

# **Xilinx AI SDK User Guide**

UG1354 (v1.2) July 3, 2019



## Revision History

The following table shows the revision history for this document.

Section	Revision Summary
<b>7/3/2019 Version 1.2</b>	
<a href="#">Chapter 5: Libraries</a>	Added Inception_V4, YOLOV2, Roadline_DEEPhi Model.
	Removed RefineDet_640x480 Model.
<a href="#">Chapter 8: Performance</a>	Updated performance data for ZCU102, ZCU104, Ultra96.
Entire document.	Editorial updates.
<b>4/29/2019 Version 1.0</b>	
Initial Xilinx release.	N/A

# Table of Contents

Revision History .....	2
Chapter 1: Introduction .....	5
Overview .....	5
Block Diagram .....	5
Features .....	6
Chapter 2: Quick Start.....	7
Downloading the Xilinx AI SDK.....	7
Setting Up the Host.....	7
Setting Up the Target.....	9
Running Xilinx AI SDK Examples .....	9
Building Your Own Application .....	11
Support .....	12
Chapter 3: Installation.....	13
Installing a Board Image.....	13
Installing DNNDK .....	14
Installing the SDK Package.....	14
Chapter 4: Cross-Compiling .....	16
Installing an OS on the Linux Server.....	16
Setting Up the Cross-Compiling Environment .....	16
Compiling the Program .....	17
Using the Provided Build Scripts.....	18
Using Makefile.....	19
Using Cmake.....	20
Chapter 5: Libraries .....	22
Classification .....	23
Face Detection .....	27
SSD Detection .....	30

Pose Detection .....	34
Semantic Segmentation .....	37
Road Line Detection.....	41
YOLOV3 Detection.....	44
YOLOv2 Detection .....	47
Openpose Detection.....	49
RefineDet Detection.....	52
DNNDK Sample .....	56
Chapter 6: Libraries Advanced Application.....	59
Introduction .....	59
How to Compile.....	59
How to Fill a Configuration File .....	59
How to Use the Code .....	64
Chapter 7: Programming APIs .....	65
Chapter 8: Performance .....	66
ZCU102 Performance.....	66
ZCU104 Performance.....	69
Ultra96 Performance.....	71
Appendix A: Legal Notices .....	74
Please Read: Important Legal Notices .....	74

## Overview

The Xilinx AI SDK is a set of high-level libraries built to be used with the Deep Neural Network Development Kit (DNNDK) and Deep-Learning Processor Unit (DPU). By encapsulating a large number of efficient and high-quality neural networks, the Xilinx AI SDK provides an easy-to-use and unified interface. 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 the business layer rather than the underlying hardware.

## Block Diagram

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

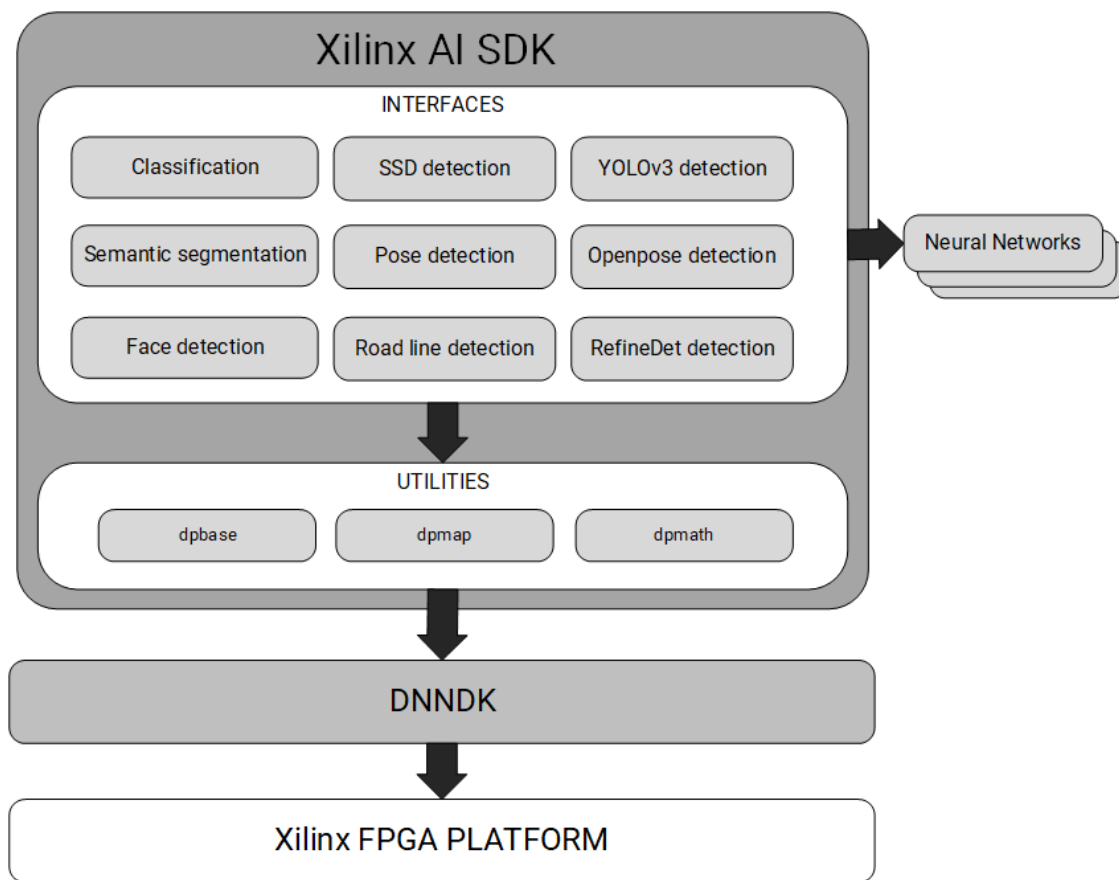


Figure 1: Xilinx AI SDK Block Diagram

---

## Features

The Xilinx AI SDK features include:

- Full stack
- Embedded
- Optimized
- Unified interface
- Practical application

### 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 `xlnx_dnndk_v3.0_190430.tar.gz`.
2. Extract the package contents.

```
$tar zxvf xlnx_dnndk_v3.0_190430.tar.gz
$cd xlnx_dnndk_v3.0_190430
$tar zxvf XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26.tar.gz
```

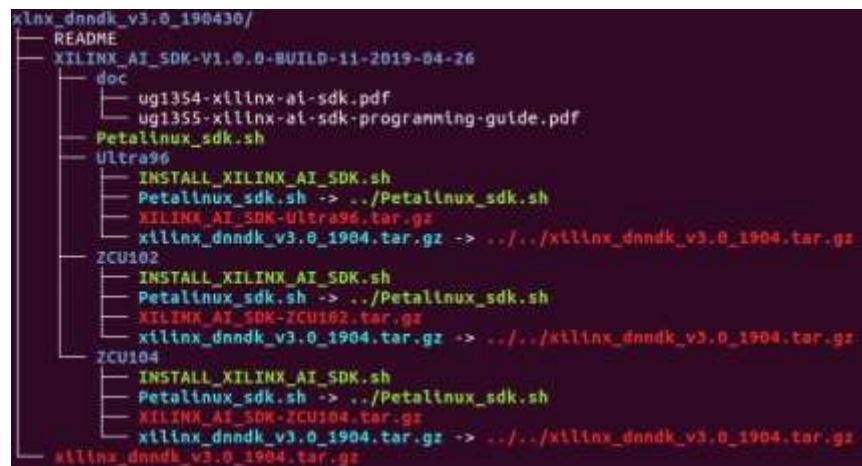


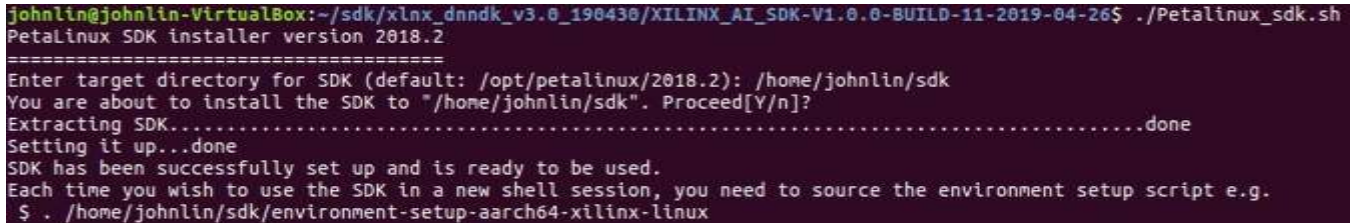
Figure 2: SDK Directory Structure

### 3. Install the PetaLinux SDK.

```
$cd XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26
$./Petalinux_sdk.sh
```

