

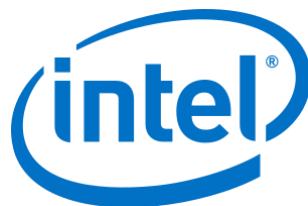
Exercise Manual *for* **Inference with FPGAs**

Software Requirements

64-bit Linux* Software Development Environment with g++
Intel® Distribution of OpenVINO™ toolkit Linux for FPGA version R3

Hardware Requirements

Intel® Programmable Acceleration Card with Intel® Arria® 10 FPGA GX FPGA



Exercise 1

Intel® Distribution of OpenVINO™ toolkit Application for the CPU

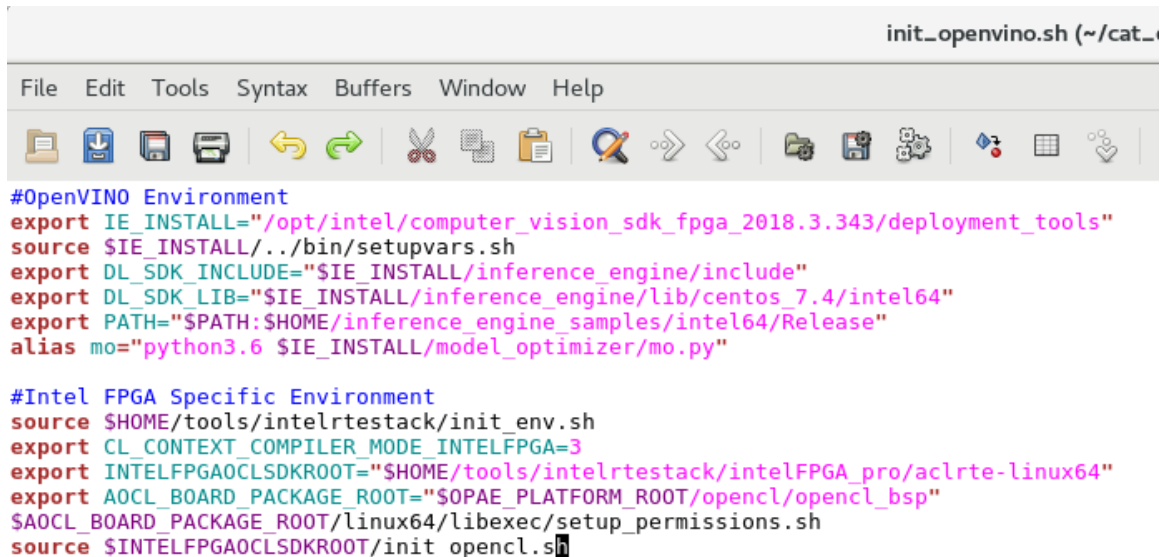
The Intel® Deep Learning Deployment Toolkit that's part of the Intel® Distribution of OpenVINO™ toolkit allows you to easily accelerate and deploy CNNs on Intel® platforms including CPUs, GPUs, VPU, and FPGAs.

In this exercise we will practice writing a simple cat classification application using the OpenVINO toolkit. The code written can be applied to any of the supported Intel® platforms but for ease of deployment we will target the CPU first and then the FPGA in the next exercise.

This application takes video or images as input and will classify the object in the image as one of 32 types of dogs or cats.

Step 1. Setup Lab Environment

- ____ 1. Open a terminal in your Linux* system
- ____ 2. Extract the exercise files associated with the training
 - a. "tar -xf cat_dog_classification.tar.gz"
- ____ 3. Change directory (cd) into the extracted directory, this directory will now be referred to as <Lab Dir>
- ____ 4. Examine the environment script init_openvino.sh



```

init_openvino.sh (~/.cat_...

File Edit Tools Syntax Buffers Window Help

#OpenVINO Environment
export IE_INSTALL="/opt/intel/computer_vision_sdk_fpga_2018.3.343/deployment_tools"
source $IE_INSTALL/./bin/setupvars.sh
export DL_SDK_INCLUDE="$IE_INSTALL/inference_engine/include"
export DL_SDK_LIB="$IE_INSTALL/inference_engine/lib/centos_7.4/intel64"
export PATH="$PATH:$HOME/inference_engine_samples/intel64/Release"
alias mo="python3.6 $IE_INSTALL/model_optimizer/mo.py"

#Intel FPGA Specific Environment
source $HOME/tools/intelrtestack/init_env.sh
export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
export INTELFPGAOCLSDKROOT="$HOME/tools/intelrtestack/intelFPGA_pro/aclrte-linux64"
export AOCL_BOARD_PACKAGE_ROOT="$SOPAE_PLATFORM_ROOT/openccl/openccl_bsp"
$AOCL_BOARD_PACKAGE_ROOT/linux64/libexec/setup_permissions.sh
source $INTELFPGAOCLSDKROOT/init_openccl.sh
  
```

Figure 1. Environment script init_openvino.sh

In the script there are two sections, OpenVINO environment and Intel® FPGA environment settings.

