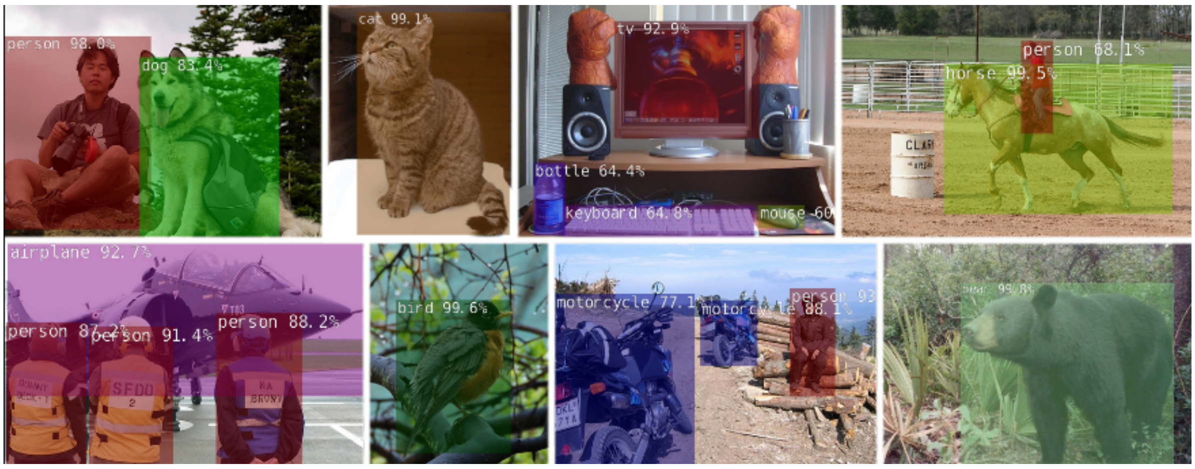
Object Detection Inference

1. Localizing Objects with DetectNet

The previous recognition example outputs class probabilities representing the entire input image. Next, we will focus on object detection and find the locations of various objects in the frame by extracting bounding boxes. Unlike image classification, object detection networks are able to detect many different objects per frame.



The detectNet object accepts an image as input and outputs a list of coordinates of the detected bounding boxes along with their class and confidence values. detectNet can be used from both Python and C++. See below for a variety of pre-trained detection models available for download. The default model used is the 91-class SSD-Mobilenet-v2 model trained on the MS COCO dataset, which achieves real-time inference performance on Jetson using TensorRT.

2. Detecting Objects from Images

First, let's try to use the detectnet program to localize objects in a static image. In addition to the input/output paths, there are some additional command-line options:

- Change the network flag of the detection model being used (default is SSD-Mobilenet-v2) (optional)
- optional --overlay flag, which can be a comma-separated combination of box, line, labels, conf, and none.

The default is --overlay=box,labels,conf, which displays the boxes, labels, and confidence values.

The cuboid option draws filled bounding boxes, while lines only draw unfilled outlines

- optional --alpha value, which sets the alpha blending value used in the overlay process (default is 120).
- threshold, which sets the minimum threshold for detection (default is 0.5).

If you are using a Docker container, it is recommended to save the output images to a directory mounted at images/test. Then, under jetsoninference/data/images/test, you can easily view these images from the host device.

Note: Before running the example, if you build your own environment, you need to download the resnet-18.tar.gz model file to the network folder to run the above program. You can directly enter the above program using the image we provide.

