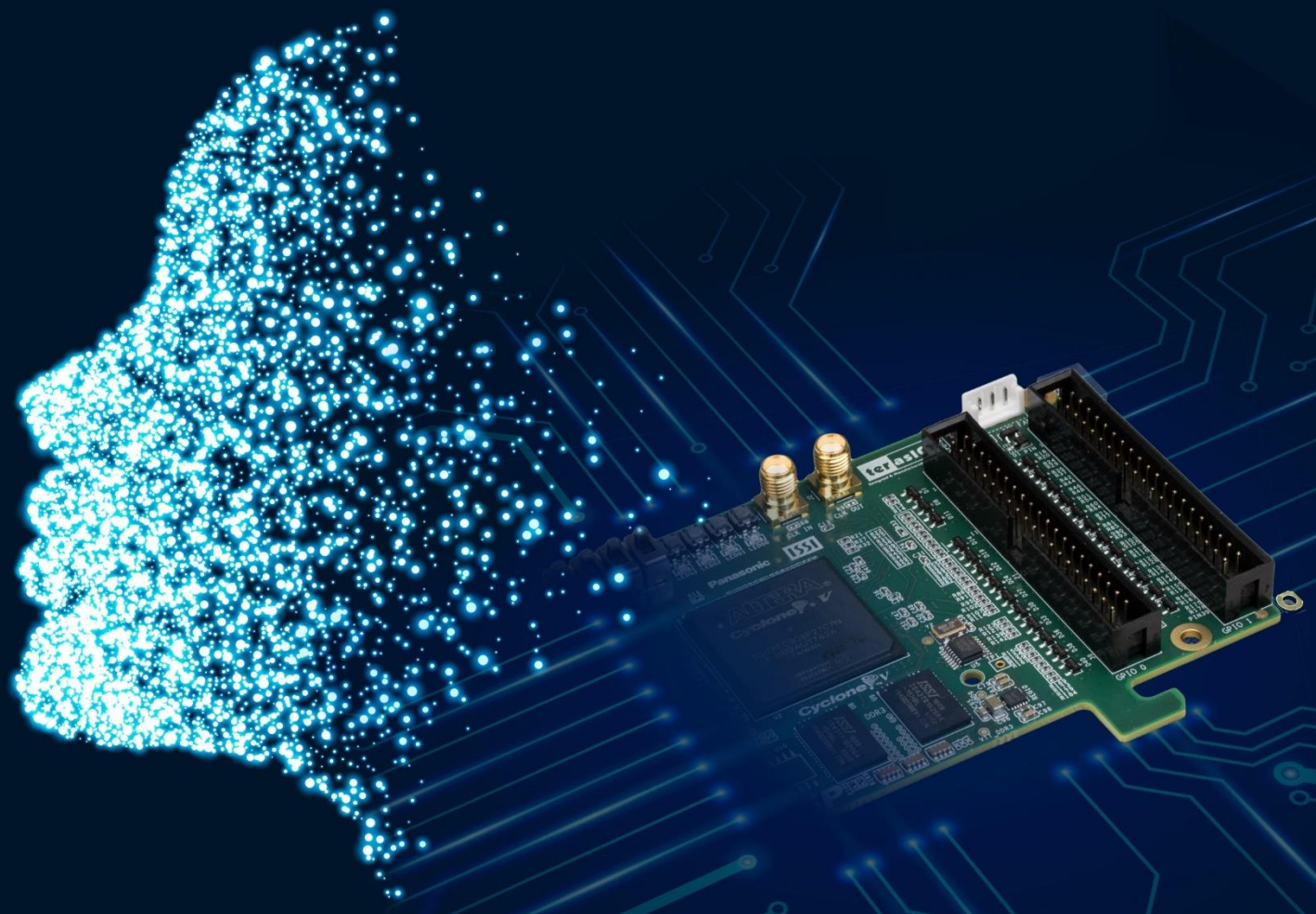
OpenVINO
Development Guide



terasIC
www.terasic.com

目录

Chapter 1

OpenVINO 工具套件及开发过程简介

1.1 关于本教程

OpenVINO(即开放可视化推理及神经网络优化工具包)开发指导教程涵盖了 OpenVINO 工具包的工作流程及实验指导。本教程将向用户演示如何使用英特尔 OpenVINO 工具包部署模拟人类视觉的应用程序和解决方案。利用异构执行加速实现了基于 CNN 的深度学习。使用易于使用的计算机视觉函数库，OpenVINO 工具包加快了产品的上市时间。本教程还向用户演示了如何快速设置在 FPGA 上运行的基于 CNN 的深度学习应用程序。

本教程是基于 Terasic OpenVINO Starter Kit 来编写的（用户也可以参考本教程来进行 DE5a-Net-DDR4 和 DE5a-Net 开发板的 OpenVINO 开发），涵盖以下内容：

- OpenVINO 工具包的简介
- OpenVINO 工作流程
- 模型优化器和推理引擎
- 在 OpenVINO Starter Kit 上运行案例
- OpenVINO Starter Kit 实验

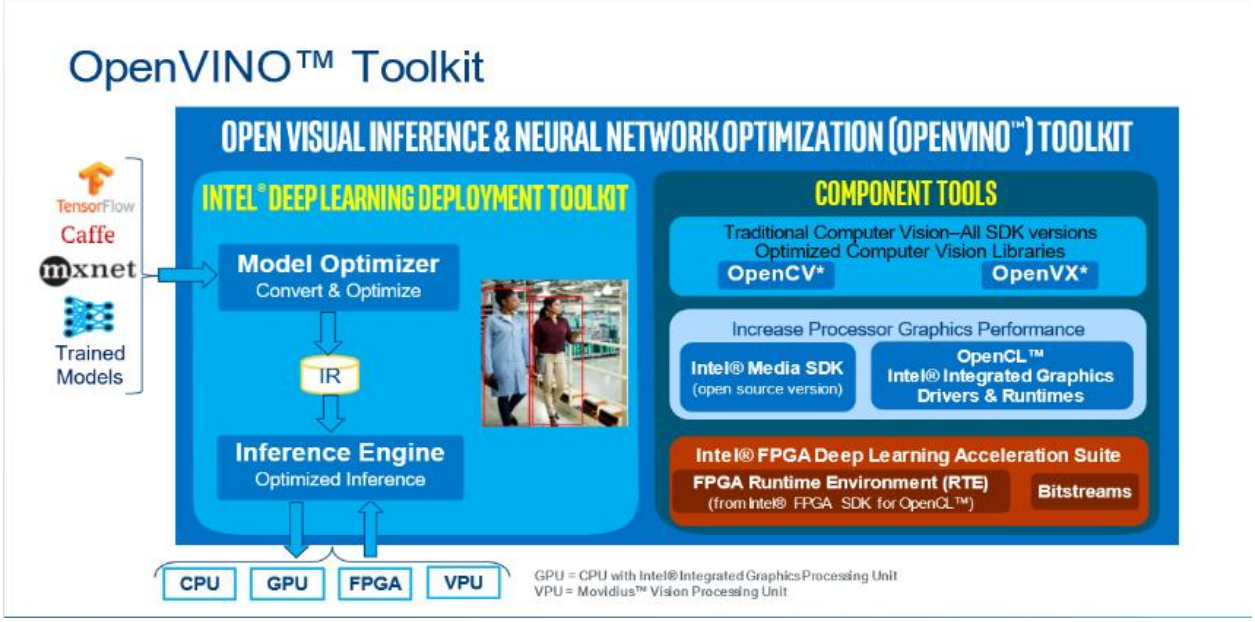
1.2 OpenVINO 工具包的特点

OpenVINO (开放可视化推理及神经网络优化) 工具包可以提高计算机视觉的性能，缩短产品上市所需的时间。它可以帮助用户轻松地利用 Terasic FPGA 开发板的优势，包括提高性能，降低功耗，显著提高 FPGA 的利用率等。用户效率可以事半功倍，并开辟新的设计可能性。主要特点是:

- 支持在边缘设备上进行 CNN 深度学习推理
- 支持跨 Intel 计算机视觉加速器的异构加速执行，CPU，英特尔集成图形化，英特尔神经计算棒及 FPG。
- 使用统一的 API 通过易于使用的计算机视觉函数和预先优化的内核加速上市时间。
- 包括对 OpenCV*, OpenCL™, and OpenVX*等对算机视觉标准的优化调用。

1.3 OpenVINO 工具包包含哪些内容

OpenVINO 使用统一的基于通用的开发标准（如 OpenCL、OpenCV 和 OpenVX）的 API。



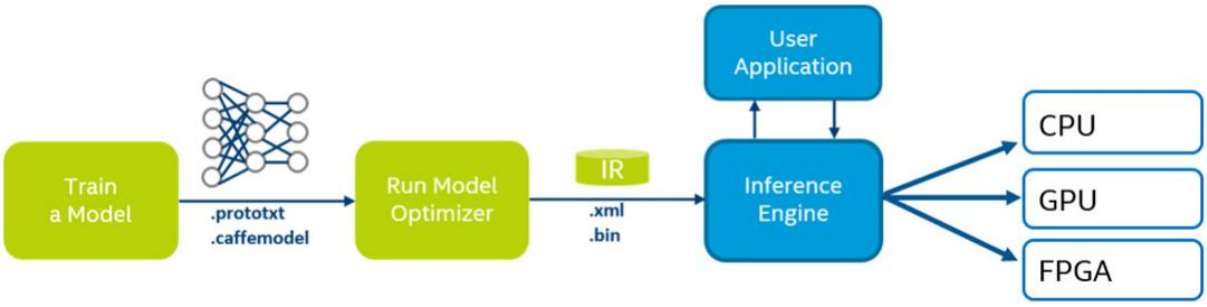
该工具包包括：

1. 深度学习部署工具包，主要有以下两个模块：
   - 模型优化器：基于 Python 的命令行工具从主流的深度学习框架（如 caffe，TensorFlow, Apache MXNet 等）导入训练过的模型。 输入训练模型，优化网络拓扑，并将其转换为 IR(中间表示)文件
   - 推理引擎：推理引擎使用统一的 API 在您选择的平台（如 CPU, GPU, VPU， FPGA）等提供优化的推理解决方案，来进行异构处理和异步执行以节省开发时间。
2. 用于 OpenCV, OpenVX 的优化的计算机视觉库，以及用于 CPU,GPU 的图像视觉
3. 改进了英特尔处理器显卡组件在 Linux 中的性能，包括英特尔媒体 SDK 开源版本、OpenCL 图形驱动程序和运行库环境。
4. 运行库环境支持在 FPGA 上运行 OpenCL,支持配置 FPGA 的比特流。

1.4 OpenVINO 的工作流程

以下是使用 OpenVINO 优化并部署训练后的模型的步骤：

1. 为您的框架配置模型优化器
2. 转换训练后的模型，根据训练的网络拓扑、权重和偏置值生成模型的优化中间表示 （IR）文件
3. 通过验证应用程序或示例应用程序，在目标环境中使用推理引擎以中间表示格式测试模型
4. 在应用程序中集成推理引擎，以在目标环境中部署模型

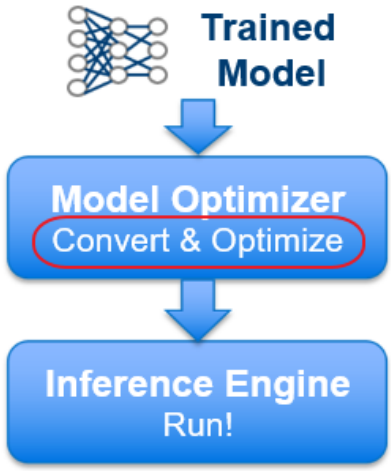


1.5 Model Optimizer 模型优化器

模型优化器是一个跨平台的命令行工具，可以促使训练和部署环境之间的转换，执行静态模型分析，调整深度学习模型，以便在端目标设备上实现最佳执行。

模型优化器生成 OpenVINO 支持的框架作为训练过的模型输入，并将网络的中间表示形式(IR)作为输出。中间表示是描述整个模型的一对文件:

- .xml:描述网络拓扑
- .bin:包含权重和偏差二进制数据



■ 模型优化器是如何工作的？

模型优化器将模型加载到内存中，然后读取模型并构建模型的内部表示形式。然后，模型优化器对其进行优化并生成中间表示格式。中间表示格式是推理引擎接受的唯一格式。模型优化器有两个主要目的：

1) 生成有效的中间表示格式

如果此转换项目无效，那么推理引擎将无法运行。模型优化器的主要职责是生成构成中间表示格式的两个文件。

2) 生成优化的中间表示格式

预训练的模型包含对训练很重要的层，例如 dropout 层。这些层在推理时是无用的，可能会增加推理时间。

在许多情况下，这些层可以自动从生成的中间表示格式中删除。但是，如果一组层可以表示为一个数学运算，从而表示为单层，则模型优化器可识别此类模式，并将这些层替换为单层。结果是