### 4. Follow the prompts to install. Figure 3 shows the installation process.

Note that if the installation is in the default path, the installation setting will be for a standard user. Because of this setting, 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:~/sdk/xlnx_dnnmk_v3.0_190430/XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26$ ./Petalinux_sdk.sh
Petalinux SDK installer version 2018.2
=====
Enter target directory for SDK (default: /opt/petalinux/2018.2): /home/johnlin/sdk
You are about to install the SDK to "/home/johnlin/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/sdk/environment-setup-aarch64-xilinx-linux
```

**Figure 3: PetaLinux Installation Process**

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

```
$ . /home/johnlin/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. Select a platform and enter its directory; then, install the Xilinx AI SDK.

```
$cd ZCU102
$./INSTALL_XILINX_AI_SDK.sh
```

### 7. Compile the samples in the Xilinx AI SDK.

```
$cd /home/johnlin/sdk/sysroots/aarch64-xilinx-
linux/usr/share/XILINX_AI_SDK/samples/classification
$sh build.sh
```

If you want to see print information for the compilation process, execute the following instructions.

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

Then, the executable program is produced.

For detailed installation of the host environment, refer to Chapter 4.



---

## Setting Up the Target

Before setting up the target, make sure that the system image of the evaluation board is the official version and that the DNNDK v3.0 has been installed.

If not, refer to the detailed installation instructions in Chapter 3.

1. Enter the platform directory, then copy the `XILINX_AI_SDK-ZCU102.tar.gz` to the board via SCP.

```
#cd XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26/ZCU102
#scp XILINX_AI_SDK-ZCU102.tar.gz root@IP_OF_CARD:~/
```

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

```
#tar zxvf XILINX_AI_SDK-ZCU102.tar.gz -C /
```

---

## 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 the Quick Start phase, we recommend compiling and running the examples directly on the target machine.

To compiling and run the examples directly on the target machine, the following steps must be followed:

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_facedetect_dense_box_640x360
sample_facedetect_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 build.sh
```

3. View the running results.

There are two ways to view the results. One is to view the results by printing the information, while the other is to view images by downloading the `sample_facedetect_dense_box_640x360_result.jpg` image (see Figure 4).



**Figure 4: Face Detection Example**

4. If you want to run the video example, execute the following command:

```
#./test_video_facedetect_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>
```

Make sure to enable X11 forwarding with the command **below. An example is provided in which the host machine IP address is 192.168.0.10.** When logging in to the board using an SSH terminal, all the video examples require a Linux Windows system to work properly.

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

5. If you want to test the performance of the model, run the following command:

```
#./test_performance_facedetect_dense_box_640x360
test_performance_facedetect_dense_box_640x360.list -t 1
-t: <num_of_threads>
```

6. For more parameter information, enter -h for viewing.

On the following page, Figure 5 shows the result of performance testing in one thread.

```

root@zcu102:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_performance_facedetect_dense_box_640x360 test_perfo
rmance_facedetect_dense_box_640x360.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:20:19.552323 19991 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 10:20:24.552664 19991 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 10:20:29.552846 19991 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 10:20:34.553023 19991 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 10:20:39.553202 19991 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 10:20:44.553385 19991 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 10:20:49.553619 19991 benchmark.hpp:150] waiting for threads terminated
I0404 10:20:49.558184 19991 benchmark.hpp:174] writing report to <STDOUT>
FPS=148.433
E2E_MEAN=6737.14
DPU_MEAN=2680.42

```

**Figure 5: Face Detection Performance Test Result**

7. If you want to check the version of Xilinx AI SDK, run the following command:

```
#xilinx_ai
```

## Building Your Own Application

Before developing your own applications, make sure your development environment is ready. Whether you are programming on the host side or target side, you will need to install the corresponding Xilinx AI SDK.

To quickly build your own application using the Xilinx AI SDK library, use the YOLOv3 library as an example.

### 1. Include the necessary head file.

```

#include <xilinx/yolov3/yolov3.hpp> //Necessary head file
#include <iostream>
#include <map>
#include <string>
//The necessary opencv lib
#include <opencv2/opencv.hpp>

```

### 2. Load an image.

```
Mat img = cv::imread(argv[1]);
```

### 3. Initiate an instance.

```
auto yolo = xilinx::yolov3::YOLOv3::create_ex("yolov3_voc_416", true);
```

### 4. Run the command.

```
auto results = yolo->run(img);
```

### 5. Draw and show the result.

```

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);
}
```

For more details about the sample code, refer to

[/usr/share/XILINX\\_AI\\_SDK/samples/yolov3/test\\_customer\\_provided\\_model\\_yolov3.cpp](/usr/share/XILINX_AI_SDK/samples/yolov3/test_customer_provided_model_yolov3.cpp)

For more details about the advanced application, refer to Chapter 6.

---

## Support

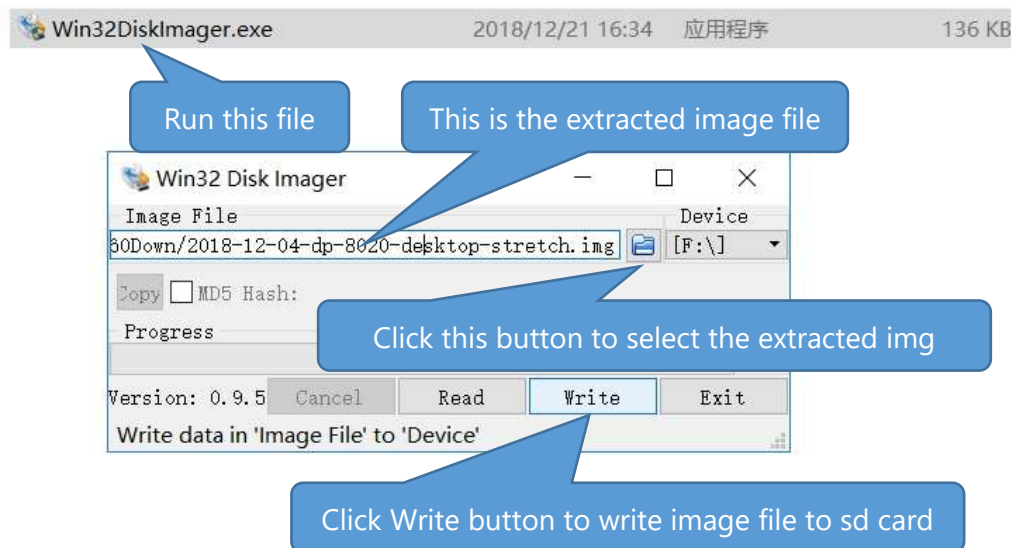
You can visit the Xilinx AI SDK community forum on the Xilinx website at the following address:

<https://forums.xilinx.com/t5/Deephi-DNNDK/bd-p/Deephi>. This resource includes topic discussions, knowledge sharing, FAQs, and requests for technical support.

This chapter provides a detailed description of the SDK installation.

### Installing a Board Image

1. Download an image file from a corresponding card 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.



**Figure 6: Win32DiskImager Usage Steps**

3. Insert the SD card with the 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.

---

## Installing DNNDK

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. Copy the corresponding sub-directory from the extracted package to the destination board. For example:

```
scp -r ZCU102 root@ip_of_card_zcu102:~/
```

3. Log in to the board (via SSH or serial port), enter the ZCU102 directory, and run `install.sh` to set up the running environment of DNNDK in the board.
4. Also, enter the ZCU102 directory in the host computer and run `host_x86/install.sh` to set up the development environment in the host computer.

---

## Installing the SDK Package

1. Download the corresponding SDK package from <https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge>, such as `xlnx_dnndk_v3.0_190430.tar.gz`, then extract it.

