Xilinx AI SDK User Guide

UG1354 (v2.0) August 13, 2019

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
08/13/2019 Version 2.0	
Entire document	Updated framework figure. Added "About this document" and "Release Notes" in Chapter 1. Updated Installation in Chapter 2. Added Programming Examples chapter. Added Application demos chapter. Added Resnet18, face landmark and ReID model. Updated Performance data for ZCU102, ZCU104, Ultra96. Removed the original Chapter 3: Installation. Removed the original Chapter 4: Cross-Compiling Removed the original Chapter 6: Libraries Advanced Application. Removed Roadline_deephi Model.
5/31/2019 Version 1.2	
Chapter 5: Libraries	Added Inception_V4, YOLOV2, Roadline_deephi model Removed RefineDet_640x480 model.
Chapter 8: Performance	Updated Performance data for ZCU102, ZCU104, Ultra96.
5/24/2019 Version 1.1	
Entire document	Editorial updates.
4/29/2019 Version 1.0	
Initial Xilinx release.	N/A

Table of Contents

About This Document

Related Version

The following table lists the SDK version related to this document.

SDK Name	Version
xilinx_ai_sdk_v2.0.x.tar.gz	Compatible with all v2.0.x versions, such as v2.0.5

Intended Audience

The main users using SDK are as follows:

- Users who want to use Xilinx's models to quickly build applications.
- Users who use their own models which are retrained by their own data under the SDK support network list.
- Users who have custom models, similar to the model supported by the SDK and use the SDK's post processing library.

Note: If the users have custom models that are totally different from the model supported by the SDK or has a special post-processing part, users can also use our samples and libraries implementation for reference. Of course, in this case, users can also use DNNDK for development.

Document Navigation

This document describes how users install, use, and develop with the SDK.

- Chapter 1 is the introduction of the SDK. By reading this chapter, users will have a clear understanding of the SDK in general, its framework, supported networks, supported hardware platforms and so on.
- Chapter 2 describes how to install the SDK and run the example. By reading this chapter, users will quickly set up the host and target environments, compile and execute the SDK related examples.
- Chapter 3 describes, in detail, each model library supported by the SDK. By reading this chapter, users will understand the model libraries supported by the SDK, the purpose of each library, how to test the library with images or videos and how to test the performance of the library.

- Chapter 4 describes, in detail, how to develop applications with SDK libraries. By reading this chapter, users will understand the following:
  - how to develop with Xilinx models
  - how to develop with user-retrained models
  - how to customize pre-processing
  - how to use the configuration file as pre and post processing parameters
  - how to use the SDK post-processing library
  - how to implement user post-processing code
- Chapter 5 describes how to set up a test environment and run the application demos. There are two application demos provided with the SDK.
- Chapter 6 describes how to find the programming APIs.
- Chapter 7 describes, in detail, the performance of the SDK on different boards.

Overview

The Xilinx AI SDK is a set of high-level libraries and APIs built for efficient AI inference with Deep-Learning Processor Unit (DPU). It provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks. This simplifies the use of deep-learning neural networks, even for users without knowledge of deep-learning or FPGAs. The Xilinx AI SDK allows users to focus more on the development of their applications, rather than the underlying hardware.

The Xilinx AI SDK and DNNDK are two development packages which can be used for AI inference with DPU, but with different positioning. DNNDK provides low-level APIs which focuses on control and operate DPU in finer granularity, while the AI SDK provides high-level APIs which are focused on the whole AI applications. Using the Xilinx AI SDK, users can not only use DPU efficiently, but also use the variety of pre-processing and post-processing implementations from many models which are key to end-to-end AI applications. The Xilinx AI SDK can help customers for fast prototyping and quickly deploying their models to DPU.

For the intended audience for the SDK, please refer to the About This Document section.

Block Diagram

The Xilinx AI SDK contains four parts: the base libraries, the model libraries, the library test samples and the application demos.

The base libraries mainly provide the operation interface with the DPU and the post-processing module of each model. "dpbase" is the interface library for DPU operations. "xnnpp" is the post-processing library of each model, with build-in modules such as optimization and acceleration. Note that **the base libraries are closed sources**.

The model libraries implement most of the neural network deployment which **are open sources**. They mainly include common types of network, such as classification, detection, segmentation, etc. These libraries provide an easy-to-use and fast development way in a unified interface, which are applicable to the Xilinx models or custom models.

The library test samples are used to quickly test and evaluate the model libraries.

The application demos show users how to use SDK libraries to develop applications.

The Xilinx AI SDK block diagram is shown in Figure 1.

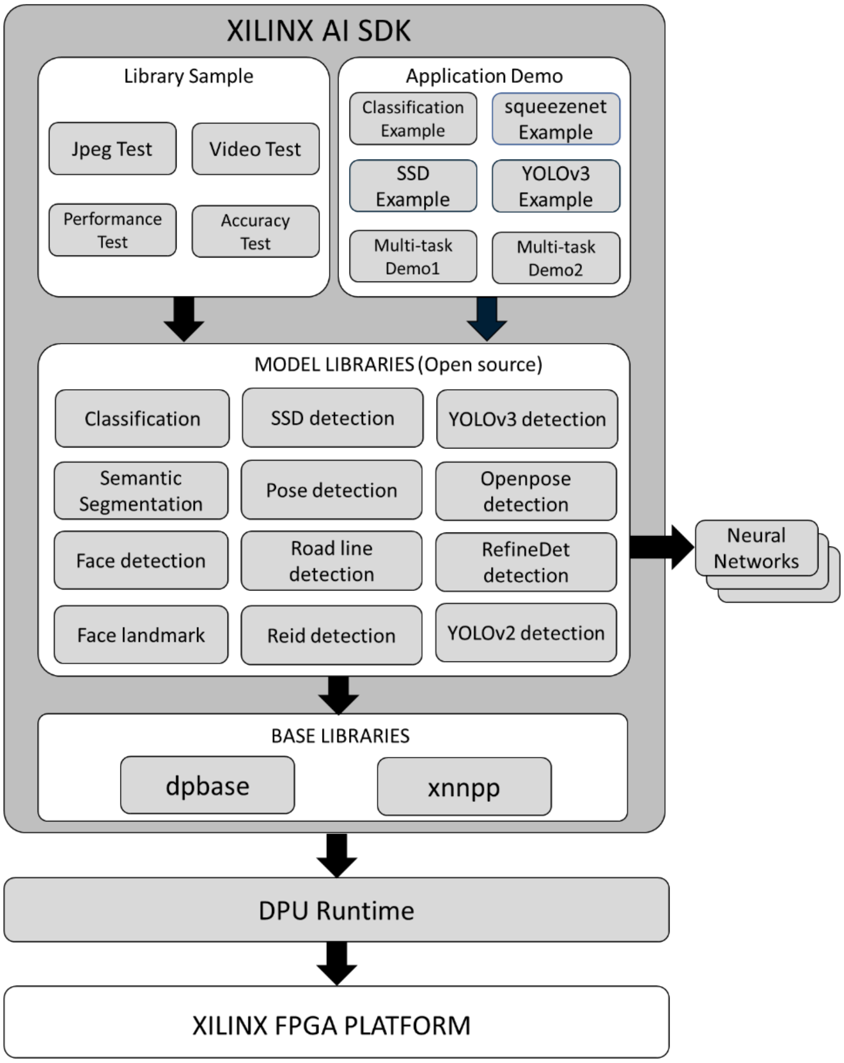


Figure 1 : Xilinx AI SDK Block Diagram

Features

The Xilinx AI SDK features include:

- A full-stack application solution from top to buttom
- Optimized pre- and post-processing functions/libraries
- Open-source model libraries
- Unified operation interface with the DPU and the pre- and post-processing interface of the model
- Practical, application-based model libraries, pre- and post-processing libraries, and application examples

Release Notes

Release 2.0.5

Released July 2019

What's New

Architectural Change

- The new SDK is separate from the DNNDK and can be installed independently

AI Library

- Up to 12 open-source AI libraries
- Add xnnpp library which unifies post-processing interface
- Add more models, such face landmark, ReID and ResNet-18 etc.

Example

- Add four examples to show how to use AI SDK
- Add two demos about multi-task/ multi-model application

Supported Neural Network and Platforms

Supported Neural Network

The following neural networks are supported by the Xilinx AI SDK.

Table 1: Neural Network Supported by the SDK

No.	Neural Network	Application
1	ResNet-18	Image Classification
2	ResNet-50	
3	Inception-v1	
4	Inception-v2	
5	Inception-v3	
6	Inception-v4	
7	MobileNet-v1_TF[1]	
8	MobileNet-v2	

9	SqueezeNet	
10	ResNet-50_TF[1]	
11	Inception-v1_TF[1]	
12	Resnet-18_TF[1]	
13	MobileNet-v2_TF[1]	
14	SSD_ADAS_VEHICLE	ADAS Vehicle Detection
15	SSD_ADAS_PEDESTRIAN	ADAS Pedestrian Detection
16	SSD_TRAFFIC	Traffic Detection
17	SSD_MobileNet-v2	Object Detection
18	SSD_VOC_TF[1]	Object Detection
19	DenseBox_320x320	Face Detection
20	DenseBox_640x360	
21	YOLOV3_ADAS_512x256	ADAS Detection
22	YOLOV3_ADAS_512x288	
23	YOLOV3_VOC	Object Detection
24	YOLOV3_VOC_TF[1]	
25	YOLOV2_BASELINE	
26	YOLOV2_COMPRESS22G	
27	YOLOV2_COMPRESS24G	
28	YOLOV2_COMPRESS26G	
29	RefineDet	
30	RefineDet_10G	
31	RefineDet_5G	
32	FPN (segmentation)	ADAS Segmentation
33	VPGnet (roadline detection)	ADAS Lane Detection

34	Sp-net (pose detection)	Pose Estimation
35	Openpose_368x368	
36	Face Landmark	Face Detection and Recognition
37	ReID	Object tracking

[1] These neural network models are trained based on the Tensorflow framework.

Supported Host Operating Systems

The following operating system have been validated to work with the XILINX AI SDK.

Operating System	Version
Ubuntu	16.04 (64-bit) 18.04 (64-bit)

Supported Platforms

The following platforms and EVBs are supported by the XILINX AI SDK

Platform	EVB	Version
Zynq UltraScale+ MPSoC ZU9EG	Xilinx ZCU102	V1.1
Zynq UltraScale+ MPSoC ZU7EV	Xilinx ZCU104	V1.0
Zynq UltraScale+ MPSoC ZU3EG	Ultra96	V1

Downloading the Xilinx AI SDK

The Xilinx AI SDK package can be freely downloaded after registration on the Xilinx website: https://www.xilinx.com/products/design-tools/ai-inference/edge-ai-platform.html.

Using a Xilinx AI SDK-supported evaluation board is recommended to allow you to become familiar with the product. Refer to https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge for more details about the Xilinx AI SDK-supported evaluation boards.

The evaluation boards supported for this release are:

- Xilinx ZCU102
- Xilinx ZCU104
- Avnet Ultra96

Setting Up the Host

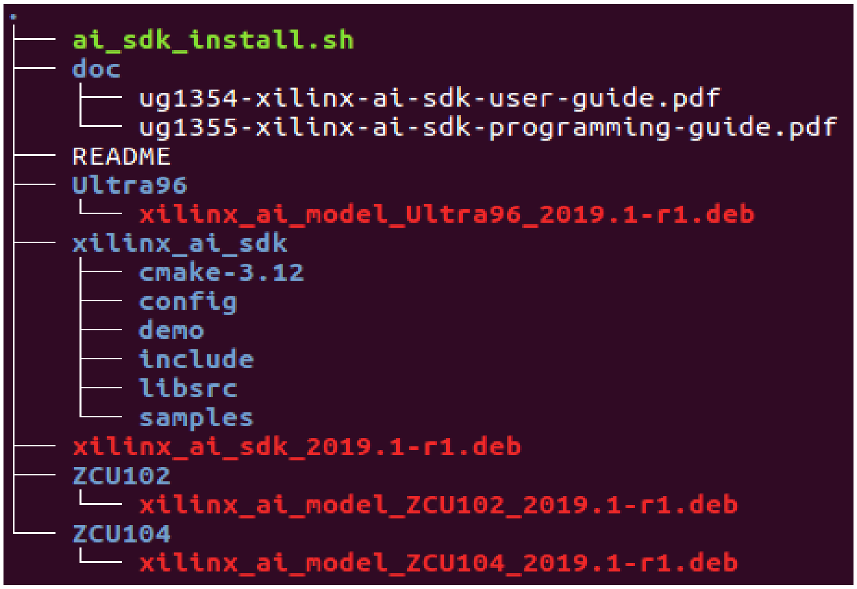
1. Download the corresponding SDK package from https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge, such as `xilinx_ai_sdk_v2.0.5.tar.gz`.

   Note that the instructions for the rest of this document refer to a version of the SDK as xilinx_ai_sdk_v2.0.x.

2. Extract the package contents.

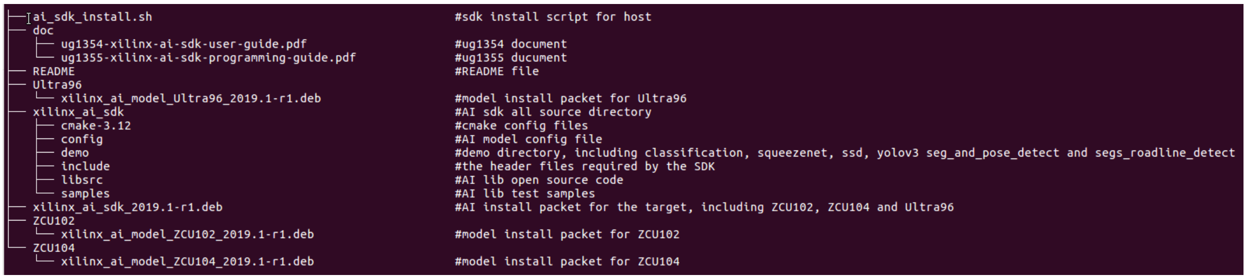
```
$cd ~
$tar zxvf xilinx_ai_sdk_v2.0.x.tar.gz
```

   Figure 2 is the directory structure diagram after the package is unzipped and Figure 3 is the description of directory structure.



```
.
├── ai_sdk_install.sh
├── doc
│   ├── ug1354-xilinx-ai-sdk-user-guide.pdf
│   └── ug1355-xilinx-ai-sdk-programming-guide.pdf
├── README
├── Ultra96
│   └── xilinx_ai_model_Ultra96_2019.1-r1.deb
├── xilinx_ai_sdk
│   ├── cmake-3.12
│   ├── config
│   ├── demo
│   ├── include
│   ├── libsrc
│   └── samples
├── xilinx_ai_sdk_2019.1-r1.deb
├── ZCU102
│   └── xilinx_ai_model_ZCU102_2019.1-r1.deb
└── ZCU104
    └── xilinx_ai_model_ZCU104_2019.1-r1.deb
```

Figure 2: SDK Directory Structure



```
├── ai_sdk_install.sh                                 #sdk install script for host
├── doc
│   ├── ug1354-xilinx-ai-sdk-user-guide.pdf           #ug1354 document
│   └── ug1355-xilinx-ai-sdk-programming-guide.pdf    #ug1355 ducument
├── README                                            #README file
├── Ultra96
│   └── xilinx_ai_model_Ultra96_2019.1-r1.deb         #model install packet for Ultra96
├── xilinx_ai_sdk                                     #AI sdk all source directory
│   ├── cmake-3.12                                    #cmake config files
│   ├── config                                        #AI model config file
│   ├── demo                                          #demo directory, including classification, squeezenet, ssd, yolov3 seg_and_pose_detect and segs_roadline_detect
│   ├── include                                       #the header files required by the SDK
│   ├── libsrc                                        #AI lib open source code
│   └── samples                                       #AI lib test samples
├── xilinx_ai_sdk_2019.1-r1.deb                       #AI install packet for the target, including ZCU102, ZCU104 and Ultra96
├── ZCU102
│   └── xilinx_ai_model_ZCU102_2019.1-r1.deb          #model install packet for ZCU102
└── ZCU104
    └── xilinx_ai_model_ZCU104_2019.1-r1.deb          #model install packet for ZCU104
```

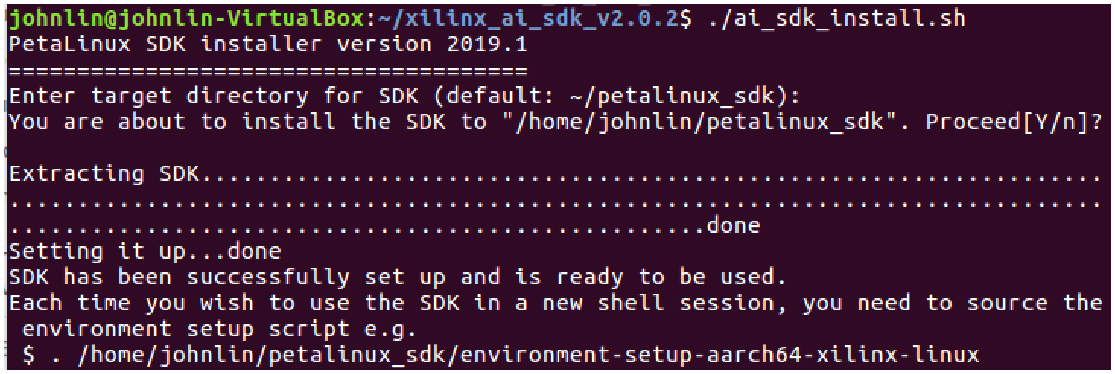
Figure 3: SDK Directory Structure Description

3. Install the AI SDK.

```
$cd xilinx_ai_sdk_v2.0.x
$./ai_sdk_install.sh
```

4. Follow the prompts to install. The following figure shows the installation process.

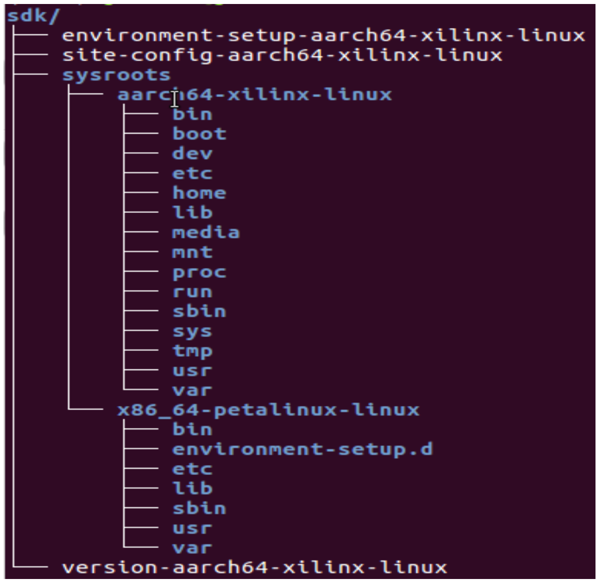
   Note that if the installation is in the default path, ensure that the path has permissions for read and write capabilities. It is recommended that users change the path for installation as they see fit.



```
johnlin@johnlin-VirtualBox:~/xilinx_ai_sdk_v2.0.2$ ./ai_sdk_install.sh
PetaLinux SDK installer version 2019.1
======================================
Enter target directory for SDK (default: ~/petalinux_sdk):
You are about to install the SDK to "/home/johnlin/petalinux_sdk". Proceed[Y/n]?

Extracting SDK.....................................................................
...................................................................................
...................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
 environment setup script e.g.
 $ . /home/johnlin/petalinux_sdk/environment-setup-aarch64-xilinx-linux
```

Figure 4: SDK Installation Process

The following figure shows the directory structure of the cross-compile system after installation.



```
sdk/
├── environment-setup-aarch64-xilinx-linux
├── site-config-aarch64-xilinx-linux
├── sysroots
│   ├── aarch64-xilinx-linux
│   │   ├── bin
│   │   ├── boot
│   │   ├── dev
│   │   ├── etc
│   │   ├── home
│   │   ├── lib
│   │   ├── media
│   │   ├── mnt
│   │   ├── proc
│   │   ├── run
│   │   ├── sbin
│   │   ├── sys
│   │   ├── tmp
│   │   ├── usr
│   │   └── var
│   └── x86_64-petalinux-linux
│       ├── bin
│       ├── environment-setup.d
│       ├── etc
│       ├── lib
│       ├── sbin
│       ├── usr
│       └── var
└── version-aarch64-xilinx-linux
```

Figure 5: Directory Structure of the Cross-Compile System

5. When the installation is complete, follow the prompts and enter the following command:

```
$ . ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
```

   Note that if you close the current terminal, you need to re-execute the above instructions in the new terminal interface.

6. Cross compile the demo in the SDK, take yolov3 as example,

```
$cd ~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/yolov3
$sh -x build.sh
```

   If you don't want to print information during compilation, execute the following command.

```
$sh build.sh
```

   If the compilation process does not report any error and the executable file "`sample_yolov3`" is generated, the host environment is installed correctly.

7. To compile the library sample in the SDK, take `classification` for example, execute the following command.

```
$cd ~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/classification
$sh -x build.sh
```

   Then, the executable program is produced.

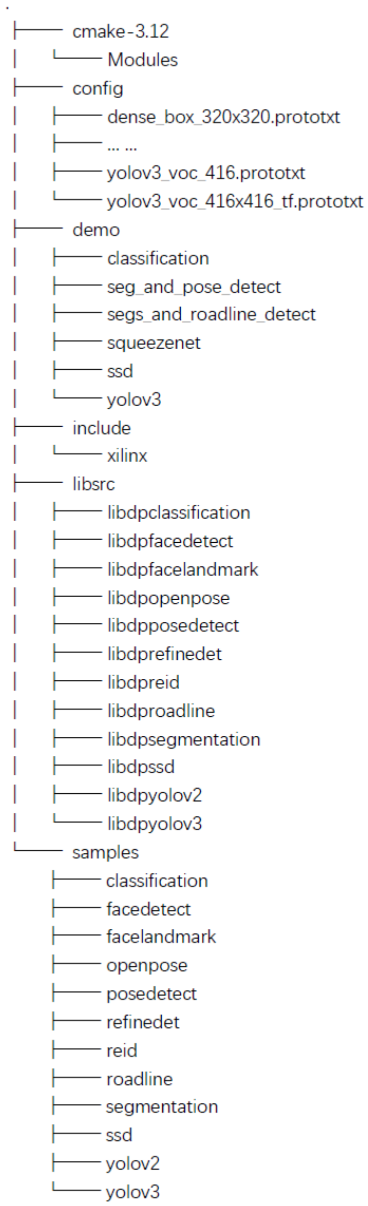
8. To modify the library source code, view and modify them under `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc.`

   If you want to recompile the library, take `libdpclassification` for example, execute the following command:

```
$cd ~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc/libdpclassification
$sh -x build.sh
```

Then the `libdpclassification.so`, the library's test program and the library's example programs are generated. If you want to change the compilation rules, check and change the `CMakeLists.txt` in the library's directory.

