Aim:

To perform simple image processing using openCV

Algorithm:

1. Read and Display the Image: Load an image from a specified path and display it.
2. Convert to Grayscale and Display: Convert the original image to grayscale and display the grayscale image.
3. Resize the Image and Display: Resize the original image to a specified size and display the resized image.
4. Blur the Image and Display: Apply Gaussian blur to the original image and display the blurred image.
5. Edge Detection, Drawing Shapes, and Display: Apply Canny edge detection to the grayscale image and display the result. Draw a rectangle and a line on the original image, then display the modified image.

Program:

```
import cv2
from google.colab.patches import cv2_imshow
# Reading an image
Image=cv2.imread('set image path&#39;)
# Displaying the image
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Display the grayscale image
cv2_imshow(gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
resized_image = cv2.resize(image,(200,200))
# Display the resized image
cv2_imshow(resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
#Blurred image
blurred_image = cv2.GaussianBlur(image, (15, 15), 0)
cv2_imshow(blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
# Apply Canny edge detection
edges = cv2.Canny(gray_image, 100, 200)
cv2_imshow(edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
#draw rectangle
cv2.rectangle(image, (50,50), (300,300), (255, 0, 0), 2)
#Draw line
cv2.line(image, (60,60), (300,300), (0, 0, 255), 2)
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result:

Thus successfully performed simple image processing using openCV

Aim:

To write a python program to Create a 3D Model from 2D Images

Algorithm:

Here is a step-by-step explanation of what the code does:

1. Import Libraries: Import necessary libraries such as OpenCV, NumPy, and Matplotlib.

2. Load Images: Load two grayscale images (`img1` and `img2`) from specified paths.

3. Initialize SIFT Detector: Create a SIFT (Scale-Invariant Feature Transform) detector.

4. Detect Keypoints and Descriptors: Detect keypoints and compute descriptors for both images using SIFT.

5. Match Descriptors Using FLANN Matcher: Use FLANN (Fast Library for Approximate Nearest Neighbors) to find matches between descriptors from the two images.

6. Lowe's Ratio Test: Apply Lowe's ratio test to filter out good matches from the list of all matches.

7. Draw Matches: Draw lines between matching keypoints from the two images and display the result using Matplotlib.

8. Extract Locations of Good Matches: Extract the locations of the good matches to use for further processing.

9. Find Essential Matrix: Compute the essential matrix using the matched keypoints to understand the relative position between the two camera views.

10. Recover Pose: Recover the relative camera rotation and translation from the essential matrix.

11. Define a Function to Shift Image Based on Depth

- Convert the base image to RGBA and depth image to grayscale.
- Calculate the shift amount for each pixel based on its depth value.
- Create a new image with the pixels shifted according to their depth values.

12. Apply the Shift Function: Apply the `shift_image` function to the images and display the resulting shifted image using OpenCV.

Program:

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

# Load images
img1 = cv2.imread('/content/cube1.jpeg', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('/content/cube2.jpeg', cv2.IMREAD_GRAYSCALE)

# Initialize the SIFT detector
sift = cv2.SIFT_create()

# Detect keypoints and descriptors
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# Match descriptors using FLANN matcher
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)  # or pass empty dictionary
flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1, des2, k=2)

# Store all the good matches as per Lowe's ratio test.
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good_matches.append(m)

# Draw matches
```

```
img_matches = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img_matches)
plt.show()

# Extract location of good matches
pts1 = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1,1,2)
pts2 = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1,1,2)
# Find the essential matrix

E, mask = cv2.findEssentialMat(pts1, pts2, focal=1.0, pp=(0., 0.), method=cv2.RANSAC,
prob=0.999, threshold=1.0)

# Recover pose
_, R, t, mask = cv2
```

```
from PIL import Image
import numpy as np
import cv2

from google.colab.patches import cv2_imshow
def shift_image(img, depth_img, shift_amount=10):
    # Ensure base image has alpha
    img = img.convert("RGBA")
    data = np.array(img)

    # Ensure depth image is grayscale (for single value)
    depth_img = depth_img.convert("L")
    depth_data = np.array(depth_img)
    deltas = ((depth_data / 255.0) * float(shift_amount)).astype(int)

    # This creates the transparent resulting image.
    # For now, we're dealing with pixel data.
    shifted_data = np.zeros_like(data)
    height, width, _ = data.shape
    for y, row in enumerate(deltas):
        for x, dx in enumerate(row):
            if x + dx < width and x + dx >= 0:
                shifted_data[y, x + dx] = data[y, x]

    # Convert the pixel data to an image.
    shifted_image = Image.fromarray(shifted_data.astype(np.uint8))
    return shifted_image

img = Image.open("/content/cube1.jpeg")
depth_img = Image.open("/content/cube2.jpeg")
shifted_img = shift_image(img, depth_img, shift_amount=10)
#shifted_img.show()
shifted_img_array = np.array(shifted_img)
cv2_imshow(shifted_img_array)
cv2.waitKey(0)  # Wait for a key press
cv2.destroyAllWindows()  # Close the image window
```