```
$tar zxvf xlnx_dnndk_v3.0_190430.tar.gz
$cd xlnx_dnndk_v3.0_190430
```

```
$tar zxvf XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26.tar.gz
```

2. Enter the platform directory, then copy the `XILINX_AI_SDK-ZCU102.tar.gz` to the board via SCP.

```
$cd XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26/ZCU102
$scp XILINX_AI_SDK-ZCU102.tar.gz root@IP_OF_CARD:~/
```

3. Log in to the board (via SSH or serial port) and execute the following command:

```
#tar zxvf XILINX_AI_SDK-ZCU102.tar.gz -C /
```

4. Enter the extracted directory of the example in the target board.

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

5. Run the example.

```
#./test_jpeg_facedetect_dense_box_640x360
sample_facedetect_dense_box_640x360.jpg
```



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

```
#sh build.sh
```

7. If you want to compile the example in host, refer to the cross-compiling section.

Note the following after the installation is complete:

- Model and Library files are stored in `/usr/lib`
- The header files are stored in `/usr/include/Xilinx`
- Model profiles are stored in `/etc/XILINX_AI_SDK.conf.d`
- Samples are stored in `/usr/share/XILINX_AI_SDK/samples`
- Documents are stored in `/usr/share/XILINX_AI_SDK/doc`

Readers will need basic cross-environment knowledge to understand this section, as the whole developing environment exists in the board, including `g++`, `gdb`, etc. To improve the compiling speed, use the cross-compiling environment.

### Installing an OS on the Linux Server

It is recommended that you install the latest stable release of an OS on the Linux server (such as Ubuntu 16 or CentOS 7) to ensure easily-available technical support. The following list includes some recommended OS versions:

- Later versions of Ubuntu 64bit, such as Ubuntu 16.04, Ubuntu 18.04
- Earlier versions of Ubuntu 64bit, such as Ubuntu 14.04

### Setting Up the Cross-Compiling Environment

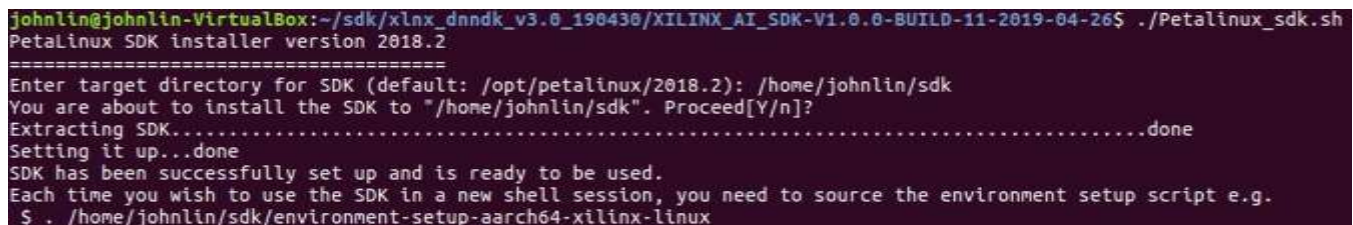
1. Download the corresponding SDK package from <https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge>, such as `xlnx_dnndk_v3.0_190430.tar.gz`, and extract it.

```
$tar zxvf xlnx_dnndk_v3.0_190430.tar.gz
$cd xlnx_dnndk_v3.0_190430
$tar zxvf XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26.tar.gz
```

2. Install the PetaLinux SDK.

```
$cd XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26
$./Petalinux_sdk.sh
```

3. Follow the prompts to install. Figure 7 shows the installation process.
4. Note that if the installation is in the default path, the installation setting will be for a standard user. Because of this setting, 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:~/sdk/xlnx_dnndk_v3.0_190430/XILINX_AI_SDK-V1.0.0-BUILD-11-2019-04-26$ ./Petalinux_sdk.sh
Petalinux SDK installer version 2018.2
=====
Enter target directory for SDK (default: /opt/petalinux/2018.2): /home/johnlin/sdk
You are about to install the SDK to "/home/johnlin/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/sdk/environment-setup-aarch64-xilinx-linux
```

Figure 7: PetaLinux Installation Process



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

```
$ . /home/johnlin/sdk/environment-setup-aarch64-xilinx-linux
```

If you close the current terminal, you will need to re-execute the above instructions in the new terminal interface.

6. Select a platform and enter its directory. Then, install the Xilinx AI SDK.

```
$cd ZCU102
```

```
$./INSTALL_XILINX_AI_SDK.sh
```

Note the following after the host installation is complete,

- Model and Library files are stored in:  
`/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/lib`
- The header files are stored in:  
`/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/include/Xilinx`
- Model profiles are stored in:  
`/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/etc/XILINX_AI_SDK.conf.d`
- Samples are stored in:  
`/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/share/XILINX_AI_SDK/samples`

---

## Compiling the Program

On the host side, there are several methods to compile a target program. You can choose one of the following ways to compile the program.

- Using the provided build scripts
- Using Makefile
- Using Cmake

All the compile tool chains are stored in `/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux` directory, such as `aarch64-linux-gnu-g++`. Furthermore, the target system root is stored in `/home/johnlin/sdk/sysroots/x86_64-petalinux-linux`.

In addition, you can see how the cross-compilation system is configured by looking at the `environment-setup-aarch64-xilinx-linux` script which is stored in `/home/johnlin/sdk`. For reference, see Figure 8 on the following page.

```
johnlin@johnlin-VirtualBox:~/sdk$ cat environment-setup-aarch64-xilinx-linux
# Check for LD_LIBRARY_PATH being set, which can break SDK and generally is a bad practice
# http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80
# http://xahlee.info/UnixResource_dir/_/ldpath.html
# Only disable this check if you are absolutely know what you are doing!
if [ ! -z "$LD_LIBRARY_PATH" ]; then
    echo "Your environment is misconfigured, you probably need to 'unset LD_LIBRARY_PATH'"
    echo "but please check why this was set in the first place and that it's safe to unset."
    echo "The SDK will not operate correctly in most cases when LD_LIBRARY_PATH is set."
    echo "For more references see:"
    echo " http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80"
    echo " http://xahlee.info/UnixResource_dir/_/ldpath.html"
    return 1
fi
export SDKTARGETSYSROOT=/home/johnlin/sdk/sysroots/aarch64-xilinx-linux
export PATH=/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/bin:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/sbin:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/bin:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/sbin:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/bin/./x86_64-petalinux-linux/bin:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux:/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux-musl:$PATH
export PKG_CONFIG_SYSROOT_DIR=$SDKTARGETSYSROOT
export PKG_CONFIG_PATH=$SDKTARGETSYSROOT/usr/lib/pkgconfig:$SDKTARGETSYSROOT/usr/share/pkgconfig
export CONFIG_SITE=/home/johnlin/sdk/site-config-aarch64-xilinx-linux
export DECORE_NATIVE_SYSROOT="/home/johnlin/sdk/sysroots/x86_64-petalinux-linux"
export DECORE_TARGET_SYSROOT="$SDKTARGETSYSROOT"
export DECORE_ACLOCAL_OPTS="-I /home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/share/aclocal"
unset command_not_found_handle
export CC="aarch64-xilinx-linux-gcc --sysroot=$SDKTARGETSYSROOT"
export CXX="aarch64-xilinx-linux-g++ --sysroot=$SDKTARGETSYSROOT"
export CPP="aarch64-xilinx-linux-gcc -E --sysroot=$SDKTARGETSYSROOT"
export AS="aarch64-xilinx-linux-as"
export LD="aarch64-xilinx-linux-ld --sysroot=$SDKTARGETSYSROOT"
export GDB=aarch64-xilinx-linux-gdb
export STRIP=aarch64-xilinx-linux-strip
export RANLIB=aarch64-xilinx-linux-ranlib
export OBJCOPY=aarch64-xilinx-linux-objcopy
export OBJDUMP=aarch64-xilinx-linux-objdump
export AR=aarch64-xilinx-linux-ar
export NM=aarch64-xilinx-linux-nm
```