Note that all the source code, samples, demos, head files and model config files can be found in `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk`. The following figure is the directory structure diagram of xilinx_ai_sdk.



```
.
├── cmake-3.12
|   └── Modules
├── config
|   ├── dense_box_320x320.prototxt
|   ├── ... ...
|   ├── yolov3_voc_416.prototxt
|   └── yolov3_voc_416x416_tf.prototxt
├── demo
|   ├── classification
|   ├── seg_and_pose_detect
|   ├── segs_and_roadline_detect
|   ├── squeezenet
|   ├── ssd
|   └── yolov3
├── include
|   └── xilinx
├── libsrc
|   ├── libdpclassification
|   ├── libdpfacedetect
|   ├── libdpfacelandmark
|   ├── libdpopenpose
|   ├── libdpposedetect
|   ├── libdprefinedet
|   ├── libdpreid
|   ├── libdproadline
|   ├── libdpsegmentation
|   ├── libdpssd
|   ├── libdpyolov2
|   └── libdpyolov3
└── samples
    ├── classification
    ├── facedetect
    ├── facelandmark
    ├── openpose
    ├── posedetect
    ├── refinedet
    ├── reid
    ├── roadline
    ├── segmentation
    ├── ssd
    ├── yolov2
    └── yolov3
```

Figure 6: Directory Structure of the Xilinx_AI_SDK

Table 2 shows the SDK file location after the installation is complete.

Table 2: SDK File Location List

Files	Location
Source code of the libraries	`~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc`
Library files	`~/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib`
Header files	`~/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/include/xilinx/ai`
Model profiles	`~/petalinux_sdk/sysroots/aarch64-xilinx-linux/etc/dpu_model_param.conf.d`
Samples	`~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples`
Demos	`~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo`
Test	`~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc/[model]/test`
Documents	`~/xilinx_ai_sdk_v2.0.x/doc`

Note: The following symbols/abbreviations are used.

"~/" is the path to extract the SDK compressed package.
"~/petalinux_sdk" is the SDK installation path.
"Samples" is used for rapid application construction and evaluation, and it is for users.
"Demos" provides more practical examples for user development, and it is for users.
"Test" is a test example for each model library which is for library developers.

Setting Up the Target

To set up the target, there are three steps that need to be done. The first step is to install the board image, the second step is to install the AI model packet and the third step is to install the AI SDK packet.

Note that the version of the board image should be 2019.1 or above.

Step 1: Installing a Board Image

1. Download the SD card system image files from https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge (such as ZCU102, ZCU104, or Ultra-96).
2. Use Win32DiskImager (free opensource software) to burn the image file onto the SD card.

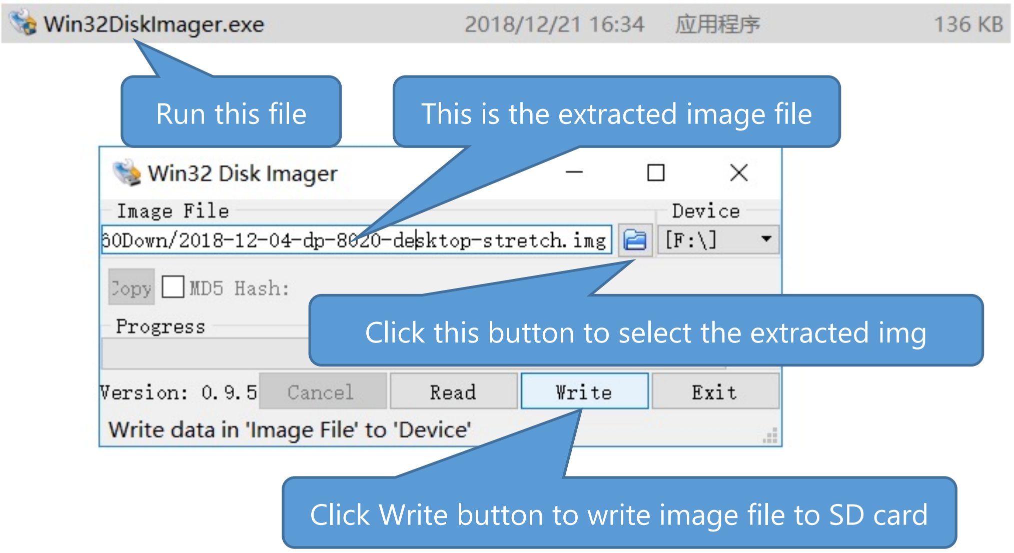


Image File: 60Down/2018-12-04-dp-8020-desktop-stretch.img

Device: [F:\]

Version: 0.9.5

Write data in 'Image File' to 'Device'

Run this file

This is the extracted image file

Click this button to select the extracted img

Click Write button to write image file to SD card

Figure 7: Win32DiskImager Usage Steps

3. Insert the SD card with image into the destination board. Plug in the power and boot the board, using the serial port to operate on the system.
4. Set up the IP information of the board via the serial port. Then you can operate on the board via SSH.

Step 2: Installing AI Model Package

1. Enter the platform directory, then copy the `xilinx_ai_model_ZCU102_2019.1-r1.deb` to the board via scp.

```
$cd ~/xilinx_ai_sdk_v2.0.x/ZCU102
$scp xilinx_ai_model_ZCU102_2019.1-r1.deb  root@IP_OF_BOARD:~/
```

Note that the deb package can be taken as normal archive, and users can extract the contents on the host side if users only need part of the models. The operation command is shown below.

```
$mkdir extract
$dpkg -X xilinx_ai_model_ZCU102_2019.1-r1.deb extract
```

2. Log in the board (via ssh or serial port), install the model package, and execute below:

```
#dpkg -i xilinx_ai_model_ZCU102_2019.1-r1.deb
```

Note the following after the installation is complete.

- Model files are stored in `/usr/lib` on the target side.

Step 3: Installing AI SDK Package

1. Enter the following directory, then copy the `xilinx_ai_sdk_2019.1-r1.deb` to the board via scp.

```
$cd ~/xilinx_ai_sdk_v2.0.x
$scp xilinx_ai_sdk_2019.1-r1.deb  root@IP_OF_BOARD:~/
```

Note that the deb package can be taken as normal archive, and users can extract the contents on the host side if users only need part of the libraries. Only model libraries can be separated dependently, others are common libraries. The operation command is shown below.

```
$mkdir extract
$dpkg -X xilinx_ai_sdk_2019.1-r1.deb extract
```

2. Log in the board (via ssh or serial port), install the Xilinx AI SDK, and execute below:

```
#dpkg -i xilinx_ai_sdk_2019.1-r1.deb
```

Note the following after the installation is complete:

- Library files are stored in `/usr/lib`
- The header files are stored in `/usr/include/xilinx/ai`
- Model profiles are stored in `/etc/dpu_model_param.conf.d`
- Samples are stored in `/usr/share/XILINX_AI_SDK/samples`
- Demos are stored in `/usr/share/XILINX_AI_SDK/demo`

Running Xilinx AI SDK Examples

There are two ways to compile a program. One is to cross-compile the program through the host and the other is to compile the program directly on the target board. Both methods have advantages and disadvantages. In this section, we compile and run the examples directly on the target machine.

1. Enter the extracted directory of example in target board and then compile each of the examples.

```
#cd /usr/share/XILINX_AI_SDK/samples/facedetect
```

2. Run the example.

```
#./test_jpeg_face_detect_dense_box_640x360
 sample_face_detect_dense_box_640x360.jpg
```

If the above executable program does not exist, run the following command to compile and generate the corresponding executable program.

```
#sh -x build.sh
```

3. View the running results.

There are two ways to view the results. One is to view the results by printing information, while the other is to view images by downloading the `sample_face_detect_dense_box_640x360_result.jpg` image (see the following figure).

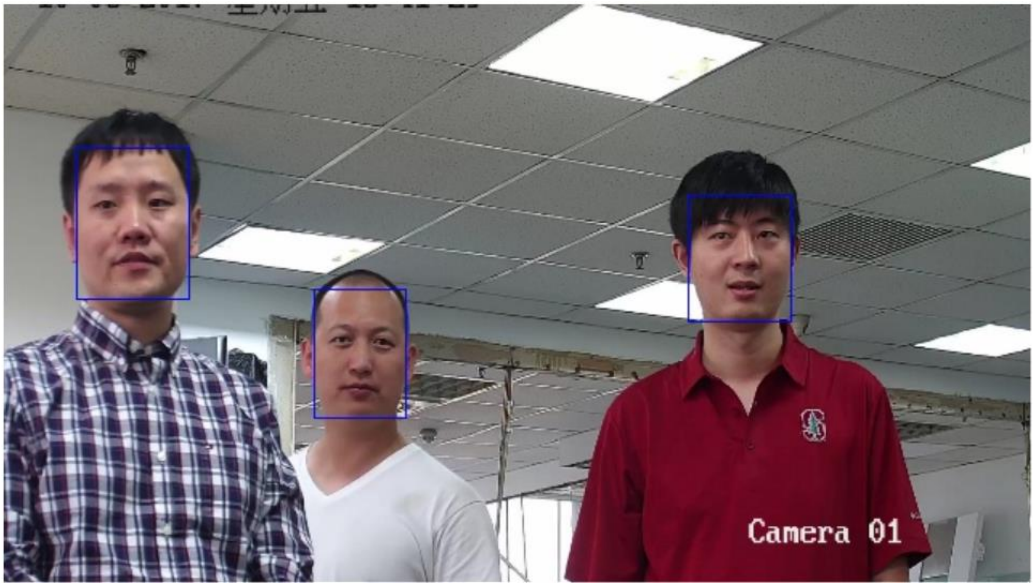


Figure 8: Face Detection Example

4. To run the video example, run the following command:

```
#./test_video_face_detect_dense_box_640x360 video_input.mp4 -t 8

Video_input.mp4: The video file’s name for input.
                 The user needs to prepare the video file.
-t: <num_of_threads>
```

To test the program with a USB camera as input, run the following command:

```
#./test_video_face_detect_dense_box_640x360 0 -t 8

0: The first USB camera device node. If you have multiple USB camera, the
value might be 1,2,3 etc.
-t: <num_of_threads>
```

Make sure to enable X11 forwarding with the following command (suppose in this example that the host machine IP address is 192.168.0.10) when logging in to the board using an SSH terminal because all the video examples require a Linux windows system to work properly.

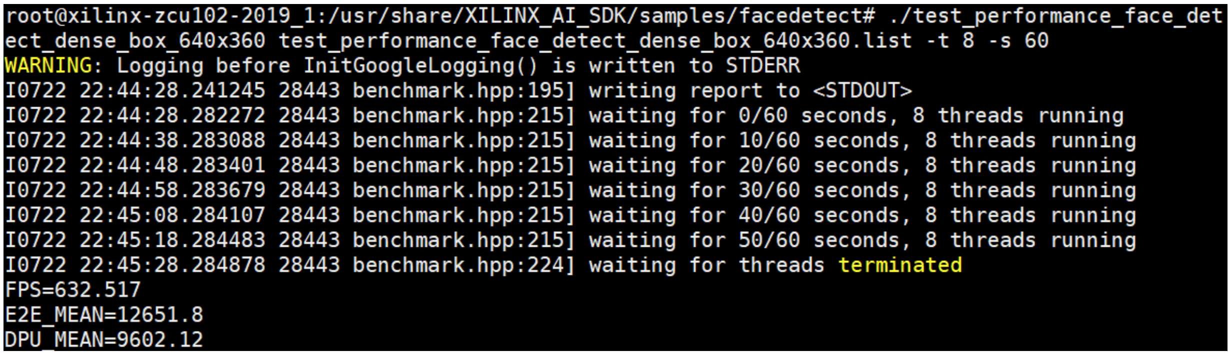
```
#export DISPLAY=192.168.0.10:0.0
```

5. To test the performance of model, run the following command:

```
#./test_performance_face_detect_dense_box_640x360
test_performance_face_detect_dense_box_640x360.list -t 8 -s 60

-t: <num_of_threads>

-s: <num_of_seconds>
```

For more parameter information, enter -h for viewing.

The following figure shows the result of performance testing in 8 threads.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_performance_face_det
ect_dense_box_640x360 test_performance_face_detect_dense_box_640x360.list -t 8 -s 60
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 22:44:28.241245 28443 benchmark.hpp:195] writing report to <STDOUT>
I0722 22:44:28.282272 28443 benchmark.hpp:215] waiting for 0/60 seconds, 8 threads running
I0722 22:44:38.283088 28443 benchmark.hpp:215] waiting for 10/60 seconds, 8 threads running
I0722 22:44:48.283401 28443 benchmark.hpp:215] waiting for 20/60 seconds, 8 threads running
I0722 22:44:58.283679 28443 benchmark.hpp:215] waiting for 30/60 seconds, 8 threads running
I0722 22:45:08.284107 28443 benchmark.hpp:215] waiting for 40/60 seconds, 8 threads running
I0722 22:45:18.284483 28443 benchmark.hpp:215] waiting for 50/60 seconds, 8 threads running
I0722 22:45:28.284878 28443 benchmark.hpp:224] waiting for threads terminated
FPS=632.517
E2E_MEAN=12651.8
DPU_MEAN=9602.12
```

Figure 9: Face Detection Performance Test Result

6. To check the version of XILINX AI SDK, run the following command:

   ```
   #xilinx_ai
   ```

7. To check the information of dpu, run the following command:

   ```
   #xilinx_verreg
   ```

8. To run the demo, refer to chapter 5.

Support

You can visit the Xilinx AI SDK community forum on the Xilinx website https://forums.xilinx.com/t5/Deephi-DNNDK/bd-p/Deephi for topic discussions, knowledge sharing, FAQs, and requests for technical support.

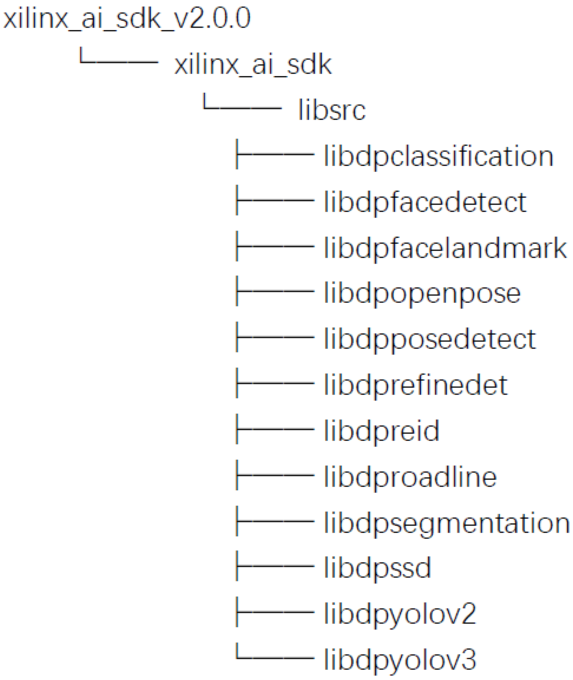
The Xilinx AI SDK contains the following types of neural network libraries based on Caffe framework:

- Classification
- Face detection
- Face landmark detection
- SSD detection
- Pose detection
- Semantic segmentation
- Road line detection
- YOLOV3 detection
- YOLOV2 detection
- Openpose detection
- RefineDet detection
- ReID detection

Also, the Xilinx AI SDK contains the following types of neural network libraries based on Tensorflow framework:

- Classification (ResNet-18, ResNet-50, Inception-V1, MobileNet-V1, MobileNet-V2)
- SSD detection
- YOLOv3 detection

The related libraries are open source and can be modified as needed. The open source codes are store in the `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc` directory, as shown in the figure below.



```
xilinx_ai_sdk_v2.0.0
    └─── xilinx_ai_sdk
             └─── libsrc
                    ├─── libdpclassification
                    ├─── libdpfacedetect
                    ├─── libdpfacelandmark
                    ├─── libdpopenpose
                    ├─── libdpposedetect
                    ├─── libdprefinedet
                    ├─── libdpreid
                    ├─── libdproadline
                    ├─── libdpsegmentation
                    ├─── libdpssd
                    ├─── libdpyolov2
                    └─── libdpyolov3
```

Figure 10: Directory Structure of the LIBSRC

The Xilinx AI SDK provides image test samples and video test samples for all the above networks. In addition, the kit provides the corresponding performance test program. For video based testing, we recommend to use raw video for evaluation. Because decoding by software libraries on ARM CPU may have inconsistent decoding time, which may affect the accuracy of evaluation.

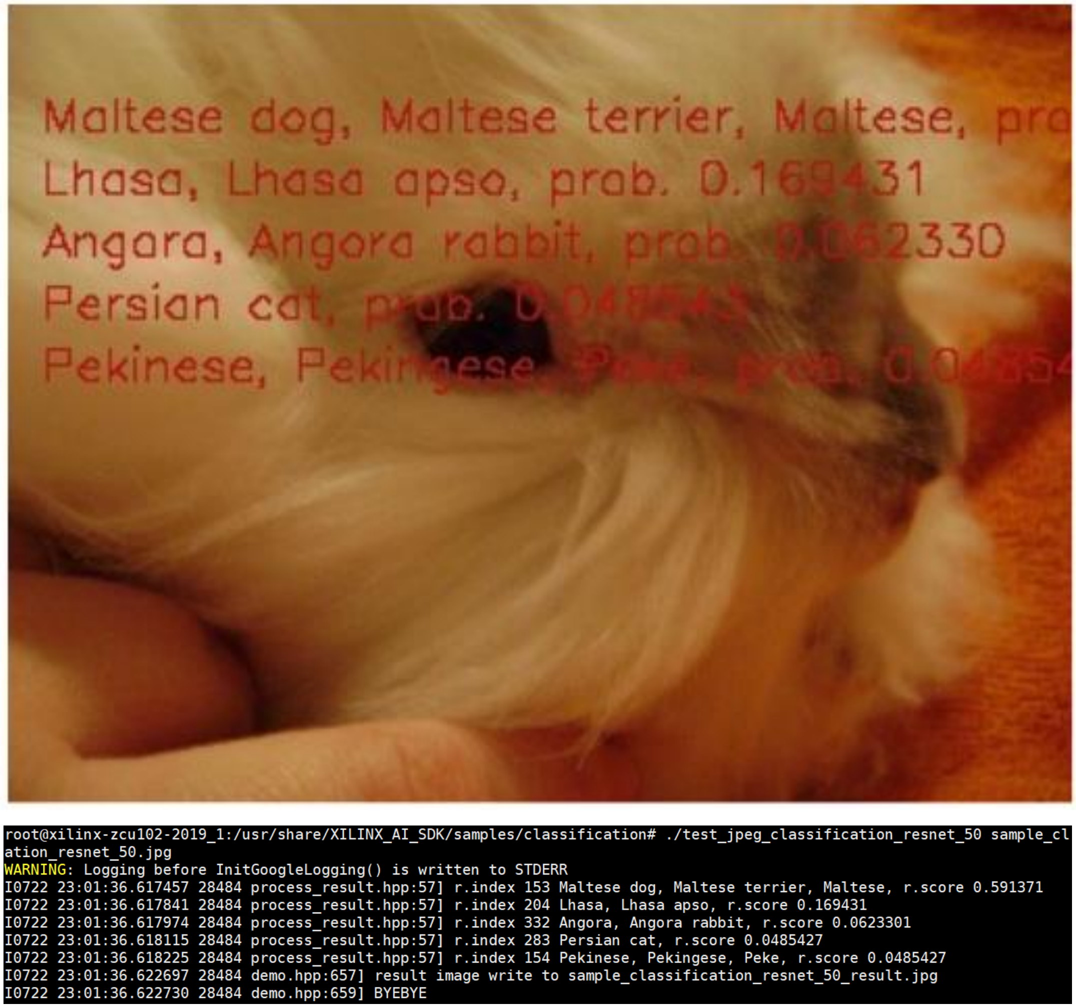
Note that all the sample programs can only run on the target side, but all the sample programs can be cross compiled on the host side or compiled on the target side.

Classification

Introduction

This lib is used to classify images. Such neural networks are trained on ImageNet for ILSVRC and they can identify the objects from its 1000 classification.

The SDK V2.0.x integrated Resnet18, Resnet50, Inception_v1, Inception_v2, Inception_v3, Inception_v4, mobilenet_v1, mobilenet_v2 into our lib. Input is a picture with an object. Output is the top-K most probable category.


Maltese dog, Maltese terrier, Maltese, pro
Lhasa, Lhasa apso, prob. 0.169431
Angora, Angora rabbit, prob. 0.062330
Persian cat, prob. 0.048543
Pekinese, Pekingese, Peke, prob. 0.04854

```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/classification# ./test_jpeg_classification_resnet_50 sample_cl
ation_resnet_50.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:01:36.617457 28484 process_result.hpp:57] r.index 153 Maltese dog, Maltese terrier, Maltese, r.score 0.591371
I0722 23:01:36.617841 28484 process_result.hpp:57] r.index 204 Lhasa, Lhasa apso, r.score 0.169431
I0722 23:01:36.617974 28484 process_result.hpp:57] r.index 332 Angora, Angora rabbit, r.score 0.0623301
I0722 23:01:36.618115 28484 process_result.hpp:57] r.index 283 Persian cat, r.score 0.0485427
I0722 23:01:36.618225 28484 process_result.hpp:57] r.index 154 Pekinese, Pekingese, Peke, r.score 0.0485427
I0722 23:01:36.622697 28484 demo.hpp:657] result image write to sample_classification_resnet_50_result.jpg
I0722 23:01:36.622730 28484 demo.hpp:659] BYEBYE
```

Figure 11: Classification Example

Sample

The Xilinx AI SDK provides classification samples based on image or video. Also, the Xilinx AI SDK provides a sample performance test of the classified network.

1. Before you run the classification example, you can choose one of the following executable programs to run:
   - test_jpeg_classification_resnet_50
   - test_jpeg_classification_resnet_18
   - test_jpeg_classification_inception_v1
   - test_jpeg_classification_inception_v2
   - test_jpeg_classification_inception_v3
   - test_jpeg_classification_inception_v4
   - test_jpeg_classification_mobilenet_v2
   - test_jpeg_classification_resnet_50_tf
   - test_jpeg_classification_inception_v1_tf
   - test_jpeg_classification_mobilenet_v1_tf
   - test_jpeg_classification_mobilenet_v2_tf
   - test_jpeg_classification_resnet_18_tf
   - test_jpeg_classification_squeezenet
2. If the executable program does not exist, it can be compiled and generated as follows:

   ```
   #sh -x build.sh
   ```
3. Run the example.

   ```
   #./test_jpeg_classification_restnet_50 sample_classification_resnet_50.jpg
   ```
4. You will see the print result on the terminal (see Figure 11).
5. Run the video example.

   ```
   #./test_video_classification_restnet_50 video_input.mp4 -t 8
   ```
6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the classification performance example, you can choose one of the following executable programs to run:
   - test_performance_classification_resnet_50
   - test_performance_classification_resnet_18
   - test_performance_classification_inception_v1
   - test_performance_classification_inception_v2

- test_performance_classification_inception_v3
- test_performance_classification_inception_v4
- test_performance_classification_mobilenet_v2
- test_performance_classification_resnet_50_tf
- test_performance_classification_inception_v1_tf
- test_performance_classification_mobilenet_v1_tf
- test_performance_classification_mobilenet_v2_tf
- test_performance_classification_resnet_18_tf
- test_performance_classification_squeezenet

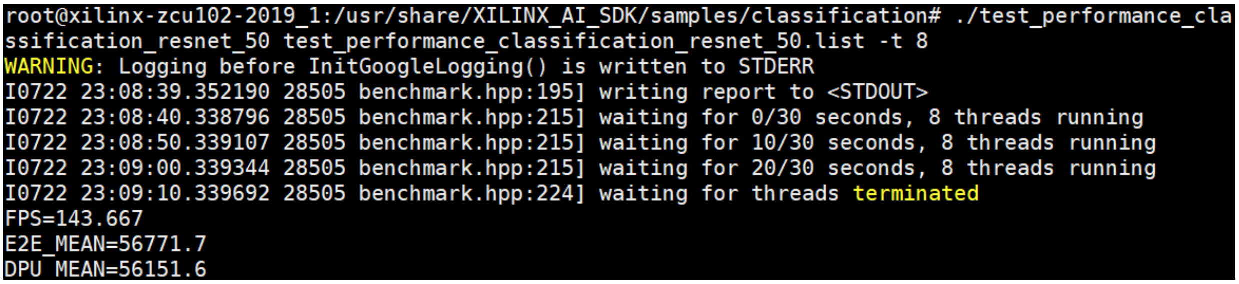
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the performance test example.