Result:

Thus successfully Created a 3D Model from 2D Images

Aim:

To write a python program to Build your own Vehicle Detection Model

Algorithm:

Import Libraries: Import necessary libraries, including OpenCV and time.

Load Pre-Trained Car Classifier: Load the pre-trained Haar cascade classifier for car detection from a specified XML file.

Initialize Video Capture: Open the video file for reading frames.

Loop Through Video Frames:

- Introduce a small delay to control the frame rate.
- Read frames from the video one by one.
- Convert each frame to grayscale.

Detect Cars: Use the car classifier to detect cars in each frame.

Draw Bounding Boxes: Draw rectangles around the detected cars in each frame.

Display the Frame: Show the frame with the detected cars using `cv2_imshow`.

Exit Condition: Break the loop if the Enter key (ASCII code 13) is pressed.

Release Resources: Release the video capture object and close all OpenCV windows.

Program:

```
import cv2
import time
from google.colab.patches import cv2_imshow

# Load the pre-trained car classifier
car_classifier = cv2.CascadeClassifier('/content/haarcascade_car.xml')

# Initialize video capture for the video file
cap = cv2.VideoCapture('/content/cars.avi')

while cap.isOpened():
    time.sleep(0.05)
```

```
    # Read frame-by-frame
    ret, frame = cap.read()
    if not ret:
        break  # Exit the loop if no frame is read (end of video or error)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect cars in the frame
    cars = car_classifier.detectMultiScale(gray, 1.4, 2)

    # Draw bounding boxes around detected cars
    for (x, y, w, h) in cars:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)

    # Display the frame with detected cars
    cv2_imshow(frame)

    # Break the loop if the Enter key (13) is pressed
    if cv2.waitKey(1) & 0xFF == 13:
        break

# Release the video capture object and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Result:

Thus successfully built Build Own Vehicle Detection Model

Aim:

To write a python program to perform Contour based Segmentation

Algorithm:

Load the Image: Read the image from a specified file path.

Convert to Grayscale: Convert the image to a grayscale format.

Apply Gaussian Blur: Apply Gaussian blur to the grayscale image to reduce noise and improve edge detection.

Perform Edge Detection: Use the Canny edge detector to find edges in the blurred image and display the result.

Morphological Operations: Apply morphological operations to close gaps in the detected edges using a rectangular kernel.

Find Contours: Detect contours in the processed image and filter out small contours to reduce noise.

Draw Bounding Boxes: For each significant contour, get its bounding box, draw it on the original image, and label it as 'Vehicle.'

Show Result: Display the image with bounding boxes and labels, then wait for a key press before closing all windows.

Program:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Load the image
image = cv2.imread('/content/car.jpg')

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

# Apply Gaussian blur to the grayscale image
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
# Perform edge detection using Canny
edges = cv2.Canny(blurred, 50, 150)
cv2_imshow(edges)

# Perform morphological operations to close gaps in edges
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
closed = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

# Find contours in the closed edges
contours, _ = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

for contour in contours:
    # Ignore small contours to reduce noise
    if cv2.contourArea(contour) < 500:
        continue

    # Get the bounding box for each contour
    (x, y, w, h) = cv2.boundingRect(contour)
    # Draw the bounding box on the original image
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.putText(image, 'Vehicle', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
0), 2)
# Show the result
cv2_imshow (image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result:

Thus successfully performed Contour based Segmentation

Aim:

To write a python program to Implement of Shape Detection using Hough Transform

Algorithm:

Load and Display the Image: Load the image from the specified path, check for successful loading, and display the original image.

Convert to Grayscale and Blur: Convert the image to grayscale and apply median blur to reduce noise.

Edge Detection: Detect edges in the image using the Canny edge detector.

Hough Line Transform: Detect and draw lines using the Hough Line Transform.

Hough Circle Transform: Detect and draw circles using the Hough Circle Transform.

Contour Detection and Shape Approximation: Find contours, approximate their shapes, and draw contours around detected shapes, distinguishing between triangles, rectangles/squares, and circles/ellipses.

Display the Result: Show the final image with shapes highlighted, then wait for a key press before closing the image window.

Program:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
def detect_shapes(image_path):
    # Load the image
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Failed to load image from {image_path}.")
        return
    cv2_imshow(image)
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Blur to reduce noise
    gray_blurred = cv2.medianBlur(gray, 5)
    # Edge detection
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    # Hough Line Transform
    lines = cv2.HoughLines(edges, 1, np.pi / 180, 150)
    if lines is not None:
        for rho, theta in lines[:, 0]:
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            x1 = int(x0 + 1000 * (-b))
            y1 = int(y0 + 1000 * (a))
            x2 = int(x0 - 1000 * (-b))
            y2 = int(y0 - 1000 * (a))
            cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)
    # Hough Circle Transform
    circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1, 20,
                               param1=50, param2=30, minRadius=1, maxRadius=40)
    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            cv2.circle(image, (i[0], i[1]), i[2], (0, 255, 0), 2)
            cv2.circle(image, (i[0], i[1]), 2, (0, 0, 255), 3)
    # Contour Detection for Other Shapes
    contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        # Approximate the contour
        epsilon = 0.02 * cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, epsilon, True)
        # Draw the contours for different shapes
        if len(approx) == 3:
            # Triangle
            cv2.drawContours(image, [approx], 0, (0, 255, 255), 2)
        elif len(approx) == 4:
            # Rectangle or Square
            cv2.drawContours(image, [approx], 0, (255, 0, 0), 2)
        elif len(approx) > 4:
            # Circle or Ellipse
            area = cv2.contourArea(contour)
            if area > 100:
                cv2.drawContours(image, [approx], 0, (255, 255, 0), 2)
    # Display the result
    cv2_imshow(image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
# Example usage

detect_shapes("/content/circle.png")
```