Figure 8: Environment Configuration Script

## Using the Provided Build Scripts

We provide the script in each sample directory of `build.sh`. Figure 9 shows the contents of the `build.sh` script.

```
johnlin@johnlin-VirtualBox:~/sdk/sysroots/aarch64-xilinx-linux/usr/share/XILINX_AI_SDK/samples/roadline$ cat build.sh
CXX=${CXX:-g++}
$CXX -std=c++11 -I. -o test_accuracy_roadline test_accuracy_roadline.cpp -lopencv_core -lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui -ldproadline -lglog
$CXX -std=c++11 -I. -o test_jpeg_roadline test_jpeg_roadline.cpp -lopencv_core -lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui -ldproadline -lglog
$CXX -std=c++11 -I. -o test_performance_roadline test_performance_roadline.cpp -lopencv_core -lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui -ldproadline -pthread -ldpbase -lglog
$CXX -std=c++11 -I. -o test_video_roadline test_video_roadline.cpp -lopencv_core -lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui -ldproadline -pthread -lglog
```

Figure 9: Build.sh Script

When using this script, make sure that the cross-compile environment is configured. If not, or if the last terminal interface is still present, you will need to execute the following command again:

```
$ . /home/johnlin/sdk/environment-setup-aarch64-xilinx-linux
```

If you want to compile the samples in the Xilinx AI SDK, use the commands below:

```
$cd /home/johnlin/sdk/sysroots/aarch64-xilinx-  
linux/usr/share/XILINX_AI_SDK/samples/classification  
  
$sh -x build.sh
```

The executable program is produced once the steps above have been completed.

If you want to compile your own program, use the `helloworld.cpp` as an example and change the `build.sh` as follows:

```
CXX=${CXX:-g++}  
$CXX -std=c++11 -I. -o helloworld helloworld.cpp
```

Then, put the `build.sh` and the `helloworld.cpp` in the same directory. When this is completed, run the following command to generate the program.

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

---

## Using Makefile

1. A typical Makefile is shown below. Only list the parts which need modification.

```
CXX      :=    g++  
CC       :=    gcc  
OBJ      :=    main.o  
  
# linking libraries of OpenCV  
LDFLAGS  =    $(shell pkg-config --libs opencv)  
# linking libraries of DNNNDK  
LDFLAGS  +=    -lhineon -ln2cube -ldputils -lpthread  
  
CUR_DIR  =    $(shell pwd)  
SRC      =    $(CUR_DIR)/src  
  
BUILD    =    $(CUR_DIR)/build  
VPATH    =    $(SRC)  
MODEL    =    $(CUR_DIR)/model/dpu_densebox.elf  
  
ARCH     =    $(shell uname -m | sed -e s/arm.*/armv71/ -e s/aarch64.*/aarch64/)
```

2. Change the CXX item.

```
CXX      :=    /home/johnlin/sdk/sysroots/x86_64-petalinux-  
linux/usr/bin/aarch64-xilinx-linux/aarch64-linux-gnu-g++
```

3. The original LDFALGS gets the OpenCV library information via a shell command of `pkg-config`. You will need to change it from the cross-compiling environment by following the action below:

```
LDFLAGS  =    --sysroot=/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/lib  
-lopencv_highgui -lopencv_video -lopencv_videoio -lopencv_imgcodecs -  
lopencv_imgproc -lopencv_core
```

4. ARCH needs to be modified to access the destination board type instead of using the shell command.

```
ARCH      =   aarch64
```

The final Makefile modification is as below:

```
CXX      :=   /home/johnlin/sdk/sysroots/x86_64-petalinux-
linux/usr/bin/aarch64-xilinx-linux/aarch64-linux-gnu-g++
CC       :=   /home/johnlin/sdk/sysroots/x86_64-petalinux-
linux/usr/bin/aarch64-xilinx-linux/aarch64-linux-gnu-gcc
OBJ      :=   main.o

# linking libraries of OpenCV
LDFLAGS  = --sysroot=/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/usr/lib
-lopencv_highgui -lopencv_video -lopencv_videoio -lopencv_imgcodecs -
lopencv_imgproc -lopencv_core
# linking libraries of DNNNDK
LDFLAGS  += -lhineon -ln2cube -ldputils -lpthread

CUR_DIR  =   $(shell pwd)
SRC      =   $(CUR_DIR)/src
BUILD    =   $(CUR_DIR)/build
VPATH    =   $(SRC)
MODEL    =   $(CUR_DIR)/model/dpu_densebox.elf

ARCH     =   aarch64
```

5. After above modification, run “make” to produce the executable program, and copy it to the board to execute.

## Using Cmake

Let’s use Facedetect in the sample directory as one example.

1. Enter the directory of /home/johnlin/sdk/sysroots/x86\_64-petalinux-linux/usr/share/XILINX\_AI\_SDK/samples/facedetect,
2. Create a new file of CMakeLists.txt including the below content:

```
cmake_minimum_required(VERSION 3.5)
project(facedetect VERSION 2.0.2 LANGUAGES C CXX)

set(OpenCV_LIBS opencv_core opencv_video opencv_videoio opencv_imgproc
opencv_imgcodecs opencv_highgui)
set(SDK_LIBS dpfacedetect)
add_executable(test_jpeg_facedetect_dense_box_320x320
test_jpeg_facedetect_dense_box_320x320.cpp)
target_link_libraries(test_jpeg_facedetect_dense_box_320x320
${OpenCV_LIBS} ${SDK_LIBS} glog)
```

3. Run the following shell command:

```
# mkdir build; cd build;  
# cmake -DCMAKE_TOOLCHAIN_FILE=/home/johnlin/sdksysroot/x86_64-petalinux-  
linux/usr/share/cmake/OEToolchainConfig.cmake ..  
# make
```

4. Then, the compiling process completes correctly and the executable program is produced.

## Chapter 5: Libraries

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

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

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

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

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. There are several points to be noted, as follows:

- When testing with video files, different video file formats will lead to inconsistent decoding time of the CPU, resulting in varying performance rates. For example, when testing with an H264 file, there may be instantaneous frame loss or stuttering during decoding. We recommend using raw video for video evaluation.
- In addition, if the user is accustomed to using DNNDK, we still provide several examples of DNNDK in the Xilinx AI SDK, such as ResNet-50, YOLOv3 and Semantic Segmentation. This document is described in detail with ResNet-50 as an example. Refer to the "DNNDK Sample" section for details.



## Classification

### Introduction

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

Now, we have integrated Resnet50, Inception\_v1, Inception\_v2, Inception\_v3, Inception\_v4, mobilenet\_v2 into our library. Input is a picture with an object, whereas output is the top-K most probable category.



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/classification# ./test_jpeg_classification_inception_v3 sample_classification_inception_v3.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:38:06.134699 20216 process_result.hpp:10] r.index 949 strawberry, r.score 0.470298
I0404 10:38:06.135076 20216 process_result.hpp:10] r.index 953 pineapple, ananas, r.score 0.222153
I0404 10:38:06.135190 20216 process_result.hpp:10] r.index 954 banana, r.score 0.173013
I0404 10:38:06.135272 20216 process_result.hpp:10] r.index 957 pomegranate, r.score 0.0234148
I0404 10:38:06.135371 20216 process_result.hpp:10] r.index 956 custard apple, r.score 0.0142018
I0404 10:38:06.141217 20216 demo.hpp:362] result image write to sample_classification_inception_v3_result.jpg
I0404 10:38:06.141254 20216 demo.hpp:364] BYEBYE
```

Figure 10: 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.

For images, the calling method is as follows:

```
#include <xilinx/classification/classification.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::classification::Classification::create(xilinx::classification::INCEP
TION_V1); }, process_result);
}
```

1. Before you run the classification example, you can choose one of the following executable program to run:

- test\_jpeg\_classification\_resnet\_50
- 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\_v2\_tf

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

```
#sh build.sh
```

3. Run the example.

```
#./test_jpeg_classification_resnet_50 sample_classification_RESNET_50.jpg
```

4. You will see the print result on the terminal (see Figure 10).



For video, the calling method is as follows:

```
#include <xilinx/classification/classification.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::classification::Classification::create(xilinx::classification::INCEP
TION_V1); },      process_result);
}
```

## 5. Run the video example.