```
#./test_performance_classification_resnet_50
test_performance_classification_resnet_50.list -t 8
```

9. You will see the printing result on the terminal, see the following figure.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/classification# ./test_performance_cla
ssification_resnet_50 test_performance_classification_resnet_50.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:08:39.352190 28505 benchmark.hpp:195] writing report to <STDOUT>
I0722 23:08:40.338796 28505 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0722 23:08:50.339107 28505 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0722 23:09:00.339344 28505 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0722 23:09:10.339692 28505 benchmark.hpp:224] waiting for threads terminated
FPS=143.667
E2E_MEAN=56771.7
DPU_MEAN=56151.6
```

Figure 12: Classification Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples /classification` directory in the SDK.

Notes

Classification supports the following types of networks. Refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/include/xilinx/ai/classification.hpp` for more details.

```
/// Resnet50 neural network, input size is 224x224
static const char CLASSIFICATION_RESNET_50[] = "resnet_50";
/// Resnet18 neural network, input size is 224x224
static const char CLASSIFICATION_RESNET_18[] = "resnet_18";
/// Inception_v1 neural network, input size is 224x224
static const char CLASSIFICATION_INCEPTION_V1[] = "inception_v1";
/// Inception_v2 neural network, input size is 224x224
```

```
    static const char CLASSIFICATION_INCEPTION_V2[] = "inception_v2";
    /// Inception_v3 neural network, input size is 299x299
    static const char CLASSIFICATION_INCEPTION_V3[] = "inception_v3";
    /// Inception_v4 neural network, input size is 299x299
    static const char CLASSIFICATION_INCEPTION_V4[] = "inception_v4";
    /// Mobilenet_v2 neural network, input size is 224x224
    static const char CLASSIFICATION_MOBILENET_V2[] = "mobilenet_v2";
    /// Resnet50 Tensorflow model, input size is 224x224
    static const char CLASSIFICATION_RESNET_50_TF[] = "resnet_50_tf";
    /// Resnet18 Tensorflow model, input size is 224x224
    static const char CLASSIFICATION_RESNET_18_TF[] = "resnet_18_tf";
    /// Inception_v1 Tensorflow model, input size is 224x224
    static const char CLASSIFICATION_INCEPTION_V1_TF[] = "inception_v1_tf";
    /// Mobilenet_v1 Tensorflow model, input size is 224x224
    static const char CLASSIFICATION_MOBILENET_V1_TF[] = "mobilenet_v1_tf";
    /// Mobilenet_v2 Tensorflow model, input size is 224x224
    static const char CLASSIFICATION_MOBILENET_V2_TF[] = "mobilenet_v2_tf";
```

Face Detection

Introduction

This lib uses DenseBox neuron network to detect human face. Input is a picture with some faces you'd like to detect. Output is a vector of the result struct contain each box's information.

The following image show the result of face detection.

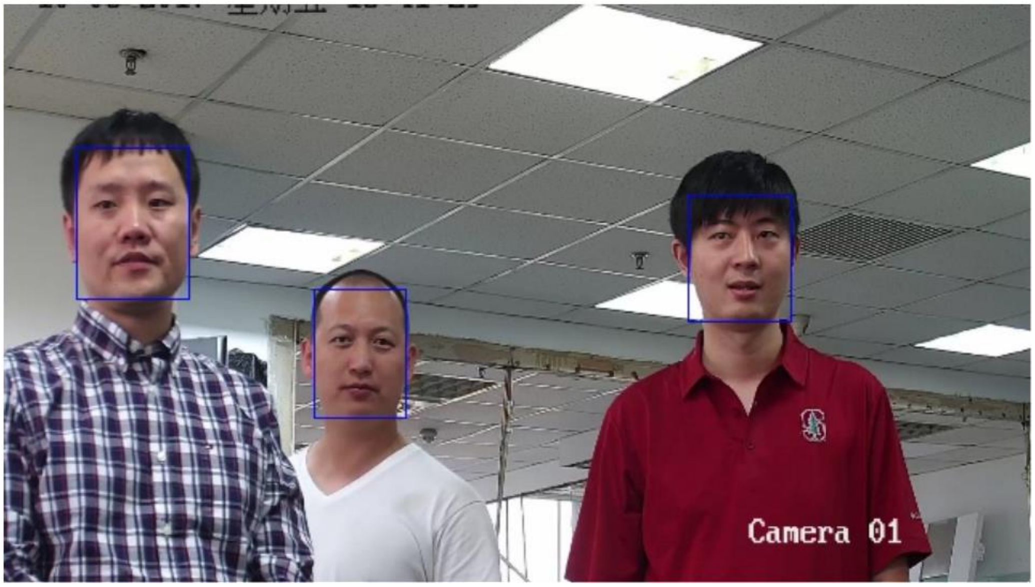


Figure 13: Face Detection Example

Sample

The Xilinx AI SDK provides face detection samples based on image or video. Also, the Xilinx AI SDK provides a sample performance test of the face detection.

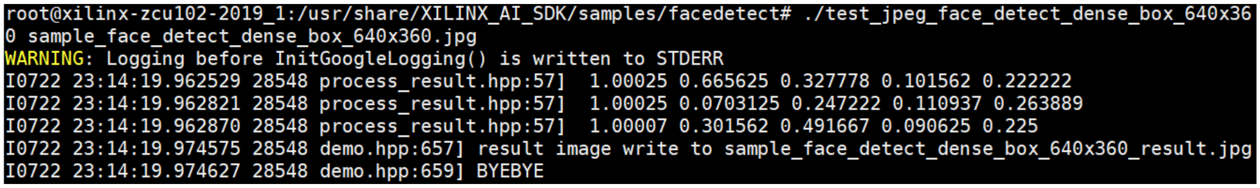
1. Before you run the face detection example, you can choose one of the following executable programs to run.
    - `test_jpeg_face_detect_dense_box_320x320`
    - `test_jpeg_face_detect_dense_box_640x360`
2. If the executable program does not exist, it can be compiled and generated as follows:

    ```
    #sh -x build.sh
    ```

3. Run the example.

    ```
    #./test_jpeg_face_detect_dense_box_640x360
    sample_face_detect_dense_box_640x360.jpg
    ```

4. You will see the print result on the terminal, see the following figure. Also, you can view the output image: sample_facedetect_dense_box_640x360_result.jpg, such as figure 13.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_jpeg_face_detect_dense_box_640x36
0 sample_face_detect_dense_box_640x360.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:14:19.962529 28548 process_result.hpp:57]  1.00025 0.665625 0.327778 0.101562 0.222222
I0722 23:14:19.962821 28548 process_result.hpp:57]  1.00025 0.0703125 0.247222 0.110937 0.263889
I0722 23:14:19.962870 28548 process_result.hpp:57]  1.00007 0.301562 0.491667 0.090625 0.225
I0722 23:14:19.974575 28548 demo.hpp:657] result image write to sample_face_detect_dense_box_640x360_result.jpg
I0722 23:14:19.974627 28548 demo.hpp:659] BYEBYE
```

Figure 14: Face Detection Running Result

5. To test the video data, execute the following command.

    ```
    #./test_video_face_detect_dense_box_640x360 video_input.mp4 -t 8
    ```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the face detection performance example, you can choose one of the following executable program to run.
    - `test_performance_face_detect_dense_box_320x320`
    - `test_performance_face_detect_dense_box_640x360`

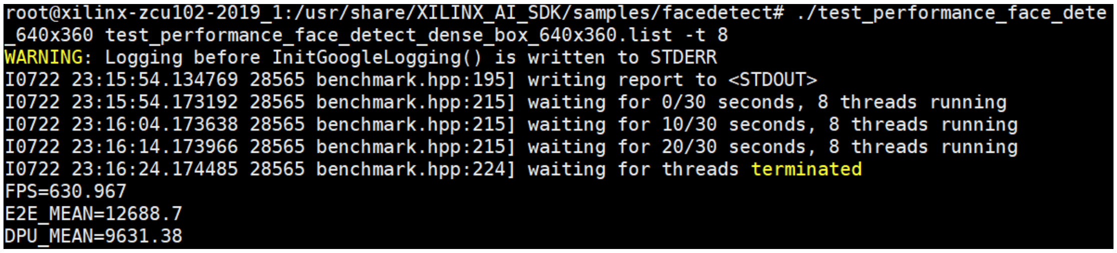
    If the executable program does not exist, it can be compiled and generated as follows:

    ```
    #sh -x build.sh
    ```

7. Run the example.

    ```
    #./test_performance_face_detect_dense_box_640x360
    test_performance_face_detect_dense_box_640x360.list -t 8
    ```

8. You will see the print result on the terminal, see the following figure.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_performance_face_dete
_640x360 test_performance_face_detect_dense_box_640x360.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:15:54.134769 28565 benchmark.hpp:195] writing report to <STDOUT>
I0722 23:15:54.173192 28565 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0722 23:16:04.173638 28565 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0722 23:16:14.173966 28565 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0722 23:16:24.174485 28565 benchmark.hpp:224] waiting for threads terminated
FPS=630.967
E2E_MEAN=12688.7
DPU_MEAN=9631.38
```

Figure 15: Face Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/facedetect` directory in the SDK.

Notes

Face detection supports the following two resolution as input. Refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/include/xilinx/ai/facedetect.hpp` for more details.

```
    /// Densebox for face-detection, input size is 320x320.
    static const char FACE_DETECT_DENSE_BOX_320x320[] = "dense_box_320x320";
    /// Densebox for face-detection, input size is 640x320.
    static const char FACE_DETECT_DENSE_BOX_640x360[] = "dense_box_640x360";
```

It's best to use a 16:9 image as input.

Face Landmark Detection

Introduction

Face landmark network is used to detect five key points of a face. The five points include the left eye, the right eye, nose, left lip of mouth, right lip of mouth. This network is used to correct face direction before face feature extraction. The input image should be a face which is detected by the face detection network. The outputs of the network are 5 key points. The 5 key points are normalized.

The following image show the result of face detection.

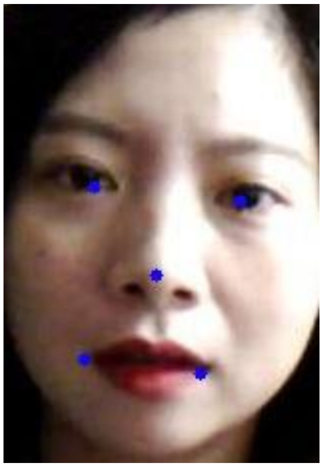


Figure 16: Face landmark Detection Example

Sample

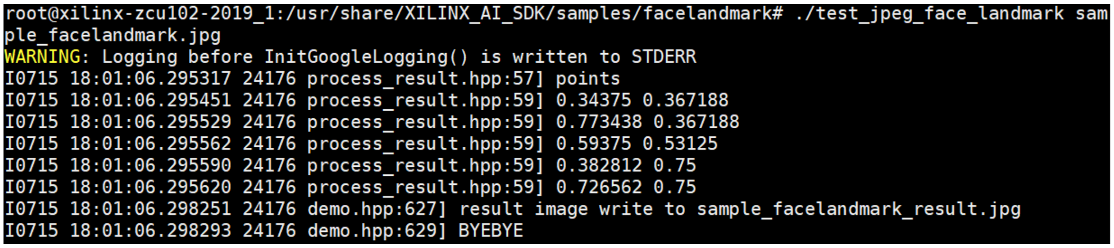
1. Before you run the face landmark detection example, make sure the following executable program exist.
   - `test_jpeg_face_landmark`
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the image example.

```
#./test_jpeg_face_landmark sample_facelandmark.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_facelandmark_result.jpg` (see Figure 16).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/facelandmark# ./test_jpeg_face_landmark sam
ple_facelandmark.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0715 18:01:06.295317 24176 process_result.hpp:57] points
I0715 18:01:06.295451 24176 process_result.hpp:59] 0.34375 0.367188
I0715 18:01:06.295529 24176 process_result.hpp:59] 0.773438 0.367188
I0715 18:01:06.295562 24176 process_result.hpp:59] 0.59375 0.53125
I0715 18:01:06.295590 24176 process_result.hpp:59] 0.382812 0.75
I0715 18:01:06.295620 24176 process_result.hpp:59] 0.726562 0.75
I0715 18:01:06.298251 24176 demo.hpp:627] result image write to sample_facelandmark_result.jpg
I0715 18:01:06.298293 24176 demo.hpp:629] BYEBYE
```

Figure 17: Face landmark Running result

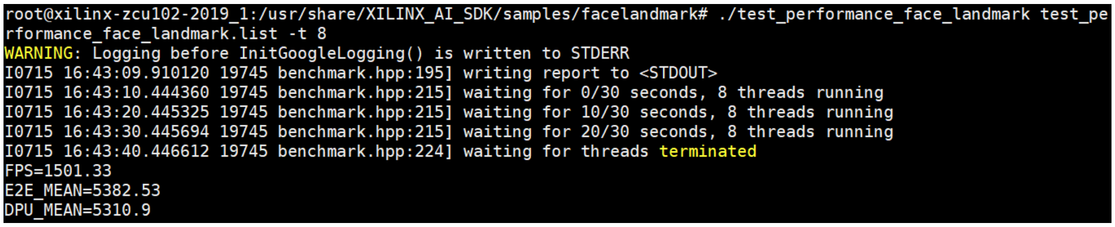
5. To test the video data, execute the following command.

```
#./test_video_face_landmark video_input.mp4 -t 8
```

6. To test performance, execute the following command.

```
#./test_performance_face_landmark test_performance_face_landmark.list -t 8
```

7. The following figure shows the performance testing result.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/facelandmark# ./test_performance_face_landmark test_pe
rformance_face_landmark.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0715 16:43:09.910120 19745 benchmark.hpp:195] writing report to <STDOUT>
I0715 16:43:10.444360 19745 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0715 16:43:20.445325 19745 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0715 16:43:30.445694 19745 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0715 16:43:40.446612 19745 benchmark.hpp:224] waiting for threads terminated
FPS=1501.33
E2E_MEAN=5382.53
DPU_MEAN=5310.9
```

Figure 18: Face landmark Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/facelandmark` directory in the SDK.

Notes

The face landmark model's name is shown below.

```
    /// face landmark model.
    static const char FACE_LANDMARK[] = "face_landmark";
```

SSD Detection

Introduction

This lib is in common use to SSD neuron network. SSD is a neural network which is used to detect objects. Input is a picture with some objects you'd like to detect. Output is a vector of the result struct contain each box's information.

The following image shows the result of SSD detection.

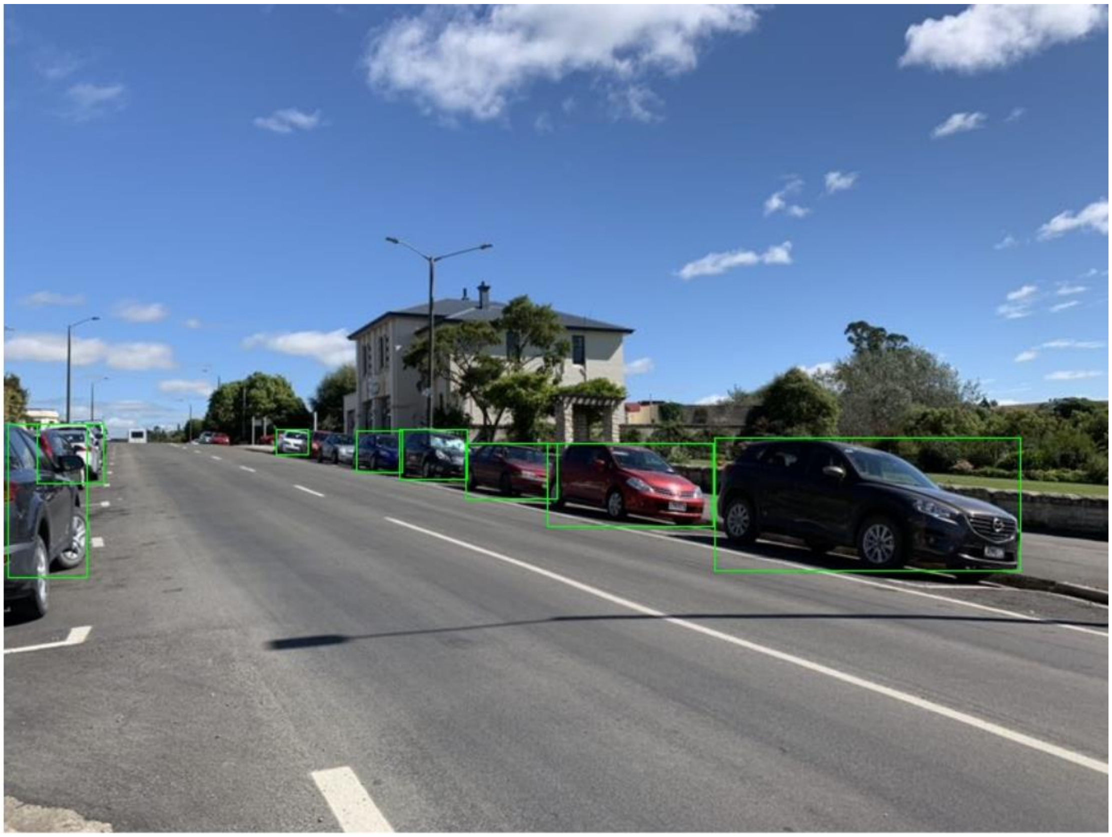


Figure 19: SSD Detection Example

Sample

The Xilinx AI SDK provides SSD detection samples based on image or video. Also, the Xilinx AI SDK provides a sample performance test of SSD detection.

1. Before you run the ssd detection example, you can choose one of the following executable programs to run:
    - `test_jpeg_ssd_adas_pedestrian_640x360`
    - `test_jpeg_ssd_adas_vehicle_v3_480x360`

- `test_jpeg_ssd_mobilenet_v2_480x360`
- `test_jpeg_ssd_traffic_480x360`
- `test_jpeg_ssd_voc_300x300_tf`

2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the example.

```
#./test_jpeg_ssd_adas_vehicle_v3_480x360
sample_ssd_adas_vehicle_v3_480x360.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_ssd_adas_vehicle_v3_480x360_result.jpg` (see Figure 19).

5. To test the video data, execute the following command.

```
#./test_video_ssd_adas_vehicle_v3_480x360 video_input.mp4 -t 8
```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the SSD detection performance example, you can choose one of the following executable program to run.

   - test_performance_ssd_adas_pedestrian_640x360
   - test_performance_ssd_adas_vehicle_v3_480x360
   - test_performance_ssd_mobilenet_v2_480x360
   - test_performance_ssd_traffic_480x360
   - test_performance_ssd_voc_300x300_tf

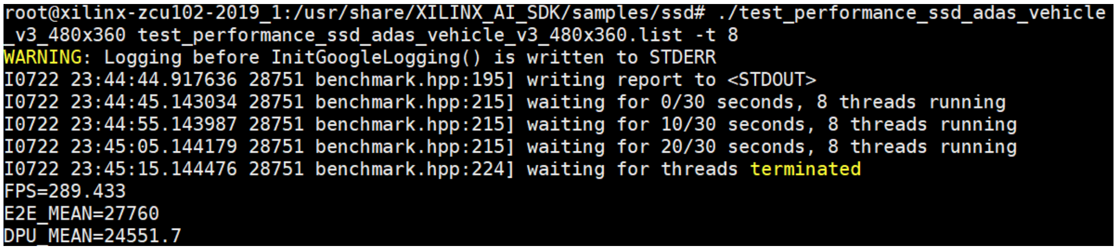
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_ssd_adas_vehicle_v3_480x360
test_performance_ssd_adas_vehicle_v3_480x360.list -t 8
```

9. You will see the printing result on the terminal, see the following figure.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/ssd# ./test_performance_ssd_adas_vehicle
_v3_480x360 test_performance_ssd_adas_vehicle_v3_480x360.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:44:44.917636 28751 benchmark.hpp:195] writing report to <STDOUT>
I0722 23:44:45.143034 28751 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0722 23:44:55.143987 28751 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0722 23:45:05.144179 28751 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0722 23:45:15.144476 28751 benchmark.hpp:224] waiting for threads terminated
FPS=289.433
E2E_MEAN=27760
DPU_MEAN=24551.7
```

Figure 20: SSD Detection Performance Test Result

For more details about the sample, refer to the `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk//samples/ssd` directory in the SDK.

Notes

SSD detection supports the following types of networks. Refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/include/xilinx/ai/ssd.hpp` for more details.

```
    /// Adas vehicle model, input size is 480x360.
    static const char SSD_ADAS_VEHICLE_V3_480x360[] = "ssd_vehicle_v3_480x360";
    /// Video structurization model, Ops is 11.6g. Input size is 480x360.
    static const char SSD_TRAFFIC_480x360[] = "ssd_traffic_480x360";
    /// Pedestrian detect model, OPs is 5.9g. Input size is 640x360, sight is adas.
    static const char SSD_ADAS_PEDESTRIAN_640x360[] = "ssd_pedestrian_640x360";
    /// Mobilenet_v2 ssd detection model, 11 classes.
    static const char SSD_MOBILENET_V2_480x360[] = "ssd_mobilenet_v2_480x360";
    /// Tensorflow ssd model, dataset is VOC.
    static const char SSD_VOC_300x300_TF[] = "ssd_voc_300x300_tf";
```

Pose Detection

Introduction

This library is used to detect posture of the human body. This library includes a neural network which can marking 14 key points of human body (You can use our SSD detection library). Input is a picture which is detected by pedestrian detection neural network. Output is a struct contain each point coordinates.

The following image shows the result of pose detection.

