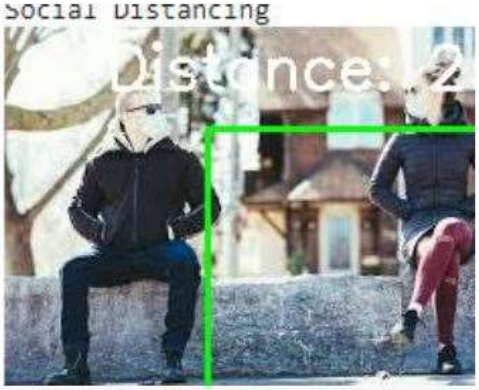
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
def calculate_distance(bbox1, bbox2):
    center1 = (bbox1[0] + bbox1[2] // 2, bbox1[1] + bbox1[3] // 2)
    center2 = (bbox2[0] + bbox2[2] // 2, bbox2[1] + bbox2[3] // 2)
    distance = np.sqrt((center1[0] - center2[0])**2 + (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)
```

Social Distancing

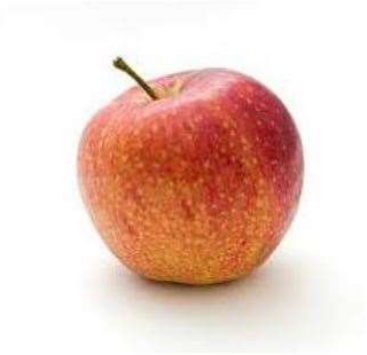


Not maintaining Social Distancing

```
!pip install opencv-python-headless
!pip install matplotlib
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab import files
from google.colab.patches import cv2_imshow
def upload_image():
  uploaded = files.upload()
  for filename in uploaded.keys():
    image = cv2.imread(filename)
  return image
def detect_shapes(image):
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  blurred = cv2.GaussianBlur(gray, (9, 9), 2)
  edges = cv2.Canny(blurred, 50, 150, apertureSize=3)
  lines = cv2.HoughLines(edges, 1, np.pi / 180, 200)
  if lines is not None:
    for line in lines:
      rho, theta = line[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(blurred, cv2.HOUGH_GRADIENT,
                dp=1.2,minDist=30, param1=50, param2=30, minRadius=15, maxRadius=100)
  if circles is not None:
    circles = np.round(circles[0, :]).astype("int")
    for (x, y, r) in circles:
      cv2.circle(image, (x, y), r, (0, 255, 0), 4)
```

```
    return image
print("Upload an image file:")
image = upload_image()
print("Original Image:")
cv2_imshow(image)
detected_shapes_image = detect_shapes(image)
print("Detected Shapes:")
cv2_imshow(detected_shapes_image)
```

Input Image:



Output Image:

```
import cv2
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
import numpy as np
image_path = 'input2.jpg'
img = cv2.imread(image_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                     'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(img_rgb, scaleFactor=1.1,
                                      minNeighbors=5, minSize=(30, 30))
pil_image = Image.open(image_path)
draw = ImageDraw.Draw(pil_image)
for (x, y, w, h) in faces:
    draw.rectangle([(x, y), (x+w, y+h)], outline="red", width=2)
plt.figure(figsize=(8, 6))
plt.imshow(pil_image)
plt.axis('off')
plt.show()
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}")
```

```
!pip install opencv-python
!pip install pytesseract
!sudo apt-get install tesseract-ocr
import cv2
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r'/usr/bin/tesseract'
img = cv2.imread("sample4.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
dilation = cv2.dilate(thresh1, rect_kernel, iterations = 1)
contours, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
im2 = img.copy()
file = open("recognized.txt", "w+")
file.write("")
file.close()
for cnt in contours:
    x, y, w, h = cv2.boundingRect(cnt)
    rect = cv2.rectangle(im2, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cropped = im2[y:y + h, x:x + w]
    file = open("recognized.txt", "a")
    text = pytesseract.image_to_string(cropped)
    file.write(text)
    file.write("\n")
    file.close()
from google.colab import drive
drive.mount('/content/drive')
with open('/content/recognized.txt', 'r') as file:
    contents = file.read()
    print(contents)
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.8
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: pytesseract in /usr/local/lib/python3.10/dist-packages (0.3.1
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (fro
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tesseract-ocr is already the newest version (4.1.1-2.1build1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/
```

```
Text is at different regions
```

```
This is SAMPLE TEXT
```

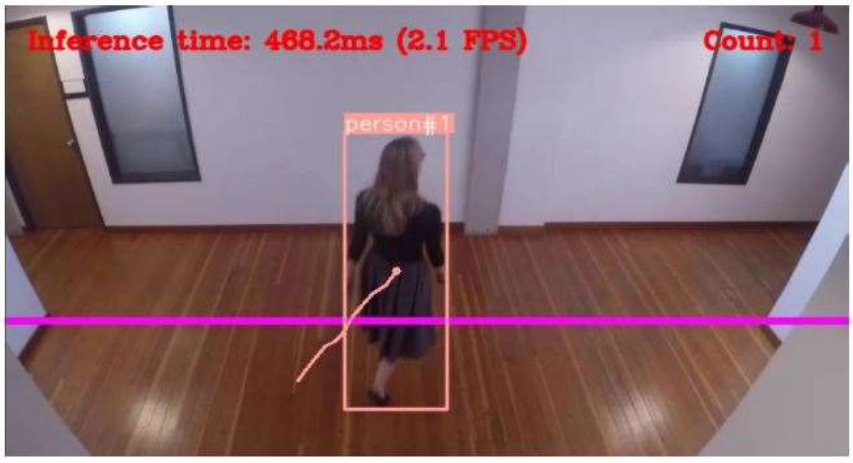
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("./Data/test_img.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray_img = cv2.dilate(gray_img, kernel=np.ones((5, 5), np.uint8))
canny = cv2.Canny(gray_img, 100, 200)
roi_vertices = [(270, 670), (600, 400), (1127, 712)]
def roi(image, vertices):
    mask = np.zeros_like(image)
    mask_color = 255
    cv2.fillPoly(mask, vertices, mask_color)
    masked_img = cv2.bitwise_and(image, mask)
    return masked_img
roi_image = roi(canny, np.array([roi_vertices], np.int32))
lines = cv2.HoughLinesP(roi_image, 1, np.pi/180, 100, minLineLength=100, maxLineGap=10)
def draw_lines(image, hough_lines):
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
    return image
final_img = draw_lines(img, lines)
plt.imshow(final_img)
plt.xticks([])
plt.yticks([])
plt.show()
```