```
#./test_video_classification_resnet_50 video_input.mp4 -t 8
```

## 6. If you want to test model performance, you can refer to the following call example. By default, we prepared a set of images for model performance testing which are stored in the ./images directory.

```
#include <xilinx/classification/classification.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::classification::Classification
::create(xilinx::classification::INCEPTION_V1);
        }
    });
}
```

## 7. Before you run the classification performance example, you can choose one of the following executable program to run:

- test\_performance\_classification\_resnet\_50
- 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\_v2\_tf

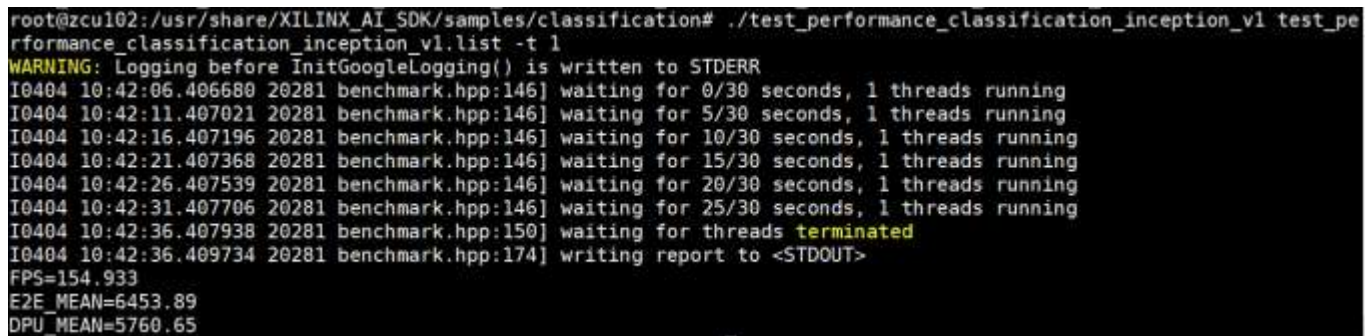
- If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

- Run the example.

```
#!/test_performance_classification_inception_v1
test_performance_classification_inception_v1.list -t 1
```

- You will see the printing result on the terminal, see Figure 11.



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/classification# ./test_performance_classification_inception_v1 test_per
formance_classification_inception_v1.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:42:06.406680 20281 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 10:42:11.407021 20281 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 10:42:16.407196 20281 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 10:42:21.407368 20281 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 10:42:26.407539 20281 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 10:42:31.407706 20281 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 10:42:36.407938 20281 benchmark.hpp:150] waiting for threads terminated
I0404 10:42:36.409734 20281 benchmark.hpp:174] writing report to <STDOUT>
FPS=154.933
E2E_MEAN=6453.89
DPU_MEAN=5760.65
```

**Figure 11: Classification Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/classification` directory in the SDK.

## Notes

Classification supports the following types of networks. Its enumeration definition is shown below. Refer to `/usr/include/xilinx/classification/classification.hpp` for more details.

```
/**
 * @brief Classification Network Type
 */
enum Type {
    /// Resnet50 neural network, input size is 224x224
    RESNET_50,
    /// Inception_v1 neural network, input size is 224x224
    INCEPTION_V1,
    /// Inception_v2 neural network, input size is 224x224
    INCEPTION_V2,
    /// Inception_v3 neural network, input size is 299x299
    INCEPTION_V3,
    /// Inception_v4 neural network, input size is 299x299
    INCEPTION_V4,
    /// Mobilenet_v2 neural network, input size is 224x224
    MOBILENET_V2,
    /// Resnet50 Tensorflow model, input size is 224x224
    RESNET_50_TF,
    /// Inception_v1 Tensorflow model, input size is 224x224
```

```
INCEPTION_V1_TF,
/// Mobilenet_v2 Tensorflow model, input size is 224x224
MOBILENET_V2_TF,

NUM_OF_TYPE

};
```

## Face Detection

### Introduction

This library uses DenseBox neuron network to detect human faces. Input is a picture with some faces requiring detection, whereas output is a vector of the result struct containing each box's information.

The following image shows the result of face detection:



**Figure 12: 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.

For image, the calling method is as follows:

```
#include <xilinx/facedetect/facedetect.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
```

```
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::facedetect::FaceDetect::create(xilinx::facedetect::DENSE_BOX_320x320
); }, process_result);
}
```

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

- test\_jpeg\_facedetect\_dense\_box\_320x320
- test\_jpeg\_facedetect\_dense\_box\_640x360

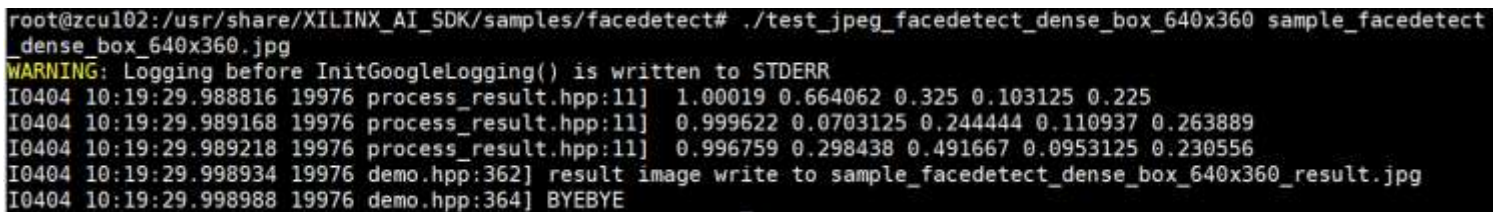
2. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

3. Run the example.

```
#!/test_jpeg_facedetect_dense_box_640x360
sample_facedetect_dense_box_640x360.jpg
```

4. You will see the print result on the terminal (see Figure 13 below as an example). Also, you can view the output image: sample\_facedetect\_dense\_box\_640x360\_result.jpg, such as Figure 12.



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_jpeg_facedetect_dense_box_640x360 sample_facedetect_dense_box_640x360.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:19:29.988816 19976 process_result.hpp:11] 1.00019 0.664062 0.325 0.103125 0.225
I0404 10:19:29.989168 19976 process_result.hpp:11] 0.999622 0.0703125 0.244444 0.110937 0.263889
I0404 10:19:29.989218 19976 process_result.hpp:11] 0.996759 0.298438 0.491667 0.0953125 0.230556
I0404 10:19:29.998934 19976 demo.hpp:362] result image write to sample_facedetect_dense_box_640x360_result.jpg
I0404 10:19:29.998988 19976 demo.hpp:364] BYEBYE
```

Figure 13: Face Detection Running Result

For video, the calling method is as follows:

```
#include <xilinx/facedetect/facedetect.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
```

```
using namespace std;
```

```
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::facedetect::FaceDetect::create(xilinx::facedetect::DENSE_BOX_320x320
); }, process_result);
}
```

5. Run the video example.

```
#./test_video_facedetect_dense_box_640x360 video_input.mp4 -t 8
```

6. If you want to test model performance, you can refer to the following call example. By default, we prepared a set of images for model performance testing, which are stored in the ./images directory.

```
#include <xilinx/facedetect/facedetect.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::classification::Classification
                ::create(xilinx::classification::INCEPTION_V1);
        }
    });
}
```

7. Before running the face detection performance example, choose one of the following executable programs to run:

- test\_performance\_facedetect\_dense\_box\_320x320
- test\_performance\_facedetect\_dense\_box\_640x360

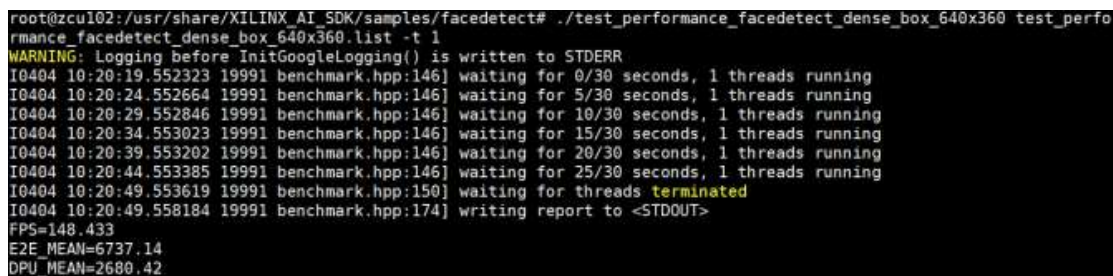
8. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

9. Run the example.