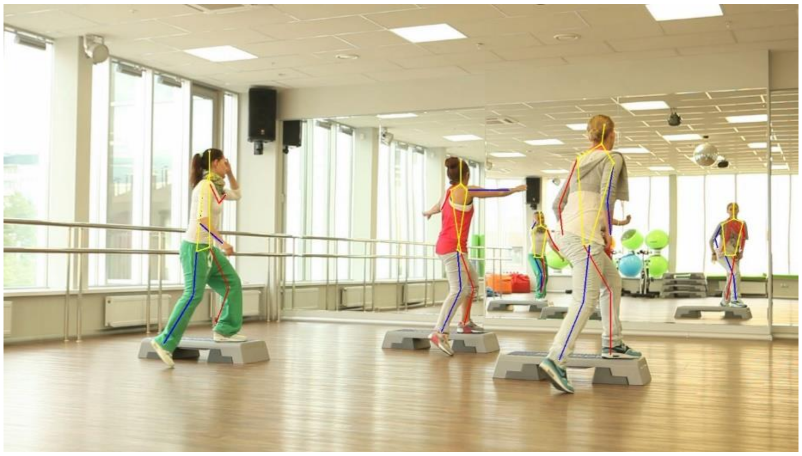


Figure 21: Pose Detection Example

Sample

The Xilinx AI SDK provides pose detection samples based on image or video. Also, the Xilinx AI SDK provides a sample performance test of pose detection.

1. Before you run the pose detection example, you can choose one of the following executable programs to run:

   - test_jpeg_pose_detect_with_ssd
   - test_jpeg_pose_detect

2. If the executable program does not exist, it can be compiled and generated as follows:

   ```
   #sh -x build.sh
   ```

3. Run the example.

   ```
   #./test_jpeg_pose_detect_with_ssd sample_pose_detect_with_ssd.jpg
   ```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_pose_detect_with_ssd_result.jpg`, see figure 21.

5. To test the video data, execute the following command.

   ```
   #./test_jpeg_pose_detect_with_ssd video_input.mp4 -t 8
   ```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the pose detection performance example, you can choose one of the following executable program to run.

   - `test_performance_pose_detect_with_ssd`
   - `test_performance_pose_detect`

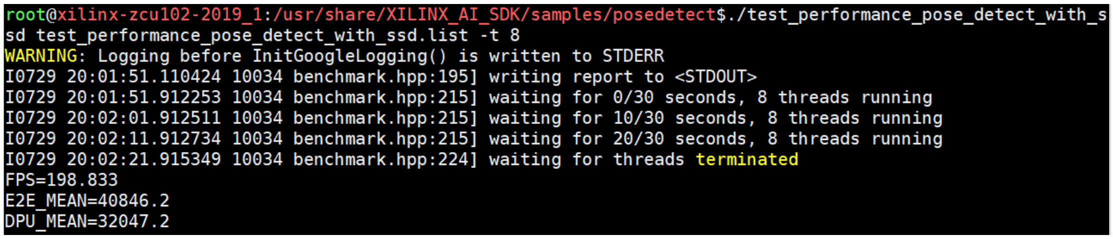
7. If the executable program does not exist, it can be compiled and generated as follows:

   ```
   #sh -x build.sh
   ```

8. Run the example.

   ```
   #./test_performance_pose_detect_with_ssd test_performance_pose_detect.list -
   t 8
   ```

9. You will see the print result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/posedetect$./test_performance_pose_detect_with_s
sd test_performance_pose_detect_with_ssd.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0729 20:01:51.110424 10034 benchmark.hpp:195] writing report to <STDOUT>
I0729 20:01:51.912253 10034 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0729 20:02:01.912511 10034 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0729 20:02:11.912734 10034 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0729 20:02:21.915349 10034 benchmark.hpp:224] waiting for threads terminated
FPS=198.833
E2E_MEAN=40846.2
DPU_MEAN=32047.2
```

Figure 22: Pose Detection Performance Test Result

For more details about the sample, refer to the `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/posedetect` directory in the SDK.

Notes

If the input image is arbitrary and the user does not know the exact location of the person, we must perform the SSD detection first. Refer to the `test_jpeg_pose_detect_with_ssd.cpp`.

If the input picture is the picture of the person who has been cut out, we can only do the pose detection. Refer to the `test_jpeg_pose_detect.cpp`.

Semantic Segmentation

Introduction

The semantic segmentation of image is to assign a semantic category to each pixel in the input image, so as to obtain the pixelated intensive classification. Libsegmentation is a segmentation lib which can be used in ADAS field. It offers simple interfaces for developer to deploy segmentation task on Xilinx FPGA.

The following is an example of semantic segmentation, where the goal is to predict class labels for each pixel in the image.

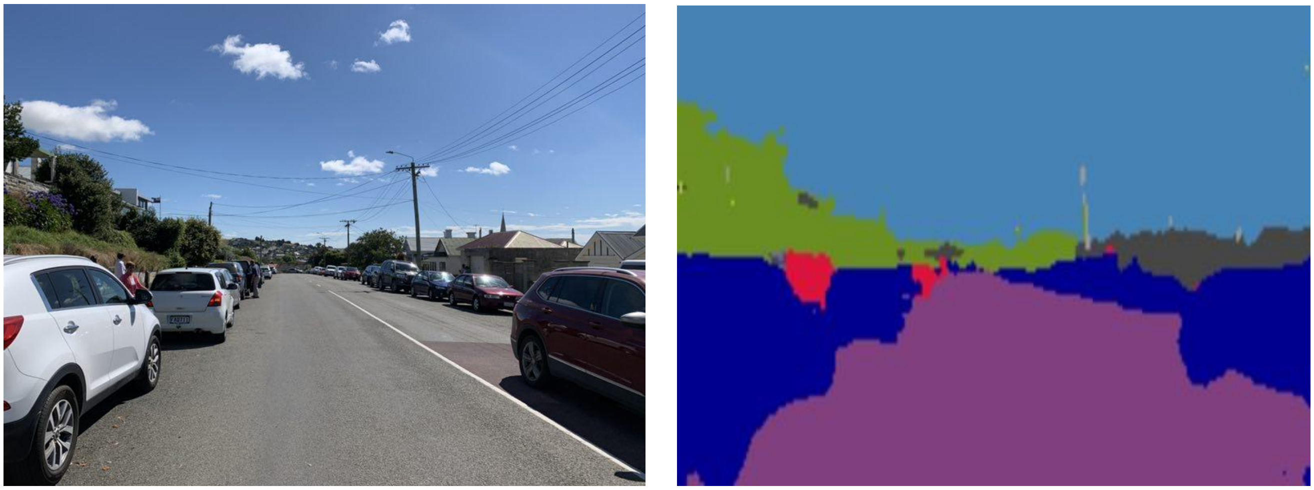


Figure 23: Semantic Segmentation Example

Sample

The Xilinx AI SDK provides semantic segmentation samples based on image or video. Also, this product provides a sample performance test of semantic segmentation.

1. Before you run the semantic segmentation example, you can choose one of the following executable programs to run.
   - test_jpeg_segmentation_fpn_8UC3
   - test_jpeg_segmentation_fpn_8UC1

2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the example.

```
#./test_jpeg_segmentation_fpn_8UC3 sample_segmentation_fpn_8UC3.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_segmentation_fpn_8UC3_result.jpg` (see Figure 23).
5. To test the video data, execute the following command.

```
#./test_video_segmentation_fpn_8UC3 video_input.mp4 -t 8
```

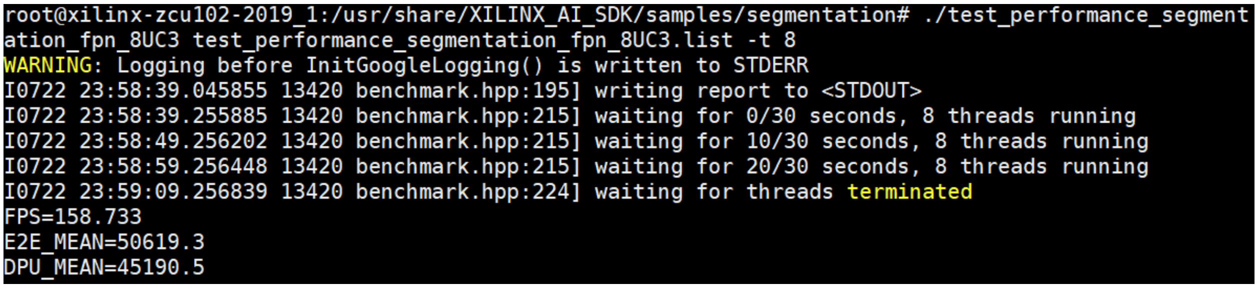
6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the semantic segmentation performance example, you can choose one of the following executable programs to run.
   - `test_performance_segmentation_fpn_8UC3`
   - `test_performance_segmentation_fpn_8UC1`
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_segmentation_fpn_8UC3
test_performance_segmentation_fpn_8UC3.list -t 8
```

9. You will see the print result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/segmentation# ./test_performance_segment
ation_fpn_8UC3 test_performance_segmentation_fpn_8UC3.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0722 23:58:39.045855 13420 benchmark.hpp:195] writing report to <STDOUT>
I0722 23:58:39.255885 13420 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0722 23:58:49.256202 13420 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0722 23:58:59.256448 13420 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0722 23:59:09.256839 13420 benchmark.hpp:224] waiting for threads terminated
FPS=158.733
E2E_MEAN=50619.3
DPU_MEAN=45190.5
```

Figure 24: Semantic Segmentation Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/segmentation` directory in the SDK.

Notes

The difference between `test_jpeg_segmentation_fpn_8UC3` and `test_jpeg_segmentation_fpn_8UC1` is the output of result. The `test_jpeg_segmentation_fpn_8UC3` output is color rendering display, while `test_jpeg_segmentation_fpn_8UC1` output is grayscale image.

Road Line Detection

Introduction

The library is used to draw lane lines in the adas library and each lane line is represented by a number type representing the category and a vector<Point> used to draw the lane line. In the test code, color map is used.

Different types of lane lines are represented by different colors. The point is stored in the container vector, and the polygon interface `cv::polylines()` of OpenCv is used to draw the lane line.

The following image show the result of road line detection.

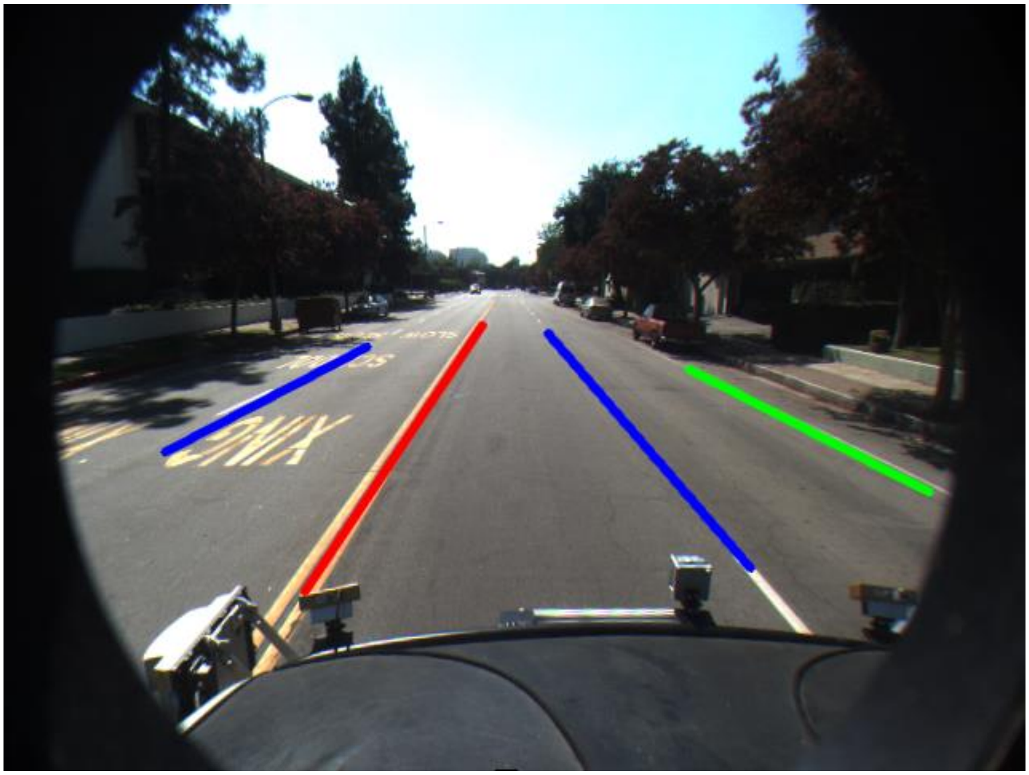


Figure 25: Road Line Detection Example

Sample

The Xilinx AI SDK provides road line detection samples based on image or video. A sample performance test of road line detection is also provided.

1. Before you run the road line detection example, make sure the following executable program exists.
    - `test_jpeg_road_line_vpg`
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the example.

```
#./test_jpeg_road_line_vpg sample_road_line.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_road_line_result.jpg` (see Figure 25).

5. To test the video data, execute the following command.

```
#./test_video_road_line_vpg video_input.mp4 -t 8
```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the road line detection performance example, make sure the following executable program exists.

    - *test_performance_road_line_vpg*

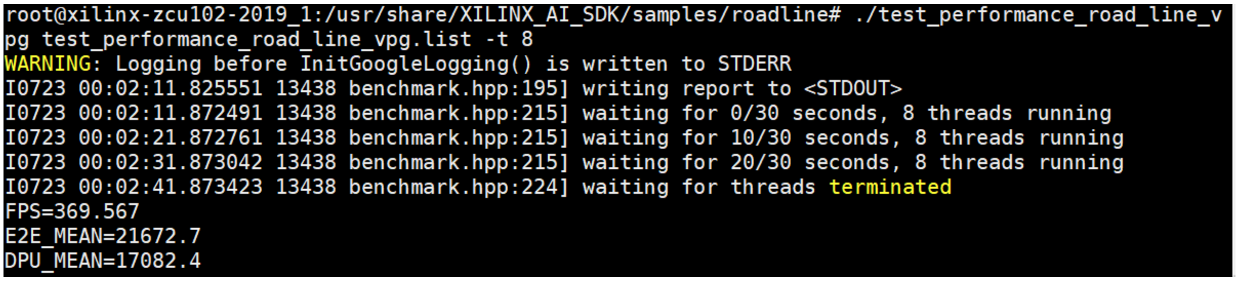
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_road_line_vpg test_performance_road_line_vpg.list -t 8
```

9. You will see the print result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/roadline# ./test_performance_road_line_v
pg test_performance_road_line_vpg.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0723 00:02:11.825551 13438 benchmark.hpp:195] writing report to <STDOUT>
I0723 00:02:11.872491 13438 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0723 00:02:21.872761 13438 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0723 00:02:31.873042 13438 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0723 00:02:41.873423 13438 benchmark.hpp:224] waiting for threads terminated
FPS=369.567
E2E_MEAN=21672.7
DPU_MEAN=17082.4
```

Figure 26: Road Line Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/roadline` directory in the SDK.

Notes

The input of the image is fixed at 640x480 and images of other sizes need to be resized.

YOLOV3 Detection

Introduction

This lib is in common use to YOLO neuron network.

YOLO is a neural network which is used to detect objects. Now its version is v3.

Input is a picture with one or more objects. Output is a vector of the result struct which is composed of the detected information.

The following image shows the result of YOLOv3 detection:

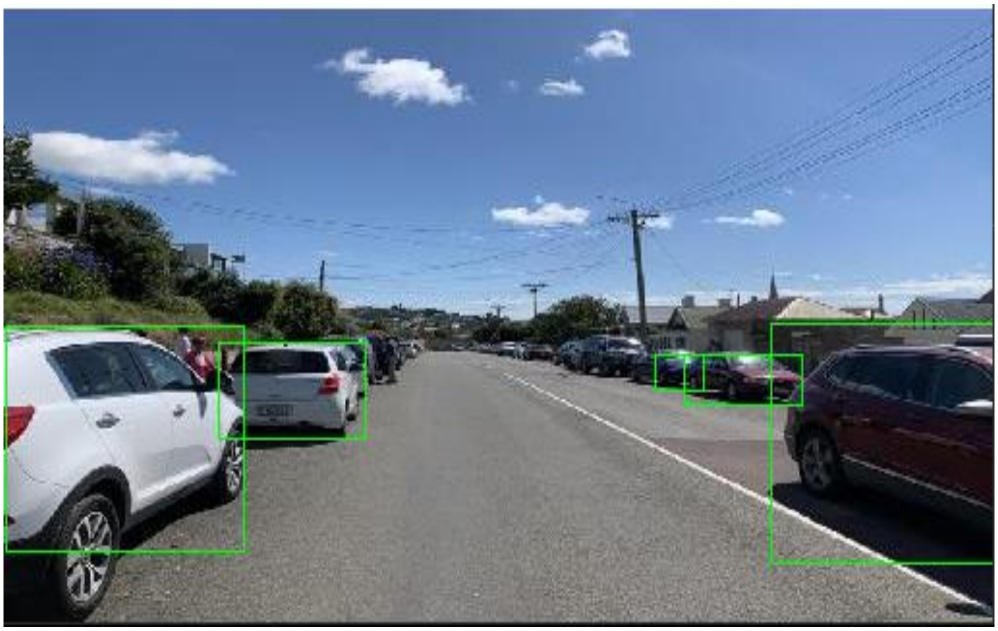


Figure 27: YOLOv3 Detection Example

Sample

The Xilinx AI SDK provides YOLOV3 detection samples based on image or video. A sample performance test of YOLOV3 detection is also provided.

1. Before you run the YOLOV3 detection example, you can choose one of the following executable programs to run:
   - `test_jpeg_yolov3_adas_512x256`
   - `test_jpeg_yolov3_adas_512x288`
   - `test_jpeg_yolov3_voc_416x416`
   - `test_jpeg_yolov3_voc_416x416_tf`
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the example.

```
#./test_jpeg_yolov3_adas_512x256 sample_yolov3_adas_512x256.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_yolov3_result.jpg` (see Figure 27).
5. To test the video data, execute the following command.

```
#./test_video_yolov3_adas_512x256 video_input.mp4 -t 8
```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the YOLOV3 detection performance example, you can choose one of the following executable programs to run.

    - `test_performance_yolov3_adas_512x256`
    - `test_performance_yolov3_adas_512x288`
    - `test_performance_yolov3_voc_416x416`
    - `test_performance_yolov3_voc_416x416_tf`

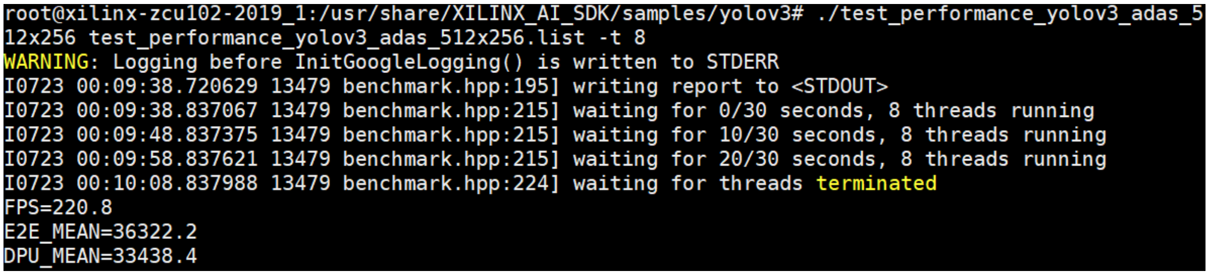
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_yolov3_adas_512x256
test_performance_yolov3_adas_512x256.list -t 8
```

9. You will see the printing result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/yolov3# ./test_performance_yolov3_adas_5
12x256 test_performance_yolov3_adas_512x256.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0723 00:09:38.720629 13479 benchmark.hpp:195] writing report to <STDOUT>
I0723 00:09:38.837067 13479 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0723 00:09:48.837375 13479 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0723 00:09:58.837621 13479 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0723 00:10:08.837988 13479 benchmark.hpp:224] waiting for threads terminated
FPS=220.8
E2E_MEAN=36322.2
DPU_MEAN=33438.4
```

Figure 28: YOLOV3 Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/yolov3` directory in the SDK.

Notes

YOLOV3 supports the following types of models. Refer to `/usr/include/xilinx/yolov3/yolov3.hpp` for more details.

```
/// VOC detect model,20 classes, input size is 416x416.
static const char YOLOV3_VOC_416x416[] = "yolov3_voc_416";
/// ADAS vehicle detect model, 3 classes, input size is 512x256.
static const char YOLOV3_ADAS_512x256[] = "yolov3_adas_512x256";
/// ADAS_vehicle detect model, 10 classes, no pruned.
static const char YOLOV3_ADAS_512x288[] = "yolov3_adas_512x288";
/// ADAS_vehicle detect model, tiny yolo.
static const char YOLOV3_ADAS_TINY_416x416[] = "tiny_yolov3_416x416";
///  VOC detect model, train by tensorflow, input size is 416x416
```

```
static const char YOLOV3_VOC_416x416_TF[] = "yolov3_voc_416x416_tf";
```

YOLOV2 Detection

Introduction

YOLOV2 does the same thing as YOLOV3, which is an upgraded version of YOLOV2.

Sample

1. Before you run the YOLOV2 detection example, you can choose one of the following executable programs to run:
   - `test_jpeg_yolov2_voc_baseline`
   - `test_jpeg_yolov2_voc_compress22g`
   - `test_jpeg_yolov2_voc_compress24g`
   - `test_jpeg_yolov2_voc_compress26g`
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the image example.

```
#./test_jpeg_yolov2_voc_baseline sample_yolov2_voc.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image: `sample_yolov2_voc_result.jpg` (see the following figure).

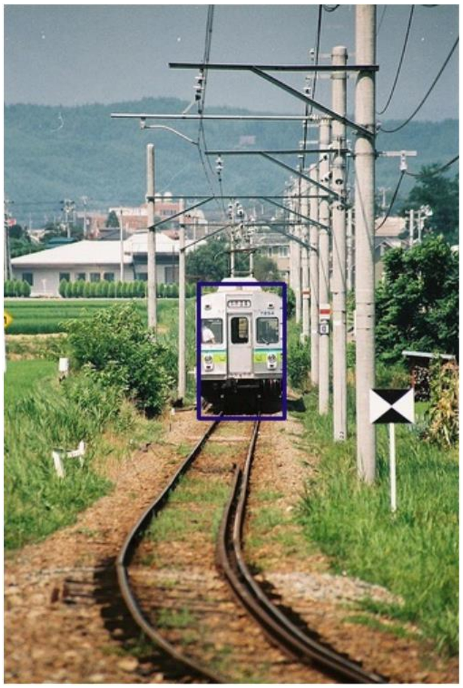


Figure 29: YOLOv2 Detection Example

5. To test the video data, execute the following command.