- ____ 5. Edit the script to match your environment
 - a. Make sure IE_INSTALL is set to the location of the installed deployment tools

- b. If you're using one of the Programmable Acceleration Cards, ensure to run the `init_env.sh` in the script
- c. Set the `INTELFPGAOCCLSDKROOT` to the Intel FPGA OpenCL™ runtime
- d. Set `AOCL_BOARD_PACKAGE_ROOT` to the `opencl_bsp` directory with in the Acceleration Stack directory.

____ 6. Source `init_opencl.sh` from the terminal

- a. `source init_openvino.sh`

____ 7. Create a build directory in the <Lab Dir> and `cd` into it

- a. `mkdir build`
- b. `cd build`
- c. `cmake ..`

This will create the build environment.

____ 8. Build the sample application

- a. `make all`

____ 9. Test the application

- a. Type: `../bin/intel64/Release/demo -h`

You should see the options for running the demo as shown in the following figure.

```

Terminal
File Edit View Search Terminal Help
bash-4.2$ ../bin/intel64/Release/demo -h
Options:
  -h                Print a usage message.
  -m <path>         Path to IR .xml file. Required.
  -i <path>         Path to input image or video. If no input path provided application will use web-cam as video source.
  -l <path>         Path to label file. Required.
  -mean <value>     Mean value. Eg: 127,127,129.
  -scale <value>    Scale value. Eg: 0.5,0.6,0.7.
  -d CPU|GPU|FPGA  Device name. CPU by default.
bash-4.2$

```

Figure 2. Screen capture of the options for running the demo

Step 2. Write the Inference Engine Code for the Application

The Cat and Dog classification application we're going to be writing will run an image through a provided deep learning network. The network is the output of the Model Optimizer provided in an Intermediate Representation (IR) format (.xml and .bin). We're going to be completing the code for the following inference process. Should you encounter any issues you may also consult the solution directory.

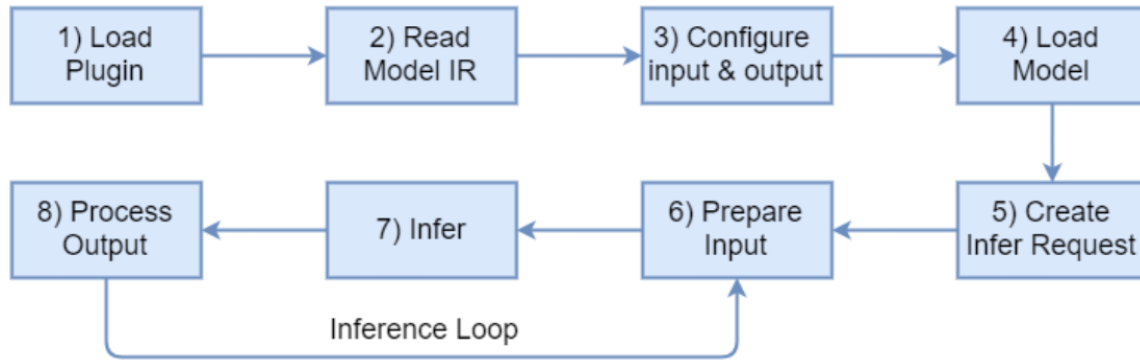


Figure 3. Inference setup and workflow

- ___ 1. Change directory into the source code directory

- a. `cd <Lab Dir>/demo`

- ___ 2. Open main.cpp in your favorite text editor

- ___ 3. Look for the comment TODO: Step 1, Load the Plugin

The first step in the inference process is to load the appropriate plugin. The OpenVINO API can be used with a variety of devices so you can easily switch between inference accelerators with the same network and source code. Different plugins support different accelerator devices.

- ___ 4. Assign the variable **plugin** to the plugin gotten from the device flag **FLAGS_d** passed in from the command line

```
plugin = dispatcher.getPluginByDevice(FLAGS_d);
```

The device defaults to CPU, which is what we'll be using in this exercise, in exercise 2, we will attempt to run the inference on the FPGA using the -d device flag.

- ___ 5. Look for the comment TODO: Step 2, Read IR Generated by Model Optimizer

The next step in the inference process is to read the IR generated by the optimizer. The IR is passed in by the -m flag and is already stored in the variable modelPath.