中间表示格式具有比原始模型更少的层。这减少了推理时间。

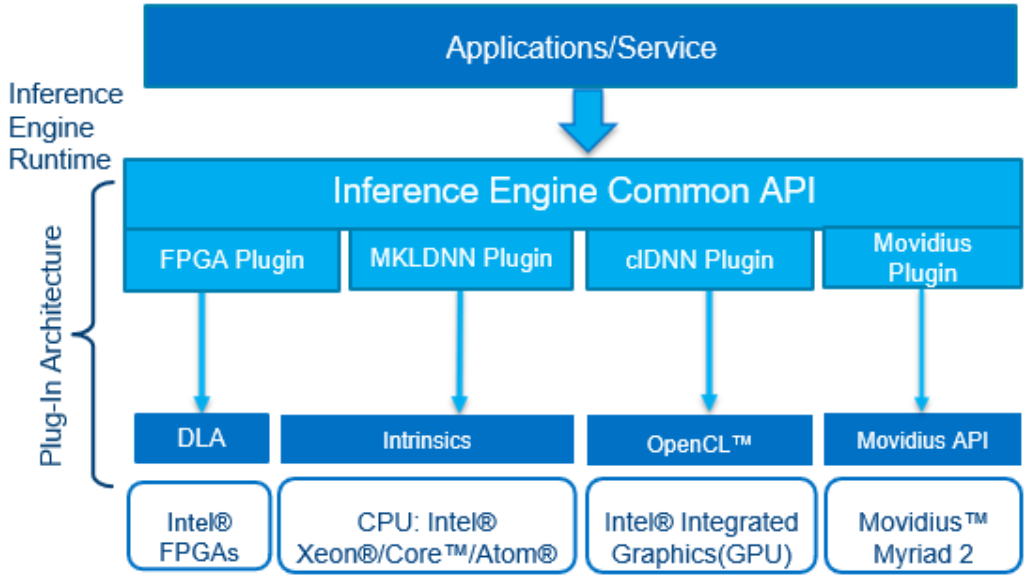
许多通用层存在于已知的框架和神经网络拓扑中。这些层的示例包括卷积、池化和激活。模型优化器必须能够使用这些层来读取原始模型并生成模型的中间表示格式，层列表因框架而异。关于每个框架所支持的拓扑，请参考 Caffe*、TensorFlow*和 MXNet*的文档。如果您的拓扑只包含层列表中的层（大多数用户使用的拓扑就是这种情况），那么模型优化器可以轻松地创建中间表示格式。之后，用户可以继续使用推理引擎。

但是，如果对模型优化器无法识别的层使用拓扑，请参阅模型优化器中的自定义图层，了解如何使用自定义图层。

1.6 推理引擎

模型优化器创建中间表示格式之后，就可以通过推理引擎输入数据。

推理引擎是一个具有一组 C++类的 C++库，用来推断输入数据(图像)并得到结果。C++库提供了一个 API 来读取中间表示格式，设置输入和输出格式，并在设备上执行模型。



每个支持的目标设备都有一个插件，每个插件都是一个 DLL/共享库。异构插件允许您跨设备分配计算工作负载。需要确保这些库在主机 PC 的路径中指定，或者在指向插件加载器的位置指定。此外，每个插件的相关库必须包含在 LD_LIBRARY_PATH 中。当推理引擎调用基于 FPGA 的 DLA 插件时，将调用 DLA 运行库软件层以使用 DLA API。这些 API 将被转换为在 FPGA 设备上执行的相应模块。这部分将在深度学习网络中的不同层中进行执行。

使用推理引擎的统一工作流程：

1. 读取中间表示格式
   使用推理引擎：：CNNNetReaderclass，并将中间表示文件读入 CNNNetwork 类中。此类表示主机内存中的网络。

2. 准备输入和输出格式：
   加载网络后，使用 CNNNetwork：：getInputInfoinfo（） 和 CNNNetwork：：getOutputInfo（）指定输入和输出精度以及网络上的布局。

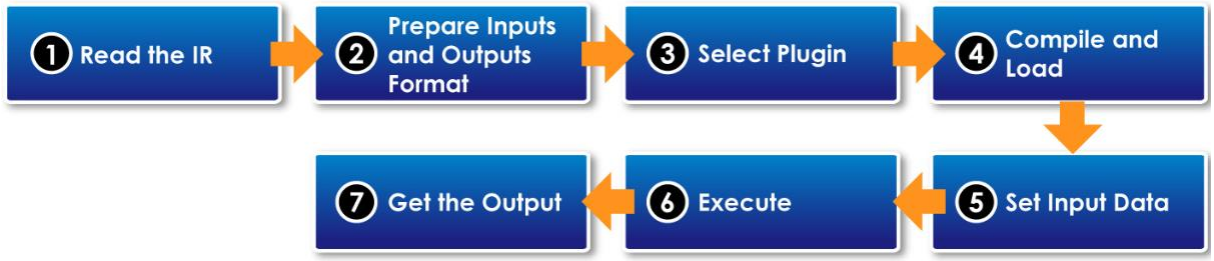
3. 选择插件
   选择插件以加载您的网络。使用 InferenceEngine:: PluginDispatcher 插件加载助手类创建插件。通过每个设备加载配置特定于这个设备和注册扩展到这个设备。
4. 编译并加载
   使用插件接口包装类 InferenceEngine::InferencePlugin 调用 LoadNetwork （） API 来编译和加载设备上的网络。为该编译和加载操作传入每个目标的加载配置。。
5. 设置输入数据
   这里有一个可执行的网络对象。

   使用此对象创建一个推论请求（InferRequest），在该推论请求中，您将向输入缓冲区发出信号，以用于输入和输出。指定一个设备分配的内存并将其直接复制到设备内存中，或者告诉设备使用您的应用程序内存来保存一个副本。
6. 执行
   选择定义了输入和输出内存的执行模式:

   同步——Infer()方法。阻塞，直到推理结束。

   异步——StartAsync()方法。使用 wait()方法(0 超时)、wait 或指定完成回调来检查状态。
7. 获取输出
   获取输出内存或在推理完成后读取前面提供的内存。

   这可以通过使用 InferRequest GetBlob API 来完成。



关于在应用程序中集成推理引擎的更多信息，请参阅 Inference Engine Developer Guide.

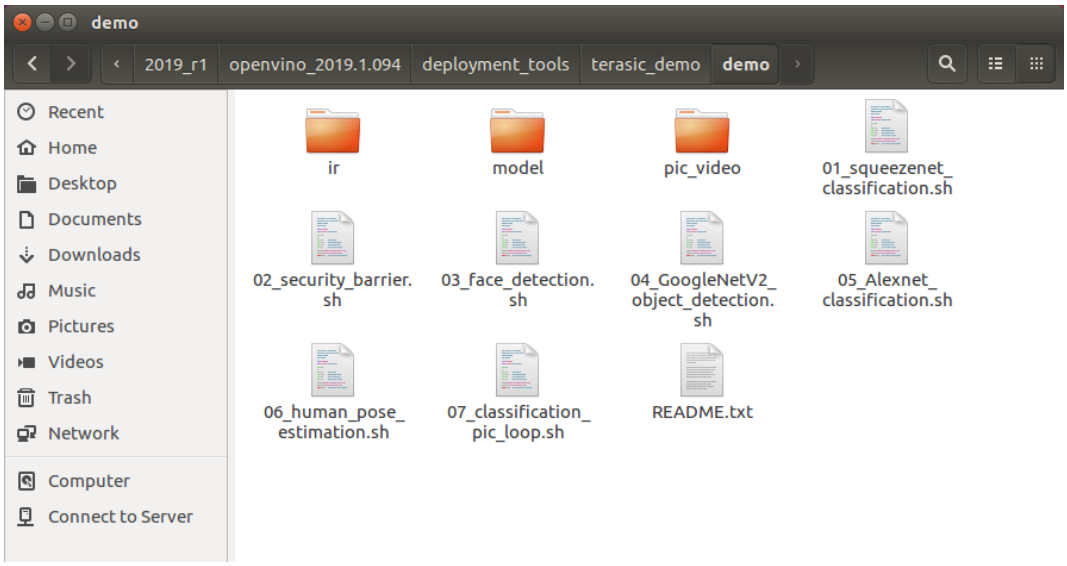
Chapter 2

在 OpenVINO 上运行例程

本章描述了如何在 OpenVINO Starter Kit 上运行例程，并展示了例程的执行结果。在运行这些例程之前，用户需要参考手册“OpenVINO_Installation_Guide”来完成 OpenVINO 开发包的安装。

2.1 简介

如下图所示，terasic_demo/demo 文件夹中有一些 shell 脚本。



下面是关于 demo 文件夹的简介

1. 如何使用这些 Shell 脚本文件

   用户可以使用默认参数运行任意一个 shell 脚本文件，还可以使用 cpu、vpu 或 fpga 来指定要运行例程的目标设备。还有其他参数可供使用，用户可以运行带有'-h'的 shell 脚本文件以获得更多细节。默认参数是 cpu。

2. 该例程所需的图像和视频位于 pic_video 文件夹中。

3. 从网上下载的 Caffe 框架位于 model 文件夹中

   - alexnet
   - squeezenet1.1
   - GoogleNetV2
   - 用户可以参考脚本的编写规则来添加 Caffe 框架。请注意路径和名称。
   - 请参考 OpenVINO-Using-TensorFlow 来传输 Tensorflow 模型

4. IR 文件夹

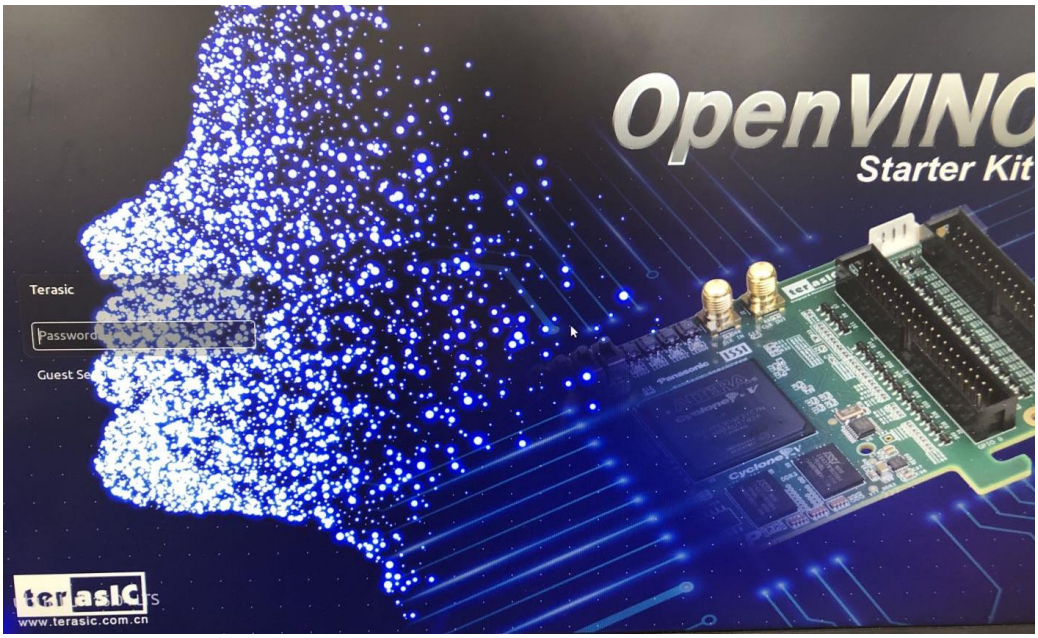
   运行演示时，如果有需要，将自动生成相应模型的 IR 文件。

- FP16 文件夹下生成的模型用于 FPGA
- FP32 文件夹下生成的模型用于 CPU

2.2 执行例程

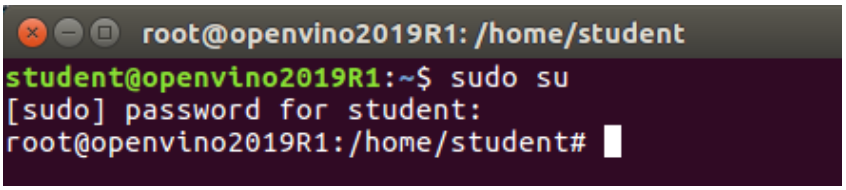
本实验描述了如何在 FPGA 上运行例程，并展示了例程的执行结果。

在演示例程之前，需要用预装有 Linux 系统的 U 盘在主机上启动 Linux 系统：将 U 盘插入主机 USB 口，并打开主机电源，启动系统后，会出现如下图所示界面，输入 terasic 键进入桌面。



OpenVINO 工具包中包含了七个演示例程。要运行这些例程，用户需要打开终端并输入 sudo su，通过 source 脚本文件来设置运行环境。

1. 桌面上单击右键，选择 Open Terminal 打开终端，输入命令“sudo su”切换到 root 超级用户权限，输入密码“terasic”。



2. 使用 cd /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo 命令切换 terasic_demo 路径。
3. 执行 source setup_board_osk.sh 设置环境变量。

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo
student@openvino2019R1:~$ sudo su
[sudo] password for student:
root@openvino2019R1:/home/student# cd /opt/intel/2019_r1/openvino/deployment_too
ls/terasic_demo/
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo# s
ource setup_board_osk.sh
```

4. 输入“y” 配置环境。

```
aoc was not found, but aocl was found. Assuming only RTE is installed.