```
#./test_video_yolov2_voc_baseline video_input.mp4 -t 8
```

6. To run the performance example, you can choose one of the following executable programs to run.

   - `test_performance_yolov2_voc_baseline`
   - `test_performance_yolov2_voc_compress22g`
   - `test_performance_yolov2_voc_compress24g`
   - `test_performance_yolov2_voc_compress26g`

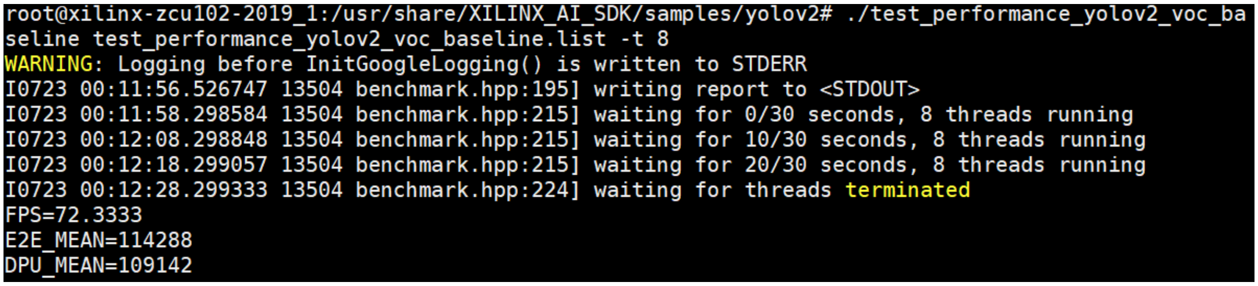
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the performance example.

```
#./test_performance_yolov2_voc_baseline
test_performance_yolov2_voc_baseline.list -t 8
```

9. You will see the printing result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/yolov2# ./test_performance_yolov2_voc_ba
seline test_performance_yolov2_voc_baseline.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0723 00:11:56.526747 13504 benchmark.hpp:195] writing report to <STDOUT>
I0723 00:11:58.298584 13504 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0723 00:12:08.298848 13504 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0723 00:12:18.299057 13504 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0723 00:12:28.299333 13504 benchmark.hpp:224] waiting for threads terminated
FPS=72.3333
E2E_MEAN=114288
DPU_MEAN=109142
```

Figure 30: YOLOV2 Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/yolov2` directory in the SDK.

Notes

YOLOV2 supports the following types of models. Refer to `/usr/include/xilinx/yolov2/yolov2.hpp` for more details.

```
/// VOC bashline model
static const char YOLOV2_VOC_BASELINE[] = "yolov2_voc_baseline";
/// VOC compress model, Ops is 22g.
static const char YOLOV2_VOC_COMPRESS22G[] = "yolov2_voc_compress22G";
/// VOC compress model, Ops is 24g.
static const char YOLOV2_VOC_COMPRESS24G[] = "yolov2_voc_compress24G";
/// VOC compress model, Ops is 26g.
static const char YOLOV2_VOC_COMPRESS26G[] = "yolov2_voc_compress26G";
```

Openpose Detection

Introduction

The library is used to draw the person's posture. It is represented by the line between the point and the point, with points stored as pairs. Every pair represents a connection and the result is the set of pairs, stored as vectors.

The following image show the result of openpose detection.

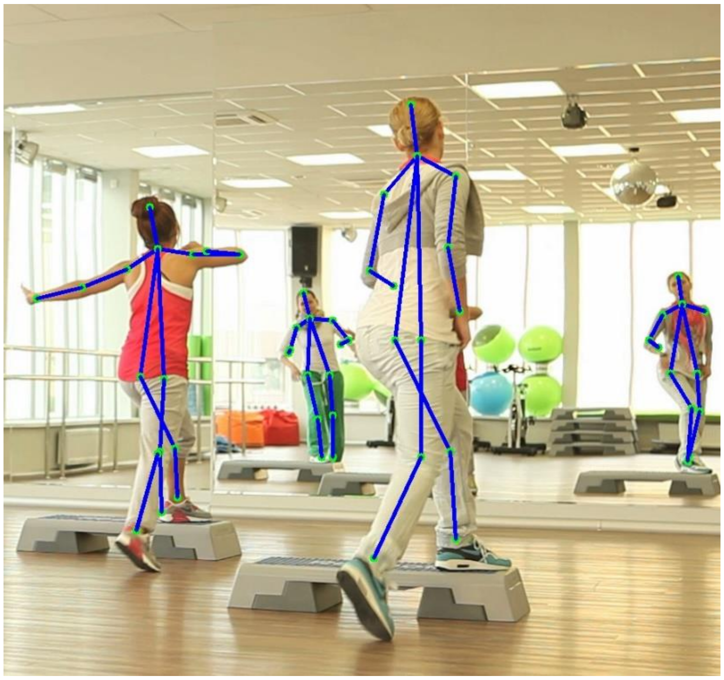


Figure 31: Openpose Detection Example

Sample

The Xilinx AI SDK provides openpose detection samples based on image or video. A sample performance test of openpose detection is also provided.

1. Before you run the openpose detection example, make sure the following executable program exists.

    - *test_jpeg_open_pose_368x368*

2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. Run the example.

```
#./test_jpeg_open_pose_368x368 sample_openpose.jpg
```

4. You will see the print result on the terminal. Also, you can view the output image:

`sample_openpose_result.jpg` (see Figure 31).

5. To test the video data, execute the following command.

```
#./test_video_open_pose_368x368 video_input.mp4 -t 8
```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the Openpose detection performance example, make sure the following executable program exists:

    - *test_performance_open_pose_368x368*

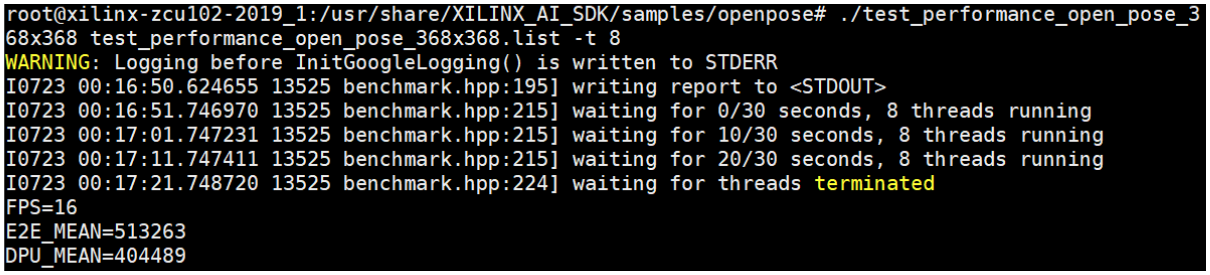
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_open_pose_368x368
test_performance_open_pose_368x368.list -t 8
```

9. You will see the print result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/openpose# ./test_performance_open_pose_3
68x368 test_performance_open_pose_368x368.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0723 00:16:50.624655 13525 benchmark.hpp:195] writing report to <STDOUT>
I0723 00:16:51.746970 13525 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0723 00:17:01.747231 13525 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0723 00:17:11.747411 13525 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0723 00:17:21.748720 13525 benchmark.hpp:224] waiting for threads terminated
FPS=16
E2E_MEAN=513263
DPU_MEAN=404489
```

Figure 32: Openpose Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/openpose` directory in the SDK.

Notes

The openpose models' name are shown below.

```
/ // openpose model, input size is 368x368.
static const char OPEN_POSE_368x368[] = "openpose_368x368";
```

RefineDet Detection

Introduction

This lib is in common use to RefineDet neuron network. RefineDet is a neural network which is used to detect human bodies. Input is a picture with some individuals you'd like to detect. Output is a vector of the result struct contain each box's information.

The following image shows the result of RefineDet detection:

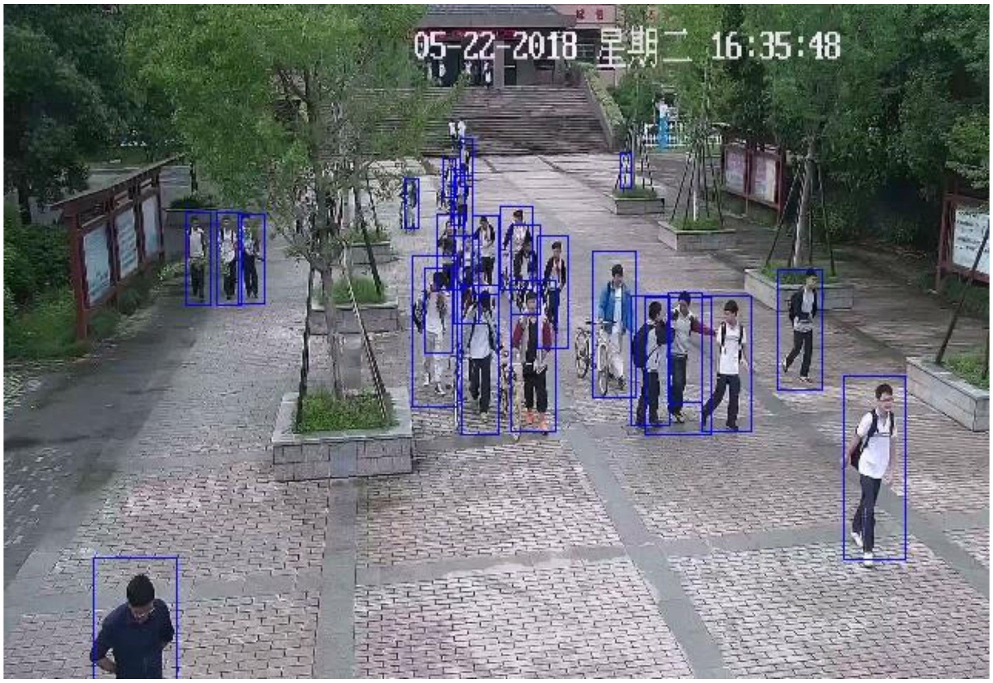


Figure 33: RefineDet Detection Example

Sample

The Xilinx AI SDK provides RefineDet detection samples based on images or videos. A sample performance test of RefineDet detection is provided.

1. Before you run the RefineDet detection example, you can choose one of the following executable programs to run:

   - `test_jpeg_refine_detect_480x360`
   - `test_jpeg_refine_detect_480x360_5g`
   - `test_jpeg_refine_detect_480x360_10g`

2. If the executable program does not exist, it can be compiled and generated as follows:

   ```
   #sh -x build.sh
   ```

3. Run the example.

   ```
   #./test_jpeg_refine_detect_480x360 sample_refine_detect_480x360.jpg
   ```

4. You will see the printing result on the terminal. Also, you can view the output image: `sample_refine_detect_480x360_result.jpg` (see Figure 33).

5. To test the video data, execute the following command.

   ```
   #./test_video_refinedet_refinedet_480x360 video_input.mp4 -t 8
   ```

6. To test the model performance, we prepared a set of images for model performance testing which are stored in the `./images` directory. Before you run the RefineDet detection performance example, you can choose one of the following executable programs to run.

- `test_performance_refine_detect_480x360`
- `test_performance_refine_detect_480x360_5g`
- `test_performance_refine_detect_480x360_10g`

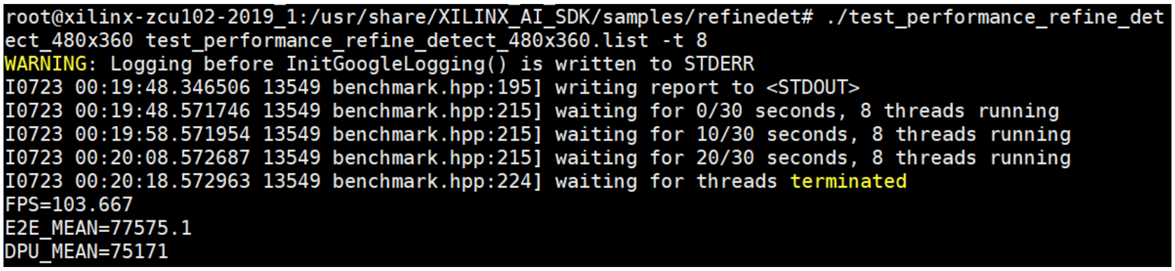
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

8. Run the example.

```
#./test_performance_refine_detect_480x360
test_performance_refine_detect_480x360.list -t 8
```

9. You will see the print result on the terminal (see the following figure).



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/refinedet# ./test_performance_refine_det
ect_480x360 test_performance_refine_detect_480x360.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0723 00:19:48.346506 13549 benchmark.hpp:195] writing report to <STDOUT>
I0723 00:19:48.571746 13549 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0723 00:19:58.571954 13549 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0723 00:20:08.572687 13549 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0723 00:20:18.572963 13549 benchmark.hpp:224] waiting for threads terminated
FPS=103.667
E2E_MEAN=77575.1
DPU_MEAN=75171
```

Figure 34: RefineDet Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/refinedet` directory in the SDK.

Notes

The RefineDet models' name are shown below.

```
//// RefineDet model, input size is 480x360, base model
static const char REFINE_DETECT_480x360[] = "refinedet_480x360";
/// RefineDet model, input size is 480x360,OPS is 10g
static const char REFINE_DETECT_480x360_10G[] = "refinedet_480x360_10G";
/// RefineDet model, input size is 480x360,OPS is 5g
static const char REFINE_DETECT_480x360_5G[] = "refinedet_480x360_5G";
```

ReID Detection

Introduction

The task of person re-identification is to identify a person of interest at another time or place. This is done by extracting the image feature and comparing the features. Images of same identity ought to have similar features and get small feature distance, while images of different identities have large feature distance. With a queried image and a pile of candidate images given, the image which has the smallest distance would be identified as the same person as the queried image.

Sample

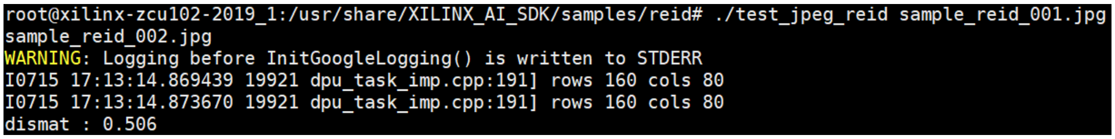
1. Before you run the ReID detection example, make sure the following executable program exist.
   - `test_jpeg_reid`
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. To test the image data, execute the following command.

```
#./test_jpeg_reid sample_reid_001.jpg sample_reid_002.jpg
```

4. You will see the print result on the terminal.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/reid# ./test_jpeg_reid sample_reid_001.jpg
sample_reid_002.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0715 17:13:14.869439 19921 dpu_task_imp.cpp:191] rows 160 cols 80
I0715 17:13:14.873670 19921 dpu_task_imp.cpp:191] rows 160 cols 80
dismat : 0.506
```

Figure 35: ReID Detection Example

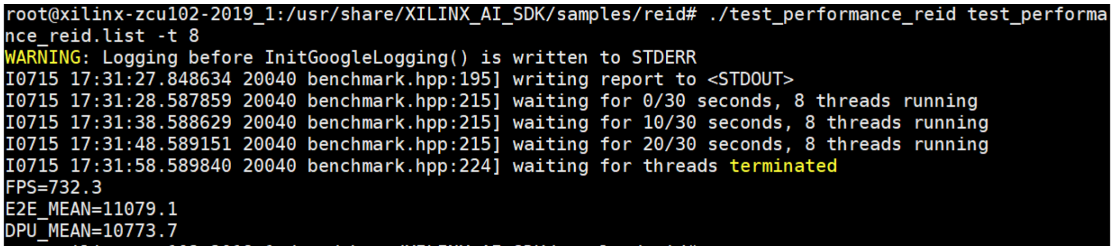
5. To test the video data, execute the following command.

```
#./test_video_reid video_input.mp4 -t 8
```

6. To test performance, execute the following command.

```
#./test_performance_reid test_performance_reid.list -t 8
```

7. The following figure shows the performance testing result.



```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/reid# ./test_performance_reid test_performa
nce_reid.list -t 8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0715 17:31:27.848634 20040 benchmark.hpp:195] writing report to <STDOUT>
I0715 17:31:28.587859 20040 benchmark.hpp:215] waiting for 0/30 seconds, 8 threads running
I0715 17:31:38.588629 20040 benchmark.hpp:215] waiting for 10/30 seconds, 8 threads running
I0715 17:31:48.589151 20040 benchmark.hpp:215] waiting for 20/30 seconds, 8 threads running
I0715 17:31:58.589840 20040 benchmark.hpp:224] waiting for threads terminated
FPS=732.3
E2E_MEAN=11079.1
DPU_MEAN=10773.7
```

Figure 36: ReID Detection Performance Test Result

For more details about the sample, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/samples/reid` directory in the SDK.

Notes

The ReID model's name is shown below.

```
/// reid model, include bn.
static const char REID[] = "reid";
```

In practice, users have many different application requirements, but it basically falls into three categories. The first is to use the ready-made models provided by Xilinx AI SDK to quickly build their own applications, and the second is to use users' own custom models which are similar to the models in the SDK and the last is to use new models that are totally different from the models in the SDK. This chapter describes the detailed development steps for the first two cases. For the third case, users can also use the SDK's samples and libraries implementation for reference. Therefore, this chapter describes the following contents:

- How to customize pre-processing
- How to use the configuration file as pre-processing and post-processing parameter
- How to use the SDK post-processing library
- How to implement user post-processing code

Developing with Xilinx Model

The Development Processes

1. Install the XILINX AI SDK on the host side, refer to Chapter 2.

   After the installation, the models can be found in the `~/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib` directory.

2. Enter the platform directory, then copy the `xilinx_ai_model_ZCU102_2019.1-r1.deb` to the board via scp.

3. Install the Xilinx Model Package on the target side.

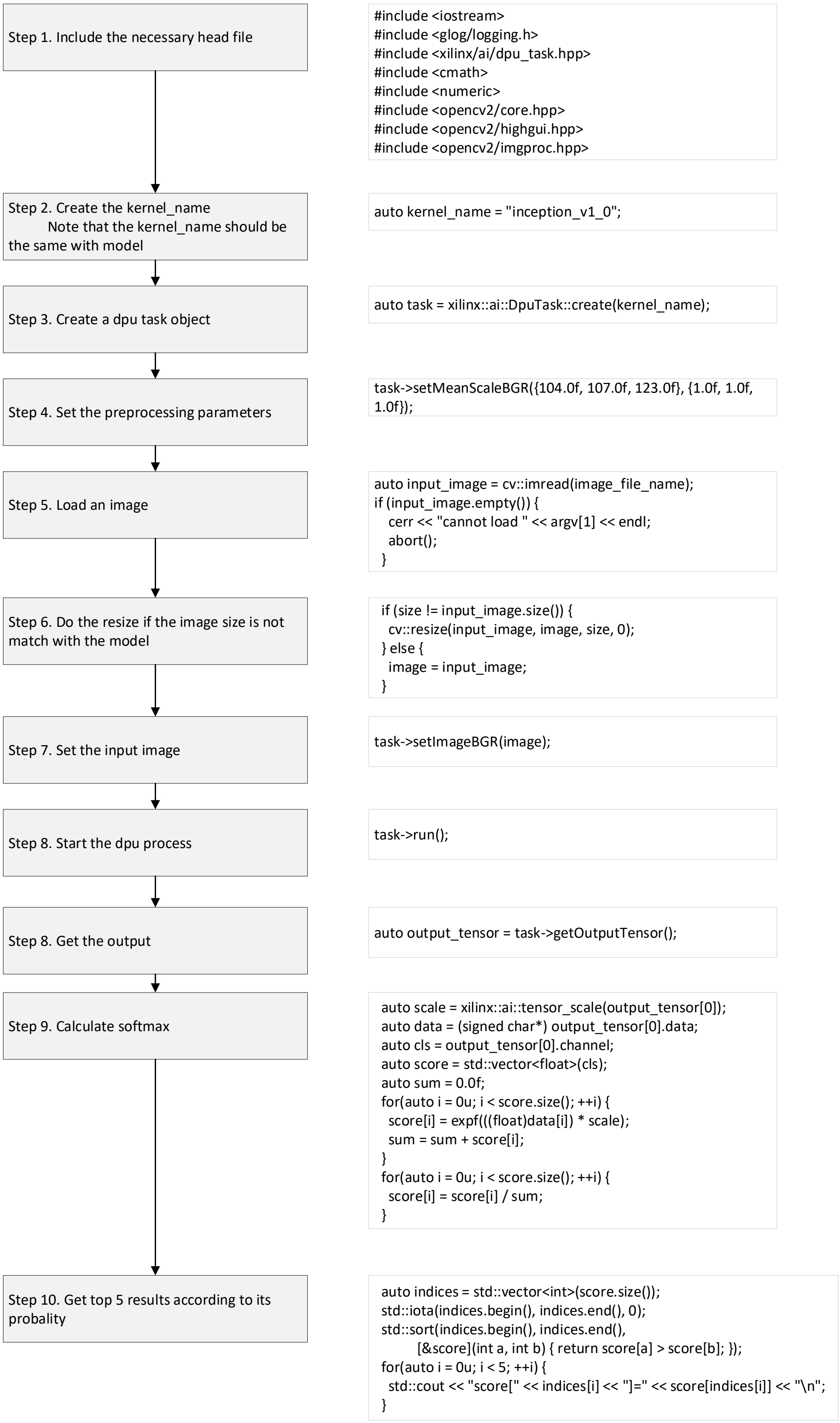
   ```
   #dpkg -i xilinx_ai_model_ZCU102_2019.1-r1.deb
   ```

   After the installation, the models can be found in the `/usr/lib` directory on the target side.

4. Create a folder under your workspace, using classification as an example.

   ```
   $mkdir classification
   ```

5. Create the classification1.cpp source file. The main flow is shown below. See `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/classification/classification1.cpp` for a complete code example.



Step 1. Include the necessary head file

```
#include <iostream>
#include <glog/logging.h>
#include <xilinx/ai/dpu_task.hpp>
#include <cmath>
#include <numeric>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
```

Step 2. Create the kernel_name
Note that the kernel_name should be the same with model

```
auto kernel_name = "inception_v1_0";
```

Step 3. Create a dpu task object

```
auto task = xilinx::ai::DpuTask::create(kernel_name);
```

Step 4. Set the preprocessing parameters

```
task->setMeanScaleBGR({104.0f, 107.0f, 123.0f}, {1.0f, 1.0f,
1.0f});
```

Step 5. Load an image

```
auto input_image = cv::imread(image_file_name);
if (input_image.empty()) {
   cerr << "cannot load " << argv[1] << endl;
   abort();
 }
```

Step 6. Do the resize if the image size is not match with the model

```
 if (size != input_image.size()) {
   cv::resize(input_image, image, size, 0);
 } else {
   image = input_image;
 }
```

Step 7. Set the input image

```
task->setImageBGR(image);
```

Step 8. Start the dpu process

```
task->run();
```

Step 8. Get the output

```
auto output_tensor = task->getOutputTensor();
```

Step 9. Calculate softmax

```
 auto scale = xilinx::ai::tensor_scale(output_tensor[0]);
 auto data = (signed char*) output_tensor[0].data;
 auto cls = output_tensor[0].channel;
 auto score = std::vector<float>(cls);
 auto sum = 0.0f;
 for(auto i = 0u; i < score.size(); ++i) {
   score[i] = expf(((float)data[i]) * scale);
   sum = sum + score[i];
 }
 for(auto i = 0u; i < score.size(); ++i) {
   score[i] = score[i] / sum;
 }
```

Step 10. Get top 5 results according to its probality

```
 auto indices = std::vector<int>(score.size());
 std::iota(indices.begin(), indices.end(), 0);
 std::sort(indices.begin(), indices.end(),
          [&score](int a, int b) { return score[a] > score[b]; });
 for(auto i = 0u; i < 5; ++i) {
   std::cout << "score[" << indices[i] << "]=" << score[indices[i]] << "\n";
 }
```

Figure 37: Main Program Flow Chart and Corresponding Code

6. Create a build.sh file as shown below, or copy one from the SDK demo and modify it.

   ```
   #/bin/sh
   CXX=${CXX:-g++}
   $CXX -std=c++11 -O3 -I. -o classification1 classification1.cpp -lopencv_core -
   lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -
   lopencv_highgui -lglog -ldpbase -ldpproto
   ```

7. Set up the cross-compile chain tool environment.

   ```
   $ . ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
   ```

8. Compile the program.

   ```
   $sh -x build.sh
   ```

9. Copy the executable program to the target board via scp

   ```
   $scp classification1 root@IP_OF_BOARD:~/
   ```

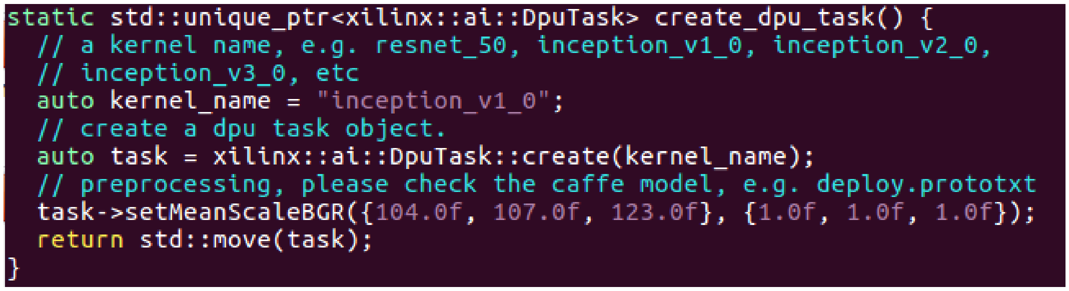
10. Execute the program on the target board. Before running the program, make sure the target board has the SDK installed, and prepare the images you want to test.

