    img_kmeans, cv2.cvtColor(img_grabcut, cv2.COLOR_BGR2RGB)
]

plt.figure(figsize=(15, 8))
for i in range(len(images)):
    plt.subplot(2, 4, i + 1)
    plt.imshow(images[i], cmap='gray' if i in [1, 2, 3] else None)
    plt.title(titles[i])
    plt.axis('off')

plt.tight_layout()
plt.show()
```

```
# Exno.4 Program to implement Object Labelling
import cv2
import numpy as np

def detect_and_label_objects(image_path):
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Image not found")
        return

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define color ranges (HSV) and corresponding labels
    color_ranges = [
        ((0, 100, 100), (10, 255, 255), "Red"),
        ((25, 100, 100), (35, 255, 255), "Yellow"),
        ((100, 100, 100), (120, 255, 255), "Blue")
    ]

    for lower, upper, label in color_ranges:
        mask = cv2.inRange(hsv, np.array(lower), np.array(upper))
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        for contour in contours:
            if cv2.contourArea(contour) < 100:
                continue
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(image, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 
0), 2)

            binary_label = np.zeros_like(image)
            cv2.rectangle(binary_label, (x, y), (x + w, y + h), (255, 255, 255), -1)
            cv2.imshow(f"Binary label - {label}", binary_label)

    cv2.imshow("Labeled Image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Run program
detect_and_label_objects(r"data/peacock.jpeg")
```

```python
# Exno.5 Implementation of Face Recognition System
import cv2
#model and config links
# https://github.com/opencv/opencv/blob/master/samples/dnn/face_detector/deploy.prototxt
#
https://github.com/opencv/opencv_3rdparty/blob/dnn_samples_face_detector_20170830/res10_300x300_ssd_iter_140000.caffemodel


def face_detection_dnn(image_path):
    model = r"data\res10_300x300_ssd_iter_140000.caffemodel"
    config = r"data\deploy.prototxt"

    net = cv2.dnn.readNetFromCaffe(config, model)
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Image not found")
        return

    h, w = image.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
                                  (300, 300), (104.0, 177.0, 123.0))
    net.setInput(blob)
    detections = net.forward()

    count = 0
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.5:
            box = detections[0, 0, i, 3:7] * [w, h, w, h]
            x1, y1, x2, y2 = box.astype("int")
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(image, "Face", (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
            count += 1

    print(f"Faces detected: {count}")
    cv2.putText(image, f"Faces: {count}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    cv2.imshow("Face Detection (DNN)", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Run
face_detection_dnn(r"data/face.jpeg")
```

```python
# Exno.6 License Plate Identification (Minimal)
import cv2
import numpy as np
import pytesseract
import os

# --- Paths ---
pytesseract.pytesseract.tesseract_cmd = r"D:\Softwares\tessaract\tesseract.exe"
cascade_path = r"data\haarcascade_russian_plate_number.xml"

if not os.path.exists(cascade_path):
    raise FileNotFoundError("Haarcascade XML file not found!")

# Load Cascade
plate_cascade = cv2.CascadeClassifier(cascade_path)

# Minimal State Dictionary
states = {
    "TN": "Tamil Nadu"
}

def extract_number_plate(img_path):
    img = cv2.imread(img_path)
    if img is None:
        print("â Error: Image not found.")
        return

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plates = plate_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in plates:
        plate = img[y:y+h, x:x+w]
        plate_gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
        _, plate_bin = cv2.threshold(plate_gray, 127, 255, cv2.THRESH_BINARY)

        # OCR using Tesseract
        text = pytesseract.image_to_string(plate_bin, config='--psm 8')
        number = ''.join(e for e in text if e.isalnum()).upper()
        state_code = number[:2] if len(number) >= 2 else "NA"
        state_name = states.get(state_code, "Unknown")

        print(f"\nDetected Number: {number}")
        print(f"State: {state_name}")

        # Draw on image
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
        cv2.putText(img, number, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255,255,255),
2)
        cv2.putText(img, state_name, (x, y+h+25), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0), 2)

        # Save & Display
        cv2.imwrite("Detected_Plate.png", plate)
        cv2.imwrite("Detected_Image.png", img)
        cv2.imshow("Detected Plate", plate)
        cv2.imshow("Full Image", img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Run
extract_number_plate(r"data\car.jpg")
```

```python
# Exno.7 Medical Image Processing (No Functions)
import cv2
import matplotlib.pyplot as plt

# --- Load image ---
file_path = r'data\brain.jpeg'
img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError(f"Cannot load image from {file_path}")

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

# --- Apply Otsu's Thresholding ---
_, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# --- Find and draw contours ---
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contoured_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.drawContours(contoured_img, contours, -1, (0, 255, 0), 2)

