```
import cv2
from google.colab.patches import cv2_imshow
image=cv2.imread('set image path')
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
resized_image = cv2.resize(image,(200,200))
cv2_imshow(resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
blurred_image = cv2.GaussianBlur(image, (15, 15), 0)
cv2_imshow(blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
edges = cv2.Canny(gray_image, 100, 200)
cv2_imshow(edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.rectangle(image, (50,50), (300,300), (255, 0, 0), 2)
cv2.line(image, (60,60), (300,300), (0, 0, 255), 2)
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
from PIL import Image
import numpy as np
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("C:\\Users\\student\\Desktop\\cube1.jpeg")
depth_img = Image.open("C:\\Users\\student\\Desktop\\cube3.jpeg")
shifted_img = shift_image(img, depth_img, shift_amount=10)
shifted_img.show()
```

```
import cv2
def detect_moving_objects(video_path):
 cap = cv2.VideoCapture(video_path)
 if not cap.isOpened():
 print("Error: Couldn't open the video file.")
 return
 bg_subtractor = cv2.createBackgroundSubtractorMOG2()
 while cap.isOpened():
 ret, frame = cap.read()
 if not ret:
 break
 fg_mask = cv2.medianBlur(bg_subtractor.apply(frame), 5)
contours, _ = cv2.findContours(fg_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
 for contour in contours:
 if cv2.contourArea(contour) > 100:r
 M = cv2.moments(contour)
 if M["m00"] != 0:
 cx = int(M["m10"] / M["m00"])
 cy = int(M["m01"] / M["m00"])
 # Draw a dot (small filled circle)
cv2.circle(frame, (cx, cy), 5, (255, 0, 0), 5)
cv2.imshow('Moving Object Detection', frame)
if cv2.waitKey(45) & 0xFF == ord('q'):
 break
 cap.release()
 cv2.destroyAllWindows()
video_path = 'input.mp4'
detect_moving_objects(video_path)
```

```
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(["C:\\Users\\student\\Downloads\\ss.jpg"])
```

```
from PIL import
Image importcv2
import numpy as np import
requests image_url =
'sample.jpg'
image = Image.open(image_url) image
= image.resize((450, 250))
image.show()
cv2.waitKey(0)
cv2.destroyAllWindows() image_arr =
np.array(image)
grey = cv2.cvtColor(image_arr, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(grey, (5, 5), 0)
dilated = cv2.dilate(blur, np.ones((3, 3))) dilated =
cv2.dilate(blur, np.ones((3, 3)))
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
closing = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE, kernel)
car_cascade_src = "vehicle.xml"
car_cascade = cv2.CascadeClassifier(car_cascade_src)
cars = car_cascade.detectMultiScale(closing, 1.1, 1)
cnt = 0

for (x, y, w, h) in cars:

   cv2.rectangle(image_arr, (x, y), (x + w, y + h), (255, 0, 0), 2)

   cnt += 1

annotated_image = Image.fromarray(image_arr)
annotated_image.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt#
Load the image
image = cv2.imread('path_to_your_image.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Apply GaussianBlur to reduce noise and improve contour detectionblurred =
cv2.GaussianBlur(gray, (5, 5), 0)
# Perform edge detection
edged = cv2.Canny(blurred, 50, 150)# Find
contours
contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
# Draw contours on the original image
contour_image = image.copy()
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)# Display
the results
plt.figure(figsize=(10, 10))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 3, 2)
plt.title('Edge Detection')
plt.imshow(edged, cmap='gray')
plt.subplot(1, 3, 3) plt.title('Contours')
plt.imshow(cv2.cvtColor(contour_image, cv2.COLOR_BGR2RGB))plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import canny
from skimage import data, segmentation, morphology, filtersfrom skimage.color
import rgb2gray, label2rgb
import scipy.ndimage as nd
plt.rcParams["figure.figsize"] = (12, 8)
%matplotlib inline rocket =
data.rocket()
rocket_wh = rgb2gray(rocket)edges =
canny(rocket_wh)
plt.imshow(edges, interpolation='gaussian')plt.title('Canny
detector')
plt.show()
fill_im = nd.binary_fill_holes(edges)
plt.imshow(fill_im)
plt.title('Region Filling')
plt.show()
elevation_map = filters.sobel(rocket_wh)
plt.imshow(elevation_map) plt.title('Elevation Map')
plt.show()
markers = np.zeros_like(rocket_wh) markers[rocket_wh <
0.1171875] = 1 # 30/255 markers[rocket_wh > 0.5859375] = 2
# 150/255plt.imshow(markers)
plt.title('Markers')plt.show()
segments = segmentation.watershed(elevation_map, markers)
plt.imshow(segments)
plt.title('Watershed Segmentation')plt.show()

segments_filled = nd.binary_fill_holes(segments - 1)label_rock, _ =
nd.label(segments_filled)
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(segments_filled, [0.8], linewidths=1.8, colors='w')
ax2.imshow(image_label_overlay)