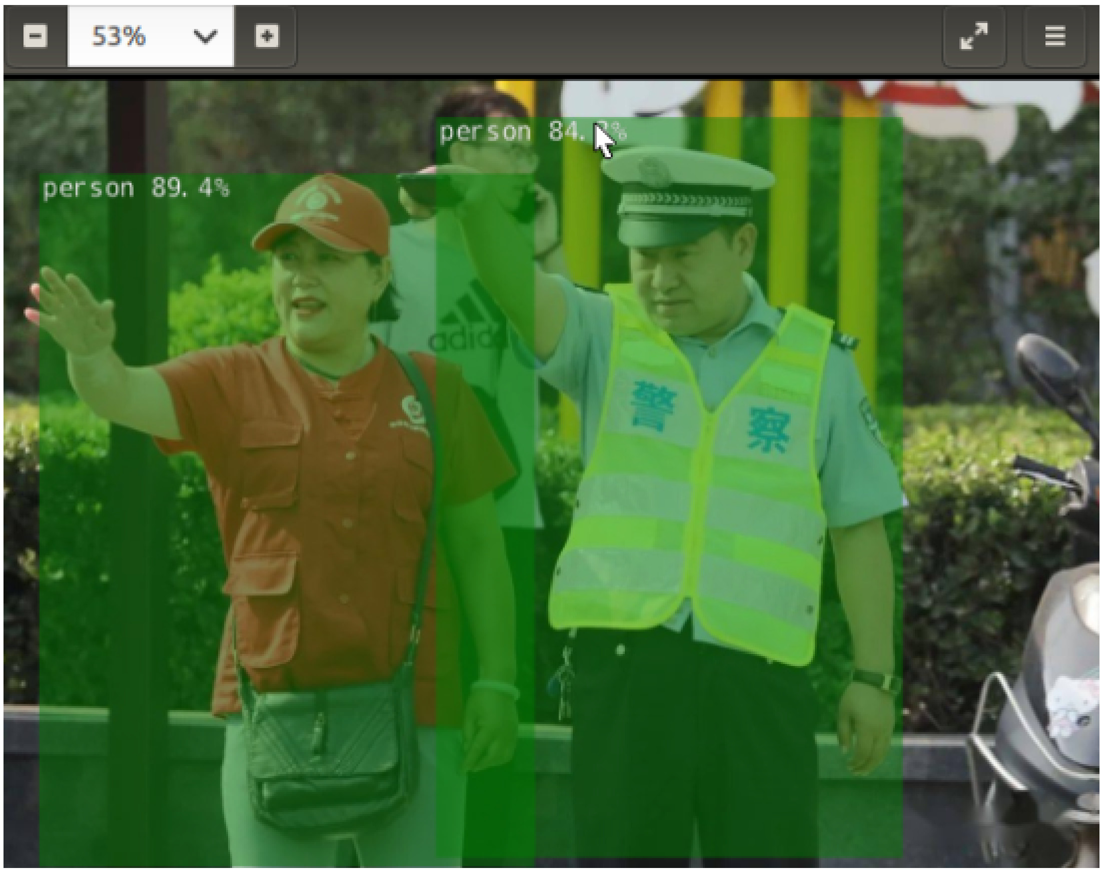
Make sure your terminal is in the aarch64/bin directory:

```
cd jetson-inference/build/aarch64/bin
```

Here are some examples of detecting pedestrians in an image using the default SSD-Mobilenet-v2 model:

```
# C++
$ ./detectnet --network=ssd-mobilenet-v2 images/xingren.png
images/test/output_xinren.png

# Python
$ ./detectnet.py --network=ssd-mobilenet-v2 images/xingren.png
images/test/output_xinren.png
```



Note: The first time you run each model, TensorRT will take several minutes to optimize the network. This optimized network file is then cached to disk, so future runs using that model will load faster.

3. Processing video files

You can store videos in the images folder, the path is

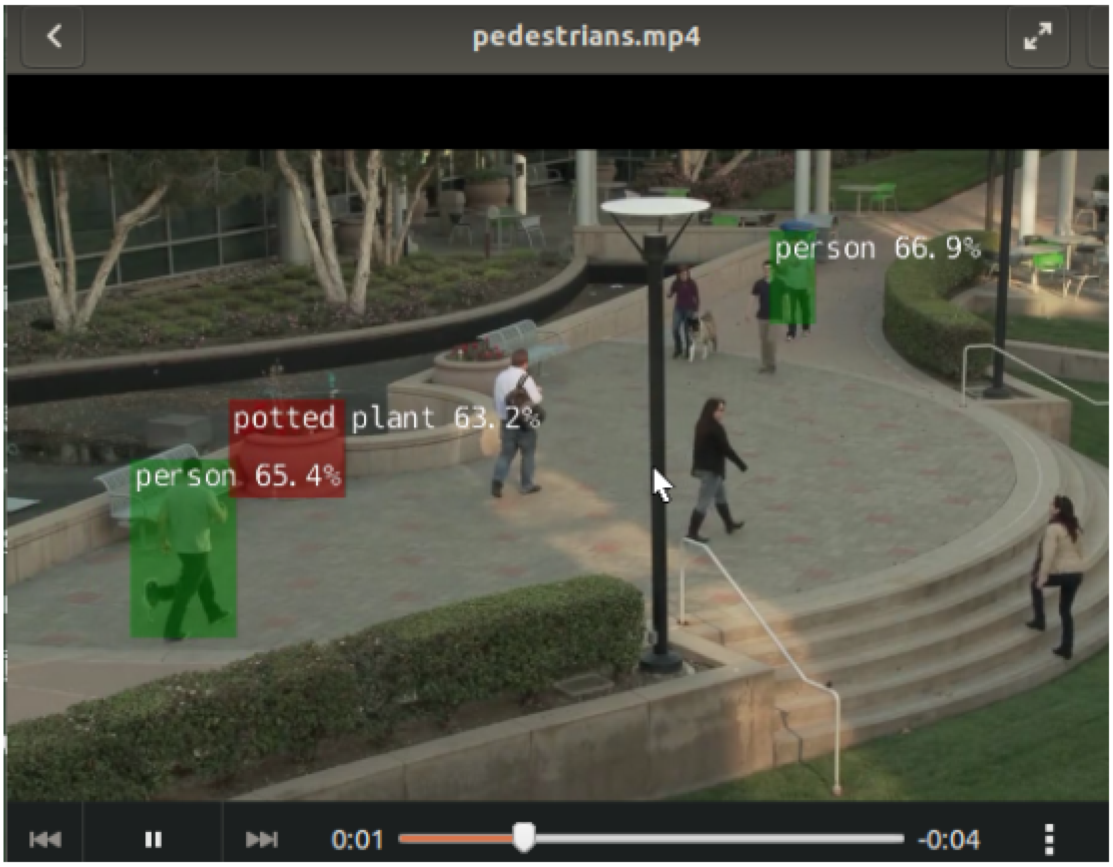
```
cd jetson-inference/build/aarch64/bin
```

Run the program:

```
# C++
./detectnet pedestrians.mp4 images/test/pedestrians_ssd.mp4

# Python
./detectnet.py pedestrians.mp4 images/test/pedestrians_ssd.mp4
```

Effect:



You can use the --threshold setting to change the detection sensitivity up or down (the default value is 0.5).