The function LoadNetwork is already written for you (in network.cpp) which makes it easy to load in both the .xml topology file and .bin weights file. The LoadNetwork function also performs the function of configuring and allocating input and output structures of the inference.

- ___ 6. Complete the code to load the network below the Step 3 comment

```
network -> LoadNetwork(modelPath, 1);
```

The arguments for the function are the path to the .xml IR file and the batch size.

- ____ 7. Look for the comment TODO: Step 4 Load Model

This step will load the read model into the plugin.

- ____ 8. Below the comment, complete the code using the already written LoadNetworkToPlugin() function

```
Network->LoadNetworkToPlugin();
```

- ____ 9. Look for while loop few lines below.

If the input is a video or a camera feed, this while loop will loop through the frames and perform inference on each one. If the source is an image then it will perform just one inference.

- ____ 10. Locate Step 6 Prepare Input

The ConvertBGR function is already written for you (in util.cpp) to convert the RGB information from the frame and convert it to the BGR format that most networks are trained with. While the function is converting, it also assigns the value to the Input data structure needed prior to performing inference.

- ____ 11. Locate the comment “TODO: Step 7 Perform Inference”

We are now ready to perform inference on the frame.

- ____ 12. Right below the comment, write the code to perform the inference

```
Network->Infer();
```

The function written for you (in network.cpp) will call the Infer function on the plugin and perform error checks.

- ____ 13. Locate “Step 8 Process Output”

This is the final step in the inference process. This function will parse the outputs to look for the top N results, after which the application will display the frame along with the classification performance in inferences/sec, the top inference results and associated confidence value.

- ____ 14. Save the file main.cpp

- ____ 15. Back in the terminal, go to the build directory

```
a. cd <Lab Dir>/build
```

- ____ 16. Type “make” to build the application

Fix any compile errors that may have arisen.

- ____ 17. Go to location of the compiled executable

```
a. cd <Lab Dir>/bin/intel64/Release/
```

- ____ 18. Execute the application with one of the provided images

19. `./demo -i cat6.jpg -l labels.txt -m breed_fp32.xml`

The exercise includes several dog and cat images, choose the image you like and pass it to the application with the `-i` flag.

Use the `-l` flag to specify the `labels.txt` which contains the named labels for the 32 different dog and cat classes.

The `-m` flag is used to specify the intermediate representation. The exercise includes two IRs for single precision and half precision floating point data types. For execution on the CPU use the 32 bit IR. The intermediate representation was generated from a Tensorflow* model using the Model Optimizer.

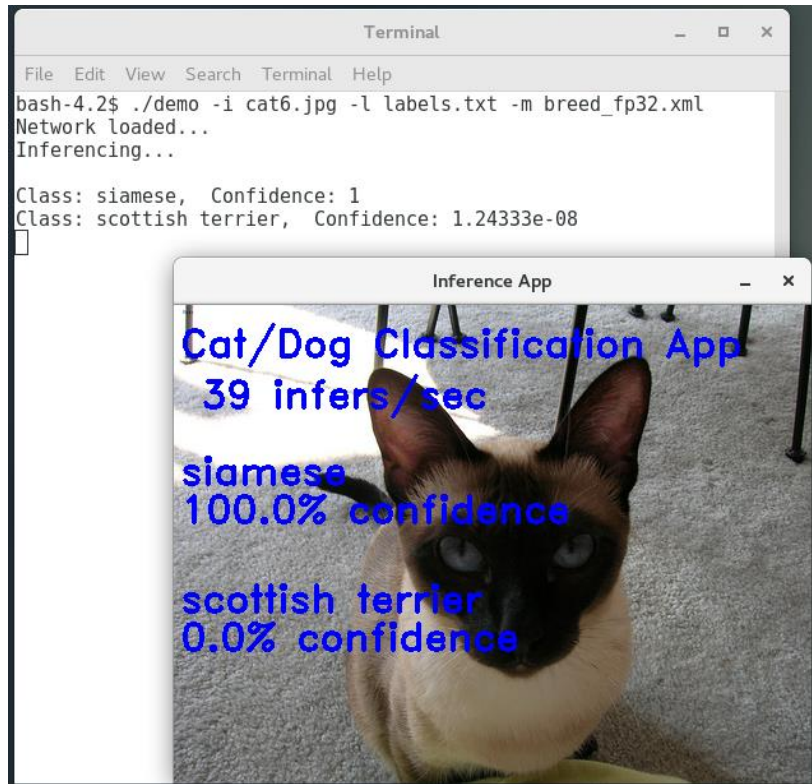


Figure 4. Cat classification results

You should see something similar to the above. Notice the performance as executed on the CPU.

Exercise Summary

- Practiced writing an OpenVINO toolkit inference engine application.

END OF EXERCISE 1