AOCL_BOARD_PACKAGE_ROOT is set to /opt/altera/aocl-pro-rte/aclrte-linux64/board/osk. Using that
.
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/bin to PATH
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/host/linux64/lib to LD_LIBRARY_PATH
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/board/osk/linux64/lib to LD_LIBRARY_PATH
Do you want to install /opt/altera/aocl-pro-rte/aclrte-linux64/board/osk? [y/n] y
```

5. 输入 cd demo 命令，切换到 demo 路径下。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo# c
d demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo#
```

6. 执行例程

■ **01_squeezenet demo**

该例程使用 squeezenet 模型来识别图片中的物体。

1) 输入*./01_squeezenet_classification.sh fpga 命令（在 FPGA 上运行）*。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo# c
d demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./01_squeezenet_classification.sh fpga
```

2） 可以看到调用的是 HETERO: FPGA, CPU 插件，提示例程在 FPGA 和 CPU 上运行。

```
File  Edit  View  Search  Terminal  Help

Run Inference Engine classification sample
Run ./classification_sample -d HETERO:FPGA,CPU -i /opt/intel/2019_r1/openvino/de
ployment_tools/terasic_demo/demo/pic_video/car.png -m /opt/intel/2019_r1/openvin
o/deployment_tools/terasic_demo/demo/ir/FP16/squeezenet1.1/squeezenet1.1.xml[ IN
FO ] InferenceEngine:
        API version ............ 1.6
        Build .................. custom_releases/2019/R1_c9b66a26e4d65bb986bb740
e73f58c6e9e84c7c2
[ INFO ] Parsing input parameters
[ INFO ] Files were added: 1
[ INFO ]     /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/pic_
video/car.png
[ INFO ] Loading plugin

        API version ............ 1.6
        Build ..................... heteroPlugin
        Description ....... heteroPlugin
[ INFO ] Loading network files:
        /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/ir/FP16/s
queezenet1.1/squeezenet1.1.xml
        /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/ir/FP16/s
queezenet1.1/squeezenet1.1.bin
[ INFO ] Preparing input blobs
```

2) 打印出前 10 项概率最高的识别结果。

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de

Image /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/pic_video/c
ar.png

classid probability label
------- ----------- -----
817     0.8363336   sports car, sport car
511     0.0946490   convertible
479     0.0419133   car wheel
751     0.0091072   racer, race car, racing car
436     0.0068162   beach wagon, station wagon, wagon, estate car, beach waggon,
 station waggon, waggon
656     0.0037564   minivan
586     0.0025741   half track
717     0.0016069   pickup, pickup truck
864     0.0012027   tow truck, tow car, wrecker
581     0.0005882   grille, radiator grille


total inference time: 25.3000632
Average running time of one iteration: 25.3000632 ms

Throughput: 39.5255930 FPS

[ INFO ] Execution successful
```

■ **02_security_barrier demo**

该例程通过使用三种模型识别汽车，车牌及车在图中的位置。

1) *./02_security_barrier.sh fpga* （在 FPGA 上运行）。

```
Demo completed successfully.

root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./02_security_barrier.sh fpga
```

2） 运行结果如下。输入 Ctrl+C 可关闭应用程序。

■ **03_face_detection**

该例程使用了四个模型，它可以识别人脸在图中的位置。它还可以根据人脸判断人的性别、年龄、表情和头部姿势。执行该例程一般需要摄像头，但是本次实验在 U 盘里面已经预置了演示的视频，我们直接执行看效果。

a) 执行“*./03_face_detection.sh fpga video*”命令，在 FPGA 上运行。（如果想实时检测来自摄像头的图像，可以直接运行命令*./03_face_detection.sh fpga 即可*）。

```
To close the application, press 'CTRL+C' or any key with focus on the output win
dow
^Croot@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/
mo# ./03_face_detection.sh fpga video
```

2） 运行结果如下所示，输入 Ctrl+c 关闭应用程序。

■ **04_GoogleNetV2_object_detection**

该例程使用 GoogleNetV2 模型识别目标对象。

1）执行“*./04_GoogleNetV2_object_detection.sh fpga video*”命令，在 FPGA 上运行。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./04_GoogleNetV2_object_detection.sh fpga video
```

2) 结果如下图所示。输入 Ctrl+c 关闭 Detection results 窗口。

■ **05_Alexnet_classification**

该例程可以使用 Alexnet 模型识别目标对象，并打印前 10 项信息（前 10 项概率最高的识别结果）。

1）执行“*./05_Alexnet_classification.sh fpga*”， 在 fpga 上运行。

```
###########################################


Demo completed successfully.

root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./05_Alexnet_classification.sh fpga
```

2）运行结果如下图所示。

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de

Image /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/pic_video/c
ar.png

classid probability label
------- ----------- -----
479     0.7561781   n02974003 car wheel
511     0.0755699   n03100240 convertible
436     0.0730273   n02814533 beach wagon, station wagon, wagon, estate car, bea
ch waggon, station waggon, waggon
817     0.0460277   n04285008 sports car, sport car
656     0.0303794   n03770679 minivan
661     0.0055283   n03777568 Model T
581     0.0031296   n03459775 grille, radiator grille
468     0.0029876   n02930766 cab, hack, taxi, taxicab
717     0.0022792   n03930630 pickup, pickup truck
627     0.0016297   n03670208 limousine, limo


total inference time: 90.5765519
Average running time of one iteration: 90.5765519 ms

Throughput: 11.0403850 FPS

[ INFO ] Execution successful
```

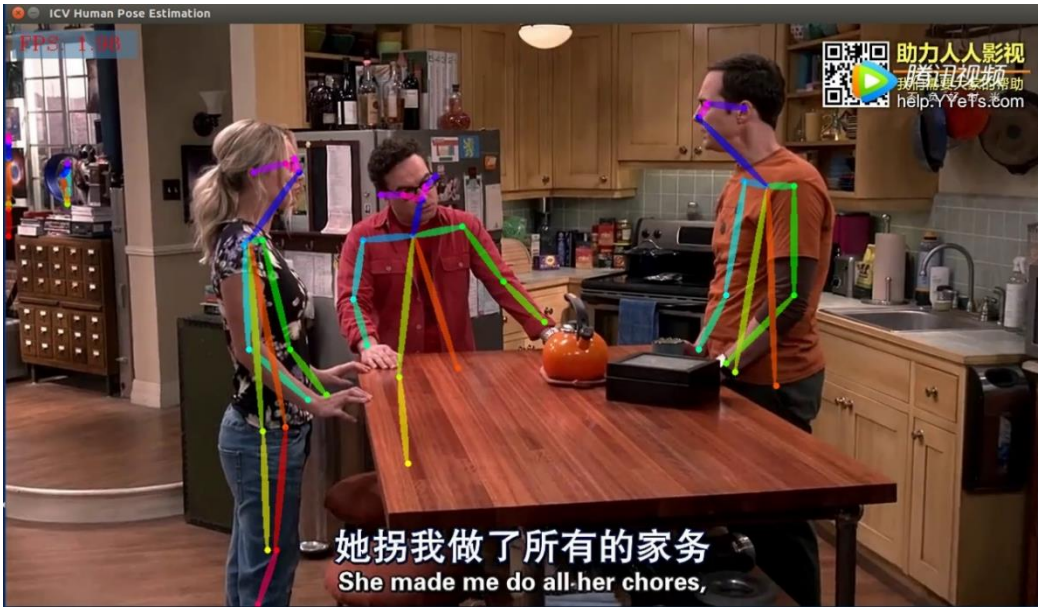
■ **06_human_pose_estimation**

该例程可以识别人体姿势并进行显示。

1）执行“./06_human_pose_estimation.sh fpga video”，在 FPGA 上运行

```
Throughput: 15.5517462 FPS

[ INFO ] Execution successful


###########################################


Demo completed successfully.

root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./06_human_pose_estimation.sh fpga video
```

2） 运行结果如下图所示。输入 Ctrl+C 关闭应用程序。



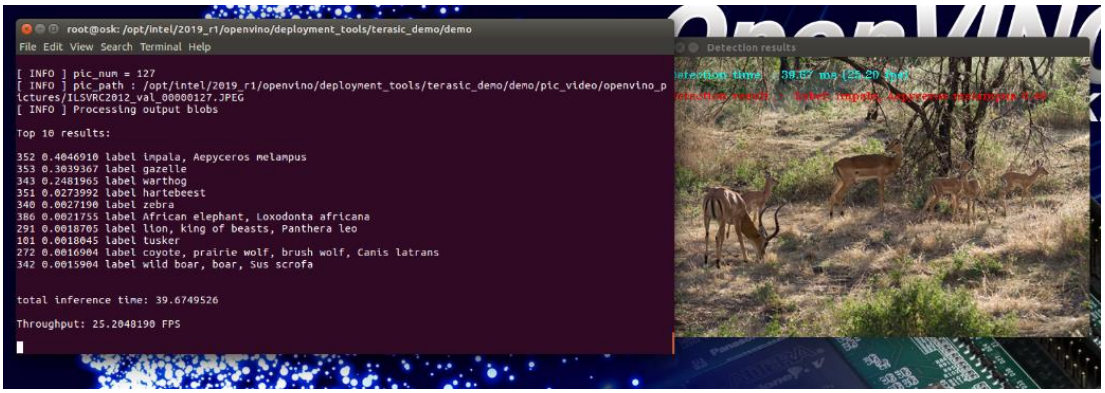
■ **07_classification_pic_loop**

该例程是基于 01_squeesnet_classification 例程，实现循环识别更多图片中的对象。

1) 执行“*./07_classification_pic_loop.sh fpga*”，在 FPGA 上运行。

```
total inference time: 30.4579064

Throughput: 32.8321975 FPS

^C
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# ./07_classification_pic_loop.sh fpga
```

2) 结果如下图所示：



Chapter 3

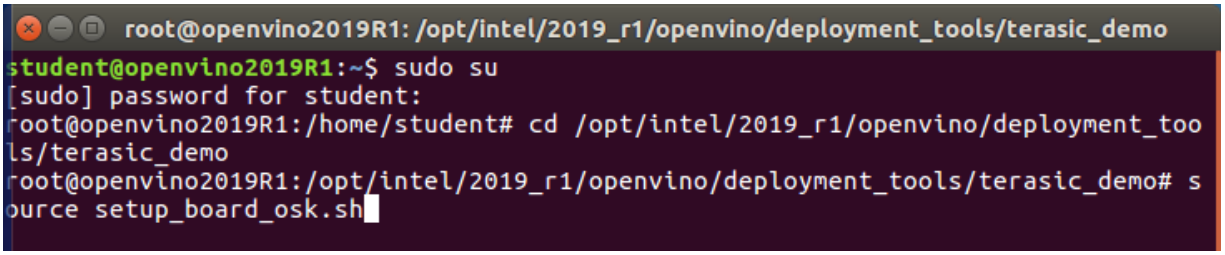
OpenVINO Starter Kit 实验

本章描述了如何在 FPGA 平台上验证实验环境，并实现用户自己的 AI 例程的加速。

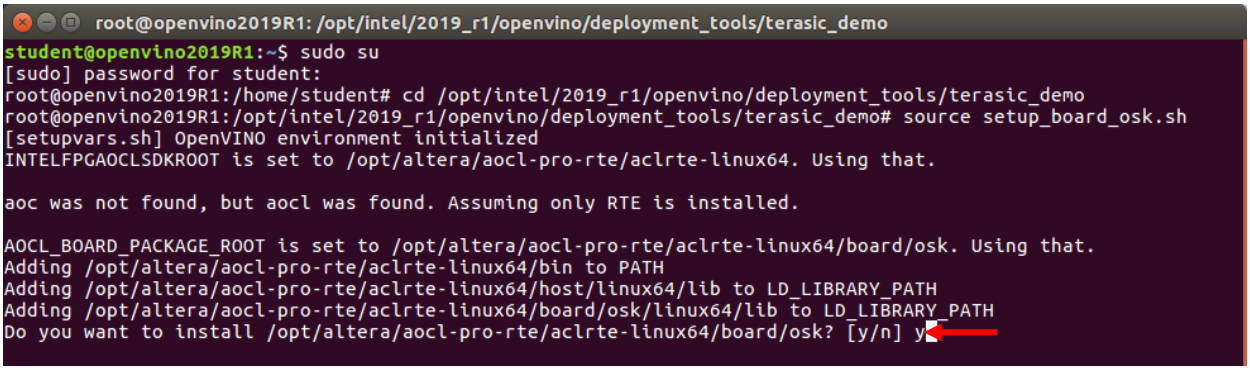
3.1 验证实验环境

本节将向用户演示如何通过运行 terasic_demo 中的例程“02_security_barrier.sh”来验证实验的环境。

1. 在桌面上右键单击鼠标，并选择 Open Termenal 打开终端。
2. 输入“sudo su”命令切换到 root 超级用户权限，password 是 terasic。
3. 输入“*cd /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo*”命令切换路径。
4. 输入“*source setup_board_osk.sh*”。



5. 输入 y 配置环境。



6. 输入 *cd /root/inference_engine_samples_build/*切换路径。
7. 输入“rm -rf CMakeCache.txt”，删除 CMakeCache.txt 文件

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo# cd /root/inferen
ce_engine_samples_build/
root@openvino2019R1:~/inference_engine_samples_build# rm -rf CMakeCache.txt
```

8. 输入“*cmake -DCMAKE_BUILD_TYPE=Release *

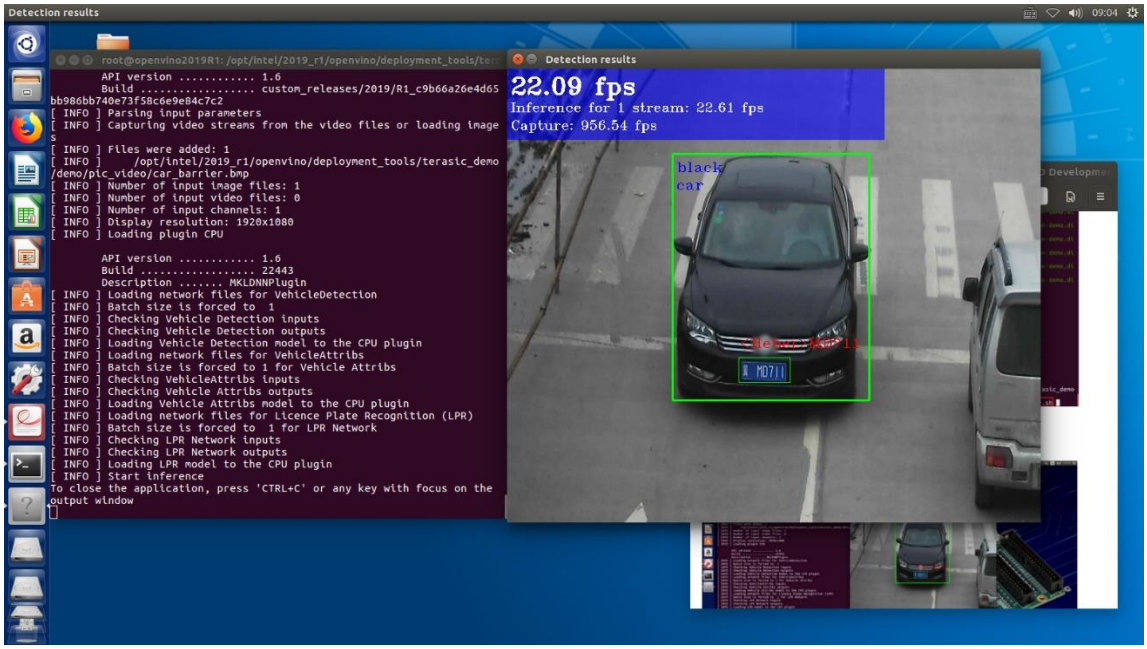
   /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/，并按下 enter 键，编译出 samples 的 makefile。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo# cd /root/inference_engine_samples_build/
root@openvino2019R1:~/inference_engine_samples_build# rm -rf CMakeCache.txt
root@openvino2019R1:~/inference_engine_samples_build# cmake -DCMAKE_BUILD_TYPE=Release \
> /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/
```