plt.show()
```

```
import cv2
import numpy as np
def detect_shapes(image_path): image =
    cv2.imread(image_path)if image is None:
        print(f"Error: Failed to load image from {image_path}.")return
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)gray_blurred
    = cv2.medianBlur(gray, 5)
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    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)
    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)
    contours, _ = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)for contour in contours:
        epsilon = 0.02 * cv2.arcLength(contour, True) approx =
        cv2.approxPolyDP(contour, epsilon, True)if len(approx) == 3:
            cv2.drawContours(image, [approx], 0, (0, 255, 255), 2)elif
        len(approx) == 4:
            cv2.drawContours(image, [approx], 0, (255, 0, 0), 2)
         elif len(approx) > 4:
            area = cv2.contourArea(contour)if area >
            100:
                cv2.drawContours(image, [approx], 0, (255, 255, 0), 2)
    cv2.imshow('Shape Detection', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
            detect_shapes("C:\\Users\\student\\Downloads\\circle.png")
```

```
import cv2
import matplotlib.pyplot as plt
image_path = 'C:/Users/student/Downloads/perfect-family-photo-session-by-rebecca-danzenbaker.webp' # Replace with your image path
image = cv2.imread(image_path)
if image is None:
 print(f"Error: Could not load image from {image_path}")
else:
 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
 faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))
 for (x, y, w, h) in faces:
 cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
 image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
 plt.imshow(image_rgb)
 plt.axis('off')
plt.show()
import cv2
import pytesseract
pytesseract.pytesseract.tesseract_cmd =
'c:\\Users\\online\\AppData\\Local\\Programs\\Tesseract-OCR\\tesseract.exe'
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 = './sample.jpeg'
extracted_text = extract_text_from_image(image_path)
print("Extracted Text:")
print(extracted_text)
```

```
import cv2
import numpy as np
def region_of_interest(img, vertices):
 mask = np.zeros_like(img)
 cv2.fillPoly(mask, vertices, 255)
 masked_image = cv2.bitwise_and(img, mask)return
 masked_image
def draw_lines(img, lines):
 if lines is not None:
 for line in lines:
 for x1, y1, x2, y2 in line:
 cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 5)def
process_image(image):
 height, width =  image.shape[:2] # Convert
 the image to grayscale
 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)# Apply Gaussian
 blur
 blur = cv2.GaussianBlur(gray, (5, 5), 0)# Apply Canny
 edge detector
 edges = cv2.Canny(blur, 50, 150)# Define the
 region of interest
 vertices = np.array([[(50, height), (width//2 - 50, height//2 + 50), (width//2 + 50,height//2 + 50), (width - 50,
height)]], dtype=np.int32)
 masked_edges = region_of_interest(edges, vertices)
lines = cv2.HoughLinesP(masked_edges, rho=1, theta=np.pi/180, threshold=50,minLineLength=50,
maxLineGap=200)
 # Draw the lines on the original imageline_image =
 np.zeros_like(image) draw_lines(line_image, lines)
 result = cv2.addWeighted(image, 0.8, line_image, 1, 0)return result
def main():
 cap =cv2.VideoCapture("C:\\Users\\student.SCASA1\\Downloads\\travel_road.mp4")
 while(cap.isOpened()):
 ret, frame = cap.read()if ret:
 result = process_image(frame) cv2.imshow('Lane
 Detection', result)if cv2.waitKey(1) & 0xFF == ord('q'):
 break
 else:
 break cap.release()
 cv2.destroyAllWindows()
if __name_____== '_main_':main()
```

```
import cv2
from pyzbar.pyzbar import decode
def decode_qr_from_image(image_path):
    frame = cv2.imread(image_path)
    if frame is None:
        print(f"Failed to load image from {image_path}")
        return
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    decoded_objects = decode(gray_frame)
    for obj in decoded_objects:
        qr_data = obj.data.decode('utf-8')
        print('Data:', qr_data)
        points = obj.polygon
        if len(points) > 4:
            hull = cv2.convexHull(np.array([point for point in points], dtype=np.float32))
            hull = list(map(tuple, np.squeeze(hull)))
        else:
            hull = points
        n = len(hull)
        for j in range(0, n):
            cv2.line(frame, hull[j], hull[(j + 1) % n], (255, 0, 0), 3)
    cv2.imshow('QR Code Scanner', frame)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
def main():
    image_path = "C:\\Users\\student.SCASA1\\Downloads\\OIP.jpg"
    decode_qr_from_image(image_path)
if__name__== "__main__":
    main
```

12

```
pip install deepface

import cv2

import matplotlib.pyplot as plt from

deepface import DeepFace