# --- Display images ---
images = [img, blurred, thresh, contoured_img]
titles = ["Original Image", "Blurred Image", "Thresholded Image", "Contours"]

import matplotlib.pyplot as plt
plt.figure(figsize=(15,5))
for i, (image, title) in enumerate(zip(images, titles)):
    plt.subplot(1, len(images), i+1)
    plt.imshow(image, cmap='gray')
    plt.title(title)
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```python
import cv2
import numpy as np

# Load image in grayscale
img = cv2.imread('data\star.jpeg', cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError("Cannot load image 'star.jpg'")

# Edge detection
edges = cv2.Canny(img, 50, 150)

# Corner detection
corners_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
corners = cv2.goodFeaturesToTrack(img, 200, 0.01, 10)
if corners is not None:
    for x, y in np.intp(corners).reshape(-1, 2):
        cv2.circle(corners_img, (x, y), 3, (0, 255, 0), -1)

# Line detection
lines_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
lines = cv2.HoughLines(edges, 1, np.pi/180, 150)
if lines is not None:
    for rho, theta in lines[:, 0]:
        a, b = np.cos(theta), np.sin(theta)
        x0, y0 = a*rho, b*rho
        x1, y1 = int(x0 + 1000*(-b)), int(y0 + 1000*(a))
        x2, y2 = int(x0 - 1000*(-b)), int(y0 - 1000*(a))
        cv2.line(lines_img, (x1, y1), (x2, y2), (0, 0, 255), 1)

# Display results
cv2.imshow("Original", img)
cv2.imshow("Edges", edges)
cv2.imshow("Corners", corners_img)
cv2.imshow("Lines", lines_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

test_image = cv2.imread(r'data\test\face.jpeg')
known_images = [cv2.imread(r'data\known\face.jpeg') for _ in range(1)]


if test_image is None or any(img is None for img in known_images):
    raise FileNotFoundError("Check that all images exist in the folder.")

# Calculate color histogram for an image
def calc_hist(img):
    hist = cv2.calcHist([img], [0, 1, 2], None, [8, 8, 8],
                        [0, 256, 0, 256, 0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist

# Recognize face by comparing histograms
test_hist = calc_hist(test_image)
distances = [cv2.compareHist(test_hist, calc_hist(kimg), cv2.HISTCMP_BHATTACHARYYA)
             for kimg in known_images]

recognized_index = np.argmin(distances)
if distances[recognized_index] <= 0.5:
    print(f"Face recognized as person {recognized_index + 1}")
else:
    print("Face not recognized")

# Display images
plt.figure(figsize=(10, 5))
plt.subplot(1, len(known_images) + 1, 1)
plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
plt.title('Test Image')

for i, kimg in enumerate(known_images):
    plt.subplot(1, len(known_images) + 1, i + 2)
    plt.imshow(cv2.cvtColor(kimg, cv2.COLOR_BGR2RGB))
    plt.title(f'Known Image {i + 1}')

plt.tight_layout()
plt.show()
```

```python
import cv2
from matplotlib import pyplot as plt

# ---- Paths ----
authorized_face_path = r'data\known\face.jpeg'   # Authorized face
test_image_path = r'data\test\face.jpeg'        # Test image

# ---- Load images ----
authorized_face_img = cv2.imread(authorized_face_path, cv2.IMREAD_GRAYSCALE)
test_img = cv2.imread(test_image_path)

if authorized_face_img is None or test_img is None:
    raise FileNotFoundError("Check that image paths are correct.")

# ---- Initialize ORB ----
orb = cv2.ORB_create()

# ---- Detect features on authorized face ----
kp1, des1 = orb.detectAndCompute(authorized_face_img, None)
if des1 is None:
    raise ValueError("No features detected in authorized face image.")

# Show authorized face with keypoints
auth_kp_img = cv2.drawKeypoints(authorized_face_img, kp1, None, color=(0, 255, 0))
plt.figure(figsize=(6, 5))
plt.title('Authorized Face Keypoints')
plt.imshow(cv2.cvtColor(auth_kp_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

# ---- Convert test image to grayscale and detect faces ----
gray_test = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray_test, 1.1, 4)

authorized = False

for (x, y, w, h) in faces:
    face_roi = gray_test[y:y+h, x:x+w]
    kp2, des2 = orb.detectAndCompute(face_roi, None)

    if des2 is not None:
        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        matches = bf.match(des1, des2)
        matches = sorted(matches, key=lambda m: m.distance)