    ```
    #./classification1 input_image.jpg
    ```

Notes:

- There are two classification examples provided in the SDK. There are stored under `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/classification/`

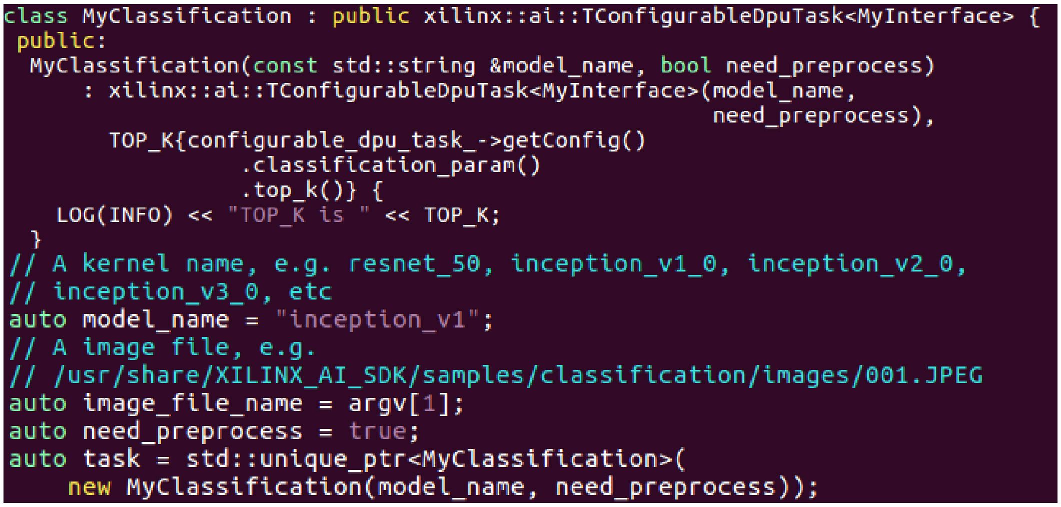
- Classification1.cpp uses user-defined pre-processing parameter as input, as shown in the following figure:



```
static std::unique_ptr<xilinx::ai::DpuTask> create_dpu_task() {
  // a kernel name, e.g. resnet_50, inception_v1_0, inception_v2_0,
  // inception_v3_0, etc
  auto kernel_name = "inception_v1_0";
  // create a dpu task object.
  auto task = xilinx::ai::DpuTask::create(kernel_name);
  // preprocessing, please check the caffe model, e.g. deploy.prototxt
  task->setMeanScaleBGR({104.0f, 107.0f, 123.0f}, {1.0f, 1.0f, 1.0f});
  return std::move(task);
}
```

Figure 38: Pre-Processing Parameters Set Code

- `Classification2.cpp` uses SDK config file as pre-process param, as show in the following figure.



```
class MyClassification : public xilinx::ai::TConfigurableDpuTask<MyInterface> {
 public:
  MyClassification(const std::string &model_name, bool need_preprocess)
      : xilinx::ai::TConfigurableDpuTask<MyInterface>(model_name,
                                                       need_preprocess),
        TOP_K{configurable_dpu_task_->getConfig()
                  .classification_param()
                  .top_k()} {
    LOG(INFO) << "TOP_K is " << TOP_K;
  }
// A kernel name, e.g. resnet_50, inception_v1_0, inception_v2_0,
// inception_v3_0, etc
auto model_name = "inception_v1";
// A image file, e.g.
// /usr/share/XILINX_AI_SDK/samples/classification/images/001.JPEG
auto image_file_name = argv[1];
auto need_preprocess = true;
auto task = std::unique_ptr<MyClassification>(
    new MyClassification(model_name, need_preprocess));
```

Figure 39: Pre-Processing Parameters Set Using SDK Configuration File

- Both of Classification examples use user post-processing code.
- If you want to use the SDK post-processing library, please check "How to Use the SDK Post-Processing Library-XNNPP" section.

For more details about the example, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo`.

Developing with User Model

When users use their own models, it is important to note that the user's model framework should be within the scope supported by the XILINX AI SDK. The following is an introduction of how to deploy a retrained SSD Caffe model to ZCU102 platform based on XILINX AI SDK step by step.

The development processes

1. Download the corresponding DNNDK package from https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge and extract it. Then, install it and refer to the document of ug1327-xilinx-dnndk-user-guide.pdf.

2. Create a folder and place the float model under it on the host side, then do the quantization.

   Following is the decent.sh script.

   ```
   $mkdir ssd_adas
   $cp float.caffe.model float.prototxt ssd_adas
   $cd ssd_adas
   $./decent.sh
   ```

   Following is the decent.sh script.

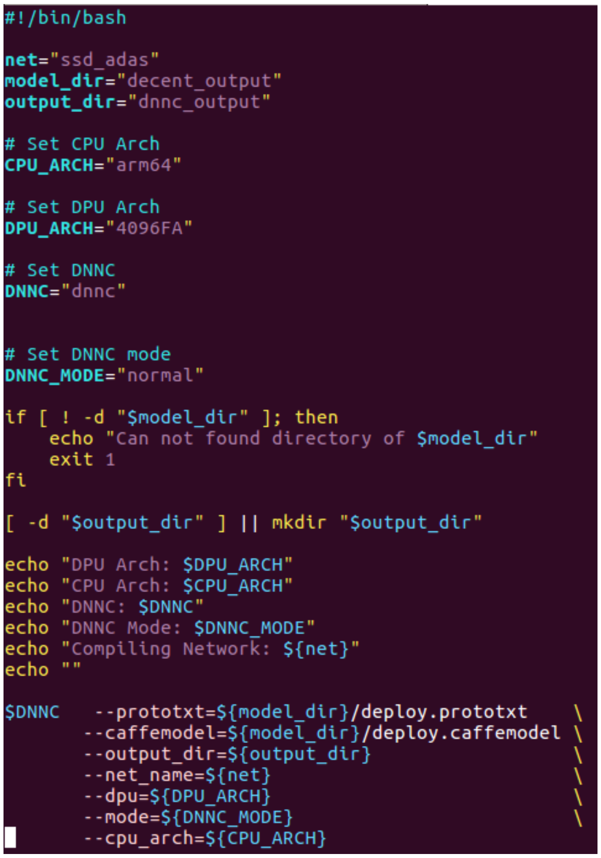
   ```
   #!/usr/bin/env bash

   #working directory
   work_dir=$(pwd)
   ```

```
    #path of float model
    model_dir=${work_dir}
    #output directory
    output_dir=${work_dir}/decent_output

    if [ -f /usr/local/bin/decent ]; then
        DECENT="decent"
    elif [ -f /usr/local/bin/decent-cpu ]; then
        DECENT="decent-cpu"
    else
        echo "Error: Please run DNNDK host_x86/install.sh first to install
decent"
        exit 1
    fi

    [ -d "$output_dir" ] || mkdir "$output_dir"

    $DECENT     quantize                                \
                -model ${model_dir}/float.prototxt      \
                -weights ${model_dir}/float.caffemodel \
                -output_dir ${output_dir}               \
                -method 1
```

3. Do the model compiling by the dnnc tool to get the elf file.

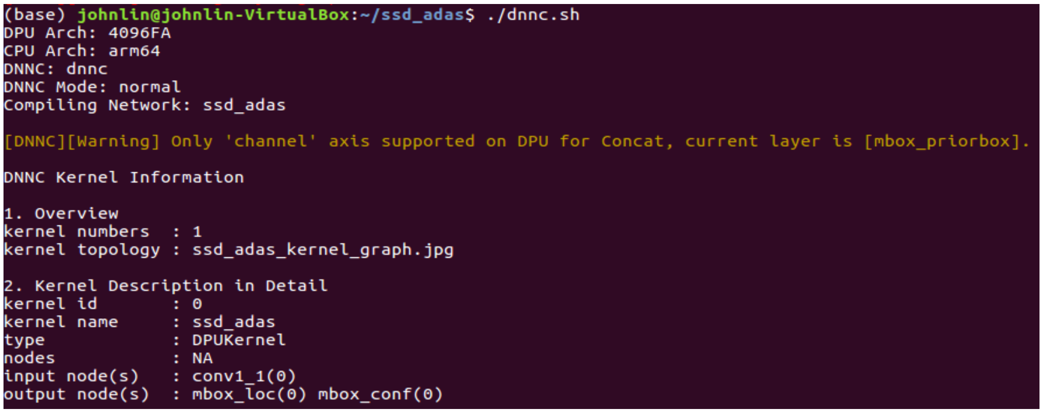
   ```
   $./dnnc.sh
   ```

   The following figure is the `dnnc.sh` script and Figure 41 is the running result of `dnnc.sh`



```
#!/bin/bash

net="ssd_adas"
model_dir="decent_output"
output_dir="dnnc_output"

# Set CPU Arch
CPU_ARCH="arm64"

# Set DPU Arch
DPU_ARCH="4096FA"

# Set DNNC
DNNC="dnnc"


# Set DNNC mode
DNNC_MODE="normal"

if [ ! -d "$model_dir" ]; then
    echo "Can not found directory of $model_dir"
    exit 1
fi

[ -d "$output_dir" ] || mkdir "$output_dir"

echo "DPU Arch: $DPU_ARCH"
echo "CPU Arch: $CPU_ARCH"
echo "DNNC: $DNNC"
echo "DNNC Mode: $DNNC_MODE"
echo "Compiling Network: ${net}"
echo ""

$DNNC    --prototxt=${model_dir}/deploy.prototxt     \
        --caffemodel=${model_dir}/deploy.caffemodel \
        --output_dir=${output_dir}                  \
        --net_name=${net}                           \
        --dpu=${DPU_ARCH}                           \
        --mode=${DNNC_MODE}                         \
        --cpu_arch=${CPU_ARCH}
```

Figure 40: dnnc.sh Script



```
(base) johnlin@johnlin-VirtualBox:~/ssd_adas$ ./dnnc.sh
DPU Arch: 4096FA
CPU Arch: arm64
DNNC: dnnc
DNNC Mode: normal
Compiling Network: ssd_adas

[DNNC][Warning] Only 'channel' axis supported on DPU for Concat, current layer is [mbox_priorbox].

DNNC Kernel Information

1. Overview
kernel numbers  : 1
kernel topology : ssd_adas_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : ssd_adas
type            : DPUKernel
nodes           : NA
input node(s)   : conv1_1(0)
output node(s)  : mbox_loc(0) mbox_conf(0)
```

Figure 41: dnnc.sh Script Run Result

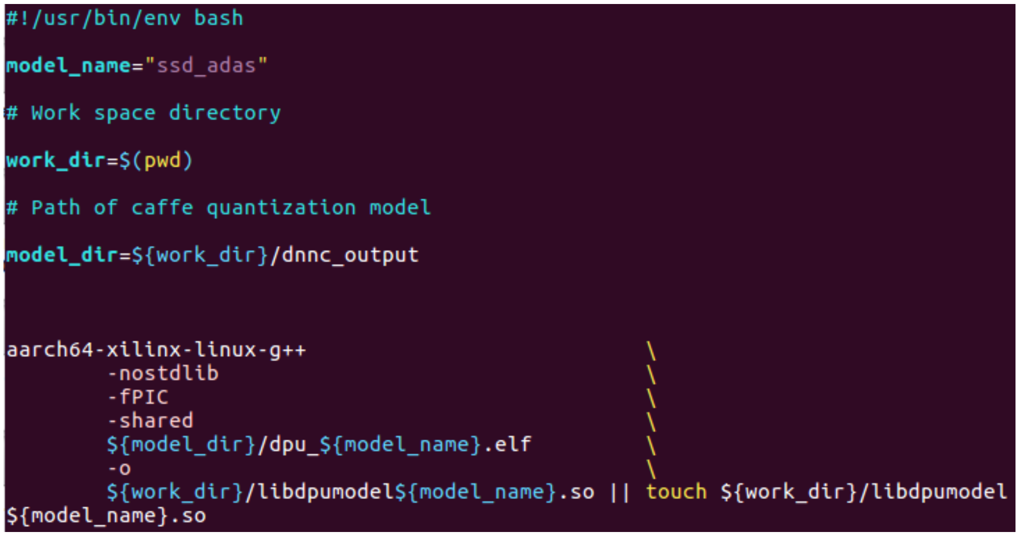
4. Set up the cross-compile chain tool environment.

   ```
   $ . ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
   ```

5. Convert the model ".elf" file to ".so" lib file. After the "`elf2so.sh`" script is executed, the `libdpumodelssd_adas.so` is generate.

   ```
   $./elf2so.sh
   ```

   Following figure is the elf2so.sh script.



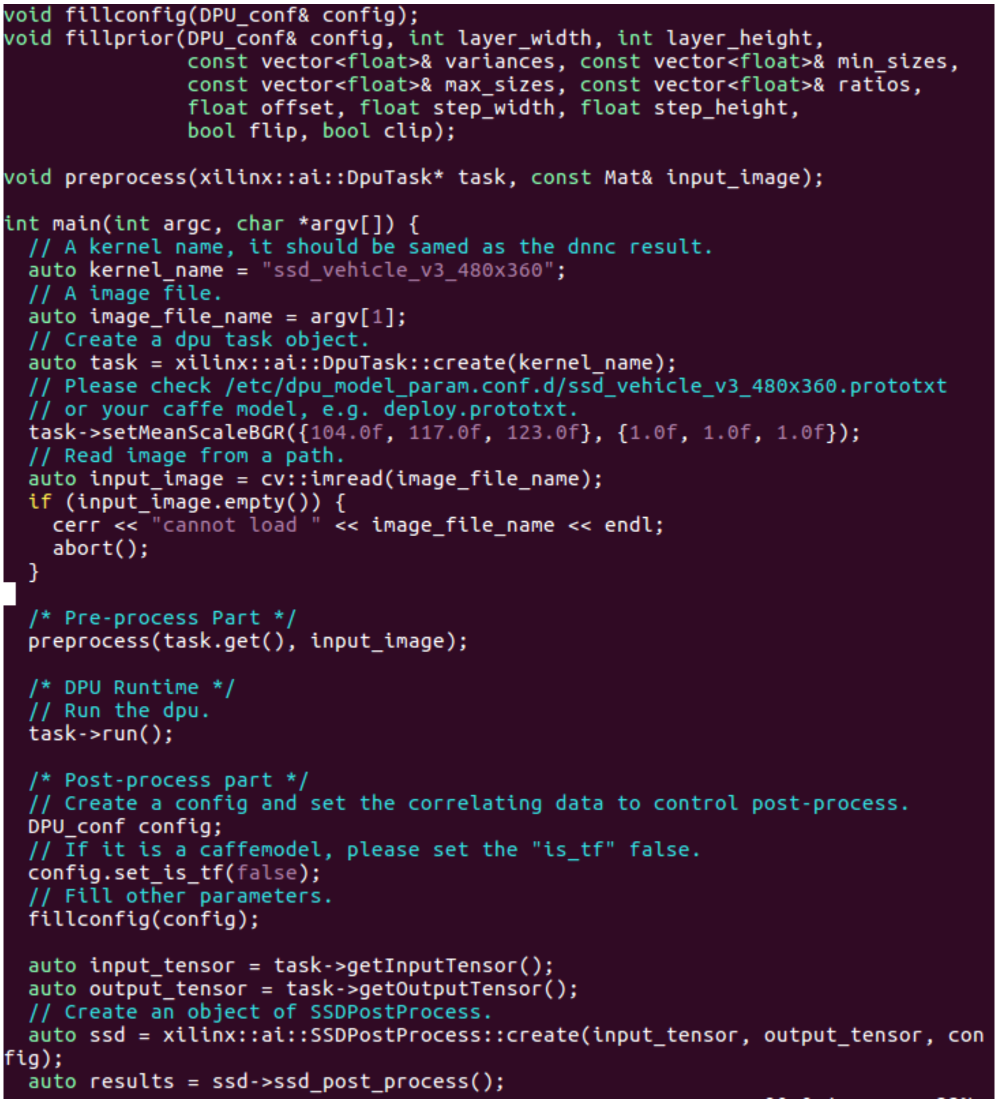
```
#!/usr/bin/env bash

model_name="ssd_adas"

# Work space directory

work_dir=$(pwd)

# Path of caffe quantization model

model_dir=${work_dir}/dnnc_output


aarch64-xilinx-linux-g++                                \
        -nostdlib                                       \
        -fPIC                                           \
        -shared                                         \
        ${model_dir}/dpu_${model_name}.elf              \
        -o                                              \
        ${work_dir}/libdpumodel${model_name}.so || touch ${work_dir}/libdpumodel
${model_name}.so
```

Figure 42: els2so.sh Script

Notes:

- `${DIR}` is the path which stores the elf
- `${MODEL_NAME}` is the elf name
- Such shared libraries should have a prefix named “libdpumodel” and the postfix “.so”.

6. Create the ssd_adas.cpp. Following is the part of ssd_adas.cpp. See `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/ssd/ssd.cpp` for reference.



```
void fillconfig(DPU_conf& config);
void fillprior(DPU_conf& config, int layer_width, int layer_height,
               const vector<float>& variances, const vector<float>& min_sizes,
               const vector<float>& max_sizes, const vector<float>& ratios,
               float offset, float step_width, float step_height,
               bool flip, bool clip);

void preprocess(xilinx::ai::DpuTask* task, const Mat& input_image);

int main(int argc, char *argv[]) {
  // A kernel name, it should be samed as the dnnc result.
  auto kernel_name = "ssd_vehicle_v3_480x360";
  // A image file.
  auto image_file_name = argv[1];
  // Create a dpu task object.
  auto task = xilinx::ai::DpuTask::create(kernel_name);
  // Please check /etc/dpu_model_param.conf.d/ssd_vehicle_v3_480x360.prototxt
  // or your caffe model, e.g. deploy.prototxt.
  task->setMeanScaleBGR({104.0f, 117.0f, 123.0f}, {1.0f, 1.0f, 1.0f});
  // Read image from a path.
  auto input_image = cv::imread(image_file_name);
  if (input_image.empty()) {
    cerr << "cannot load " << image_file_name << endl;
    abort();
  }

  /* Pre-process Part */
  preprocess(task.get(), input_image);

  /* DPU Runtime */
  // Run the dpu.
  task->run();

  /* Post-process part */
  // Create a config and set the correlating data to control post-process.
  DPU_conf config;
  // If it is a caffemodel, please set the "is_tf" false.
  config.set_is_tf(false);
  // Fill other parameters.
  fillconfig(config);

  auto input_tensor = task->getInputTensor();
  auto output_tensor = task->getOutputTensor();
  // Create an object of SSDPostProcess.
  auto ssd = xilinx::ai::SSDPostProcess::create(input_tensor, output_tensor, con
fig);
  auto results = ssd->ssd_post_process();
```

Figure 43: Part of ssd.cpp Code for Reference

7. Create a build.sh file as shown below, or copy one from the SDK demo and modify it.

```
#/bin/sh

CXX=${CXX:-g++}

$CXX -std=c++11 -O3 -I. -o ssd_adas ssd_adas.cpp -lopencv_core -lopencv_video
-lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui -lglog
-lxnnpp -ldpproto -lprotobuf -ldpbase
```

8. Compile the program, generate executable file `ssd_adas`.

```
$sh -x build.sh
```

9. Copy the `libdpumodelssd_adas.so` to the target and put it under `/usr/lib`.

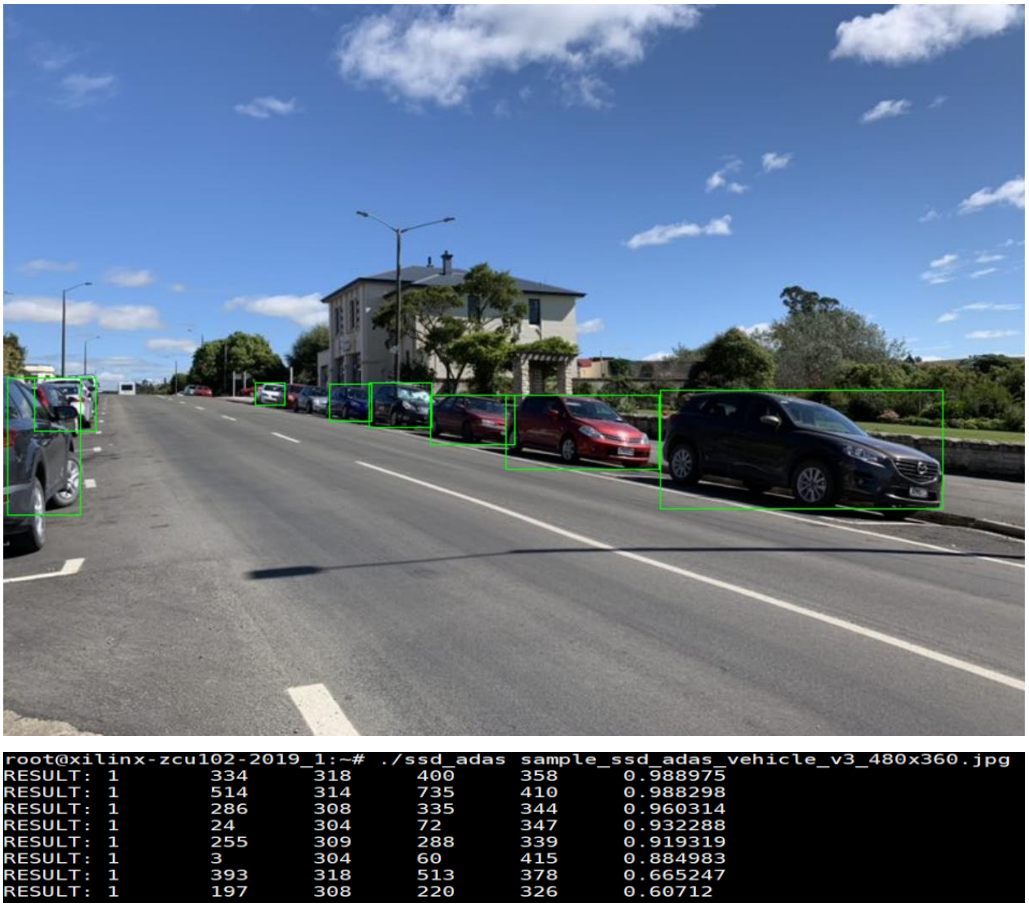
```
$scp libdpumodelssd_adas.so  root@IP_OF_BOARD:/usr/lib/
```

10. Copy the executable program to the target board via scp.

```
$scp ssd_adas root@IP_OF_BOARD:~/
```

11. Execute the program on the target board and get the following results. Before running the program, make sure the target board has the SDK installed, and prepare the images you want to test.