```
#./test_performance_facedetect_dense_box_640x360
test_performance_facedetect_dense_box_640x360.list -t 1
```

10. You will see the print result on the terminal. See Figure 14 for reference..



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/facedetect# ./test_performance_facedetect_dense_box_640x360 test_perfo
rmance_facedetect_dense_box_640x360.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:20:19.552323 19991 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 10:20:24.552664 19991 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 10:20:29.552846 19991 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 10:20:34.553023 19991 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 10:20:39.553202 19991 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 10:20:44.553385 19991 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 10:20:49.553619 19991 benchmark.hpp:150] waiting for threads terminated
I0404 10:20:49.558184 19991 benchmark.hpp:174] writing report to <STDOUT>
FPS=148.433
E2E_MEAN=6737.14
DPU_MEAN=2680.42
```

Figure 14: Face Detection Performance Test Result

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/facedetect` directory in the SDK.

## Notes

Face detection supports the following two resolutions as input. Its enumeration definition is shown below. Refer to `/usr/include/xilinx/facedetect/facedetect.hpp` for more details.

```
/**
 * @brief Facedetect Network Type
 */
enum Type {
    /// Input size is 320x320
    DENSE_BOX_320x320,
    /// Input size is 640x360
    DENSE_BOX_640x360
};
```

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

---

## SSD Detection

### Introduction

This library is commonly used in SSD neuron networks, which are used to detect objects. Input is a picture with some objects in need of detection, whereas output is a vector of the result struct containing each box's information.

The following image shows the result of SSD detection:



Figure 15: SSD Detection Example

## Sample

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

For images, the calling method is as follows:

```
#include <xilinx/ssd/ssd.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::ssd::SSD::create(xilinx::ssd::ADAS_VEHICLE_V3_480x360); },
        process_result);
}
```

1. Before running the SSD detection example, 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\_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 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 15 for reference).

For video, the calling method is as follows:

```
#include <xilinx/ssd/ssd.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include <./process_result.hpp>
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::ssd::SSD::create(xilinx::ssd::ADAS_VEHICLE_V3_480x360); },
        process_result);
}
```

##### 5. Run the video example.

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

##### 6. If you want to test model performance, you can refer to the following call example. By default, a set of images for model performance testing have been prepared and are stored in the ./images directory.

```
#include <xilinx/ssd/ssd.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::ssd::SSD
::create(xilinx::ssd::ADAS_VEHICLE_V3_480x360);
        }
    });
}
```

##### 7. Before running the SSD detection performance example, 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\_480x360
- test\_performance\_ssd\_mobilenet\_v2\_480x360
- test\_performance\_ssd\_traffic\_480x360
- test\_performance\_ssd\_voc\_300x300\_tf

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

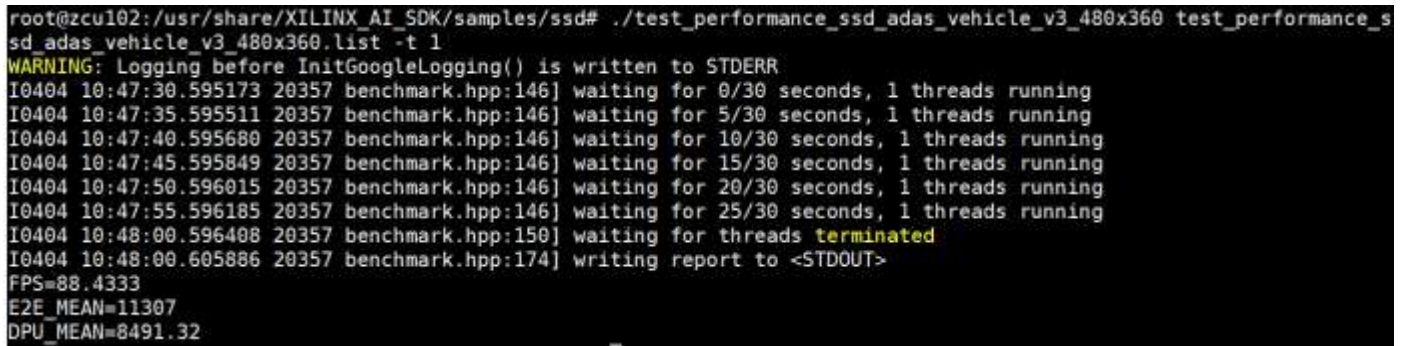
```
#sh build.sh
```



## 9. Run the example.

```
#./test_performance_ssd_adas_vehicle_v3_480x360
test_performance_ssd_adas_vehicle_v3_480x360.list -t 1
```

## 10. You will see the printing result on the terminal (see Figure 16 for reference).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/ssd# ./test_performance_ssd_adas_vehicle_v3_480x360 test_performance_ssd_adas_vehicle_v3_480x360.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 10:47:30.595173 20357 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 10:47:35.595511 20357 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 10:47:40.595680 20357 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 10:47:45.595849 20357 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 10:47:50.596015 20357 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 10:47:55.596185 20357 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 10:48:00.596408 20357 benchmark.hpp:150] waiting for threads terminated
I0404 10:48:00.605886 20357 benchmark.hpp:174] writing report to <STDOUT>
FPS=88.4333
E2E_MEAN=11307
DPU_MEAN=8491.32
```

**Figure 16: SSD Detection Performance Test Result**

For more details about the sample, refer to the `/usr/share/XILINX_AI_SDK/samples/ssd` directory in the SDK.

## Notes

SSD detection supports the following types of networks. Its enumeration definition is shown below. Refer to `/usr/include/xilinx/ssd/ssd.hpp` for more details.

```
enum Type {
    /// Adas vehicle model, input size is 480x360.
    ADAS_VEHICLE_V3_480x360,
    /// Video structurization model, input size is 480x360, it can fit 117g,
    /// 11.6g, 5.5g. This is 11.6g.
    TRAFFIC_480x360,
    /// Pedestrian detect, OPS is 5.9g. input size is 640x360, sight is adas.
    ADAS_PEDESTRIAN_640x360,
    /// Mobilenet ssd detection, 7 classes
    MOBILENET_480x360,
    /// Mobilenet_v2 ssd detection, 11 classes
    MOBILENET_V2_480x360,

    /// tensorflow ssd model:
    VOC_300x300_TF,

    NUM_OF_TYPE
};
```

## Pose Detection

### Introduction

This library is used to detect postures of the human body. This library includes a neural network which can mark 14 key points of human bodies; you can use our SSD detection library. Input will be a picture detected using the pedestrian detection neural network. Output is a struct containing each point's coordinates.

The following image shows the result of pose detection:



**Figure 17: Pose Detection Example**

### Sample

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

For images, the calling method is as follows:

```
#include <xilinx/posedetect/posedetect.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
#include "ssd_pose_detect.hpp"
using namespace std;
cv::Mat process_result_ssd(cv::Mat &image,
                          const std::vector &results,
                          bool is_jpeg) {
    for (auto& result : results) {
        process_result(image, result, is_jpeg);
    }
    return image;
}
```

```
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::posedetect::SSDPoseDetect::create(); },
        process_result_ssd);
}
```

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

- test\_jpeg\_posedetect\_with\_ssd
- test\_jpeg\_posedetect

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

```
#sh build.sh
```

3. Run the example.

```
#./test_jpeg_posedetect_with_ssd sample_posedetect_with_ssd.jpg
```

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

sample\_posedetect\_with\_ssd\_result.jpg (see Figure 17 for reference).

For video, the calling method is as follows:

```
#include <xilinx/posedetect/posedetect.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include " ./process_result.hpp"
#include " ./ssd_pose_detect.hpp"
using namespace std;
cv::Mat process_result_ssd(cv::Mat &image,
    const std::vector &results,
    bool is_jpeg) {
    for (auto& result : results) {
        process_result(image, result, is_jpeg);
    }
    return image;
}

using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] { return
xilinx::posedetect::SSDPoseDetect::create(); },
        process_result_ssd);
}
```

5. Run the video example.

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

6. If you want to test model performance, refer to the following call example. By default, a set of images has been prepared for model performance testing and are stored in the `./images` directory.