9. 输入“make”。
10. 等待编译完成。
11. 输入“cd /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo”切换工作路径。
12. 输入“*./02_security_barrier.sh fpga*”。

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo
[ 94%] Building CXX object text_detection_demo/CMakeFiles/text_detection_demo.dir/main.cpp.o
[ 94%] Linking CXX executable ../intel64/Release/text_detection_demo
[ 94%] Built target text_detection_demo
Scanning dependencies of target validation_app
[ 95%] Building CXX object validation_app/CMakeFiles/validation_app.dir/ClassificationProcessor
.cpp.o
[ 95%] Building CXX object validation_app/CMakeFiles/validation_app.dir/ObjectDetectionProcesso
r.cpp.o
[ 96%] Building CXX object validation_app/CMakeFiles/validation_app.dir/Processor.cpp.o
[ 97%] Building CXX object validation_app/CMakeFiles/validation_app.dir/VOCAnnotationParser.cpp
.o
[ 97%] Building CXX object validation_app/CMakeFiles/validation_app.dir/classification_set_gene
rator.cpp.o
[ 98%] Building CXX object validation_app/CMakeFiles/validation_app.dir/image_decoder.cpp.o
[ 98%] Building CXX object validation_app/CMakeFiles/validation_app.dir/main.cpp.o
[ 99%] Building CXX object validation_app/CMakeFiles/validation_app.dir/pugixml/pugixml.cpp.o
[100%] Linking CXX executable ../intel64/Release/validation_app
[100%] Built target validation_app
root@openvino2019R1:~/inference_engine_samples_build# cd /opt/intel/2019_r1/openvino/deployment
_tools/terasic_demo/demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo# ./02_securi
ty_barrier.sh cpu
target_device = CPU
target_precision = FP32
```

13. 运行结果如下所示。



3.2 实验一：如何使用模型优化器转换模型

本实验将向用户演示如何使用模型优化器工具从预下载的 caffe 模型文件“squeesenet1.1”中获取将被推理引擎应用程序使用的 IR 参数。

1. 输入“*cd /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/model/caffe/*”切换到预下载的模型文件夹中。
2. 输入 ls 查看文件夹中的文件信息，该文件包含了 bvlc_alexnet, squeezent1.1 和 SSD_GoogleNetV2 模型。
3. 输入 cd squeezenet1.1，选择对应的模型文件夹。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo#
 cd /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/model/caffe/
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/
model/caffe# ls
bvlc_alexnet  squeezenet1.1  SSD_GoogleNetV2
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/
model/caffe# cd squeezenet1.1/
```

4. 输入 ls， 可以看到该模型由三个文件组成：

- squeezenet1.1.caffemodel 是描述已训练模型的调整后的权重和偏差.
- squeezenet1.1.labels 是分类模型的标签文件
- squeezenet1.1.prototxt 是模型结构的描述文件。

```
model/caffe/squeezenet1.1# ls
squeezenet1.1.caffemodel  squeezenet1.1.prototxt
squeezenet1.1.labels
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/
model/caffe/squeezenet1.1#
```

5. 输入“*cd ../../../*” 返回 demo 文件夹。
6. 输入“*mkdir my_ir*”创建新文件夹以保存 IR 文件。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo
/model/caffe/squeezenet1.1# cd ../../../
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo
# mkdir my_ir
```

7. 输入“*cd /opt/intel/2019_r1/openvino/deployment_tools/model_optimizer*”，切换到 model_optimizer 文件夹。

8. 输入“*python3.5 mo_caffe.py *

   --input_model /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/

   *model/caffe/squeezenet1.1/squeezenet1.1.caffemodel *

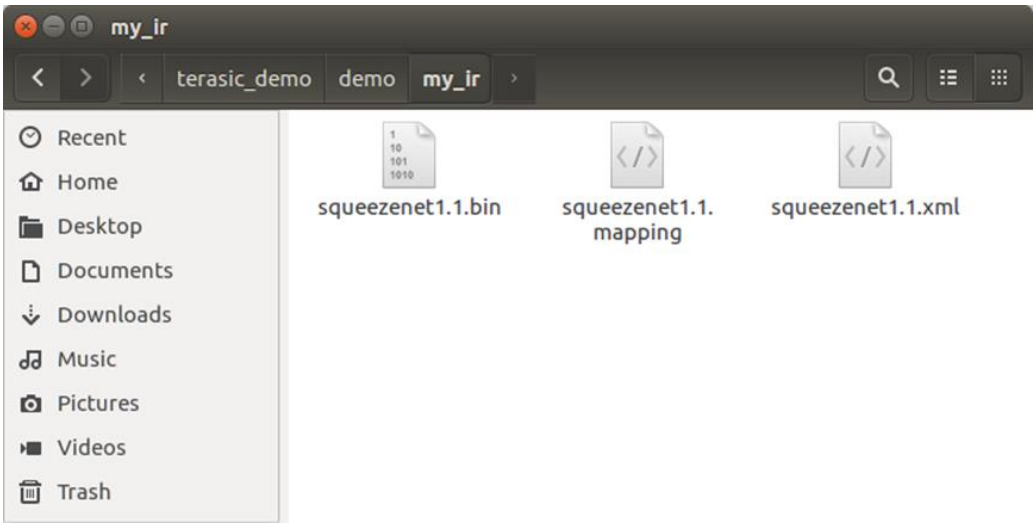
   *--output_dir /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/my_ir *

   --data_type FP16”.

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo# cd /opt/intel/2019_r1/openvino/deployment_tools/model_optimizer
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/model_optimizer
# python3.5 mo_caffe.py \
> --input_model /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/\
> model/caffe/squeezenet1.1/squeezenet1.1.caffemodel \
> --output_dir /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/my
_ir \
> --data_type FP16
```

9. 执行完成之后，在 my_ir 文件夹就会生成对应的 IR 文件。

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/model_optimizer
ls/terasic_demo/demo/model/caffe/squeezenet1.1/squeezenet1.1.caffemodel
        - Path for generated IR:        /opt/intel/2019_r1/openvino/deployment_too
ls/terasic_demo/demo/my_ir
        - IR output name:       squeezenet1.1
        - Log level:    ERROR
        - Batch:        Not specified, inherited from the model
        - Input layers:         Not specified, inherited from the model
        - Output layers:        Not specified, inherited from the model
        - Input shapes:         Not specified, inherited from the model
        - Mean values:  Not specified
        - Scale values:         Not specified
        - Scale factor:         Not specified
        - Precision of IR:      FP32
        - Enable fusing:        True
        - Enable grouped convolutions fusing:   True
        - Move mean values to preprocess section:       False
        - Reverse input channels:       False
Caffe specific parameters:
        - Enable resnet optimization:   True
        - Path to the Input prototxt:   /opt/intel/2019_r1/openvino/deployment_too
ls/terasic_demo/demo/model/caffe/squeezenet1.1/squeezenet1.1.prototxt
        - Path to CustomLayersMapping.xml:      Default
        - Path to a mean file:  Not specified
        - Offsets for a mean file:      Not specified
Model Optimizer version:        2019.1.0-341-gc9b66a2

[ SUCCESS ] Generated IR model.
[ SUCCESS ] XML file: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo/my_ir/squeezenet1.1.xml
[ SUCCESS ] BIN file: /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/de
mo/my_ir/squeezenet1.1.bin
[ SUCCESS ] Total execution time: 5.73 seconds.
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/model_optimizer#
```



10. 输入 “*cp* \

    /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/\

    model/caffe/squeezenet1.1/squeezenet1.1.labels \
    /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/my_ir/ ”命令，将.lable 文件从 Model 文件夹复制到 my_ir 文件夹。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/model_optimizer#
cp \
> /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/\
> model/caffe/squeezenet1.1/squeezenet1.1.labels \
> /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/my_ir/
```

3.3 实验二：如何编译推理引擎应用程序

1. 由于时间有限，我们直接复制现有的应用程序，重命名，并进行编译。
2. 输入“*cd ../inference_engine/samples*”切换到推理引擎 samples 文件夹下。
3. 输入 “`cp -r classification_sample my_classification_sample`” 。
4. 输入“*cd my_classification_sample*”切换到新的 app 文件夹下。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/model_optimizer#
cd ../inference_engine/samples
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cp -r classification_sample my_classification_sample
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_classification_sample/
```

5. 输入 “`gedit CMakeLists.txt`” 打开文件，将 target_name 重命名为 my_classification_sample，保存并关闭文件。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_classification_sample/
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_classification_sample# gedit CMakeLists.txt

(gedit:4708): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.
DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by a
ny .service files
```

Open *CMakeLists.txt /opt/intel/2019_r1/openvino/deployment_too...nce_engine/samples/my_classification... Save

```
# Copyright (C) 2018-2019 Intel Corporation
# SPDX-License-Identifier: Apache-2.0
#

set (TARGET_NAME "my_classification_sample")

file (GLOB SRC
        ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp
        )

# Create named folders for the sources within the .vcproj
# Empty name lists them directly under the .vcproj
source_group("src" FILES ${SRC})

link_directories(${LIB_FOLDER})

# Create library file from sources.
add_executable(${TARGET_NAME} ${SRC})
```

6. 输入“cd ../”返回到 sample 文件夹。
7. 输入“*mkdir my_build*”，创建新文件夹以保存生成的可执行程序。
8. 输入“*cd my_build*”切换到工作目录。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_classification_sample# cd ../
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# mkdir my_build
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_build/
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build#
```

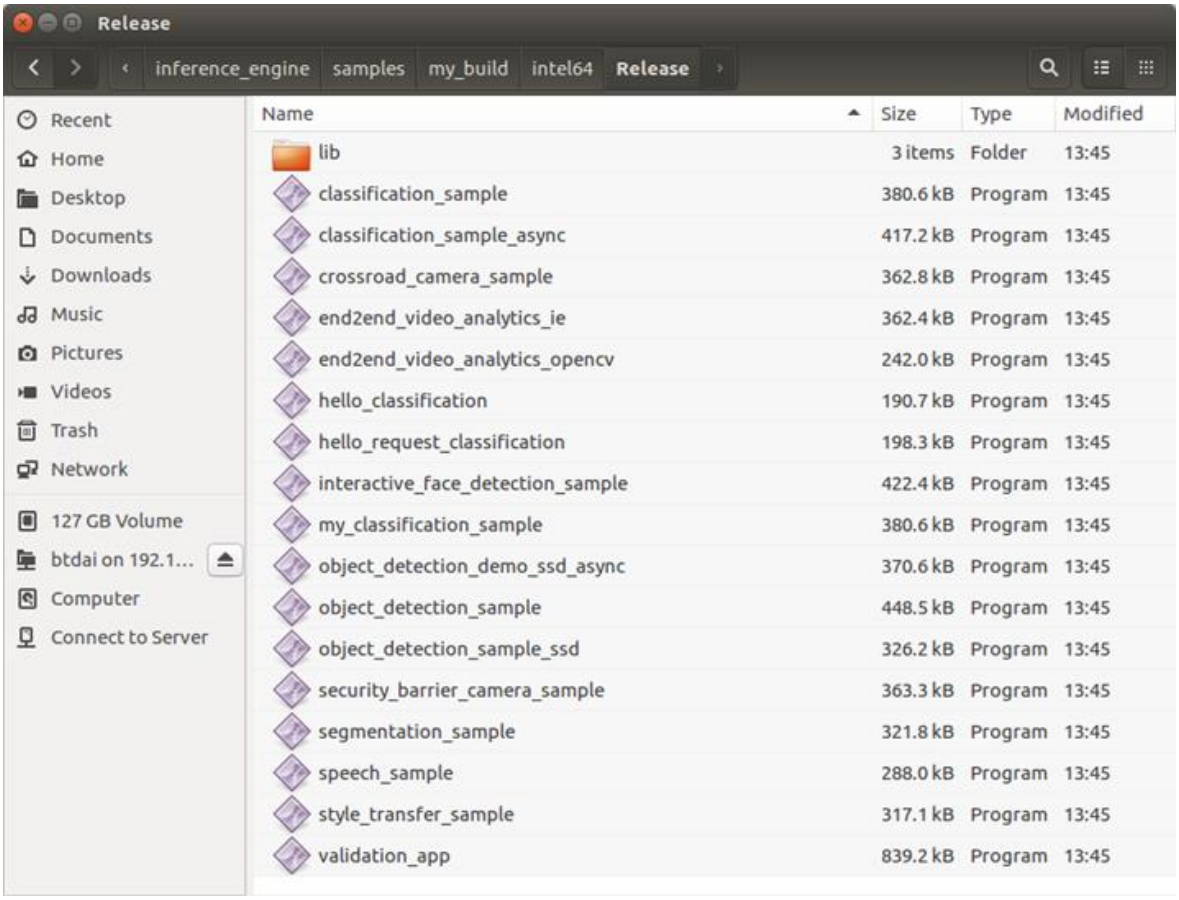
9. 输入 “*cmake -DCMAKE_BUILD_TYPE=Release \ /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/*” 命令，生成用于编译 code 的 makefile 文件。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_build/
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build# cmake -DCMAKE_BUILD_TYPE=Release \
> /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/
```

