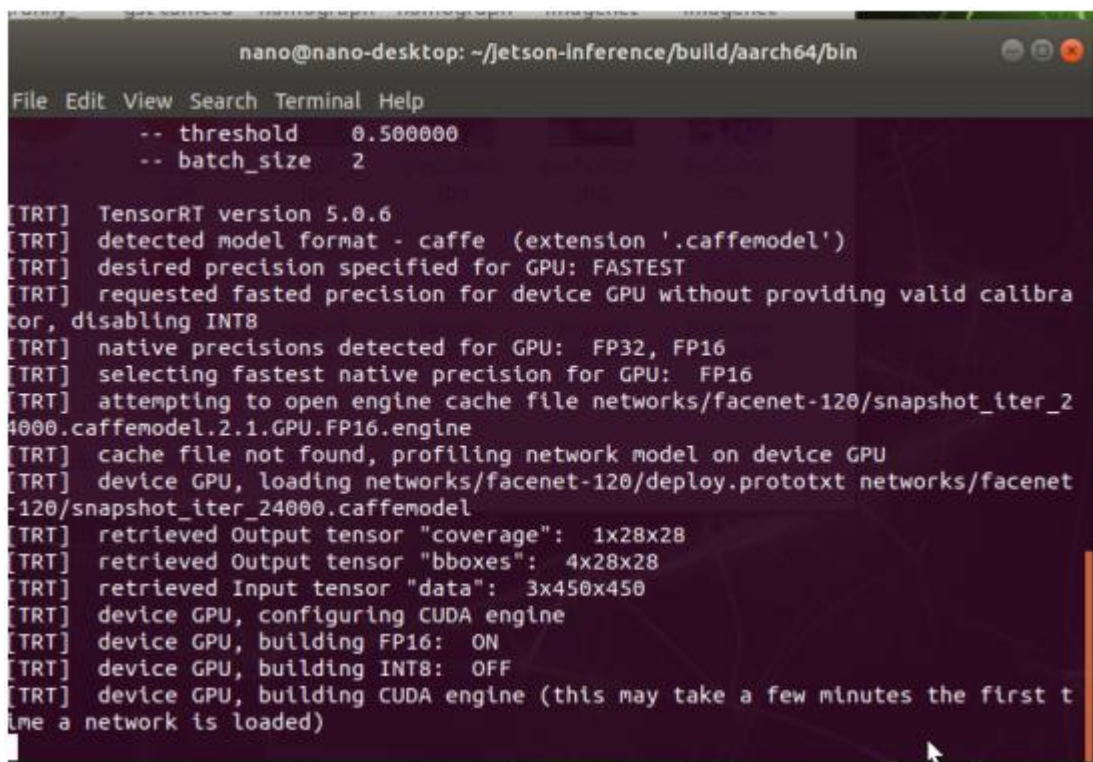


9.使用 DetectNet 运行实时摄像头检测演示

与前面的示例类似，在 `detectnet-camera` Jetson 板载摄像头的实时视频源上运行对象检测网络。从命令行启动它以及所需网络的类型：

```
$ ./detectnet-camera facenet          # 运行使用面部识别网络
$ ./detectnet-camera multyped         # 运行使用多级行人/行李探测器
$ ./detectnet-camera pednet           # 运行使用原始单级行人探测器
$ ./detectnet-camera coco-bottle      # 在摄像头下检测 瓶/汽水罐
$ ./detectnet-camera coco-dog         # 在摄像头下检测狗
$ ./detectnet-camera                  # 默认情况下，程序将运行使用 multyped
```

注意: 要在运行 `detectnet` 时获得最佳性能, 请通过运行脚本来增加 Jetson 时钟限制: `sudo ~/jetson_clocks.sh`



```
nano@nano-desktop: ~/jetson-inference/build/aarch64/bin
File Edit View Search Terminal Help
-- threshold 0.500000
-- batch_size 2

[TRT] TensorRT version 5.0.6
[TRT] detected model format - caffe (extension '.caffemodel')
[TRT] desired precision specified for GPU: FASTEST
[TRT] requested fastest precision for device GPU without providing valid calibration, disabling INT8
[TRT] native precisions detected for GPU: FP32, FP16
[TRT] selecting fastest native precision for GPU: FP16
[TRT] attempting to open engine cache file networks/facenet-120/snapshot_iter_24000.caffemodel.2.1.GPU.FP16.engine
[TRT] cache file not found, profiling network model on device GPU
[TRT] device GPU, loading networks/facenet-120/deploy.prototxt networks/facenet-120/snapshot_iter_24000.caffemodel
[TRT] retrieved Output tensor "coverage": 1x28x28
[TRT] retrieved Output tensor "bboxes": 4x28x28
[TRT] retrieved Input tensor "data": 3x450x450
[TRT] device GPU, configuring CUDA engine
[TRT] device GPU, building FP16: ON
[TRT] device GPU, building INT8: OFF
[TRT] device GPU, building CUDA engine (this may take a few minutes the first time a network is loaded)
```