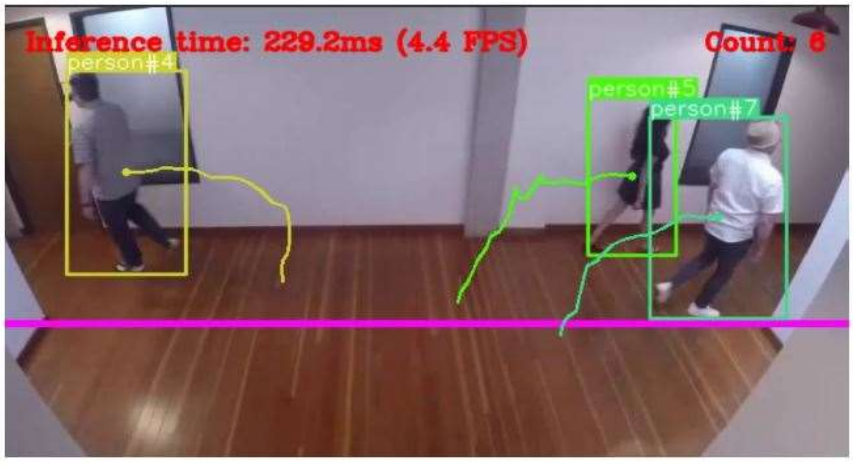
```
!pip install opencv-python-headless matplotlib deepface
import cv2
import matplotlib.pyplot as plt
from deepface import DeepFace
from google.colab import files
import io
from PIL import Image
uploaded = files.upload()
for fn in uploaded.keys():
  img_path = fn
  img = cv2.imread(img_path)
  plt.imshow(img[:,:,::-1])
  plt.axis('off')
  plt.show()
  try:
    result = DeepFace.analyze(img, actions=['emotion'])
    print(result)
    print("Dominant emotion: ", result[0]['dominant_emotion'])
  except ValueError:
    print("No face detected in the 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}.xml"
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)
```

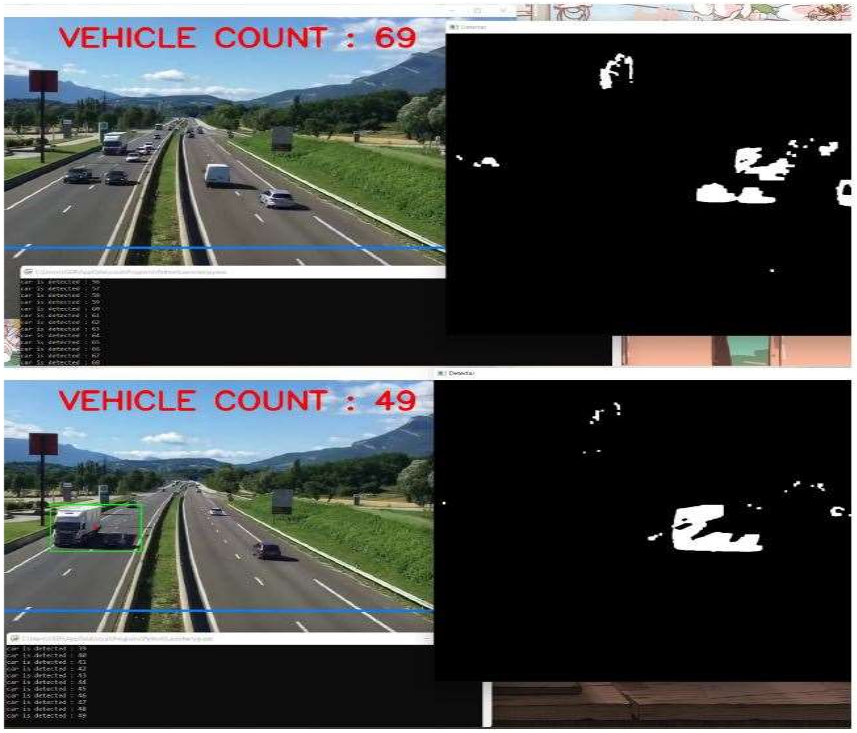
```
    display.clear_output(wait=True)
    display.display(i)
except KeyboardInterrupt:
    print("Interrupted")
cap.release()
cv2.destroyAllWindows()
VIDEO_SOURCE = "https://github.com/intel-iot-devkit/sample-videos/raw/master/people-detection.mp4"
import ipywidgets as widgets
import openvino as ov
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)
```





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



```
!pip install pyqrcode

!pip install pypng
!pip install IPython

import pyqrcode
import png
from pyqrcode import QRCode
from IPython.display import Image

s = "Mr Programmer github Link - https://github.com/dashboard"

url = pyqrcode.create(s)
url.png('myqr.png', scale=6)


Image(filename='myqr.png')
```