10. 输入“*make*”，编译应用程序，请耐心等待编译过程的执行。当前的编译设置将把 samples 文件夹中的所有应用程序编译为可执行程序。

```
-- Configuring done
-- Generating done
-- Build files have been written to: /opt/intel/2019_r1/openvino/deployment_tools/
inference_engine/samples/my_build
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build# make
```

11. 在路径:my_build/intel64/Release/下，将生成相应的 my_classification_sample 可执行程序，并创建应用程序。



关于推理引擎 API 的更多信息，用户可以参考以下链接

https://docs.openvinotoolkit.org/latest/annotated.html

https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Integrate_with_customer_application_new_API.html

3.4 实验三：执行创建的应用程序文件，使用推理引擎进行分类预测

1. 在前面的步骤中，我们将模型转换为推理引擎使用的 IR 文件，并生成相应的可执行文件。在执行文件之前，让我们大致了解一下应用程序中的操作。

2. 打开 classification_sample.h 文件，里面有一个参数“showUsage”用于执行 app， -h 表示帮助，-i 表示带有图像或相机参数的文件夹的路径，-m 表示已训练模型的路径(IR 路径)，-d 表示目标设备。

Open — classification_sample.h — /opt/intel/computer_vision_sdk_fpga_20...ngine/samples/my_classification_sample — Save

```
/// @brief Top results number (default 10) <br>
DEFINE_int32(nt, 10, ntop_message);

/// @brief Enable per-layer performance report
DEFINE_bool(pc, false, performance_counter_message);

/// @brief Define parameter for clDNN custom kernels path <br>
/// Default is ./lib
DEFINE_string(c, "", custom_cldnn_message);

/// @brief Absolute path to CPU library with user layers <br>
/// It is a optional parameter
DEFINE_string(l, "", custom_cpu_library_message);

/// @brief Iterations count (default 1)
DEFINE_int32(ni, 1, iterations_count_message);

/**
* @brief This function show a help message
*/
static void showUsage() {
    std::cout << std::endl;
    std::cout << "classification_sample [OPTION]" << std::endl;
    std::cout << "Options:" << std::endl;
    std::cout << std::endl;
    std::cout << "    -h                      " << help_message << std::endl;
    std::cout << "    -i \"<path>\"             " << image_message << std::endl;
    std::cout << "    -m \"<path>\"             " << model_message << std::endl;
    std::cout << "      -l \"<absolute_path>\"    " << custom_cpu_library_message << std::endl;
    std::cout << "           Or" << std::endl;
    std::cout << "      -c \"<absolute_path>\"    " << custom_cldnn_message << std::endl;
    std::cout << "    -pp \"<path>\"            " << plugin_path_message << std::endl;
    std::cout << "    -d \"<device>\"           " << target_device_message << std::endl;
    std::cout << "    -nt \"<integer>\"         " << ntop_message << std::endl;
    std::cout << "    -ni \"<integer>\"         " << iterations_count_message << std::endl;
    std::cout << "    -pc                     " << performance_counter_message << std::endl;
}
```

C/C++/ObjC Header — Tab Width: 8 — Ln 1, Col 1 — INS

3. 打开 inference_engine/samples/my_classification_sample 文件夹下的 main.cpp 文件，解释如下：

1）加载用于推理引擎的插件，本实验中使用的是用于 FPGA 和 CPU 的异构插件——HETERO 插件。

```
// ------------------------------ Parsing and validation of input args ---------------------------------
if (!ParseAndCheckCommandLine(argc, argv)) {
    return 0;
}

/** This vector stores paths to the processed images **/
std::vector<std::string> imageNames;
parseImagesArguments(imageNames);
if (imageNames.empty()) throw std::logic_error("No suitable images were found");
// ---------------------------------------------------------------------------------------------------

// --------------------------- 1. Load Plugin for inference engine ---------------------------------
slog::info << "Loading plugin" << slog::endl;
InferencePlugin plugin = PluginDispatcher({ FLAGS_pp, "../../../lib/intel64" , "" }).getPluginByDevice(FLAGS_d);

/** Loading default extensions **/
if (FLAGS_d.find("CPU") != std::string::npos) {
    /**
     * cpu_extensions library is compiled from "extension" folder containing
     * custom MKLDNNPlugin layer implementations. These layers are not supported
     * by mkldnn, but they can be useful for inferring custom topologies.
    **/
    plugin.AddExtension(std::make_shared<Extensions::Cpu::CpuExtensions>());
}

if (!FLAGS_l.empty()) {
    // CPU(MKLDNN) extensions are loaded as a shared library and passed as a pointer to base extension
    auto extension_ptr = make_so_pointer<IExtension>(FLAGS_l);
    plugin.AddExtension(extension_ptr);
    slog::info << "CPU Extension loaded: " << FLAGS_l << slog::endl;
}
if (!FLAGS_c.empty()) {
    // clDNN Extensions are loaded from an .xml description and OpenCL kernel files
    plugin.SetConfig({{PluginConfigParams::KEY_CONFIG_FILE, FLAGS_c}});
    slog::info << "GPU Extension loaded: " << FLAGS_c << slog::endl;
}

/** Setting plugin parameter for collecting per layer metrics **/
if (FLAGS_pc) {
    plugin.SetConfig({ { PluginConfigParams::KEY_PERF_COUNT, PluginConfigParams::YES } });
}

/** Printing plugin version **/
printPluginVersion(plugin, std::cout);
// ---------------------------------------------------------------------------------------------------
```

2）读取模型优化器生成的 IR（.xml 和.bin 文件）。本实验中，xml 文件是 squeezenet1.1.xml

```
// --------------------------- 2. Read IR Generated by ModelOptimizer (.xml and .bin files) ---
std::string binFileName = fileNameNoExt(FLAGS_m) + ".bin";
slog::info << "Loading network files:"
        "\n\t" << FLAGS_m <<
        "\n\t" << binFileName <<
slog::endl;

CNNNetReader networkReader;
/** Reading network model **/
networkReader.ReadNetwork(FLAGS_m);

/** Extracting model name and loading weights **/
networkReader.ReadWeights(binFileName);
CNNNetwork network = networkReader.getNetwork();
// -----------------------------------------------------------------------------------------
```

3） 配置输入和输出，准备输入 blobs，读取输入尺寸信息，读取图像路径，准备输出 blobs。

```
// --------------------------- 3. Configure input & output ---------------------------------------

// --------------------------- Prepare input blobs -----------------------------------------------
slog::info << "Preparing input blobs" << slog::endl;

/** Taking information about all topology inputs **/
InputsDataMap inputInfo = network.getInputsInfo();

if (inputInfo.size() != 1) throw std::logic_error("Sample supports topologies only with 1 input");
auto inputInfoItem = *inputInfo.begin();

/** Specifying the precision and layout of input data provided by the user.
 * This should be called before load of the network to the plugin **/
inputInfoItem.second->setPrecision(Precision::U8);
inputInfoItem.second->setLayout(Layout::NCHW);

std::vector<std::shared_ptr<unsigned char>> imagesData;
for (auto & i : imageNames) {
    FormatReader::ReaderPtr reader(i.c_str());
    if (reader.get() == nullptr) {
        slog::warn << "Image " + i + " cannot be read!" << slog::endl;
        continue;
    }
    /** Store image data **/
    std::shared_ptr<unsigned char> data(
            reader->getData(inputInfoItem.second->getTensorDesc().getDims()[3],
                            inputInfoItem.second->getTensorDesc().getDims()[2]));
    if (data.get() != nullptr) {
        imagesData.push_back(data);
    }
}
if (imagesData.empty()) throw std::logic_error("Valid input images were not found!");

/** Setting batch size using image count **/
network.setBatchSize(imagesData.size());
size_t batchSize = network.getBatchSize();
slog::info << "Batch size is " << std::to_string(batchSize) << slog::endl;
```

4） 将模型加载到插件。

```
// --------------------------- 4. Loading model to the plugin --------------
slog::info << "Loading model to the plugin" << slog::endl;

ExecutableNetwork executable_network = plugin.LoadNetwork(network, {});
inputInfoItem.second = {};
outputInfo = {};
network = {};
networkReader = {};
// -------------------------------------------------------------------------
```

5）创建推断请求。

```
// --------------------------- 5. Create infer request ----------------
InferRequest infer_request = executable_network.CreateInferRequest();
// -------------------------------------------------------------------
```

6）准备输入。

```
// --------------------------- 6. Prepare input ------------------------------------------------------
/** Iterate over all the input blobs **/
for (const auto & item : inputInfo) {
    /** Creating input blob **/
    Blob::Ptr input = infer_request.GetBlob(item.first);

    /** Filling input tensor with images. First b channel, then g and r channels **/
    size_t num_channels = input->getTensorDesc().getDims()[1];
    size_t image_size = input->getTensorDesc().getDims()[2] * input->getTensorDesc().getDims()[3];

    auto data = input->buffer().as<PrecisionTrait<Precision::U8>::value_type*>();

    /** Iterate over all input images **/
    for (size_t image_id = 0; image_id < imagesData.size(); ++image_id) {
        /** Iterate over all pixel in image (b,g,r) **/
        for (size_t pid = 0; pid < image_size; pid++) {
            /** Iterate over all channels **/
            for (size_t ch = 0; ch < num_channels; ++ch) {
                /**         [images stride + channels stride + pixel id ] all in bytes            **/
                data[image_id * image_size * num_channels + ch * image_size + pid ] = imagesData.at(image_id).get()[pid*num_channels + ch];
            }
        }
    }
}
inputInfo = {};
// ---------------------------------------------------------------------------------------------------
```

7）进行推断，使用 CPU 处理数据，并将结果返回到 CPU。

```
// --------------------------- 7. Do inference -------------------------------------------------------
slog::info << "Starting inference (" << FLAGS_ni << " iterations)" << slog::endl;

typedef std::chrono::high_resolution_clock Time;
typedef std::chrono::duration<double, std::ratio<1, 1000>> ms;
typedef std::chrono::duration<float> fsec;

double total = 0.0;
/** Start inference & calc performance **/
for (int iter = 0; iter < FLAGS_ni; ++iter) {
    auto t0 = Time::now();
    infer_request.Infer();
    auto t1 = Time::now();
    fsec fs = t1 - t0;
    ms d = std::chrono::duration_cast<ms>(fs);
    total += d.count();
}

/** Show performance results **/
slog::info << "Average running time of one iteration: " << total / static_cast<double>(FLAGS_ni) << " ms" << slog::endl;

if (FLAGS_pc) {
    printPerformanceCounts(infer_request, std::cout);
}
// ---------------------------------------------------------------------------------------------------
```

8） 处理输出，将推理结果与 label 文件比较，然后打印最后的识别结果以及平均推理时间。

```
// --------------------------- 8. Process output -----------------------------------------------------
slog::info << "Processing output blobs" << slog::endl;

const Blob::Ptr output_blob = infer_request.GetBlob(firstOutputName);
auto output_data = output_blob->buffer().as<PrecisionTrait<Precision::FP32>::value_type*>();

/** Validating -nt value **/
const int resultsCnt = output_blob->size() / batchSize;
if (FLAGS_nt > resultsCnt || FLAGS_nt < 1) {
    slog::warn << "-nt " << FLAGS_nt << " is not available for this network (-nt should be less than " \
               << resultsCnt+1 << " and more than 0)\n            will be used maximal value : " << resultsCnt;
    FLAGS_nt = resultsCnt;
}

/** This vector stores id's of top N results **/
std::vector<unsigned> results;
TopResults(FLAGS_nt, *output_blob, results);

std::cout << std::endl << "Top " << FLAGS_nt << " results:" << std::endl << std::endl;

/** Read labels from file (e.x. AlexNet.labels) **/
bool labelsEnabled = false;
std::string labelFileName = fileNameNoExt(FLAGS_m) + ".labels";
std::vector<std::string> labels;

std::ifstream inputFile;
inputFile.open(labelFileName, std::ios::in);
if (inputFile.is_open()) {
    std::string strLine;
    while (std::getline(inputFile, strLine)) {
        trim(strLine);
        labels.push_back(strLine);
    }
    labelsEnabled = true;
}
```

4. 现在，我们清楚了在应用程序中执行的操作，接下来，我们运行之前生成的可执行文件。

5. 输入 “*cd* \

   /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples\

   /my_build/intel64/Release” 切换到应用程序文件夹。

```
gtxml.cpp.o
[100%] Linking CXX executable ../intel64/Release/validation_app
[100%] Built target validation_app
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build# cd \
> /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples\
> /my_build/intel64/Release
```

6. 输入 “*./my_classification_sample -i* \

   /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/pic_video/car.png \

   -m /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/\

   demo/my_ir/squeezenet1.1.xml -d "HETERO:FPGA,CPU"”执行推理引擎应用程序。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engin
e/samples/my_build/intel64/Release# ./my_classification_sample -i \
> /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/pic_video/car.p
ng \
> -m /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/\
> demo/my_ir/squeezenet1.1.xml -d "HETERO:FPGA,CPU"
```