上面执行过程中，第一次执行每个都会执行更新模型，需要花费很长时间，需要大家耐心等待，下次执行就直接执行。

注意：默认情况下，Jetson 的板载 CSI 摄像机将用作视频源。如果您希望使用 USB 网络摄像头

与前面的 `detectnet-console` 示例类似，这些相机应用程序使用检测网络，只是它们处理来自相机的实时视频。`detectnet-camera` 接受各种可选的命令行参数，包括：

- `--network` 标志，该标志更改正在使用的[检测模型](#)（默认为 SSD-Mobilenet-v2）。
- `--overlay` 标志，该标志可以是逗号分隔的组合 `box`，`labels`，`conf`，和 `none`
- 默认值为 `--overlay=box,labels,conf` 显示框，标签和置信度值

- `--alpha` 设置覆盖时使用的 Alpha 混合值的值（默认值为 `120`）。
- `--threshold` 设置检测最低阈值的值（默认为 `0.5`）。
- `--camera` 标志设置要使用的摄像头设备
- 通过指定传感器索引（`0` 或 `1` 等）来使用 MIPI CSI 摄像机
- **V4L2USB** 摄像机通过指定其 `/dev/video` 节点（`/dev/video0`，`/dev/video1` 等）使用。
- 默认为使用 MIPI CSI 传感器 `0`（`--camera=0`）
- `--width` 和 `--height` 标志设置相机分辨率（默认为 `1280x720`）
- 分辨率应设置为相机支持的格式。
- 使用以下命令查询可用格式：

```
$ sudo apt-get install v4l-utils
$ v4l2-ctl --list-formats-ext
```

您可以根据需要组合使用这些标志，并且还有其他命令行参数可用于加载自定义模型。启动带有 `--help` 标志的应用程序以获取更多信息，或参阅 [Examples](#) 自述文件。

以下是启动程序的一些典型方案：

C ++

```
$ ./detectnet-camera # 使用 PedNet，默认 MIPI CSI 相机（1280x720）
$ ./detectnet-camera --network=facenet # 使用 FaceNet，默认 MIPI CSI 相机（1280x720）
$ ./detectnet-camera --camera=/dev/video1 # 使用 PedNet，V4L2 摄像机/dev/video1（1280x720）
$ ./detectnet-camera --width=640 --height=480 # 使用 PedNet，默认 MIPI CSI 摄像机（640x480）
```

Python

```
$ ./detectnet-camera.py # 使用 PedNet，默认 MIPI CSI 摄像机（1280x720）
$ ./detectnet-camera.py --network=facenet # 使用 FaceNet，默认 MIPI CSI 摄像机（1280x720）
$ ./detectnet-camera.py --camera=/dev/video1 # 使用 PedNet，V4L2 摄像机/dev/video1（1280x720）
$ ./detectnet-camera.py --width=640 --height=480 # 使用 PedNet，默认为 MIPI CSI 摄像机（640x480）
```

注意：例如摄像机使用，见杰特森维基的以下部分：

-纳米： -泽维尔： - TX1 / TX2：显影剂盒包括车载 MIPI CSI 传感器模块（OV5693）

https://eLinux.org/Jetson_Nano#Cameras

https://eLinux.org/Jetson_AGX_Xavier#Ecosystem_Products_.26_Cameras

可视化

OpenGL 窗口中显示的是实时摄像机流，上面覆盖了检测到的对象的边界框。请注意，当前基于 SSD 的型号具有最高的性能。这是使用该 **coco-dog** 模型的一种：

```
# C ++  
$ ./detectnet-camera --network=coco-dog  
  
# Python 的  
$ ./detectnet-camera.py --network=coco-dog
```

如果视频馈送中未检测到所需的对象，或者您得到的是虚假检测，请尝试使用 **--threshold** 参数降低或提高检测阈值（默认值为 **0.5**）。

执行第一个命令后如下：



可以同时检测多个人脸。

下一步是什么

这是 *Hello AI World* 教程的最后一步，该教程涵盖了 Jetson 与 TensorRT 的推理。

总结一下，我们已经涵盖了：

- 使用图像识别网络对图像进行分类
- 用 C ++编写自己的图像识别程序

- 从实时摄像机流中分类视频
- 执行对象检测以定位对象坐标

接下来，我们建议您遵循我们的完整培训+推理教程，该教程还包括在自定义数据集上重新培训这些网络。这样，您可以收集自己的数据，并让模型识别特定于您的应用程序的对象。完整的教程还包括语义分割，它类似于图像分类，但是在每个像素级别上，而不是为整个图像预测一个类。祝好运！