```
#include <xilinx/benchmark/benchmark.hpp>
#include "../ssd_pose_detect.hpp"
#include "../process_result.hpp"
using namespace std;
cv::Mat process_result_ssd(
    cv::Mat &image,
    const std::vector &results,
    bool is_jpeg) {
    for (auto& result : results) {
        process_result(image, result, is_jpeg);
    }
    return image;
}

int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        { return xilinx::posedetect::SSDPoseDetect::create(); }
    });
}
```

7. Before running the pose detection performance example, choose one of the following executable program to run:

- `test_performance_posedetect_with_ssd`
- `test_performance_posedetect`

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

```
#sh build.sh
```

9. Run the example.

```
#./test_performance_posedetect_with_ssd test_performance_posedetect.list
```

10. You will see the print result on the terminal (see Figure 18 for reference).

```

root@zcu102:/usr/share/XILINX_AI_SDK/samples/posedetect# ./test_performance_posedetect test_performance_posedetect.lis
t -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 15:01:41.905664 31025 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 15:01:46.906061 31025 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 15:01:51.906244 31025 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 15:01:56.906416 31025 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 15:02:01.906587 31025 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 15:02:06.906765 31025 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 15:02:11.907001 31025 benchmark.hpp:150] waiting for threads terminated
I0404 15:02:11.907924 31025 benchmark.hpp:174] writing report to <STDOUT>
FPS=406
E2E_MEAN=2462.24
DPU_MEAN=2103.95

```

**Figure 18: Pose Detection Performance Test Result**

For more details about the sample, refer to the `/usr/share/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, perform the SSD detection first. Refer to the following: `test_jpeg_posedetect_with_ssd.cpp`.

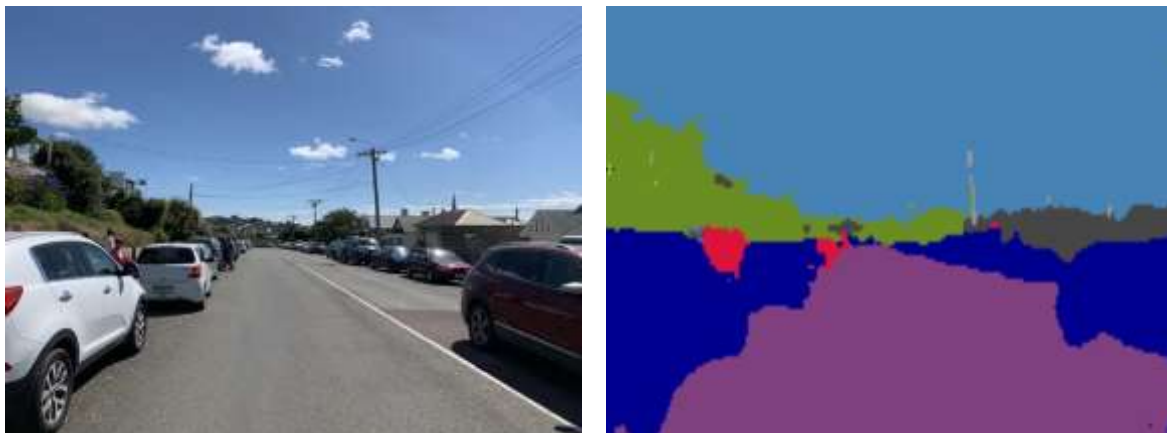
If the input image is a picture of a person who has been cut out, only the pose detection can be completed. Refer to the following: `test_jpeg_posedetect.cpp`.

# Semantic Segmentation

## Introduction

The semantic segmentation of image is to assign a semantic category to each pixel in the input image; this will allow you to obtain the pixelated intensive classification. Library segmentation can be used in an ADAS field. It offers simple interfaces for developers to deploy segmentation tasks on Xilinx FPGAs.

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



**Figure 19: Semantic Segmentation Example**

## Sample

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

For image, the calling method is as follows:

```
#include <xilinx/segmentation/segmentation.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;

int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::segmentation::Segmentation8UC3::create(Xilinx::segmentation::FPN); },
        process_result);
}
```

1. Before running the semantic segmentation example, choose one of the following executable program 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 build.sh
```

3. Run the example.

```
#./test_jpeg_segmentation_fpn_8UC3 sample_segmentation.jpg
```

4. You will see the print result on the terminal. Also you can view the output image at sample\_segmentation\_result.jpg (see Figure 19 for reference).

For video, the calling method is as follows:

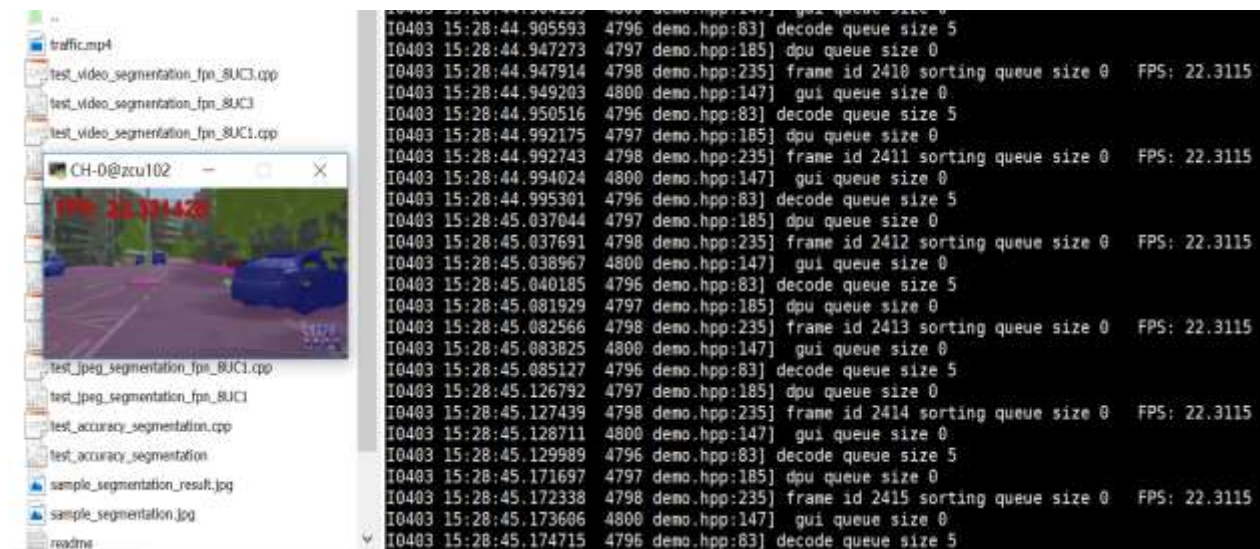
```
#include <xilinx/segmentation/segmentation.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
```

```
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::segmentation::Segmentation8UC3::create(xilinx::segmentation::FPN); },
        process_result);
}
```

5. Run the video example.

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

Figure 20, seen below, shows the video semantic segmentation running with one thread's result.