```
root@openvino2019R1: /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/s
817     0.8363336   sports car, sport car
511     0.0946490   convertible
479     0.0419133   car wheel
751     0.0091072   racer, race car, racing car
436     0.0068162   beach wagon, station wagon, wagon, estate car, beach waggon, s
tation waggon, waggon
656     0.0037564   minivan
586     0.0025741   half track
717     0.0016069   pickup, pickup truck
864     0.0012027   tow truck, tow car, wrecker
581     0.0005882   grille, radiator grille


total inference time: 25.4686959
Average running time of one iteration: 25.4686959 ms

Throughput: 39.2638871 FPS

[ INFO ] Execution successful
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build/intel64/Release#
```

3.5 进阶实验

1. 在前面的步骤中，我们已经知道了如何将模型转换为 IR，以及如何生成可被推理引擎使用的可执行文件。接下来，让我们创建一个新的例程。

2. 因为前面已经生成了 IR 文件，所以不需要再重新生成，我们继续使用之前生成的 squeezenet1.1.xml 和相应的 bin 文件。

3. 输入“*cd /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples*”切换到推理引擎 samples 文件夹。

4. 输入“*cp -r my_classification_sample my_demo*”，拷贝前面实验生成的文件，我们将修改这些文件以应用到 my_demo 中。

5. 输入“*cd my_demo*”，切换到新拷贝的 samples 文件夹。

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_build/intel64/Release# cd /opt/intel/2019_r1/openvino/deployment_tools/
inference_engine/samples
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cp -r my_classification_sample my_demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_demo#
```

6. 输入 *gedit CMakeLists.txt* 打开该文件，按照如下内容修改文件：

```
samples# cp -r my_classification_sample my_demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples# cd my_demo
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_demo# gedit CMakeLists.txt
```

Open *CMakeLists.txt /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

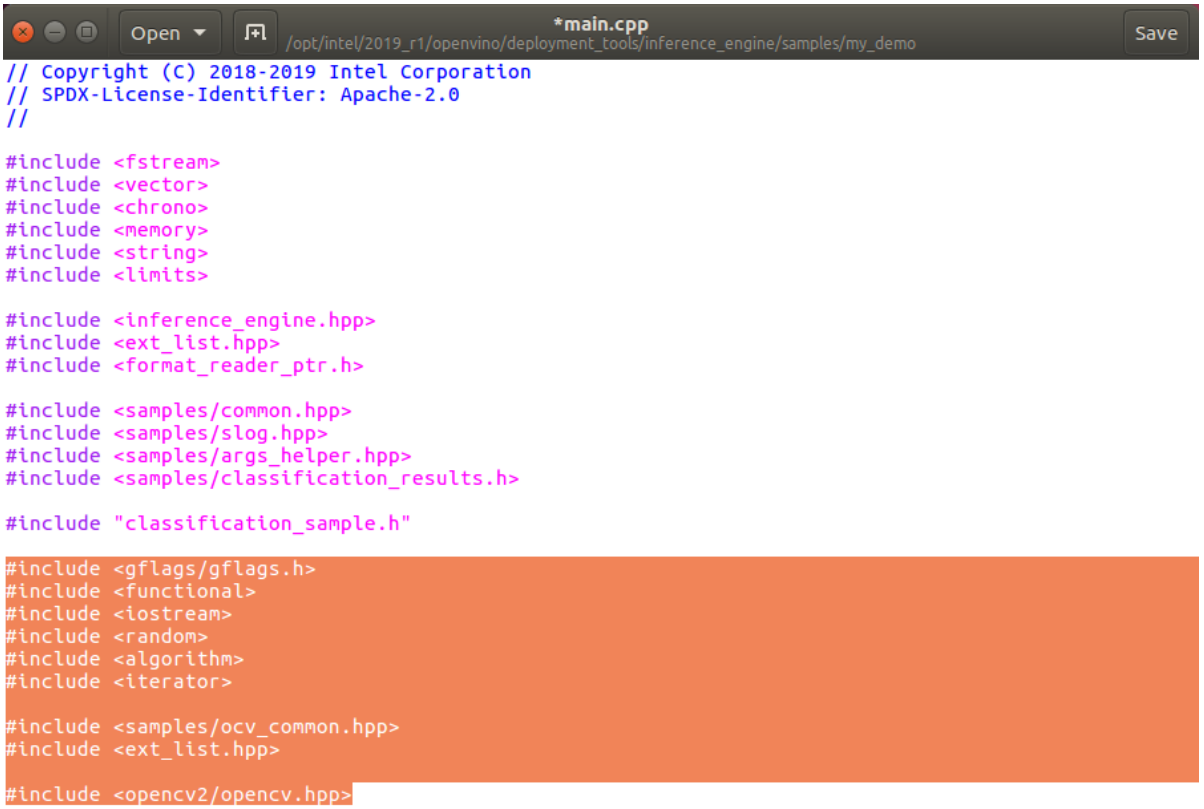
```
# Copyright (C) 2018-2019 Intel Corporation
# SPDX-License-Identifier: Apache-2.0
#

set (TARGET_NAME "my_demo")
# Find OpenCV components if exist
find_package(OpenCV COMPONENTS highgui QUIET)
if(NOT(OpenCV_FOUND))
message(WARNING "OPENCV is disabled or not found, " ${TARGET_NAME} " skipped")
return()
endif()
file (GLOB MAIN_SRC
${CMAKE_CURRENT_SOURCE_DIR}/*.cpp
)
file (GLOB MAIN_HEADERS
${CMAKE_CURRENT_SOURCE_DIR}/*.h
)
# Create named folders for the sources within the .vcproj
# Empty name lists them directly under the .vcproj
source_group("src" FILES ${MAIN_SRC})
source_group("include" FILES ${MAIN_HEADERS})
link_directories(${LIB_FOLDER})
# Create library file from sources.
add_executable(${TARGET_NAME} ${MAIN_SRC} ${MAIN_HEADERS})
add_dependencies(${TARGET_NAME} gflags)
set_target_properties(${TARGET_NAME} PROPERTIES "CMAKE_CXX_FLAGS""${CMAKE_CXX_FLAGS} -fPIE"
COMPILE_PDB_NAME ${TARGET_NAME})
#target_link_libraries(${TARGET_NAME} ${InferenceEngine_LIBRARIES} IE::ie_cpu_extension format_reader gflags)
target_link_libraries(${TARGET_NAME} IE::ie_cpu_extension ${InferenceEngine_LIBRARIES} gflags ${OpenCV_LIBRARIES})
if(UNIX)
target_link_libraries(${TARGET_NAME} ${LIB_DL} pthread)
endif()
```

CMake Tab Width: 8 Ln 14, Col 2 INS

```
set (TARGET_NAME "my_demo")

# Find OpenCV components if exist
find_package(OpenCV COMPONENTS highgui QUIET)
if(NOT(OpenCV_FOUND))
    message(WARNING "OPENCV is disabled or not found, " ${TARGET_NAME} " skipped")
    return()
endif()

file (GLOB MAIN_SRC
        ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp
        )
file (GLOB MAIN_HEADERS
        ${CMAKE_CURRENT_SOURCE_DIR}/*.h
        )

# Create named folders for the sources within the .vcproj
# Empty name lists them directly under the .vcproj
source_group("src" FILES ${MAIN_SRC})
source_group("include" FILES ${MAIN_HEADERS})

link_directories(${LIB_FOLDER})

# Create library file from sources.
add_executable(${TARGET_NAME} ${MAIN_SRC} ${MAIN_HEADERS})
add_dependencies(${TARGET_NAME} gflags)
set_target_properties(${TARGET_NAME} PROPERTIES "CMAKE_CXX_FLAGS"
"${CMAKE_CXX_FLAGS} -fPIE"
COMPILE_PDB_NAME ${TARGET_NAME})

#target_link_libraries(${TARGET_NAME} ${InferenceEngine_LIBRARIES} IE::ie_cpu_extension
format_reader gflags)
target_link_libraries(${TARGET_NAME} IE::ie_cpu_extension ${InferenceEngine_LIBRARIES} gflags
${OpenCV_LIBRARIES})
if(UNIX)
    target_link_libraries(${TARGET_NAME} ${LIB_DL} pthread)
endif()
```

7. 输入"gedit main.cpp"打开文件进行应用程序修改。在前面的步骤中生成的主机应用程序用于输入单个图片的分类。接下来，我们将修改应用程序以支持在输入循环的多个图上显示分类。

   第 1 步 给 opencv 和视频运行添加头文件。




```
#include <gflags/gflags.h>
#include <functional>
#include <iostream>
#include <random>
#include <algorithm>
#include <iterator>

#include <samples/ocv_common.hpp>
#include <ext_list.hpp>

#include <opencv2/opencv.hpp>
```

第 2 步 给图像编号定义一个宏。

```
#define PIC_NUM 199
```

```
Open  *main.cpp  /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo  Save
#include <ext_list.hpp>

#include <opencv2/opencv.hpp>


using namespace InferenceEngine;

#define PIC_NUM 199

ConsoleErrorListener error_listener;

bool ParseAndCheckCommandLine(int argc, char *argv[]) {
    // ----------------------------Parsing and validation of input
args-------------------------------------
    gflags::ParseCommandLineNonHelpFlags(&argc, &argv, true);
    if (FLAGS_h) {
        showUsage();
```

第 3 步 删除 imageNames，因为不用再使用它。

删除前：

```
Open  *main.cpp  /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo  Save
        // ------------------------------ Parsing and validation of input args
---------------------------------
        if (!ParseAndCheckCommandLine(argc, argv)) {
            return 0;
        }

        /** This vector stores paths to the processed images **/
        std::vector<std::string> imageNames;
        parseInputFilesArguments(imageNames);
        if (imageNames.empty()) throw std::logic_error("No suitable images were found");
        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 1. Load Plugin for inference engine
-------------------------------------
        slog::info << "Loading plugin" << slog::endl;
        InferencePlugin plugin = PluginDispatcher({ FLAGS_pp }).getPluginByDevice(FLAGS_d);
        if (FLAGS_p_msg) {
            static_cast<InferenceEngine::InferenceEnginePluginPtr>(plugin)->SetLogCallback
(error_listener);
```

删除后：

```
Open  *main.cpp  /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo  Save
/**
* @brief The entry point the Inference Engine sample application
* @file classification_sample/main.cpp
* @example classification_sample/main.cpp
*/
int main(int argc, char *argv[]) {
    try {
        slog::info << "InferenceEngine: " << GetInferenceEngineVersion() << slog::endl;

        // ------------------------------ Parsing and validation of input args
---------------------------------
        if (!ParseAndCheckCommandLine(argc, argv)) {
            return 0;
        }

        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 1. Load Plugin for inference engine
-------------------------------------
        slog::info << "Loading plugin" << slog::endl;
        InferencePlugin plugin = PluginDispatcher({ FLAGS_pp }).getPluginByDevice(FLAGS_d);
        if (FLAGS_p_msg) {
            static_cast<InferenceEngine::InferenceEnginePluginPtr>(plugin)->SetLogCallback
(error listener);
```

第 4 步 删除 imagesData，因为这是用于 imageNames 的。

删除前：

Open *main.cpp /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

```
        /** Taking information about all topology inputs **/
        InputsDataMap inputInfo = network.getInputsInfo();
        if (inputInfo.size() != 1) throw std::logic_error("Sample supports topologies only with 1
input");

        auto inputInfoItem = *inputInfo.begin();

        /** Specifying the precision and layout of input data provided by the user.
         * This should be called before load of the network to the plugin **/
        inputInfoItem.second->setPrecision(Precision::U8);
        inputInfoItem.second->setLayout(Layout::NCHW);

        std::vector<std::shared_ptr<unsigned char>> imagesData;
        for (auto & i : imageNames) {
            FormatReader::ReaderPtr reader(i.c_str());
            if (reader.get() == nullptr) {
                slog::warn << "Image " + i + " cannot be read!" << slog::endl;
                continue;
            }
            /** Store image data **/
            std::shared_ptr<unsigned char> data(
                    reader->getData(inputInfoItem.second->getTensorDesc().getDims()[3],
                                    inputInfoItem.second->getTensorDesc().getDims()[2]));
            if (data.get() != nullptr) {
                imagesData.push_back(data);
            }
        }
        if (imagesData.empty()) throw std::logic_error("Valid input images were not found!");

        /** Setting batch size using image count **/
        network.setBatchSize(imagesData.size());
        size_t batchSize = network.getBatchSize();
        slog::info << "Batch size is " << std::to_string(batchSize) << slog::endl;

        // ------------------------------ Prepare output blobs
-----------------------------------------------------
        slog::info << "Preparing output blobs" << slog::endl;
```

C++ Tab Width: 8 Ln 168, Col 94 INS

删除后：

Open *main.cpp /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

```
        /** Taking information about all topology inputs **/
        InputsDataMap inputInfo = network.getInputsInfo();
        if (inputInfo.size() != 1) throw std::logic_error("Sample supports topologies only with 1
input");

        auto inputInfoItem = *inputInfo.begin();

        /** Specifying the precision and layout of input data provided by the user.
         * This should be called before load of the network to the plugin **/
        inputInfoItem.second->setPrecision(Precision::U8);
        inputInfoItem.second->setLayout(Layout::NCHW);

        /** Setting batch size using image count **/
        network.setBatchSize(imagesData.size());
        size_t batchSize = network.getBatchSize();
        slog::info << "Batch size is " << std::to_string(batchSize) << slog::endl;

        // ------------------------------ Prepare output blobs
-----------------------------------------------------
        slog::info << "Preparing output blobs" << slog::endl;
```