Result:

Thus successfully performed the Implementation of Shape Detection using Hough Transform

Aim:

To write a python program to Build a QR Code Scanner

Algorithm:

1. **Import Libraries**: Import necessary libraries for image processing (`cv2`, `numpy`), QR code decoding (`pyzbar`), and displaying images (`cv2_imshow`).
2. **Define QR Code Decoding Function**:
   - Convert the image frame to grayscale.
   - Decode QR codes from the grayscale image.
3. **Load the Image**: Read the image file from a specified path.
4. **Check for Image Read Errors**: Ensure that the image was loaded successfully; if not, print an error message and exit.
5. **Detect QR Codes**: Use the `decode_qr` function to find QR codes in the image.
6. **Process Detected QR Codes**:
   - Extract and decode the data from each QR code.
   - Draw a bounding box around each QR code on the image.
   - Print the decoded QR code information.
7. **Display the Image**: Show the image with QR codes highlighted, and wait for a key press before closing the display window.

Program:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from pyzbar.pyzbar import decode
# Function to decode QR codes
def decode_qr(frame):
    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Decode QR codes
    qr_codes = decode(gray)
    return qr_codes

# Load the image file
image_path = '/content/qr.png'  # Replace with your image path
frame = cv2.imread(image_path)
if frame is None:
    print("Error: Could not read the image file")
    exit()

# Detect QR codes
qr_codes = decode_qr(frame)

# Process detected QR codes
for qr_code in qr_codes:
    # Extract QR code information
    qr_data = qr_code.data.decode('utf-8')
    # Draw bounding box around the QR code
    points = qr_code.polygon
    if len(points) > 4:
        hull = cv2.convexHull(np.array([point for point in points],dtype = np.float32))
        cv2.polylines(frame, [hull.astype(np.int32)], True, (255, 0, 0),3)
    else:
        cv2.polylines(frame, [np.array(points, dtype=np.int32)], True, (255, 0, 0), 3)

    # Print QR code information
    print("QR Code detected:", qr_data)

# Display the image with QR codes highlighted
cv2_imshow(frame)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result:

Thus successfully Built a QR Code Scanner

Aim:

To write a python program to perform Image captioning

Algorithm:

1. **Import Libraries**: Import the necessary libraries including Hugging Face Transformers, PyTorch, and PIL.
2. **Load Pre-trained Models**:
   - Load the VisionEncoderDecoderModel for image captioning.
   - Load the feature extractor to preprocess images for the model.
   - Load the tokenizer for decoding generated captions.
3. **Set Device**: Determine if a GPU (CUDA) is available and set the device accordingly. Move the model to the appropriate device.
4. **Define Generation Parameters**: Set parameters for generating captions, including maximum length and number of beams for beam search.
5. **Define Prediction Function**:
   - Convert image paths to PIL images.
   - Ensure all images are in RGB mode.
   - Extract pixel values from images.
   - Move pixel values to the appropriate device (GPU or CPU).
   - Generate captions using the model.
   - Decode the generated IDs into human-readable text.
6. **Run Prediction**: Call the `predict_step` function with a list of image paths and get

Program:

```
from transformers import VisionEncoderDecoderModel, ViTFeatureExtractor,
AutoTokenizer
import torch
from PIL import Image
import warnings
warnings.filterwarnings('ignore')
model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-
captioning")
feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-
captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
max_length = 16
num_beams = 4
gen_kwargs = {"max_length": max_length, "num_beams": num_beams}
def predict_step(image_paths):
  images = []
  for image_path in image_paths:
    i_image = Image.open(image_path)
    if i_image.mode != "RGB":
      i_image = i_image.convert(mode="RGB")
    images.append(i_image)
  pixel_values = feature_extractor(images=images, return_tensors="pt").pixel_values
  pixel_values = pixel_values.to(device)
  output_ids = model.generate(pixel_values, **gen_kwargs)
  preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
  preds = [pred.strip() for pred in preds]
  return preds
predict_step(["/content/ss.jpg"])
```

Result:

Thus successfully performed Image captioning

Aim:

To write a python program to perform Region based Segmentation

Algorithm:

Import Libraries: Import necessary libraries for image processing and visualization, including numpy, matplotlib, skimage, and scipy.

Load and Convert Image to Grayscale: Load the sample rocket image from skimage and convert it to grayscale using rgb2gray.

Apply Canny Edge Detection: Use the Canny edge detector to find edges in the grayscale image and display the result.

Fill Regions in Edge Image: Fill holes in the binary edge image to prepare for region segmentation and display the filled regions.

Prepare for Watershed Segmentation: Compute the elevation map using the Sobel filter, define markers based on grayscale intensity thresholds, and display the markers.

Perform Watershed Segmentation: Apply the watershed algorithm to segment the image and display the result.

Overlay and Contour Plot: Fill holes in the segmented image, label connected components, and overlay the segmented regions on the original grayscale image with contours.

Adjust and Display Plots: Use matplotlib to create subplots showing the original image with contours and the segmented image with labels, and adjust subplot margins for better visualization.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import canny
from skimage import data,morphology,filters
from skimage.color import rgb2gray,label2rgb
import scipy.ndimage as nd
plt.rcParams["figure.figsize"] = (12,8)
from skimage.segmentation import watershed
%matplotlib inline

# load images and convert grayscale
rocket = data.rocket()
rocket_wh = rgb2gray(rocket)
```

```
# apply edge segmentation
# plot canny edge detection
edges = canny(rocket_wh)
plt.imshow(edges, interpolation='gaussian')
plt.title('Canny detector')

# fill regions to perform edge segmentation
fill_im = nd.binary_fill_holes(edges)
plt.imshow(fill_im)
plt.title('Region Filling')

# Region Segmentation
# First we print the elevation map
elevation_map = filters.sobel(rocket_wh)
plt.imshow(elevation_map)

# Since, the contrast difference is not much. Anyways we will perform it
markers = np.zeros_like(rocket_wh)
markers[rocket_wh < 0.1171875] = 1 # 30/255
markers[rocket_wh > 0.5859375] = 2 # 150/255

plt.imshow(markers)
plt.title('markers')