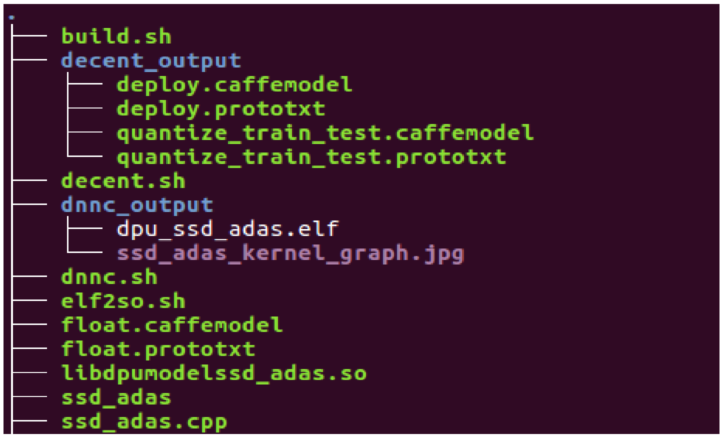
```
#./ssd_adas sample_ssd_adas_vehicle_v3_480x360.jpg
```



```
root@xilinx-zcu102-2019_1:~# ./ssd_adas sample_ssd_adas_vehicle_v3_480x360.jpg
RESULT: 1        334     318     400     358     0.988975
RESULT: 1        514     314     735     410     0.988298
RESULT: 1        286     308     335     344     0.960314
RESULT: 1        24      304     72      347     0.932288
RESULT: 1        255     309     288     339     0.919319
RESULT: 1        3       304     60      415     0.884983
RESULT: 1        393     318     513     378     0.665247
RESULT: 1        197     308     220     326     0.60712
```

Figure 44: ssd_adas Example Running Result

The following figure is the directory structure of the ssd_adas project for reference.



```
.
├── build.sh
├── decent_output
│   ├── deploy.caffemodel
│   ├── deploy.prototxt
│   ├── quantize_train_test.caffemodel
│   └── quantize_train_test.prototxt
├── decent.sh
├── dnnc_output
│   ├── dpu_ssd_adas.elf
│   └── ssd_adas_kernel_graph.jpg
├── dnnc.sh
├── elf2so.sh
├── float.caffemodel
├── float.prototxt
├── libdpumodelssd_adas.so
├── ssd_adas
├── ssd_adas.cpp
```

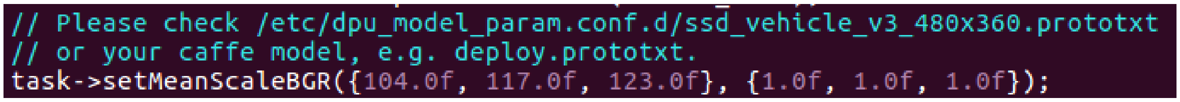
Figure 45: ssd_adas Project Directory Structure

How to Customize Pre-Processing

Before convolutional neural network processing, image data generally needs to be preprocessed. The basics of some preprocessing techniques that can be applied to any kind of data are as follows:

- Mean subtraction
- Normalization
- PCA and Whitening

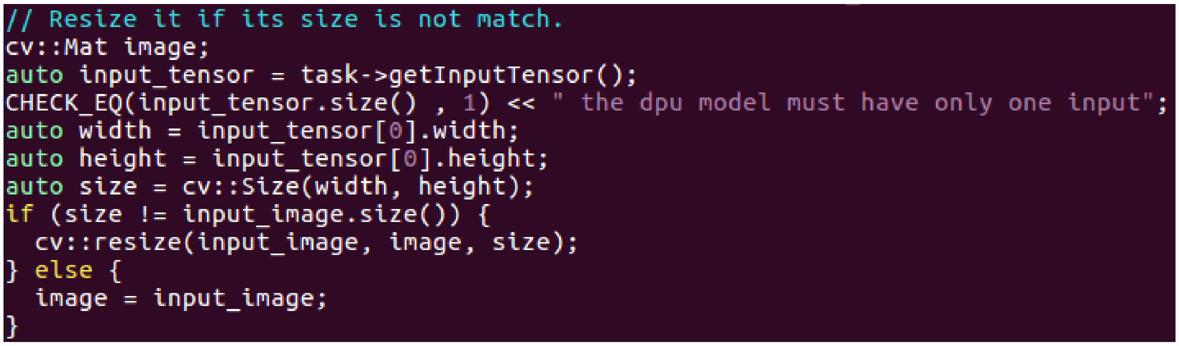
User calls the `setMeanScaleBGR` function to implement the Mean subtraction and normalization, as shown in the figure below. See `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/include/Xilinx/ai/dpu_task.hpp` for details.



```
// Please check /etc/dpu_model_param.conf.d/ssd_vehicle_v3_480x360.prototxt
// or your caffe model, e.g. deploy.prototxt.
task->setMeanScaleBGR({104.0f, 117.0f, 123.0f}, {1.0f, 1.0f, 1.0f});
```

Figure 46: setMeanScaleBGR Example

User calls the `cv::resize` function to scale the image, as shown in the figure below.



```
// Resize it if its size is not match.
cv::Mat image;
auto input_tensor = task->getInputTensor();
CHECK_EQ(input_tensor.size() , 1) << " the dpu model must have only one input";
auto width = input_tensor[0].width;
auto height = input_tensor[0].height;
auto size = cv::Size(width, height);
if (size != input_image.size()) {
  cv::resize(input_image, image, size);
} else {
  image = input_image;
}
```

Figure 47: cv::resize Example

How to use the Configuration File

Xilinx AI SDK provides another way to read model parameters by reading the configuration file. It facilitates uniform configuration management of model parameters. The configuration files are all located in `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/config`. When compiling the model to a ".so", you should name the configuration file same as the `${MODEL_NAME}.prototxt`.

Note that if you are developing on the host side, the configuration files are installed in `~/petalinux_sdk/sysroots/aarch64-xilinx-linux/etc/dpu_model_param.conf.d`.

```
model {
  name: "yolov3_voc_416"
  kernel {
     name: "yolov3_voc_416"
     mean: 0.0
     mean: 0.0
     mean: 0.0
     scale: 0.00390625
     scale: 0.00390625
     scale: 0.00390625
  }
  model_type : YOLOv3
  yolo_v3_param {
    …
  }
  is_tf: false
}
```

Table 3: Compiling Model and Kernel Parameters

Model/Kernel	Parameter Type	Description
model	name	This name should be same as the ${MODEL_NAME}.
	model_type	This type should depend on which type of model you used.
kernel	name	This name should be filled as the result of your DNNC compile. Sometimes, its name may have an extra postfix "_0", here need fill the name with such postfix. (For example: inception_v1_0)
	mean	Normally there are three lines, each of them corresponding to the mean-value of "BRG" which are pre-defined in the model.
	scale	Normally there are three lines. Each of them is corresponds to the RGB-normalized scale. If the model had no scale in training stage, here should fill with one.
	is_tf	Bool type, if your model is trained by tensorflow, please add this and set with "true". It could be blank in the prototxt or set as "false" when the model is caffe.

yolo_v3_param

```
  model_type : YOLOv3
  yolo_v3_param {
    num_classes: 20
    anchorCnt: 3
    conf_threshold: 0.3
    nms_threshold: 0.45
    biases: 10
    biases: 13
    biases: 16
    biases: 30
    biases: 33
    biases: 23
    biases: 30
    biases: 61
    biases: 62
    biases: 45
    biases: 59
    biases: 119
    biases: 116
    biases: 90
    biases: 156
    biases: 198
    biases: 373
    biases: 326
    test_mAP: false
  }
}
```

Below are the YOLOv3 model's parameters. You can modify them as your model requires.

Table 4: YOLOv3 Model Parameters

Parameter Type	Description
num_classes	The actual number of the model's detection categories
anchorCnt	The number of this model's anchor
conf_threshold	The threshold of the boxes' confidence, which could be modified to fit your practical application
nms_threshold	The threshold of NMS
biases	These parameters are same as the model's. Each bias need writes in a sperate line. (Biases amount) = anchorCnt * (output-node amount) * 2. set correct lines in the prototxt.
test_mAP	If your model was trained with letterbox and you want to test its mAP, set this as "true". Normally it is "false" for executing much faster.

SSD_param

```
model_type : SSD
ssd_param : {
    num_classes : 4
    nms_threshold : 0.4
    conf_threshold : 0.0
    conf_threshold : 0.6
    conf_threshold : 0.4
    conf_threshold : 0.3
    keep_top_k : 200
    top_k : 400
    prior_box_param {
    layer_width : 60,
    layer_height: 45,
    variances: 0.1
    variances: 0.1
    variances: 0.2
    variances: 0.2
    min_sizes: 21.0
    max_sizes: 45.0
    aspect_ratios: 2.0
    offset: 0.5
```

```
           step_width: 8.0
           step_height: 8.0
           flip: true
           clip: false
           }
       }
```

Below are the SSD parameters. The parameters of SSD-model include all kinds of threshold and PriorBox requirements. You can reference your SSD deploy.prototxt to fill them.

Table 5: SSD-Model Parameters

Parameter Type	Description
num_classes	The actual number of the model's detection categories
anchorCnt	The number of this model's anchor
conf_threshold	The threshold of the boxes' confidence. Each category could have a different threshold, but its amount must be equal to num_classes.
nms_threshold	The threshold of NMS
biases	These parameters are same as the model's. Each bias need writes in a separate line. (Biases amount) = anchorCnt * (output-node amount) * 2. Set correct lines in the prototxt.
test_mAP	If your model was trained with letterbox and you want to test its mAP, set this as "true". Normally it is "false" for executing much faster
keep_top_k	Each category of detection objects' top K boxes
top_k	All the detection object's top K boxes, except the background (the first category)
prior_box_param	There is more than one PriorBox, which could be found in the original model *(deploy.prototxt)* for corresponding each different scale. These PriorBoxes should oppose each other. (see Table 6 for Prior Box Parameters)

Table 6: PriorBox Parameters

layer_width/layer_height	The input width/height of this layer. Such numbers could be computed from the net structure.
variances	These numbers are used for boxes regression, just only to fill them as original model. There should be four variances.
min_sizes/max_size	Filled as the "deploy.prototxt", but each number should be written in a separate line.
aspect_ratios	The ratio's number (each one should be written in a separate line). Default has 1.0 as its first ratio. If you set a new number here, there will be two ratios created when the opposite is true. One is a filled number; another is its reciprocal. For example, this parameter has only one set element, "ratios: 2.0". The ratio vector has three numbers: 1.0, 2.0. 0.5
offset	Normally, the PriorBox is created by each central point of the feature map, so that offset is 0.5.

step_width/step_height	Copy from the original file. If there are no such numbers there, you can use the following formula to compute them:
	step_width = img_width ÷ layer_width
	step_height = img_height ÷ layer_height

offset	Normally, PriorBox is created by each central point of the feature map, so that the offset is 0.5.
flip	Control whether rotate the PriorBox and change the ratio of length/width.
clip	Set as false. If true, it will let the detection boxes' coordinates keep at [0, 1].

Example Code

The following is the example code.

```
Mat img = cv::imread(argv[1]);
auto yolo = xilinx::ai::YOLOv3::create("yolov3_voc_416", true);
auto results = yolo->run(img);
for(auto &box : results.bboxes){
    int label = box.label;
    float xmin = box.x * img.cols + 1;
    float ymin = box.y * img.rows + 1;
    float xmax = xmin + box.width * img.cols;
    float ymax = ymin + box.height * img.rows;
    if(xmin < 0.) xmin = 1.;
    if(ymin < 0.) ymin = 1.;
    if(xmax > img.cols) xmax = img.cols;
    if(ymax > img.rows) ymax = img.rows;
    float confidence = box.score;
    cout << "RESULT: " << label << "\t" << xmin << "\t" << ymin << "\t"
            << xmax << "\t" << ymax << "\t" << confidence << "\n";
    rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0),
                1, 1, 0);
}
imshow("", img);
waitKey(0);
```

You should use the "create" to create the YOLOv3 object.

```
    static std::unique_ptr<YOLOv3> create(const std::string& model_name,
                                          bool need_mean_scale_process = true);
```

Notes

The model_name is same as the prototxt's.

For more details about the example, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc/libdpyolov3/test/test_yolov3.cpp` .

How to Use the SDK Post-Processing Library (libxnnpp)

Post-processing is an important step in the whole process. Each neural network has different post-processing methods. The `libxnnpp.so` post-processing library is provided in SDK to facilitate user calls. It's a closed source library. It supports the following neural network post-processing.

- Classification (ResNet-18, ResNet-50, Inception-V1, Inception-V2, Inception-V3, Inception-V4, MobileNet-V1, MobileNet-V2)
- Face detection
- Face landmark detection
- SSD detection
- Pose detection
- Semantic segmentation
- Road line detection
- YOLOV3 detection
- YOLOV2 detection
- Openpose detection
- RefineDet detection
- ReId detection

There are two ways to call xnnpp.

- *One is automatic call, through* `xilinx::ai::<model>::create` *create the task, such as* `xilinx::ai::YOLOv3::create(xilinx::ai::YOLOV3_ADAS_512x256, true)`. *After <model>->run finished, xnnpp will be automatically processed, users can modify the parameters through the model configuration file.*

- One is manual call, through `xilinx::ai::DpuTask::create` to create the task. Then, create the object of the post-process and run the post-process. Take SSD post-processing as an example, the specific steps are as follows.

1. Create a config and set the correlating data to control post-process.

```
    using DPU_conf = xilinx::ai::proto::DpuModelParam;
    DPU_conf config;
```

2. If it is a caffemodel, set the "is_tf" false.

```
    config.set_is_tf(false);
```

3. Fill other parameters.

```
    fillconfig(config);
```

12. Create an object of SSDPostProcess.

```
    auto input_tensor = task->getInputTensor();
    auto output_tensor = task->getOutputTensor();
    auto ssd = xilinx::ai::SSDPostProcess::create(input_tensor, output_tensor,
 config);
```

13. Run the post-process.

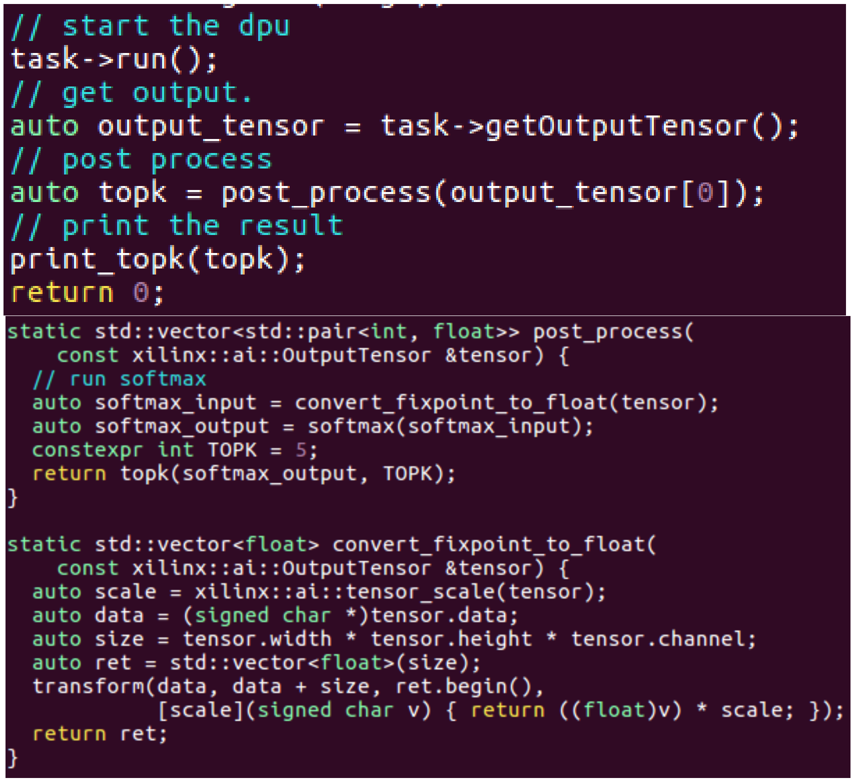
```
    auto results = ssd->ssd_post_process();
```

Notes

- The header files of the `libxnnpp.so` are stored in `~/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/include/xilinx/ai/nnpp/` on the host side.
- The libxnnpp.so is stored in `~/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib` on the host side.
- For more details about the post processing examples, refer to `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/ssd/ssd.cpp`, `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/yolov3/yolov3.cpp` and `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/libsrc/libdpyolov3/test/test_yolov3.cpp` .

How to implement user post-processing code

Users can also call their own post-processing functions on their own request. Take `classification1.cpp` as an example. First using `xilinx::ai::DpuTask::create` to create the task, Then after the DPU processing is complete, the user's post-processing function can be invoked. The post_process function in the following figure is a user post-processing code.

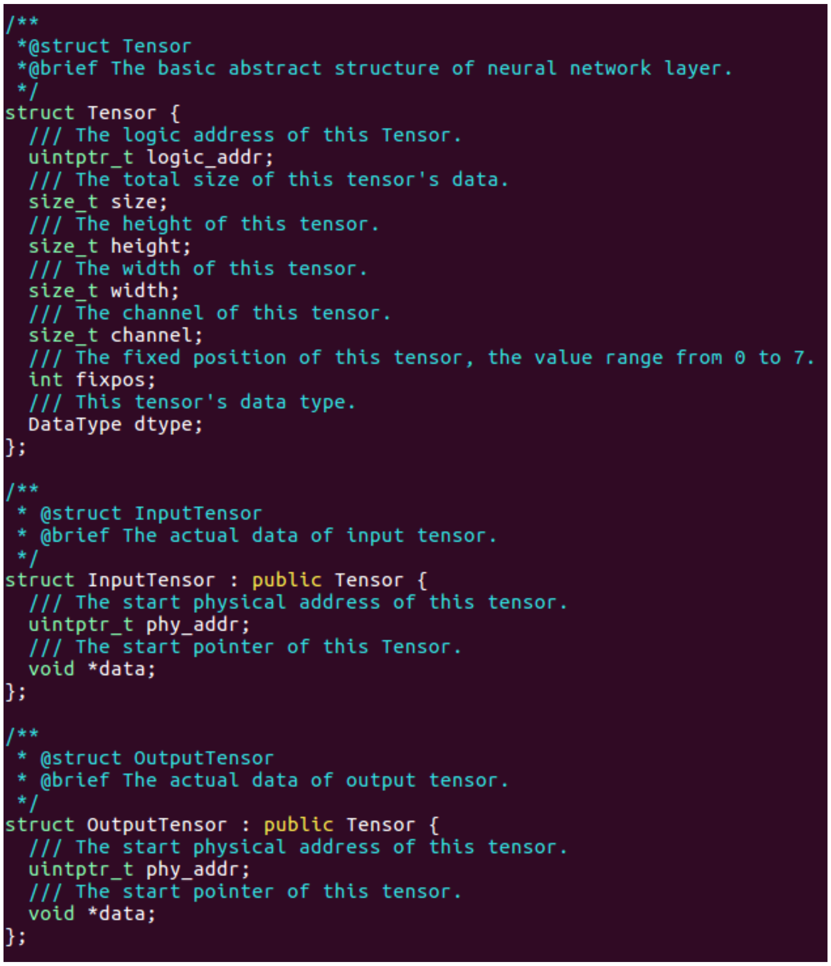


```
// start the dpu
task->run();
// get output.
auto output_tensor = task->getOutputTensor();
// post process
auto topk = post_process(output_tensor[0]);
// print the result
print_topk(topk);
return 0;
```

```
static std::vector<std::pair<int, float>> post_process(
    const xilinx::ai::OutputTensor &tensor) {
  // run softmax
  auto softmax_input = convert_fixpoint_to_float(tensor);
  auto softmax_output = softmax(softmax_input);
  constexpr int TOPK = 5;
  return topk(softmax_output, TOPK);
}

static std::vector<float> convert_fixpoint_to_float(
    const xilinx::ai::OutputTensor &tensor) {
  auto scale = xilinx::ai::tensor_scale(tensor);
  auto data = (signed char *)tensor.data;
  auto size = tensor.width * tensor.height * tensor.channel;
  auto ret = std::vector<float>(size);
  transform(data, data + size, ret.begin(),
            [scale](signed char v) { return ((float)v) * scale; });
  return ret;
}
```

Figure 48: User Post-Processing Code Example

The following figure shows the definition of OutputTensor. See `tensor.hpp` header file for details.



```
/**
 *@struct Tensor
 *@brief The basic abstract structure of neural network layer.
 */
struct Tensor {
  /// The logic address of this Tensor.
  uintptr_t logic_addr;
  /// The total size of this tensor's data.
  size_t size;
  /// The height of this tensor.
  size_t height;
  /// The width of this tensor.
  size_t width;
  /// The channel of this tensor.
  size_t channel;
  /// The fixed position of this tensor, the value range from 0 to 7.
  int fixpos;
  /// This tensor's data type.
  DataType dtype;
};

/**
 * @struct InputTensor
 * @brief The actual data of input tensor.
 */
struct InputTensor : public Tensor {
  /// The start physical address of this tensor.
  uintptr_t phy_addr;
  /// The start pointer of this Tensor.
  void *data;
};

/**
 * @struct OutputTensor
 * @brief The actual data of output tensor.
 */
struct OutputTensor : public Tensor {
  /// The start physical address of this tensor.
  uintptr_t phy_addr;
  /// The start pointer of this tensor.
  void *data;
};
```

Figure 49: The Definition of Tensor in tensor.hpp

Please refer to:

`~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/classification/classification1.cpp` and `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/classification/classification2.cpp` for more details.

Chapter 5: Application Demos

This chapter describes how to set up a test environment and how to run the application demos. There are two application demos provided within the XILINX AI SDK. Here, we take ZCU102 board as test platform.