第 5 步 将 network.setBatchSize(imagesData.size())修改为 network.setBatchSize(1)。

修改前

Open *main.cpp /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

```
        auto inputInfoItem = *inputInfo.begin();

        /** Specifying the precision and layout of input data provided by the user.
         * This should be called before load of the network to the plugin **/
        inputInfoItem.second->setPrecision(Precision::U8);
        inputInfoItem.second->setLayout(Layout::NCHW);


        /** Setting batch size using image count **/
        network.setBatchSize(imagesData.size());
        size_t batchSize = network.getBatchSize();
        slog::info << "Batch size is " << std::to_string(batchSize) << slog::endl;

        // ------------------------------ Prepare output blobs
-------------------------------------------------
        slog::info << "Preparing output blobs" << slog::endl;
```

修改后：

Open *main.cpp /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

```
        auto inputInfoItem = *inputInfo.begin();

        /** Specifying the precision and layout of input data provided by the user.
         * This should be called before load of the network to the plugin **/
        inputInfoItem.second->setPrecision(Precision::U8);
        inputInfoItem.second->setLayout(Layout::NCHW);


        /** Setting batch size using image count **/
        network.setBatchSize(1);
        size_t batchSize = network.getBatchSize();
        slog::info << "Batch size is " << std::to_string(batchSize) << slog::endl;

        // ------------------------------ Prepare output blobs
-------------------------------------------------
        slog::info << "Preparing output blobs" << slog::endl;
```

第 6 步 删除如下高亮标记的代码。
删除前：

Open *main.cpp /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo Save

```
        } else if (outputDims.size() == 4 /* NCHW */) {
            /* H = W = 1 */
            if (outputDims[2] == 1 && outputDims[3] == 1) outputCorrect = true;
        }

        if (!outputCorrect) {
            throw std::logic_error("Incorrect output dimensions for classification model");
        }
        //
-----------------------------------------------------------------------------------------------------

        // --------------------------- 4. Loading model to the plugin
----------------------------------------
        slog::info << "Loading model to the plugin" << slog::endl;

        ExecutableNetwork executable_network = plugin.LoadNetwork(network, {});
        inputInfoItem.second = {};
        outputInfo = {};
        network = {};
        networkReader = {};
        //
-----------------------------------------------------------------------------------------------------

        // --------------------------- 5. Create infer request
```

删除后：

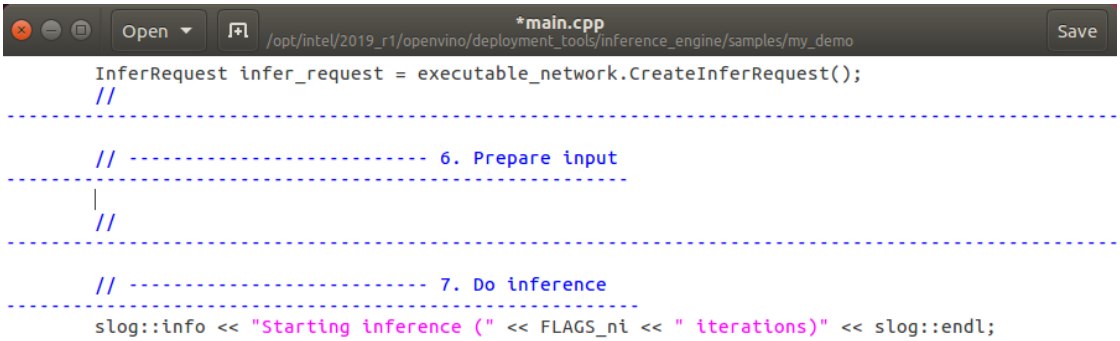
```
Open  *main.cpp  /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo  Save
        } else if (outputDims.size() == 4 /* NCHW */) {
            /* H = W = 1 */
            if (outputDims[2] == 1 && outputDims[3] == 1) outputCorrect = true;
        }

        if (!outputCorrect) {
            throw std::logic_error("Incorrect output dimensions for classification model");
        }
        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 4. Loading model to the plugin
-----------------------------------------
        slog::info << "Loading model to the plugin" << slog::endl;

        ExecutableNetwork executable_network = plugin.LoadNetwork(network, {});

        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 5. Create infer request
-------------------------------------------------
        InferRequest infer_request = executable_network.CreateInferRequest();
        //
```

第 7 步 删除"step 6, Prepare input"的操作。

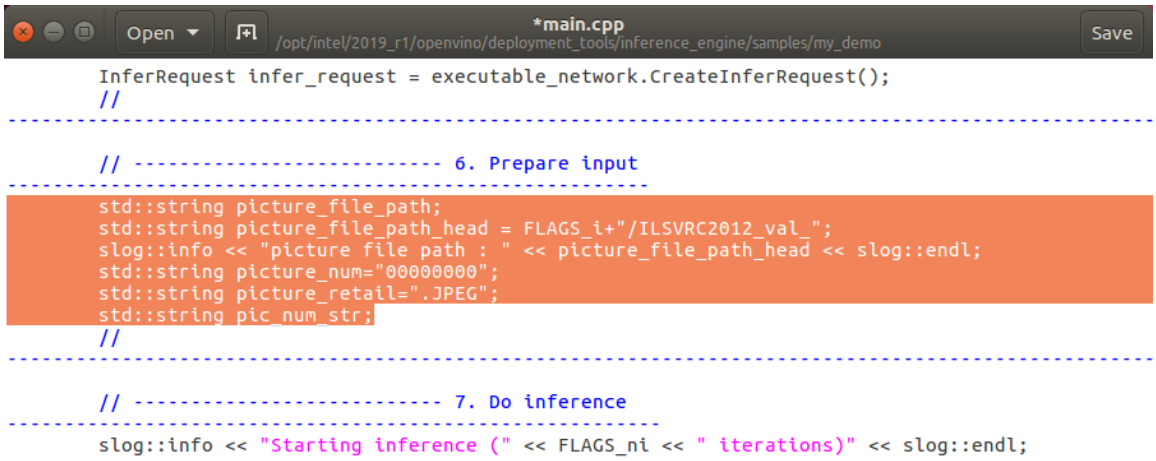
删除前：

```
Open  *main.cpp  /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples/my_demo  Save
        InferRequest infer_request = executable_network.CreateInferRequest();
        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 6. Prepare input
-------------------------------------------------------
        /** Iterate over all the input blobs **/
        for (const auto & item : inputInfo) {
            /** Creating input blob **/
            Blob::Ptr input = infer_request.GetBlob(item.first);

            /** Filling input tensor with images. First b channel, then g and r channels **/
            size_t num_channels = input->getTensorDesc().getDims()[1];
            size_t image_size = input->getTensorDesc().getDims()[2] * input->getTensorDesc
().getDims()[3];

            auto data = input->buffer().as<PrecisionTrait<Precision::U8>::value_type*>();

            /** Iterate over all input images **/
            for (size_t image_id = 0; image_id < imagesData.size(); ++image_id) {
                /** Iterate over all pixel in image (b,g,r) **/
                for (size_t pid = 0; pid < image_size; pid++) {
                    /** Iterate over all channels **/
                    for (size_t ch = 0; ch < num_channels; ++ch) {
                        /**          [images stride + channels stride + pixel id ] all in
bytes            **/
                        data[image_id * image_size * num_channels + ch * image_size + pid ] =
imagesData.at(image_id).get()[pid*num_channels + ch];
                    }
                }
            }
        }
        inputInfo = {};
        //
-------------------------------------------------------------------------------------------------------

        // --------------------------- 7. Do inference
C++  Tab Width: 8  Ln 228, Col 24  INS
```

删除后：

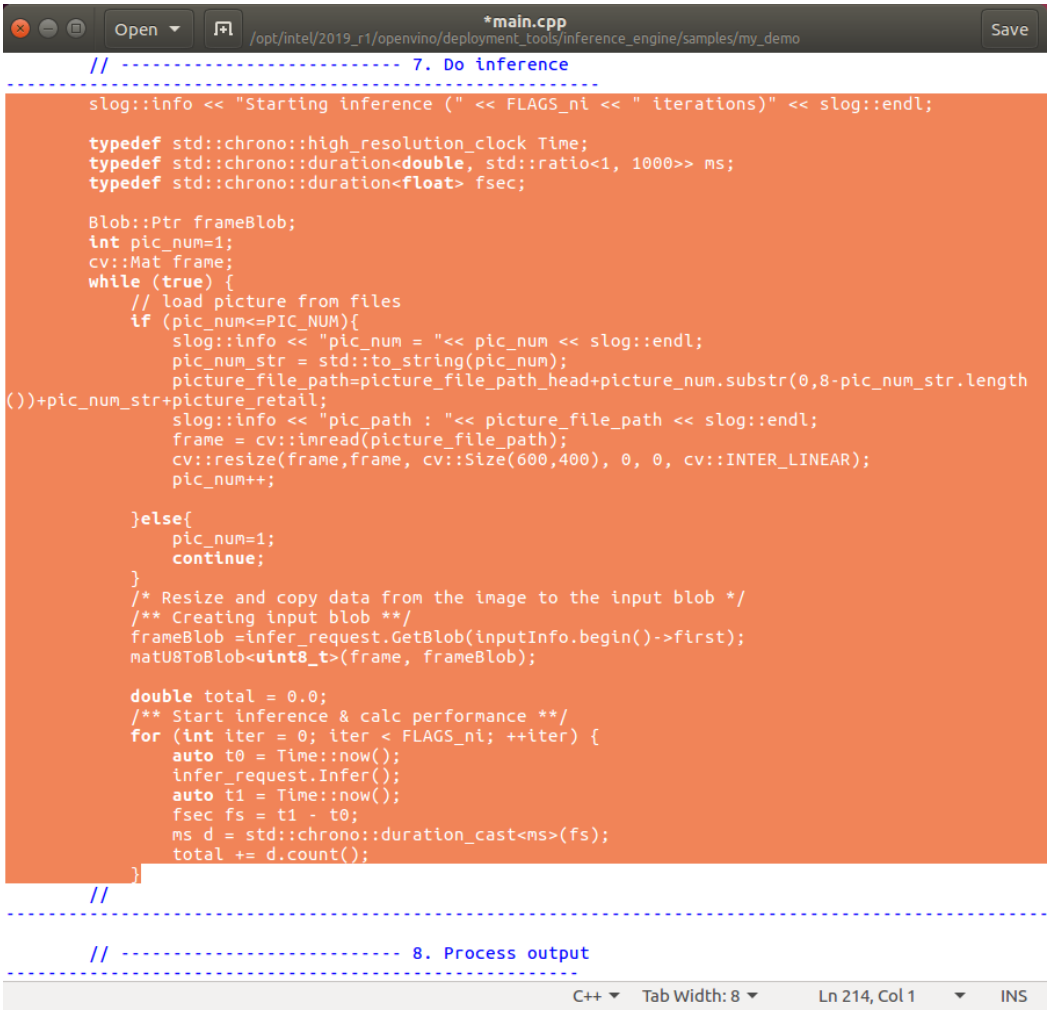


第 8 步 在"step 6, Prepare input"中添加图像相关信息。




```
std::string picture_file_path;
std::string picture_file_path_head = FLAGS_i+"/ILSVRC2012_val_";
slog::info << "picture file path : " << picture_file_path_head << slog::endl;
std::string picture_num="00000000";
std::string picture_retail=".JPEG";
std::string pic_num_str;
```

第 9 步 删除“step 7, Do inference”中的代码，并添加下面的代码。




```
slog::info << "Starting inference (" << FLAGS_ni << " iterations)" << slog::endl;

    typedef std::chrono::high_resolution_clock Time;
    typedef std::chrono::duration<double, std::ratio<1, 1000>> ms;
    typedef std::chrono::duration<float> fsec;

Blob::Ptr frameBlob;
int pic_num=1;
cv::Mat frame;
while (true) {
    // load picture from files
    if (pic_num<=PIC_NUM){
    slog::info << "pic_num = "<< pic_num << slog::endl;
    pic_num_str = std::to_string(pic_num);
    picture_file_path=picture_file_path_head+picture_num.substr(0,8-
pic_num_str.length())+pic_num_str+picture_retail;
```

```
slog::info << "pic_path : "<< picture_file_path << slog::endl;
frame = cv::imread(picture_file_path);
cv::resize(frame,frame, cv::Size(600,400), 0, 0, cv::INTER_LINEAR);
pic_num++;

}else{
pic_num=1;
continue;
}
/* Resize and copy data from the image to the input blob */
        /** Creating input blob **/
        frameBlob =infer_request.GetBlob(inputInfo.begin()->first);
matU8ToBlob<uint8_t>(frame, frameBlob);

double total = 0.0;
/** Start inference & calc performance **/
for (unsigned int iter = 0; iter < FLAGS_ni; ++iter) {
auto t0 = Time::now();
infer_request.Infer();
auto t1 = Time::now();
fsec fs = t1 - t0;
ms d = std::chrono::duration_cast<ms>(fs);
total += d.count();
}
```