# Perform watershed region segmentation
segmentation =watershed(elevation_map, markers)

plt.imshow(segmentation)
plt.title('Watershed segmentation')
```

```
# plot overlays and contour
segmentation = nd.binary_fill_holes(segmentation - 1)
label_rock, _ = nd.label(segmentation)
# overlay image with different labels
image_label_overlay = label2rgb(label_rock, image=rocket_wh)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 16), sharey=True)
ax1.imshow(rocket_wh)
ax1.contour(segmentation, [0.8], linewidths=1.8, colors='w')
ax2.imshow(image_label_overlay)
margins = dict(left=0.02, right=0.98, bottom=0.05, top=0.95, wspace=0.05, hspace=0.05)
fig.subplots_adjust(**margins)
```

Result:

Thus successfully performed Region based Segmentation

Aim:

To write a python program to perform Emotion Recognition through Facial Expressions

Algorithm:

1. **Import Libraries**: Import `cv2` for image processing, `matplotlib.pyplot` for plotting, and `DeepFace` for facial analysis.
2. **Read and Display the Image**: Load the image from a specified path using OpenCV and display it using `matplotlib`. Note that `img[:, :, ::-1]` is used to convert the image from BGR (OpenCV's default) to RGB (matplotlib's expected format).
3. **Analyze the Image**: Use the `DeepFace` library to analyze the image for facial emotions.
4. **Print the Analysis Result**: Output the result of the emotion analysis to the console.

Program:

```
import cv2
import matplotlib.pyplot as plt
from deepface import DeepFace
img = cv2.imread(r'/content/img.jpg')
plt.imshow(img[:, :, ::-1])
plt.show()
result = DeepFace.analyze(img,
actions = ['emotion'])
print(result)
```

Result:

Thus successfully performed Emotion Recognition through Facial Expressions

Aim:

To write a python program to perform Scene Text Detection

Algorithm:

Import Libraries: Import cv2 for image processing and pytesseract for optical character recognition (OCR).

Set Tesseract Path: Configure the path to the Tesseract OCR executable. This path must be set correctly for pytesseract to function.

Define Text Extraction Function:

Load the image from the specified path.

Convert the image to grayscale.

Use pytesseract to extract text from the grayscale image.

Specify Image Path: Set the path to the image file that contains the text to be extracted.

Extract and Print Text: Call the extract_text_from_image function with the image path, and print the extracted text to the console.

Program:

```
import cv2
import pytesseract
pytesseract.pytesseract.tesseract_cmd ='C:\\Program Files\\Tesseract-OCR\\tesseract.exe' #set that app path
def extract_text_from_image(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    text = pytesseract.image_to_string(gray)
    return text
image_path ='C:\\Users\\sudha\\Desktop\\text image.png' #set the path for image which contains the text
extracted_text = extract_text_from_image(image_path)
print("Extracted Text:")
print(extracted_text)
```

Result:

Thus successfully performed Scene Text Detection

Aim:

To write a python program to Build a People Counting Solution

Algorithm:

- **Import Libraries**: Import `cv2` for computer vision tasks and `numpy` for numerical operations.
- **Load Pre-trained Model**: Load the MobileNet-SSD model and the corresponding configuration file using OpenCV's `dnn` module.
- **Initialize Video Capture**: Start capturing video from the webcam or a specified video file.
- **Define Classes and Colors**: Set up the list of object classes that MobileNet-SSD can detect and assign a random color to each class.
- **Initialize People Count**: Set up a counter to keep track of the number of detected people.
- **Process Video Frames**:
  - Read frames from the video capture.
  - Prepare the frame for object detection by resizing and normalizing it.
  - Perform object detection on the frame using the MobileNet-SSD model.
- **Count and Draw Bounding Boxes**:
  - Loop through the detected objects, filter detections based on confidence, and check if the detected class is "person."
  - Draw bounding boxes and labels around detected people.
- **Update and Display People Count**: Update the people count if the number of detected people changes, and display the count on the frame.
- **Show Frame and Handle Exit**: Display the current frame with detections and text. Exit the loop if the 'q' key is pressed.
- **Clean Up**: Release the video capture object and close all OpenCV windows.

Program:

```
import cv2
import numpy as np
# Load the pre-trained MobileNet-SSD model and the prototxt file
net = cv2.dnn.readNetFromCaffe('C:\\Users\\sudha\\Desktop\\deploy.prototxt',
'C:\\Users\\sudha\\Desktop\\MobileNet-SSD-master\\MobileNet-SSD-master\mobilenet_iter_73000.caffemodel')
# Initialize the video capture (0 for webcam or provide a video file path)
cap = cv2.VideoCapture(0)
# List of classes for MobileNet-SSD
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
           "car", "cat", "chair", "cow", "diningtable", "dog", "horse",
           "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]
# Colors for each class
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
# Initialize a counter for people
people_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Prepare the frame for object detection
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 0.007843, (300, 300), 127.5)
    net.setInput(blob)
    detections = net.forward()
    current_count = 0
    # Loop over the detections
    for i in np.arange(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        # Filter out weak detections
        if confidence > 0.2:
            idx = int(detections[0, 0, i, 1])
            # Only consider the "person" class
            if CLASSES[idx] == "person":
                current_count += 1
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
                cv2.rectangle(frame, (startX, startY), (endX, endY), COLORS[idx], 2)
                y = startY - 15 if startY - 15 > 15 else startY + 15
                cv2.putText(frame, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
    # Update the people count if it changes
    if current_count != people_count:
        people_count = current_count
    # Display the count on the frame
    cv2.putText(frame, f"People Count: {people_count}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    # Show the output frame
    cv2.imshow("Frame", frame)
    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Clean up
cap.release()
cv2.destroyAllWindows()
```

Result:

Thus successfully Built a People Counting Solution

Aim:

To write a python program to perform Road Lane Detection in Autonomous Vehicles

Algorithm:

- **Import Libraries**: The code imports `cv2` for computer vision tasks, `numpy` for numerical operations, and `matplotlib.pyplot` for displaying images.
- **Load and Convert Image**: It reads an image from a file, converts it from BGR to RGB, and then converts the RGB image to grayscale.
- **Pre-process Image**: The grayscale image is dilated using a 5x5 kernel to enhance features.
- **Edge Detection**: The Canny edge detection algorithm is applied to the dilated grayscale image to find edges.
- **Define and Apply Region of Interest (ROI)**: The vertices of a polygon defining the ROI are specified. A mask is created and applied to isolate the ROI from the edge-detected image.
- **Line Detection**: Lines are detected in the ROI image using the Hough Line Transform.
- **Draw Detected Lines**: The detected lines are drawn on the original RGB image.
- **Display Final Image**: The final image, with detected lines overlaid, is displayed using `matplotlib`.

Program:

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

# Load the image
img = cv2.imread("./Data/test_img.jpg")

# Convert the image from BGR to RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Convert the image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# Apply dilation
gray_img = cv2.dilate(gray_img, kernel=np.ones((5, 5), np.uint8))

# Apply Canny edge detection
canny = cv2.Canny(gray_img, 100, 200)

# Define the region of interest (ROI) vertices
roi_vertices = [(270, 670), (600, 400), (1127, 712)]

# Define a function to apply the ROI mask
def roi(image, vertices):
    mask = np.zeros_like(image)
    mask_color = 255  # Mask color for grayscale images
    cv2.fillPoly(mask, vertices, mask_color)
    masked_img = cv2.bitwise_and(image, mask)
    return masked_img

# Apply the ROI mask to the Canny edge-detected image
roi_image = roi(canny, np.array([roi_vertices], np.int32))

# Apply Hough Line Transform to detect lines
lines = cv2.HoughLinesP(roi_image, 1, np.pi/180, 100, minLineLength=100, maxLineGap=10)

# Define a function to draw lines on the image
def draw_lines(image, hough_lines):
    for line in hough_lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

```
    return image

# Draw the detected lines on the original image
final_img = draw_lines(img, lines)

# Display the final image
plt.imshow(final_img)
plt.xticks([])
plt.yticks([])
plt.show()
```

Result:

Thus successfully performed Road Lane Detection in Autonomous Vehicles

Aim:

To write a python program to perform Basic motion detection and tracking

Algorithm:

1. **Initialize Video Capture:**
   - Start by creating a video capture object with `cv2.VideoCapture`, where you can use `0` for the webcam or provide a path to a video file.
2. **Set Up Background Subtractor:**
   - Initialize the background subtractor with `cv2.createBackgroundSubtractorMOG2()` to differentiate moving objects from the static background.
3. **Process Video Frames:**
   - Continuously capture frames from the video source using `cap.read()`. Check if the frame is read correctly, and if not, exit the loop.
4. **Apply Background Subtraction:**
   - Use the background subtractor to generate a foreground mask that highlights moving objects against the static background.
5. **Detect Contours:**
   - Find contours in the foreground mask with `cv2.findContours()`. These contours represent the boundaries of detected objects.
6. **Filter and Draw Bounding Boxes:**
   - Loop through the detected contours and filter out those with small areas to avoid noise. For each significant contour, compute the bounding box using `cv2.boundingRect()` and draw it on the original frame with `cv2.rectangle()`.
7. **Display Results:**
   - Show the original frame with bounding boxes and the foreground mask using `cv2.imshow()`.
8. **Handle User Input:**
   - Use `cv2.waitKey()` to wait for user input. Exit the loop if the 'q' key is pressed.
9. **Clean Up:**
   - Release the video capture object and close all OpenCV windows to clean up resources.

Program:

```
import cv2
import numpy as np

# Initialize the video capture object
cap = cv2.VideoCapture(0)  # Use 0 for the webcam, or replace with a video file path

# Initialize background subtractor
fgbg = cv2.createBackgroundSubtractorMOG2()

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

    # Apply the background subtractor to get the foreground mask
    fgmask = fgbg.apply(frame)

    # Find contours in the mask
    contours, _ = cv2.findContours(fgmask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw bounding boxes around detected objects
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter out small contours
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    cv2.imshow('Foreground Mask', fgmask)

    # Exit if the 'q' key is pressed
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break

# Release the video capture object and close all windows
cap.release()
cv2.destroyAllWindows()
```

Result:

Thus successfully performed Basic motion detection and tracking

Aim:

To write a python program to perform Face Detection on Your Family Photos

Algorithm:

Load the Image:

Read the image from the specified path using an image processing library (e.g., OpenCV).

Convert Image to RGB:

Convert the image from BGR to RGB color space if using OpenCV (which loads images in BGR format).

Load Face Detection Model:

Load a pre-trained Haar Cascade classifier for face detection. This classifier is trained to detect faces in images.

Detect Faces:

Apply the face detection model to the image to detect faces. This model returns bounding boxes around detected faces.

Convert to PIL Image:

Convert the image to a PIL (Python Imaging Library) image format for drawing purposes.

Draw Bounding Boxes:

Use PIL to draw rectangles around the detected faces. The rectangles are drawn using the coordinates of the bounding boxes returned by the face detection model.

Display the Image:

Use Matplotlib to display the image with annotated faces.

Save the Image:

Save the annotated image to a file.

Print Output Path:

Print the path where the annotated image is saved.

Program:

```
import cv2
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
import numpy as np

# Load the image
image_path = 'input2.jpg'
img = cv2.imread(image_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

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

# Detect faces in the image
faces = face_cascade.detectMultiScale(img_rgb, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
# Convert to PIL image for drawing
pil_image = Image.open(image_path)
draw = ImageDraw.Draw(pil_image)
# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    draw.rectangle([(x, y), (x + w, y + h)], outline="red", width=2)
# Display the result
plt.figure(figsize=(8, 6))
plt.imshow(pil_image)
plt.axis('off')
plt.show()
# Save the image with detected faces
output_image_path = 'output_image_with_faces_detected.jpg'
pil_image.save(output_image_path)
print(f"Image with faces detected saved at: {output_image_path}")
```

Result:

Thus successfully performed Face Detection on Your Family Photos

Aim:

To write a python program to Developing Social Distancing application

Algorithm:

- Load the image from the file path using OpenCV.
- Define two bounding boxes for objects in the image, specifying their coordinates and dimensions.
- Draw rectangles around the defined bounding boxes on the image.
- Compute the center points of each bounding box and calculate the Euclidean distance between these centers.
- Compare the calculated distance to a predefined threshold to determine if social distancing is maintained. If the distance is below the threshold, it indicates that the objects are too close. Otherwise, they are sufficiently apart.
- Draw a combined bounding box that encompasses both original bounding boxes, using a color that indicates whether social distancing is maintained or not.
- Overlay the calculated distance on the image as text and display the processed image.

Program:

```
import cv2
import numpy as np
import os
from google.colab.patches import cv2_imshow

def calculate_distance(bbox1, bbox2):
    center1 = (bbox1[0] + bbox1[2] // 2, bbox1[1] + bbox1[3] // 2)
    center2 = (bbox2[0] + bbox2[2] // 2, bbox2[1] + bbox2[3] // 2)
    distance = np.sqrt((center1[0] - center2[0])**2 + (center1[1] - center2[1])**2)
    return distance

def draw_bounding_box(image, bbox, color):
    cv2.rectangle(image, (bbox[0], bbox[1]), (bbox[0] + bbox[2], bbox[1] + bbox[3]), color, 2)

image_path = '/content/ii.jpg'
if not os.path.exists(image_path):
    print(f"Error: Image file '{image_path}' not found.")
else:
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Unable to load image '{image_path}'")
    else:
        bbox1 = (100, 50, 200, 150)
        bbox2 = (300, 200, 180, 120)

        draw_bounding_box(image, bbox1, (0, 255, 0))
        draw_bounding_box(image, bbox2, (0, 255, 0))