Demo Overview

There are two application demos provided within the Xilinx AI SDK. They all use the SDK libraries to build their applications. The codes and video files are stored in `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/segs_and_roadline_detect` and `~/xilinx_ai_sdk_v2.0.x/xilinx_ai_sdk/demo/seg_and_pose_detect`.

`segs_and_roadline_detect` is a demo that includes multi-task segmentation network processing, vehicle detection and road line detection. It simultaneously performs 4-channel segmentation and vehicle detection and 1-channel road lane detection.

`seg_and_pose_detect` is a demo that includes multi-task segmentation network processing and pose detection. It simultaneously performs 1-channel segmentation process and 1-channel pose detection.

Note that to achieve the best performance, the demos use the DRM(Direct Render Manager) for video display. Please Log in the board via ssh or serial port and run the demo remotely. If you don't want to use DRM for video display, set "USE_DRM=0" in the compile option.

Demo Platform and Setup

Demo Platform

- HW
  - 1 x ZCU102 Prod Silicon

  https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

  - 1 x Win7/10 laptop
  - 1 x 16GB SD card
  - 1 x Ethernet cables
  - 1 x DP 1080P compatible monitor
  - 1 x DP cable

- SW
  - ZCU102 board image

https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge

  - xilinx_ai_sdk_v2.0.x
  - Images and video files
  - Terminal software like MobaXterm, Putty

DPU Configuration & Dev Tool Used

- 3xB4096 @294MHz
- Vivado 2019.1, AI SDK V2.0.x

Demo Setup Illustration

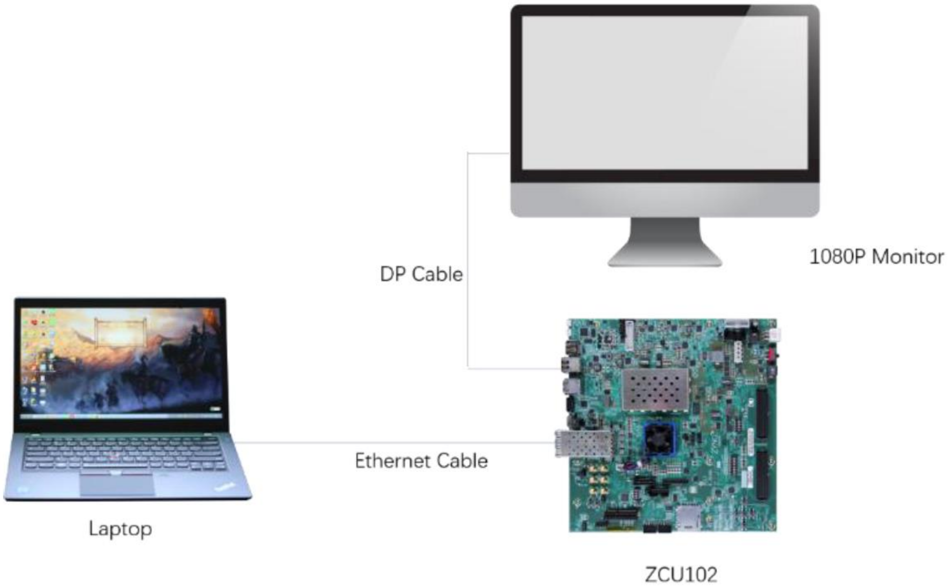


Figure 50: Demo setup Illustration

Demo 1: Multi-Task Segmentation + Car Detection and Road Line Detection

Target Application

- ADAS/AD

AI Model & Performance & Power

- FPN
  - 512x288, 4ch, 20fps
- VPGNET
  - 640x480, 1ch, 56fps

- 20W @ ZU9EG

Build & Run the Demo

```
#cd /usr/share/XILINX_AI_SDK/demo/segs_and_roadline_detect
```

```
#sh -x build_segs_and_roadline_detect.sh
```

To use DRM display, run the following command:

```
#./segs_and_roadline_detect_drm seg_512_288.avi seg_512_288.avi seg_512_288.avi
seg_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3 >/dev/null 2>&1
```

To use OpenCV display, run the following command.

```
#./segs_and_roadline_detect_x seg_512_288.avi seg_512_288.avi seg_512_288.avi
seg_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3 >/dev/null 2>&1
```

Demo Picture

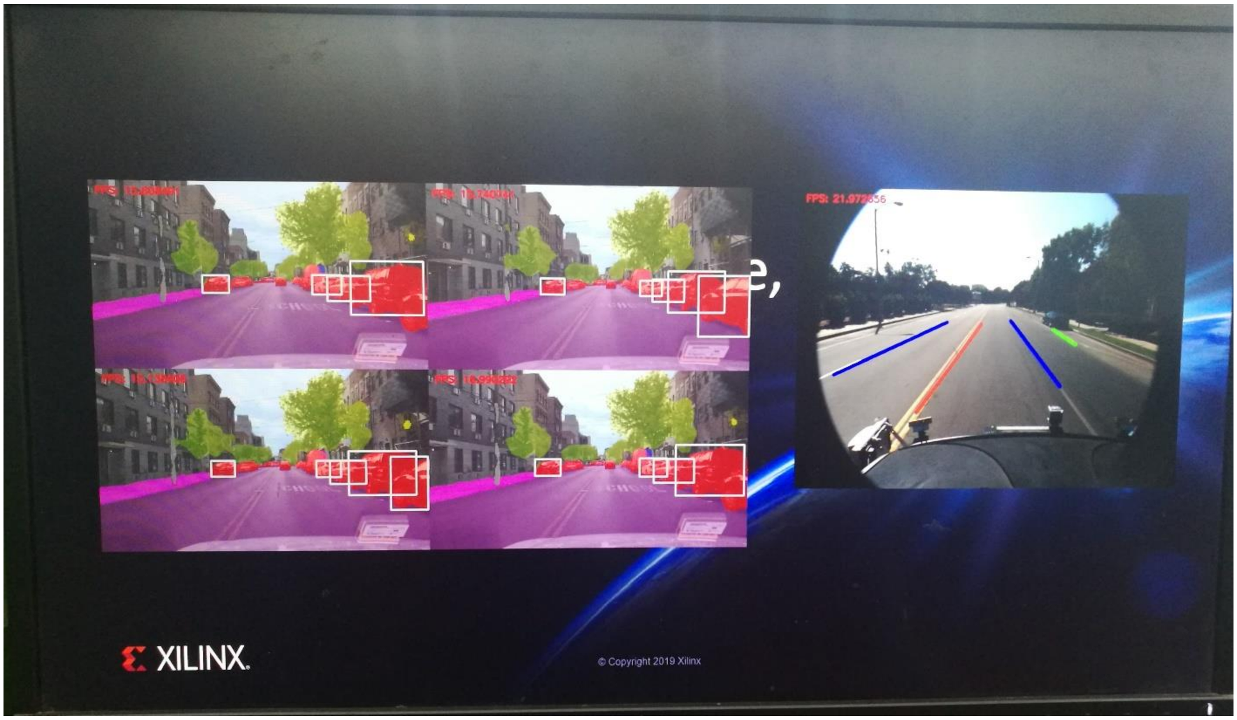


Figure 51: Segmentation and Roadline Detection Demo Picture

Demo 2: Multi-Task Segmentation+Car Detection and Pose Detection

Target Application

- ADAS/AD

- Smart city

AI Model & Performance & Power

- FPN
  - 960x540, 1ch, 22fps
- Openpose
  - 960x540, 1ch, 22fps
- 20W @ ZU9EG

Build & Run the Demo

```
#cd /usr/share/XILINX_AI_SDK/demo/seg_and_pose_detect
#sh -x build_seg_and_pose_detect.sh
```

To use DRM display, run the following command.

```
#./seg_and_pose_detect_drm seg_960_540.avi pose_960_540.avi -t 4 -t 4 >/dev/null
2>&1
```

To use OpenCV display, run the following command.

```
#./seg_and_pose_detect_x seg_960_540.avi pose_960_540.avi -t 4 -t 4 >/dev/null 2>&1
```

Demo Picture

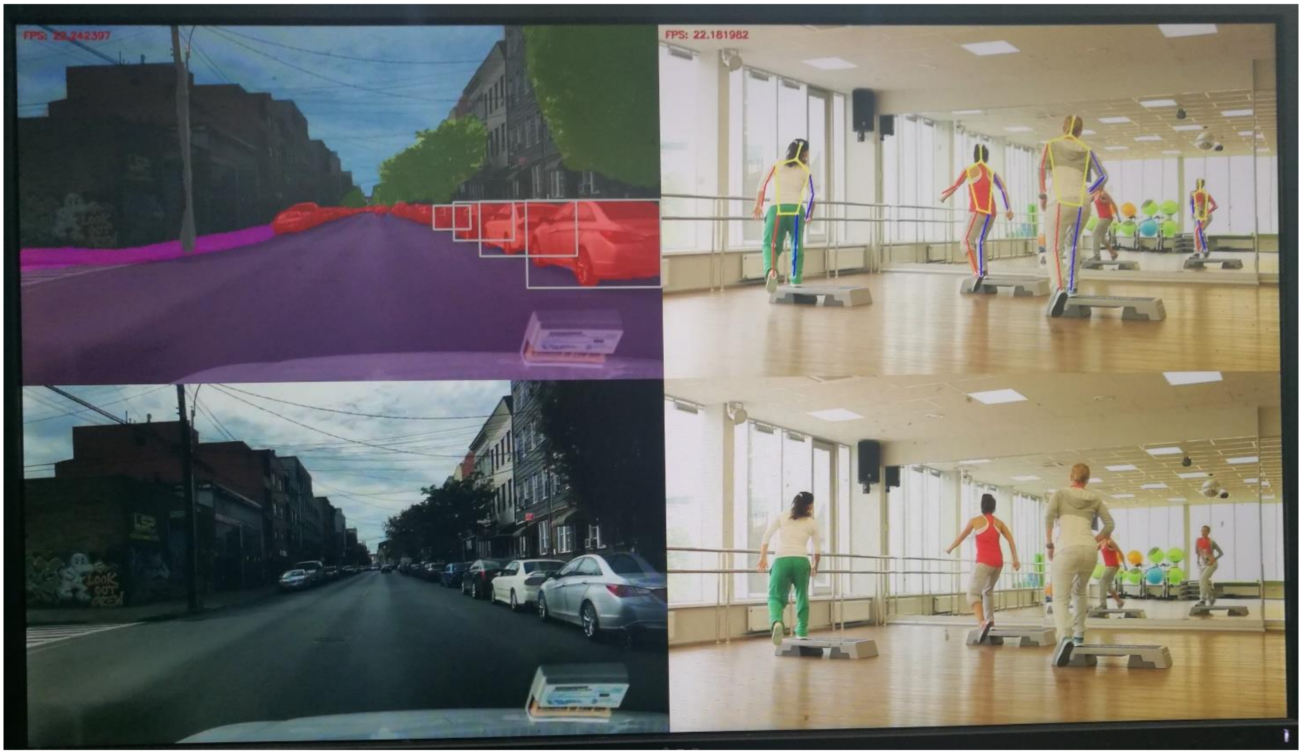


Figure 52: Segmentation and Pose Detection Demo Picture

Chapter 6: Programming APIs

For details about the Programming APIs, please refer to ug1355-xilinx-ai-sdk-programming-guide.pdf in the SDK or download it from the https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge .

This chapter describes in detail the performance of the Xilinx AI SDK on the following different boards.

- ZCU102 (0432055-04)
- ZCU102 (0432055-05)
- ZCU104
- Ultra96

ZCU102 Performance

The ZCU102 evaluation board uses the mid-range ZU9 UltraScale+ device. There are two different hardware versions of ZCU102 board, one with the serial number 0432055-04 as the header and the other with the serial number 0432055-05 as the header. The performance of the Xilinx AI SDK varies between the two hardware versions (because of different DDR performance). Table 7 is the performance of ZCU102 (0432055-04) and table 8 is the performance of ZCU102 (0432055-05). In both boards, triple B4096F DPU cores are implemented in program logic.

Refer to Table 7 for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-04) with DPU running at 287Mhz.

Table 7: ZCU102 (0432055-04) Performance

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single Thread)	Performance(fps) (Multiple Thread)
ResNet-18	224x224	3.65	196	524.4
ResNet-50	224x224	7.7	77.8	179.3
Inception-v1	224x224	3.2	182.7	485.5
Inception-v2	224x224	4.0	147.9	373.3
Inception-v3	299x299	11.4	58.8	155.4
Inception-v4	299x299	24.5	29.2	84.3
MobileNet-v1_TF[1]	224x224	1.14	312.1	876
MobileNet-v2	224x224	0.6	244.6	638.1

SqueezeNet	227x227	0.76	277.7	1080.8
ResNet-18_TF[1]	224x224	28	30.1	83.4
ResNet-50_TF[1]	224x224	7.0	83.8	191.4
Inception-v1_TF[1]	224x224	3.0	148.9	358.3
MobileNet-v2_TF1[1]	224x224	1.2	183.1	458.7
SSD_ADAS_VEHICLE	480x360	6.3	88.3	320.5
SSD_ADAS_PEDESTRIAN	640x360	5.9	77.2	314.7
SSD_TRAFFIC	480x360	11.6	57.2	218.2
SSD_MobileNet-v2	480x360	6.6	41.3	141.2
SSD_VOC_TF[1]	300x300	64.8	14.4	46.8
DenseBox (face detect)	320x320	0.49	412.2	1416.6
DenseBox_640x360	640x360	1.1	202.4	722.6
YOLOV3_ADAS	512x256	5.5	90.2	259.7
YOLOV3_ADAS	512x288	53.7	13.5	42.9
YOLOV3_VOC	416x416	65.4	14.2	44.4
YOLOV3_VOC_TF[1]	416x416	65.6	14.1	44
YOLOV2_BASELINE	448x448	34	25.2	86.4
YOLOV2_COMPRESS22G	448x448	11.6	54.3	211.2
YOLOV2_COMPRESS24G	448x448	9.9	60.9	242.4
YOLOV2_COMPRESS26G	448x448	7.8	69.2	286.7
RefineDet	480x360	25	33.8	108.2
RefineDet_10G	480x360	10.1	65	221.9
RefineDet_5G	480x360	5.1	90.6	312
FPN (segmentation)	512x256	8.9	60.3	203.9

VPGnet (roadline detection)	640x480	2.5	105.9	424.7
Sp-net (pose detection)	128x224	0.55	579.1	1620.7
Openpose	368x368	189	3.6	38.5
Face Landmark	72x96	0.14	885	1623.3
Reid	160x80	0.95	375	773.5

[1] These neural network models are trained based on the Tensorflow framework.

Refer to Table 8 for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-05) with DPU running at 287Mhz.

Table 8: ZCU102 (0432055-05) Performance

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
ResNet-18	224x224	3.65	195	485.2
ResNet-50	224x224	7.7	77	163.4
Inception-v1	224x224	3.2	181.7	452.4
Inception-v2	224x224	4.0	147	345.7
Inception-v3	299x299	11.4	58.5	144.9
Inception-v4	299x299	24.5	29	78.5
MobileNet-v1_TF[1]	224x224	1.14	309.7	809.5
MobileNet-v2	224x224	0.6	241.9	587.3
SqueezeNet	227x227	0.76	274.8	1012.2
ResNet-18_TF[1]	224x224	28	29.9	80.1
ResNet-50_TF[1]	224x224	7.0	82.9	173.3
Inception-v1_TF[1]	224x224	3.0	147.7	330.6
MobileNet-v2_TF[1]	224x224	1.2	181.1	422.2
SSD_ADAS_VEHICLE	480x360	6.3	88.3	306.3

SSD_ADAS_PEDESTRIAN	640x360	5.9	77.2	309.4
SSD_TRAFFIC	480x360	11.6	57.2	216
SSD_MobileNet-v2	480x360	6.6	40.5	124.7
SSD_VOC_TF[1]	300x300	64.8	14.4	47
DenseBox (face detect)	320x320	0.49	406.2	1311.8
DenseBox_640x360	640x360	1.1	201.4	650.1
YOLOV3_ADAS	512x256	5.5	89.6	239.7
YOLOV3_ADAS	512x288	53.7	13.5	42.1
YOLOV3_VOC	416x416	65.4	14.2	43.6
YOLOV3_VOC_TF[1]	416x416	65.6	14.1	43.1
YOLOV2_BASELINE	448x448	34	25.1	84.6
YOLOV2_COMPRESS22G	448x448	11.6	54.2	206.1
YOLOV2_COMPRESS24G	448x448	9.9	60.8	238
YOLOV2_COMPRESS26G	448x448	7.8	69.1	279.4
RefineDet	480x360	25	33.8	106.4
RefineDet_10G	480x360	10.1	64.9	215.7
RefineDet_5G	480x360	5.1	90.5	298.2
FPN (segmentation)	512x256	8.9	60	188.5
VPGnet (roadline detection)	640x480	2.5	106.5	396.9
Sp-net (pose detection)	128x224	0.55	574.8	1516.8
Openpose	368x368	189	3.6	16.6
Face Landmark	72x96	0.14	871.3	1444.3
ReID	160x80	0.95	370.3	702.8

[1] These neural network models are trained based on the Tensorflow framework.

ZCU104 Performance

The ZCU104 evaluation board uses the mid-range ZU7ev UltraScale+ device. Dual B4096F DPU cores are implemented in program logic and delivers 2.4 TOPS INT8 peak performance for deep learning inference acceleration.

Refer to Table 9 for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU104 with DPU running at 305Mhz.

Table 9: ZCU104 Performance

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
ResNet-18	224x224	3.65	206.7	428.6
ResNet-50	224x224	7.7	82.5	151.8
Inception-v1	224x224	3.2	197.3	404.9
Inception-v2	224x224	4.0	158	310.2
Inception-v3	299x299	11.4	62.4	126.3
Inception-v4	299x299	24.5	30.9	64.6
MobileNet-v1_TF[1]	224x224	1.14	330.3	728.4
MobileNet-v2	224x224	0.6	259.8	537
SqueezeNet	227x227	0.76	284	940.9
ResNet-18_TF[1]	224x224	28	32	62.8
ResNet-50_TF[1]	224x224	7.0	88.5	163.7
Inception-v1_TF[1]	224x224	3.0	157.4	305.5
MobileNet-v2_TF[1]	224x224	1.2	191.9	380.9
SSD_ADAS_VEHICLE	480x360	6.3	93.5	242.9
SSD_ADAS_PEDESTRIAN	640x360	5.9	82.5	236.1
SSD_TRAFFIC	480x360	11.6	60.7	159.6
SSD_MobileNet-v2	480x360	6.6	26.5	116.4

SSD_VOC_TF[1]	300x300	64.8	13.5	33.9
DenseBox (face detect)	320x320	0.49	428.5	1167.4
DenseBox_640x360	640x360	1.1	215	626.3
YOLOV3_ADAS	512x256	5.5	95.2	228.4
YOLOV3_ADAS	512x288	53.7	14.3	31.7
YOLOV3_VOC	416x416	65.4	15.1	33
YOLOV3_VOC_TF[1]	416x416	65.6	15	32.8
YOLOV2_BASELINE	448x448	34	26.6	63.8
YOLOV2_COMPRESS22G	448x448	11.6	57.1	158.9
YOLOV2_COMPRESS24G	448x448	9.9	64	186.9
YOLOV2_COMPRESS26G	448x448	7.8	72.6	224.9
RefineDet	480x360	25	35.9	78
RefineDet_10G	480x360	10.1	69.4	166.4
RefineDet_5G	480x360	5.1	96.2	241.8
FPN (segmentation)	512x256	8.9	63.6	177.3
VPGnet (roadline detection)	640x480	2.5	112.2	355.7
Sp-net (pose detection)	128x224	0.55	626.5	1337.3
Openpose	368x368	189	3.7	12.1
Face Landmark	72x96	0.14	977.7	1428.2
Reid	160x80	0.95	407.6	702.7

[1] These neural network models are trained based on the Tensorflow framework.

Ultra96 Performance

Ultra96™ is an ARM-based, Xilinx Zynq UltraScale+™ MPSoC development board based on the Linaro 96Boards specification. The 96Boards' specifications are open and define a standard board layout for development platforms that can be used by software application, hardware device, kernel, and other system software developers. Ultra96 represents a unique position in the 96Boards community with a wide range of potential peripherals and acceleration engines in the programmable logic that is not available from other offerings.

One B1600F DPU core is implemented in program logic of Ultra96 and delivers 459 GOPS INT8 peak performance for deep learning inference acceleration.

Refer to Table 10 for the throughput performance (in frames/sec or fps) for various neural network samples on Ultra96 with DPU running at 287Mhz.

Table 10: Ultra96 Performance with DPU running at 287Mhz

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
ResNet-18	224x224	3.65	74.2	77.8
ResNet-50	224x224	7.7	32.5	33.5
Inception-v1	224x224	3.2	71.6	75.1
Inception-v2	224x224	4.0	58.3	61.3
Inception-v3	299x299	11.4	22.7	23.4
Inception-v4	299x299	24.5	11.3	11.5
MobileNet-v1_TF[1]	224x224	1.14	167.6	186.6
MobileNet-v2	224x224	0.6	136.2	NA
SqueezeNet	227x227	0.76	167.9	283.6
ResNet-18_TF[1]	224x224	28	10.2	10.4
ResNet-50_TF[1]	224x224	7.0	35.7	36.6
Inception-v1_TF[1]	224x224	3.0	59	61.3
MobileNet-v2_TF[1]	224x224	1.2	98.3	104.3
SSD_ADAS_VEHICLE	480x360	6.3	41.2	46.2
SSD_ADAS_PEDESTRIAN	640x360	5.9	42.9	50.8
SSD_TRAFFIC	480x360	11.6	28.2	31.8
SSD_MobileNet-v2	480x360	6.6	16.5	27.8
SSD_VOC_TF[1]	300x300	64.8	5.4	5.8

DenseBox (face detect)	320x320	0.49	239.9	334.2
DenseBox_640x360	640x360	1.1	117	167.2
YOLOV3_ADAS	512x256	5.5	43.9	49.7
YOLOV3_ADAS	512x288	53.7	5.2	5.3
YOLOV3_VOC	416x416	65.4	5.4	5.5
YOLOV3_VOC_TF[1]	416x416	65.6	5	5.1
YOLOV2_BASELINE	448x448	34	8.5	8.9
YOLOV2_COMPRESS22G	448x448	11.6	26.7	30.7
YOLOV2_COMPRESS24G	448x448	9.9	32.3	38.4
YOLOV2_COMPRESS26G	448x448	7.8	38	46.8
RefineDet	480x360	25	15.1	NA
RefineDet_10G	480x360	10.1	31.1	33.7
RefineDet_5G	480x360	5.1	49.3	55.3
FPN (segmentation)	512x256	8.9	27.5	NA
VPGnet (roadline detection)	640x480	2.5	71.9	NA
Sp-net (pose detection)	128x224	0.55	261.6	277.4
Openpose	368x368	189	1.8	NA
Face Landmark	72x96	0.14	339.2	347.6
Reid	160x80	0.95	159.2	166.6

[1] These neural network models are trained based on the Tensorflow framework.

Appendix A: Legal Notices

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS “XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (“SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (“SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.