第 10 步 修改“step 8, Process output”添加如下红框中所示代码。

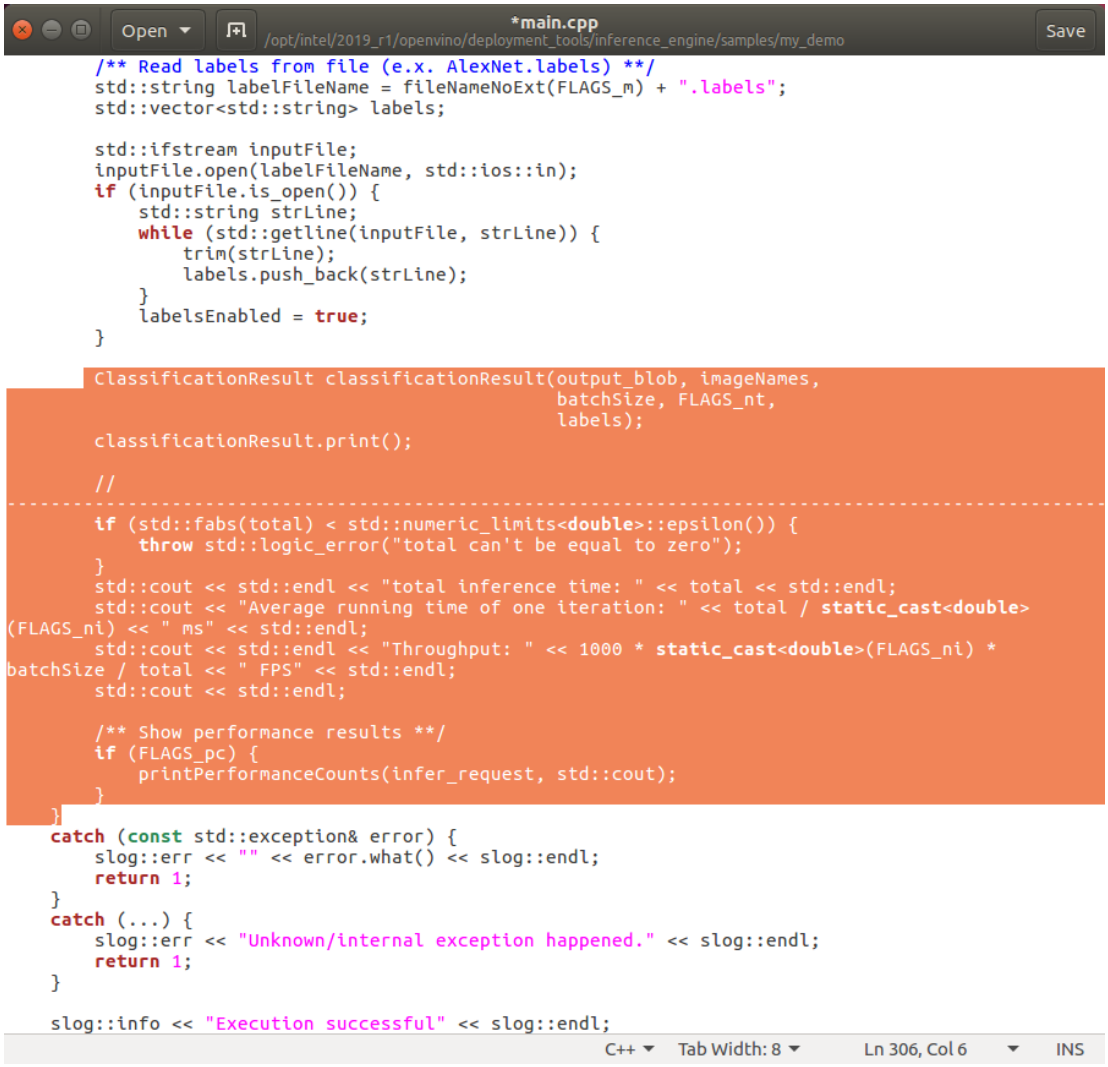
```
// --------------------------- 8. Process output -------------------------------------------------------
slog::info << "Processing output blobs" << slog::endl;

const Blob::Ptr output_blob = infer_request.GetBlob(firstOutputName);
auto output_data = output_blob->buffer().as<PrecisionTrait<Precision::FP32>::value_type*>();

/** Validating -nt value **/
const size_t resultsCnt = output_blob->size() / batchSize;
if (FLAGS_nt > resultsCnt || FLAGS_nt < 1) {
    slog::warn << "-nt " << FLAGS_nt << " is not available for this network (-nt should be less than " \
               << resultsCnt+1 << " and more than 0)\n            will be used maximal value : " << resultsCnt;
    FLAGS_nt = resultsCnt;
}

std::vector<unsigned> results;
TopResults(FLAGS_nt, *output_blob, results);
std::cout << std::endl << "Top " << FLAGS_nt << " results:" << std::endl << std::endl;
bool labelsEnabled = false;
/** Read labels from file (e.x. AlexNet.labels) **/
std::string labelFileName = fileNameNoExt(FLAGS_m) + ".labels";
std::vector<std::string> labels;

std::ifstream inputFile;
inputFile.open(labelFileName, std::ios::in);
if (inputFile.is_open()) {
    std::string strLine;
    while (std::getline(inputFile, strLine)) {
        trim(strLine);
        labels.push_back(strLine);
    }
labelsEnabled = true;
}


ClassificationResult classificationResult(output_blob, imageNames,
                                          batchSize, FLAGS_nt,
                                          labels);
classificationResult.print();

// -----------------------------------------------------------------------------------------------------
if (std::fabs(total) < std::numeric_limits<double>::epsilon()) {
    throw std::logic_error("total can't be equal to zero");
}
std::cout << std::endl << "total inference time: " << total << std::endl;
std::cout << "Average running time of one iteration: " << total / static_cast<double>(FLAGS_ni) << " ms" << std::endl;
std::cout << std::endl << "Throughput: " << 1000 * static_cast<double>(FLAGS_ni) * batchSize / total << " FPS" << std::en
```

```
auto output_data = output_blob->buffer().as<PrecisionTrait<Precision::FP32>::value_type*>();
```

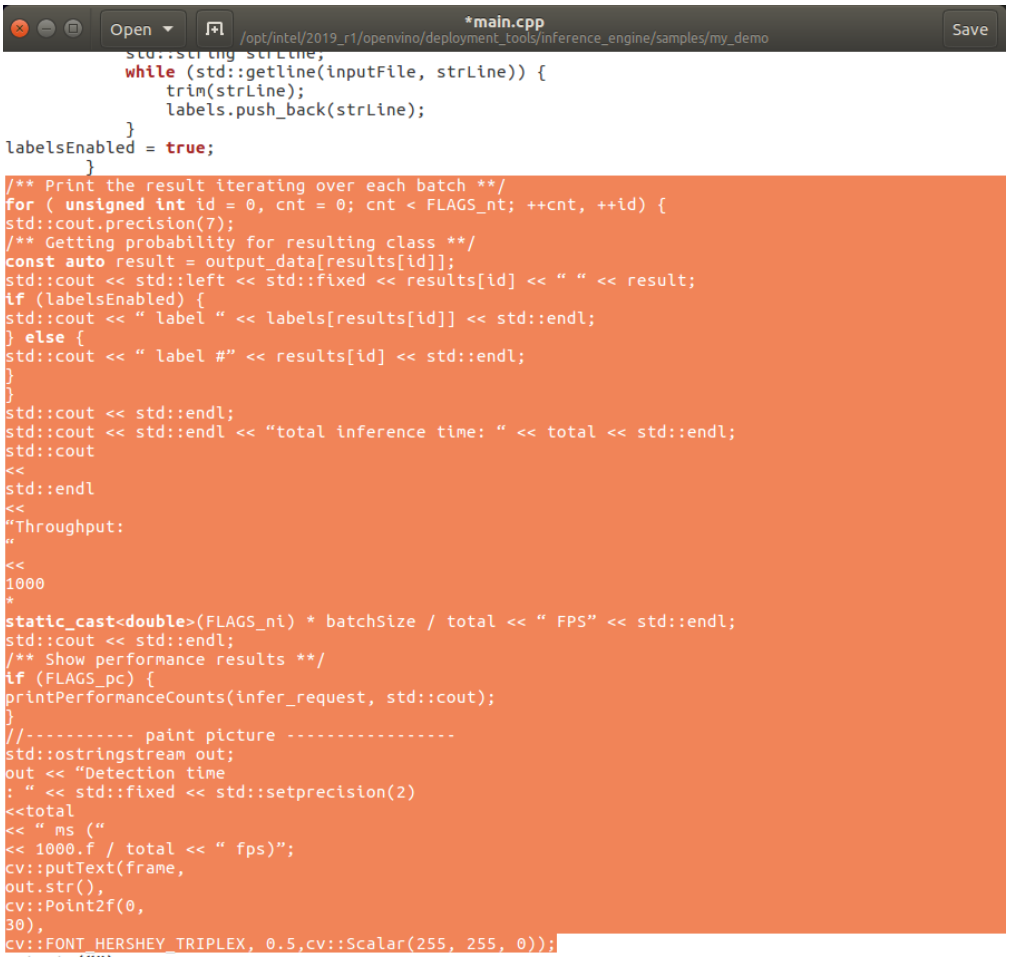
```
std::vector<unsigned> results;
TopResults(FLAGS_nt, *output_blob, results);


std::cout << std::endl << "Top " << FLAGS_nt << " results:" << std::endl << std::endl;
bool labelsEnabled = false;
```

```
labelsEnabled = true;
```

删除如下代码：



第 11 步 添加如下图片所示的代码。




```
/** Print the result iterating over each batch **/
            for (unsigned int id = 0, cnt = 0; cnt < FLAGS_nt; ++cnt, ++id) {
                std::cout.precision(7);
                /** Getting probability for resulting class **/
                const auto result = output_data[results[id]];
                std::cout << std::left << std::fixed << results[id] << " " << result;
                if (labelsEnabled) {
                    std::cout << " label " << labels[results[id]] << std::endl;
                } else {
                    std::cout << " label #" << results[id] << std::endl;
                }
            }
            std::cout << std::endl;

        std::cout << std::endl << "total inference time: " << total << std::endl;
            std::cout << std::endl << "Throughput: " << 1000 * static_cast<double>(FLAGS_ni)
* batchSize / total << " FPS" << std::endl;
            std::cout << std::endl;
```

```
            /** Show performance results **/
            if (FLAGS_pc) {
                printPerformanceCounts(infer_request, std::cout);
            }

        //---------- paint picture ----------------
        std::ostringstream out;
            out << "Detection time   : " << std::fixed << std::setprecision(2) <<total
                << " ms ("
                << 1000.f / total << " fps)";
            cv::putText(frame, out.str(), cv::Point2f(0, 30), cv::FONT_HERSHEY_TRIPLEX,
0.5,
                        cv::Scalar(255, 255, 0));
        out.str("");
        out << "Detection result   : " << std::fixed << std::setprecision(2) << " Label: " <<
labels[results[0]] << " "<< output_data[results[0]];
            cv::putText(frame, out.str(), cv::Point2f(0, 60), cv::FONT_HERSHEY_TRIPLEX,
0.5,
                        cv::Scalar(0, 0, 255));

        //---------------- picture display ------------------
        cv::imshow("Detection results", frame);
        const int key = cv::waitKey(1000);
        if (27 == key)   // Esc
                break;
    }
  }
```

第 12 步 修改后，保存 main.cpp。

8. 输入“*cd /root/inference_engine_samples_build/*”进入 sample_build 文件夹
9. 输入“*rm CMakeCache.txt*”清除 build 缓存。
10. 输入 “*cmake -DCMAKE_BUILD_TYPE=Release \ /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples*”

```
root@openvino2019R1:/opt/intel/2019_r1/openvino/deployment_tools/inference_engine/
samples/my_demo# cd /root/inference_engine_samples_build/
root@openvino2019R1:~/inference_engine_samples_build# rm CMakeCache.txt
root@openvino2019R1:~/inference_engine_samples_build# cmake -DCMAKE_BUILD_TYPE=Rel
ease \
> /opt/intel/2019_r1/openvino/deployment_tools/inference_engine/samples
```

11. 输入“*make my_demo*”，编译应用程序。

```
root@openvino2019R1:~/inference_engine_samples_build# make my_demo
```

12. 输入“*cd intel64/Release*”切换到 app 目录。在该目录下，会生成与 my_demo 对应的可执行文件，完成新应用程序。

```
root@openvino2019R1:~/inference_engine_samples_build# cd intel64/Release/
root@openvino2019R1:~/inference_engine_samples_build/intel64/Release# ls
benchmark_app                       multi-channel-face-detection-demo
calibration_tool                    multi-channel-human-pose-estimation-demo
classification_sample               my_classification_sample
classification_sample_async         my_demo
classification_sample_for_c5p       object_detection_demo
classification_sample_for_pic_loop  object_detection_demo_ssd_async
crossroad_camera_demo               object_detection_demo_yolov3_async
end2end_video_analytics_ie          object_detection_sample_ssd
end2end_video_analytics_opencv      pedestrian_tracker_demo
hello_autoresize_classification     perfcheck
hello_classification                security_barrier_camera_demo
hello_request_classification        segmentation_demo
hello_shape_infer_ssd               smart_classroom_demo
human_pose_estimation_demo          speech_sample
interactive_face_detection_demo     style_transfer_sample
lenet_network_graph_builder         super_resolution_demo
lib                                 text_detection_demo
mask_rcnn_demo                      validation_app
```

13. 输入 “*./my_demo -d "HETERO:FPGA,CPU" -i *
/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/
*pic_video/openvino_pictures -m *
/opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/ir/
FP16/squeezenet1.1/squeezenet1.1.xml”来运行主机应用程序。

```
root@openvino2019R1:~/inference_engine_samples_build/intel64/Release# ./my_demo
-d "HETERO:FPGA,CPU" -i \/opt/intel/2019_r1/openvino/deployment_tools/terasic_de
mo/demo/\
> pic_video/openvino_pictures -m \
> /opt/intel/2019_r1/openvino/deployment_tools/terasic_demo/demo/ir/\
> FP16/squeezenet1.1/squeezenet1.1.xml
```

14. 运行结果如下。