        # Show top 10 matches
        match_img = cv2.drawMatches(authorized_face_img, kp1, face_roi, kp2, matches[:10],
None, flags=2)
        plt.figure(figsize=(8, 6))
        plt.title('Top 10 Matches')
        plt.imshow(cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()

        # Check good matches
        good_matches = [m for m in matches if m.distance < 50]
        if len(good_matches) > 10:
            authorized = True

# ---- Output Result ----
print("Authorized" if authorized else "Unauthorized")
```

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

# ---- Paths ----
left_image_path = r"data\face.jpeg"
right_image_path = r"data\car.jpeg"

# ---- Load images in grayscale ----
left_img = cv2.imread(left_image_path, cv2.IMREAD_GRAYSCALE)
right_img = cv2.imread(right_image_path, cv2.IMREAD_GRAYSCALE)

# Ensure both images are loaded
if left_img is None or right_img is None:
    raise FileNotFoundError("Check image paths.")

# Resize right image if needed
if left_img.shape != right_img.shape:
    right_img = cv2.resize(right_img, (left_img.shape[1], left_img.shape[0]))

# ---- StereoSGBM parameters ----
stereo = cv2.StereoSGBM_create(
    minDisparity=0,
    numDisparities=16*6,  # must be divisible by 16
    blockSize=7,
    P1=8*3*7**2,
    P2=32*3*7**2,
    disp12MaxDiff=1,
    uniquenessRatio=10,
    speckleWindowSize=100,
    speckleRange=32
)

# ---- Compute disparity map ----
disparity = stereo.compute(left_img, right_img).astype(np.float32) / 16.0

# Normalize for visualization
disp_norm = cv2.normalize(disparity, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# ---- Display results ----
plt.figure(figsize=(12, 5))
plt.subplot(1, 3, 1)
plt.title("Left Image")
plt.imshow(left_img, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Right Image")
plt.imshow(right_img, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Disparity Map")
plt.imshow(disp_norm, cmap='plasma')
plt.axis('off')

plt.tight_layout()
plt.show()
```

```
import cv2 as cv
import matplotlib.pyplot as plt

# ---- Load pre-trained TensorFlow pose model ----
net = cv.dnn.readNetFromTensorflow(r"data\graph_opt.pb")

# ---- Parameters ----
inWidth = 368
inHeight = 368
thr = 0.2  # confidence threshold

# ---- Body parts and skeleton connections ----
BODY_PARTS = {
    "Nose": 0, "Neck": 1,
    "Right Shoulder": 2, "Right Elbow": 3, "Right Wrist": 4,
    "Left Shoulder": 5, "Left Elbow": 6, "Left Wrist": 7,
    "Right Hip": 8, "Right Knee": 9, "Right Ankle": 10,
    "Left Hip": 11, "Left Knee": 12, "Left Ankle": 13,
    "Right Eye": 14, "Left Eye": 15,
    "Right Ear": 16, "Left Ear": 17
}

POSE_PAIRS = [
    ("Neck", "Right Shoulder"), ("Neck", "Left Shoulder"),
    ("Right Shoulder", "Right Elbow"), ("Right Elbow", "Right Wrist"),
    ("Left Shoulder", "Left Elbow"), ("Left Elbow", "Left Wrist"),
    ("Neck", "Right Hip"), ("Right Hip", "Right Knee"), ("Right Knee", "Right Ankle"),
    ("Neck", "Left Hip"), ("Left Hip", "Left Knee"), ("Left Knee", "Left Ankle"),
    ("Neck", "Nose"),
    ("Nose", "Right Eye"), ("Right Eye", "Right Ear"),
    ("Nose", "Left Eye"), ("Left Eye", "Left Ear")
]

# ---- Load image ----
frame = cv.imread(r"data\human.jpeg")
frameHeight, frameWidth = frame.shape[:2]

# ---- Prepare input for network ----
blob = cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight),
                            (127.5, 127.5, 127.5), swapRB=True, crop=False)
net.setInput(blob)
out = net.forward()[:, :len(BODY_PARTS), :, :]

# ---- Detect keypoints ----
points = []
for i in range(len(BODY_PARTS)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x = int((frameWidth * point[0]) / out.shape[3])
    y = int((frameHeight * point[1]) / out.shape[2])
    points.append((x, y) if conf > thr else None)

# ---- Draw skeleton ----
for pair in POSE_PAIRS:
    partFrom, partTo = pair
    idFrom = BODY_PARTS[partFrom]
    idTo = BODY_PARTS[partTo]
    if points[idFrom] and points[idTo]:
        cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
        cv.circle(frame, points[idFrom], 3, (0, 0, 255), -1)
        cv.circle(frame, points[idTo], 3, (0, 0, 255), -1)

# ---- Display result ----
plt.figure(figsize=(10, 6))
plt.imshow(cv.cvtColor(frame, cv.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Human Pose Estimation")
plt.show()
```

```
import cv2
import numpy as np

# ---- Paths ----
weights = r"data\yolov4.weights"
cfg     = r"data\yolov4.cfg"
names   = r"data\coco.names"
video   = r"data\traffic.mp4"
# ---- Load YOLO ----
net = cv2.dnn.readNet(weights, cfg)
layers = [net.getLayerNames()[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
classes = [line.strip() for line in open(names)]

cap = cv2.VideoCapture(video)
count = 0

while True:
    ret, frame = cap.read()
    if not ret: break
    h, w = frame.shape[:2]

    blob = cv2.dnn.blobFromImage(frame, 1/255, (608,608), swapRB=True, crop=False)
    net.setInput(blob)
    outs = net.forward(layers)

    boxes, confs, ids = [], [], []
    for out in outs:
        for det in out:
            scores = det[5:]
            cid = np.argmax(scores)
            conf = scores[cid]
            if conf > 0.3 and classes[cid] in ["car","truck","bus","motorbike"]:
                cx, cy = int(det[0]*w), int(det[1]*h)
                bw, bh = int(det[2]*w), int(det[3]*h)
                x, y = max(0,cx-bw//2), max(0,cy-bh//2)
                boxes.append([x,y,bw,bh]); confs.append(float(conf)); ids.append(cid)

    for i in cv2.dnn.NMSBoxes(boxes, confs, 0.3, 0.4).flatten():
        x,y,bw,bh = boxes[i]
        cv2.rectangle(frame,(x,y),(x+bw,y+bh),(0,255,0),2)
        cv2.putText(frame, classes[ids[i]], (x,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(0,255,0),2)
        if y > h//2: count += 1

    cv2.putText(frame,f'Count: {count}',(10,30),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv2.imshow("Traffic", frame)
    if cv2.waitKey(10) & 0xFF == ord('q'): break

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

