```
import cv2
import numpy as np
import tensorflow as tf

model = tf.saved_model.load('path_to_your_saved_model')
def detect_objects(frame):
    input_tensor = tf.convert_to_tensor(frame)
    input_tensor = input_tensor[tf.newaxis, ...]
    detections = model(input_tensor)
    return detections
video_path = 'path_to_your_video_file.mp4'
cap = cv2.VideoCapture(video_path)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    detections = detect_objects(frame)
 cv2.imshow('Object Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
tracker = cv2.TrackerCSRT_create()
def initialize_tracker(frame, bbox):
    tracker.init(frame, bbox)
def update_tracker(frame):
    ok, bbox = tracker.update(frame)
    if ok:
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (0, 255, 0), 2, 1)
    else:
        cv2.putText(frame, "Tracking failure detected", (100,80), cv2.FONT_HERSHEY_SIMPLEX,
0.75,(0,0,255),2)
    return frame
for detection in detections:
    bbox = detection['bbox']  # Extract bounding box coordinates
    initialize_tracker(frame, bbox)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    frame = update_tracker(frame)
    cv2.imshow('Object Tracking', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()


img_path = "images/blueangels.jpg"
prev_time = time.time()
img = Image.open(img_path)
detections = detect_image(img)
inference_time = datetime.timedelta(seconds=time.time() - prev_time)
print ('Inference Time: %s' % (inference_time))
cmap = plt.get_cmap('tab20b')
colors = [cmap(i) for i in np.linspace(0, 1, 20)]
img = np.array(img)
plt.figure()
fig, ax = plt.subplots(1, figsize=(12,9))
ax.imshow(img)
pad_x = max(img.shape[0] - img.shape[1], 0) * (img_size / max(img.shape))
```

```
pad_y = max(img.shape[1] - img.shape[0], 0) * (img_size / max(img.shape))
unpad_h = img_size - pad_y
unpad_w = img_size - pad_x
if detections is not None:
   unique_labels = detections[:, -1].cpu().unique()
   n_cls_preds = len(unique_labels)
   bbox_colors = random.sample(colors, n_cls_preds)
    for x1, y1, x2, y2, conf, cls_conf, cls_pred in detections:
       box_h = ((y2 - y1) / unpad_h) * img.shape[0]
       box_w = ((x2 - x1) / unpad_w) * img.shape[1]
       y1 = ((y1 - pad_y // 2) / unpad_h) * img.shape[0]
       x1 = ((x1 - pad_x // 2) / unpad_w) * img.shape[1]
       color = bbox_colors[int(np.where(
           unique_labels == int(cls_pred))[0])]
       bbox = patches.Rectangle((x1, y1), box_w, box_h,
           linewidth=2, edgecolor=color, facecolor='none')
       ax.add_patch(bbox)
       plt.text(x1, y1, s=classes[int(cls_pred)],
              color='white', verticalalignment='top',
              bbox={'color': color, 'pad': 0})
plt.axis('off')
plt.savefig(img_path.replace(".jpg", "-det.jpg"),
               bbox_inches='tight', pad_inches=0.0)
plt.show()
```