img = cv2.imread(r'/content/321.jpg')

plt.imshow(img[:, :, : : -1]) plt.show()

result = DeepFace.analyze(img,

                actions = ['emotion'])

print(result)
```

```
import cv2
import numpy as np
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)
```

```
!pip install "openvino>=2024.0.0" "ultralytics==8.2.18" "torch>=2.1" "ipywidgets==7.7.1"  from
pathlib import Path
  from ultralytics import YOLO
  models_dir = Path("./models")
  models_dir.mkdir(exist_ok=True)
  DET_MODEL_NAME =
  "yolov8n"
  det_model = YOLO(models_dir / f"{DET_MODEL_NAME}.pt")
  label_map = det_model.model.names
  res = det_model()
  det_model_path=models_dir/f"{DET_MODEL_NAME}_openvino_model/{DET_MODEL_NAME}.x
  ml"  if not det_model_path.exists():
     det_model.export(format="openvino", dynamic=True, half=True)
  from ultralytics import YOLO, solutions
  import cv2  import
  time  import
  collections
  import numpy as np
  from IPython import display
  import torch
  import openvino as ov
  import ipywidgets as widgets
  def run_inference(source, device):
     core = ov.Core()
     det_ov_model = core.read_model(det_model_path)
     ov_config = {}
  if "GPU" in device.value or ("AUTO" in device.value and "GPU" in core.available_devices):
     ov_config = {"GPU_DISABLE_WINOGRAD_CONVOLUTION": "YES"}
  compiled_model = core.compile_model(det_ov_model, device.value, ov_config)
  def infer(*args):
     result    =    compiled_model(args)
     return torch.from_numpy(result[0])
  det_model.predictor.inference  =  infer
  det_model.predictor.model.pt  =  False
  try:
     cap = cv2.VideoCapture(source)
     assert cap.isOpened(), "Error reading video file"
     line_points = [(0, 300), (1080, 300)]
     classes_to_count = [0]
  counter = solutions.ObjectCounter(view_img=False,reg_pts=line_points,
                              classes_names=det_model.names,draw_tracks=True,
                              line_thickness=2,view_in_counts=False,view_out_counts=False)
```

```
processing_times = collections.deque(maxlen=200)
while cap.isOpened():
    success, frame =
    cap.read()  if not success:
        print("Video frame is empty or video processing has been successfully completed.")
        break
    start_time = time.time()
    tracks = det_model.track(frame, persist=True, show=False,
                        classes=classes_to_count,verbose=False)
    frame = counter.start_counting(frame, tracks)
    stop_time = time.time()
    processing_times.append(stop_time - start_time)
    _, f_width = frame.shape[:2]
    processing_time = np.mean(processing_times) *
    1000  fps = 1000 / processing_time
    cv2.putText(img=frame,text=f"Inference time: {processing_time:.1f}ms ({fps:.1f} FPS)",
            org=(20, 40),fontFace=cv2.FONT_HERSHEY_COMPLEX,fontScale=f_width / 1000,color=(0, 0, 255),
            thickness=2,lineType=cv2.LINE_AA)
    counts = counter.out_counts
    text = f"Count: {counts}"
    fontFace = cv2.FONT_HERSHEY_COMPLEX
    fontScale = 0.75
    thickness = 2
    (text_width, text_height), _ = cv2.getTextSize(text, fontFace, fontScale, thickness)
    top_right_corner = (frame.shape[1] - text_width - 20, 40)
   cv2.putText(img=frame,text=text,org=(top_right_corner[0],
top_right_corner[1]),fontFace=fontFace,fontScale=fontScale,color=(0,0,255),thickness=thickness,lineType=cv2.LINE_AA)
    _, encoded_img = cv2.imencode(ext=".jpg", img=frame, params=[cv2.IMWRITE_JPEG_QUALITY,
    100])   i = display.Image(data=encoded_img)
```

```
import cv2
import numpy as np
from time import sleep
largura_min=80 #Largura minima do retangulo
altura_min=80 #Altura minima do retangulo
offset=6 #Erro permitido entre pixel
pos_linha=550 #Posição da linha de contagem
delay= 60 #FPS do vídeo
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
cap = cv2.VideoCapture('video.mp4')
subtracao = cv2.bgsegm.createBackgroundSubtractorMOG()
while True:
ret , frame1 = cap.read()
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)
contorno,h=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(contorno):
(x,y,w,h) = cv2.boundingRect(c)
validar_contorno = (w >= largura_min) and (h >= altura_min)
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:
break
cv2.destroyAllWindows()
cap.release()
cap.release()
```

```
    display.clear_output(wait=True)  display.display(i)
except KeyboardInterrupt:
   print("Interrupted")  cap.release()
cv2.destroyAllWindows()
VIDEO_SOURCE = "https://github.com/intel-iot-devkit/sample-videos/raw/master/people-detection.mp4"  import
ipywidgets as widgets
import openvino as ov  core =
ov.Core()
device = widgets.Dropdown(options=core.available_devices + ["AUTO"], value="AUTO", description="Device:",
disabled=False)
device
run_inference(source=VIDEO_SOURCE, device = device)
```