4. Run the real-time camera recognition demo

The detectnet.cpp/detectnet.py sample we used earlier can also be used for real-time camera streams. Supported camera types include:

- MIPI CSI cameras (csi://0)
- V4L2 cameras (/dev/video0)
- RTP/RTSP streams (rtsp://username:password@ip:port)

Here are some typical scenarios for launching a program on a camera feed.

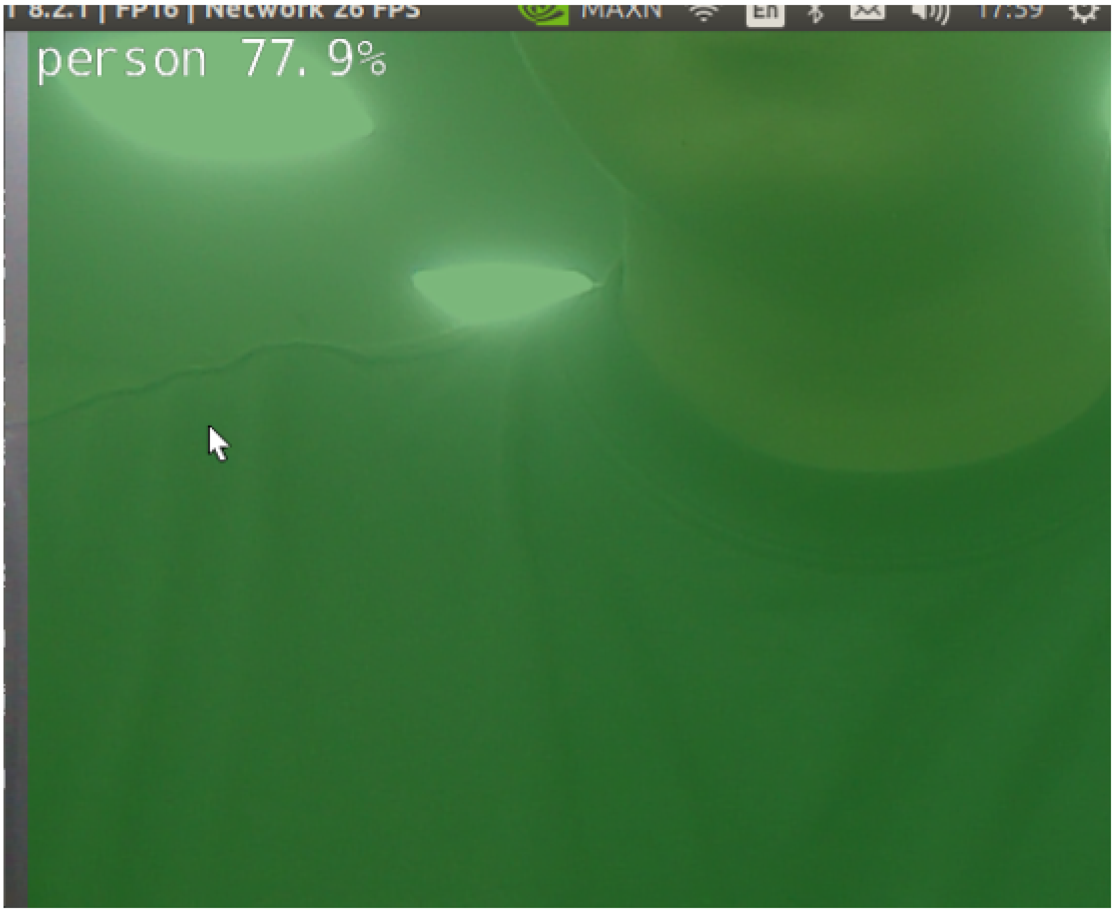
C++

```
$ ./detectnet csi://0 # MIPI CSI camera
$ ./detectnet /dev/video0 # V4L2 camera
$ ./detectnet /dev/video0 output.mp4 # save to video file
```

python

```
$ ./detectnet.py csi://0 # MIPI CSI camera
$ ./detectnet.py /dev/video0 # V4L2 camera
$ ./detectnet.py /dev/video0 output.mp4 # save to video file
```

The OpenGL window displays the live camera stream overlaid with bounding boxes of detected objects. Note that SSD-based models currently have the highest performance. Here is an example of using the model:



If the desired objects are not detected in the video feed, or you get spurious detections, try lowering or increasing the detection threshold using the --threshold parameter (default is 0.5).