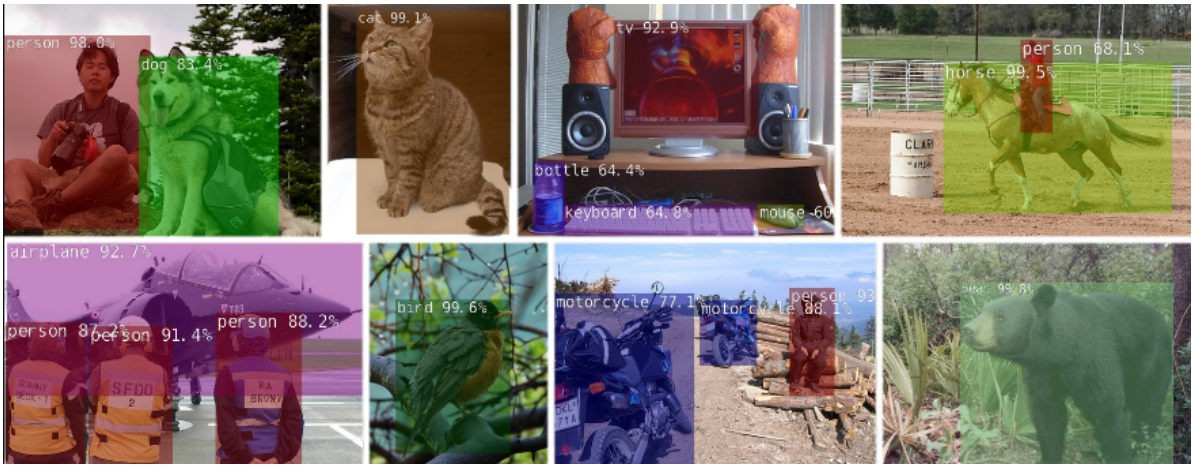
Target detection inference

1.Using DetectNet to locate objects

The previous recognition example output represents the class probability of the entire input image. Next, we will focus on object detection and find the positions of various objects in the frame by extracting bounding boxes. Unlike image classification, object detection networks can detect many different objects per frame.



The detectNet object accepts images as input and outputs a list of coordinates for the detected bounding boxes, along with their categories and confidence values. DetectNet can be used from Python and C++. For various pre trained detection models available for download, please refer to the following text. The default model used is the 91 class SSD-Mobilenet-v2 model trained on the MS COCO dataset, which achieved real-time inference performance using TensorRT on Jetson.

2.Detect objects from images

Firstly, let's try using the detectnet program to locate objects in static images. In addition to the input/output path, there are also some additional command-line options:

- Change the network flag of the detection model being used (default to SSD Mobilenet v2) (optional)
- The optional -- overlay flag can be a combination of box, line, labels, conf, and none separated by commas.The default value is -- overlay=box, labels, conf, which displays the box, label, and confidence value.The box option draws filled bounding boxes, while the line only draws unfilled contours
- Optional - Alpha value, which sets the alpha blend value used during the stacking process (default value is 120).
- Threshold, used to set the minimum threshold for detection (default value is 0.5).

If you are using a Docker container, it is recommended to save the output image to the directory mounted on images/test. Then, under Jetson inference/data/images/test, these images can be easily viewed from the host device

Note: Before running the case, if you are building your own environment, you must download the resnet-18.tar.gz model file to the network folder to run the above program. You can directly input the above program using the image we provide.

Please ensure that your terminal is located in the aarch64/bin directory:

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

Here are some examples of using the default SSD-Mobilenet-v2 model to detect pedestrians in images:

```
# 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: TensorRT will take a few minutes to optimize the network when running each model for the first time. This optimized network file is then cached on disk, so future runs using this model will load faster.

3.Processing video files

You can store videos in the images folder at the path

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

Run 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 (default value is 0.5).

4.Run real-time camera recognition demonstration

The detectnet.cpp/detectnet.py sample we previously used can also be used for real-time camera streaming. The 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 programs on camera feeds

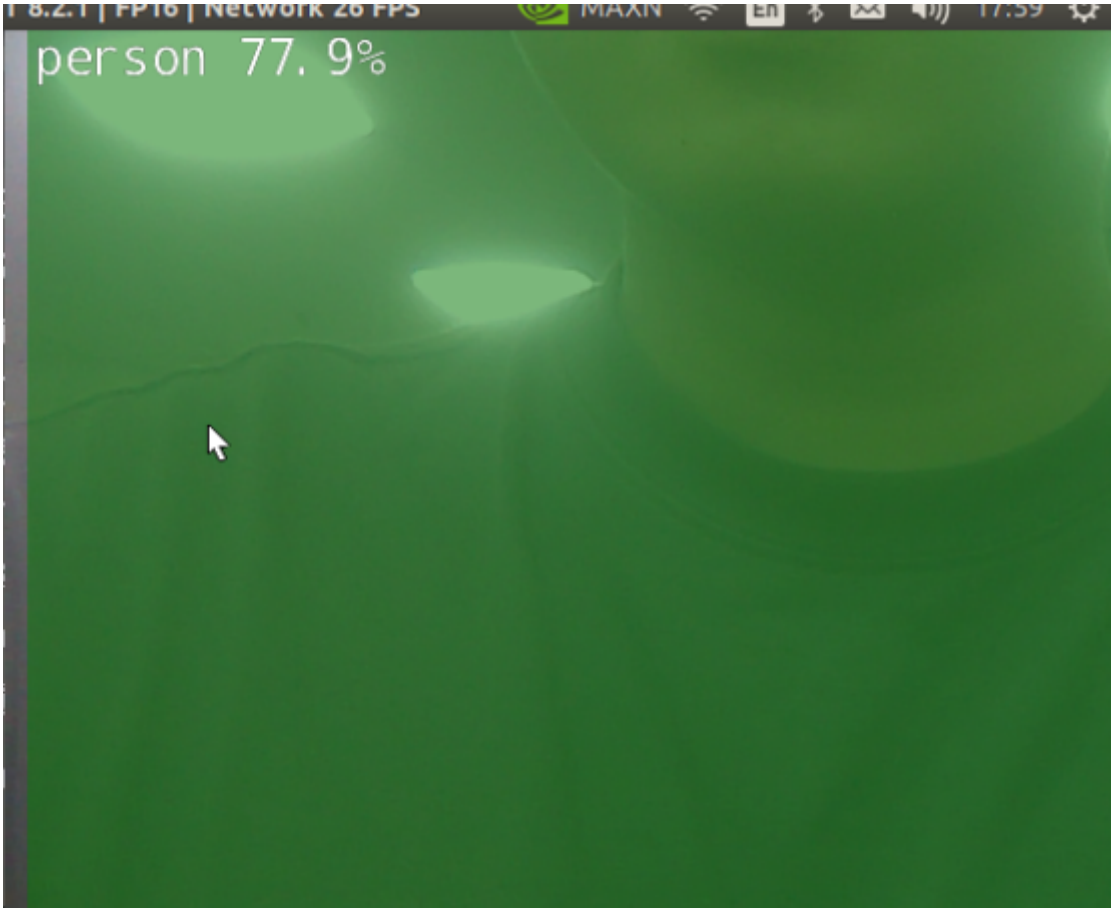
.C++

```
$ ./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 a real-time camera stream that covers the bounding box of the detected object. Please note that SSD based models currently have the highest performance. This is an example of using a model:



If the desired object is not detected in the video feed, or if you receive false detection, try using the -- threshold parameter to lower or increase the detection threshold (default value is 0.5).