```python
import numpy as np
import cv2
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import os

# ----- Paths -----
MODEL_PATH = r'data/action_recognition_model.h5'
TEST_IMAGE = r'data//run.jpg'
action_classes = ['walking', 'running', 'jumping', 'standing', 'sitting', 'falling',
'other']

# Load model
if not os.path.exists(MODEL_PATH):
    raise FileNotFoundError(f"Model not found: {MODEL_PATH}")
model = load_model(MODEL_PATH)

# Determine expected input size & channels
_, H, W, C = model.input_shape

# Load and preprocess image
img = cv2.imread(TEST_IMAGE)
if C == 1:
    img_proc = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_proc = cv2.resize(img_proc, (W, H))[:, :, np.newaxis]
else:
    img_proc = cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), (W, H))

x = np.expand_dims(img_proc.astype('float32') / 255.0, axis=0)

# Predict action
preds = model.predict(x)[0]
top_label = action_classes[np.argmax(preds)]

# Display results
print(f"Predicted Action: {top_label} ({np.max(preds)*100:.1f}%)")
cv2.putText(img, top_label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```python
import cv2
import numpy as np

# --- Load image ---
image_path = r'data\road.jpeg'
image = cv2.imread(image_path)
if image is None:
    raise ValueError("Image not found or path is incorrect")

# --- Grayscale and blur ---
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# --- Edge detection ---
edges = cv2.Canny(blurred, 50, 150)

# --- Region of interest mask ---
height, width = edges.shape
mask = np.zeros_like(edges)
roi = np.array([[
    (0, height),
    (width*0.1, height*0.5),
    (width*0.9, height*0.5),
    (width, height)
]], dtype=np.int32)
cv2.fillPoly(mask, roi, 255)
masked_edges = cv2.bitwise_and(edges, mask)

# --- Hough Line Transform ---
lines = cv2.HoughLinesP(masked_edges, 1, np.pi/180, 30, minLineLength=50, maxLineGap=30)

line_image = np.copy(image)
left_lines, right_lines = [], []

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        if x2 != x1:
            slope = (y2 - y1) / (x2 - x1)
            if slope < -0.2:
                left_lines.append(line)
            elif slope > 0.2:
                right_lines.append(line)

# --- Draw lane lines ---
for line in left_lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(line_image, (x1, y1), (x2, y2), (255, 0, 0), 2)
for line in right_lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

# --- Overlay on original image ---
result = cv2.addWeighted(image, 0.8, line_image, 1, 0)

# --- Display ---
cv2.imshow('Original', image)
cv2.imshow('Road Margins', result)
cv2.waitKey(0)
cv2.destroyAllWindows()

# --- Save result ---
cv2.imwrite('road_with_margins.jpg', result)
```