        distance = calculate_distance(bbox1, bbox2)

        if distance < 200:
            color = (0, 255, 0)
            print("Social Distancing")
        else:
            color = (0, 0, 255)
            print("Not maintaining Social Distancing")
        bbox_combined = (min(bbox1[0], bbox2[0]), min(bbox1[1], bbox2[1]),
                         max(bbox1[0] + bbox1[2], bbox2[0] + bbox2[2]) - min(bbox1[0], bbox2[0]),
                         max(bbox1[1] + bbox1[3], bbox2[1] + bbox2[3]) - min(bbox1[1], bbox2[1]))
        draw_bounding_box(image, bbox_combined, color)

        cv2.putText(image, f'Distance: {distance:.2f} pixels', (50, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)

        cv2_imshow(image)
```

Result:

Thus successfully Developed Social Distancing application

Aim:

To write a python program to Count Vehicles in Images and Video

Algorithm:

1. Import the libraries: `cv2`, `numpy`, and `time`.
2. Define parameters for minimum bounding box width and height, allowed offset, position of the counting line, and frames per second.
3. Create a function to calculate the center of a bounding box.
4. Open the video file using `cv2.VideoCapture`.
5. Initialize a background subtractor using `cv2.bgsegm.createBackgroundSubtractorMOG()`.
6. Start a loop to process each video frame:
   - Capture the frame.
   - If the frame is not read, break the loop.
7. Preprocess the frame by converting to grayscale, applying Gaussian blur, and background subtraction. Perform dilation on the mask.
8. Apply morphological closing to the mask.
9. Detect contours in the processed mask.
10. Draw a counting line on the frame.
11. For each detected contour, calculate the bounding rectangle, check if it meets the minimum size criteria, and draw the bounding box and center on the frame.
12. For each detected center, check if it crosses the counting line within the allowed offset. Increment the vehicle count if it does and draw the counting line in a different color.
13. Display the vehicle count on the frame and show the original frame and processed mask.
14. Check if the 'Esc' key is pressed to exit the loop.
15. Release the video capture object and close all OpenCV windows.

Program:

```
import cv2
import numpy as np
from time import sleep

# Configuration
largura_min = 80  # Minimum width of rectangle
altura_min = 80  # Minimum height of rectangle
offset = 6  # Allowed error between pixels
pos_linha = 550  # Position of counting line
delay = 60  # FPS of the video

detec = []
carros = 0

def pega_centro(x, y, w, h):
    x1 = int(w / 2)
    y1 = int(h / 2)
    cx = x + x1
    cy = y + y1
    return cx, cy

# Open video file
cap = cv2.VideoCapture('video.mp4')
subtracao = cv2.bgsegm.createBackgroundSubtractorMOG()

while True:
    ret, frame1 = cap.read()
    if not ret:
        break

    tempo = float(1 / delay)
    sleep(tempo)

    grey = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(grey, (3, 3), 5)
    img_sub = subtracao.apply(blur)
    dilat = cv2.dilate(img_sub, np.ones((5, 5)))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    dilatada = cv2.morphologyEx(dilat, cv2.MORPH_CLOSE, kernel)
    dilatada = cv2.morphologyEx(dilatada, cv2.MORPH_CLOSE, kernel)
```

```
    contours, _ = cv2.findContours(dilatada, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.line(frame1, (25, pos_linha), (1200, pos_linha), (255, 127, 0), 3)
    for (i, c) in enumerate(contours):
        (x, y, w, h) = cv2.boundingRect(c)
        validar_contorno = (w >= largura_min) and (h >= altura_min)
        if not validar_contorno:
            continue
        cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
        centro = pega_centro(x, y, w, h)
        detec.append(centro)
        cv2.circle(frame1, centro, 4, (0, 0, 255), -1)
    for (x, y) in detec:
        if (y < (pos_linha + offset)) and (y > (pos_linha - offset)):
            carros += 1
            cv2.line(frame1, (25, pos_linha), (1200, pos_linha), (0, 127, 255), 3)
            detec.remove((x, y))
    print("Car is detected: " + str(carros))
    cv2.putText(frame1, "VEHICLE COUNT: " + str(carros), (450, 70),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 5)
    cv2.imshow("Video Original", frame1)
    cv2.imshow("Detectar", dilatada)
    if cv2.waitKey(1) == 27:  # Press 'Esc' to exit
        break

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

Result:

Thus successfully developed code to Count Vehicles in Images and Video