**Figure 20: Video Semantic Segmentation Running Result**

6. If you want to test model performance, refer to the following call example. A set of default images for model performance testing has been prepared and are stored in the `./images` directory.

```
#include <xilinx/segmentation/segmentation.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::segmentation::Segmentation8UC3
                ::create(xilinx::segmentation::FPN);
        }
    });
}
```



7. Before running the semantic segmentation performance example, choose one of the following executable programs to run:

- `test_performance_segmentation_fpn_8UC3`
- `test_performance_segmentation_fpn_8UC1`

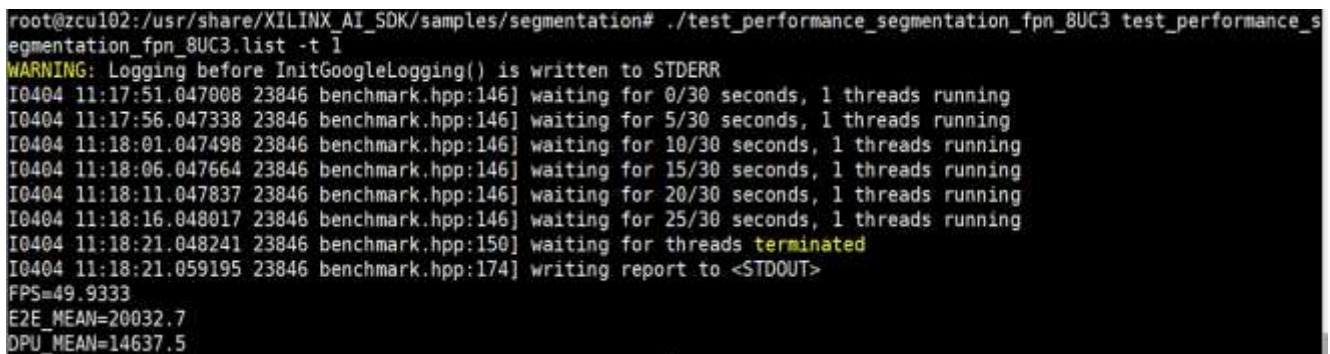
8. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

9. Run the example.

```
#!/test_performance_segmentation_fpn_8UC3
test_performance_segmentation_fpn_8UC3.list -t 1
```

10. You will see the print result on the terminal (see Figure 21).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/segmentation# ./test_performance_segmentation_fpn_8UC3 test_performance_s
egmentation_fpn_8UC3.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 11:17:51.047008 23846 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 11:17:56.047338 23846 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 11:18:01.047498 23846 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 11:18:06.047664 23846 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 11:18:11.047837 23846 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 11:18:16.048017 23846 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 11:18:21.048241 23846 benchmark.hpp:150] waiting for threads terminated
I0404 11:18:21.059195 23846 benchmark.hpp:174] writing report to <STDOUT>
FPS=49.9333
E2E_MEAN=20032.7
DPU_MEAN=14637.5
```

**Figure 21: Semantic Segmentation Performance Test Result**

For more details about the sample, refer to `/usr/share/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 the result.

The `test_jpeg_segmentation_fpn_8UC3` output is a color-rendering display, while `test_jpeg_segmentation_fpn_8UC1` output is a grayscale image.



## Road Line Detection

### Introduction

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

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

The following image shows the result of road line detection:



Figure 22: Road Line Detection Example

### Sample

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

For images, the calling method is as follows:

```
#include <xilinx/roadline/roadline.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
```

```
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;

int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return xilinx::roadline::RoadLine::create(); },
        process_result);
}
```

1. Before running the road line detection example, make sure the following executable program exists:

- test\_jpeg\_roadline

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

```
#sh build.sh
```

3. Run the example.

```
#./test_jpeg_roadline sample_roadline.jpg
```

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

sample\_roadline\_result.jpg (see Figure 22 for reference).

For video, the calling method is as follows:

```
#include <xilinx/roadline/roadline.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return xilinx::roadline::RoadLine::create(); },
        process_result);
}
```

5. Run the video example.

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

6. To test model performance, refer to the call example on the following page. A default set of images for model performance testing have been prepared and are stored in the ./images directory.

```
#include <xilinx/roadline/roadline.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::roadline ::RoadLine
                ::create();
        }
    });
}
```

- Before running the road line detection performance example, make sure the following executable program exists:

```
test_performance_roadline
```

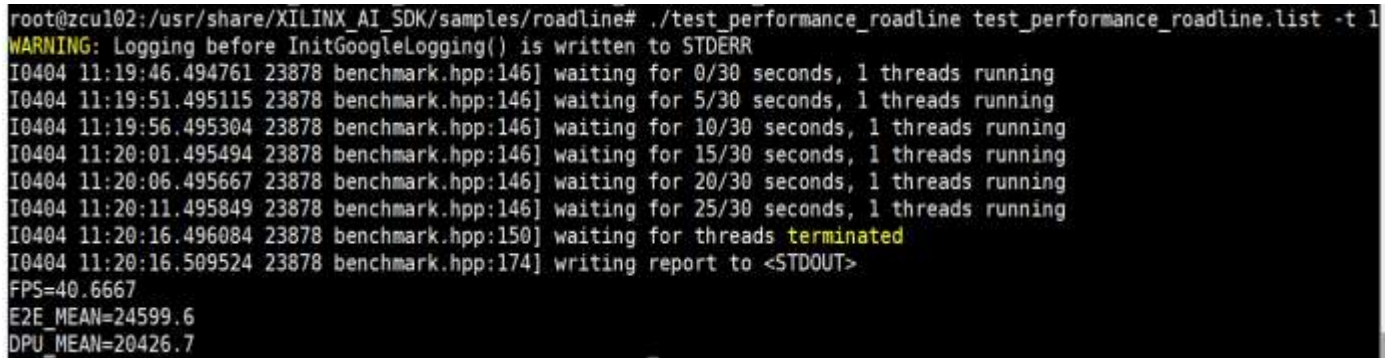
- If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

- Run the example.

```
#./test_performance_roadline test_performance_roadline.list -t 1
```

- You will see the print result on the terminal (see Figure 23 for reference).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/roadline# ./test_performance_roadline test_performance_roadline.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 11:19:46.494761 23878 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 11:19:51.495115 23878 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 11:19:56.495304 23878 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 11:20:01.495494 23878 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 11:20:06.495667 23878 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 11:20:11.495849 23878 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 11:20:16.496084 23878 benchmark.hpp:150] waiting for threads terminated
I0404 11:20:16.509524 23878 benchmark.hpp:174] writing report to <STDOUT>
FPS=40.6667
E2E_MEAN=24599.6
DPU_MEAN=20426.7
```

**Figure 23: Road Line Detection Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/roadline` directory in the SDK.

## Notes

The input of the image is fixed at 640 x 480, so all images of other sizes need to be resized.

## YOLOV3 Detection

### Introduction

This library is in common use in the YOLO neuron network, which is used to detect objects and is currently in its third version. Input will be a picture with one or more objects, whereas output will be a vector of the result struct, which is composed of the detected information.

The following image shows the result of YOLOv3 detection:

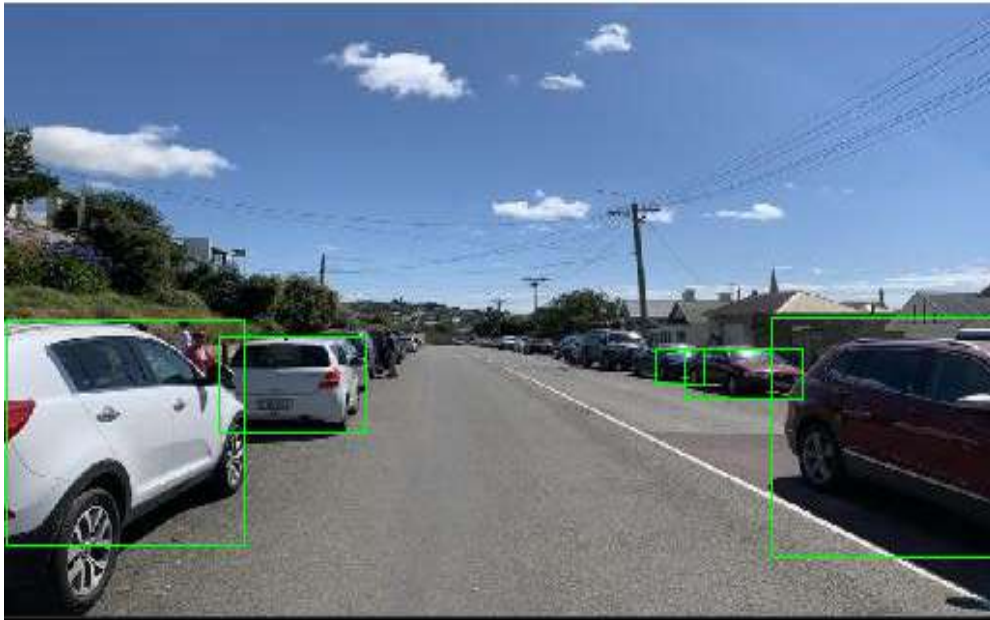


Figure 24: YOLOv3 Detection Example

### Sample

The Xilinx AI SDK provides YOLOv3 detection samples based on an image or a video. A sample performance test of YOLOv3 detection is also provided.

For images, the calling method is as follows:

```
#include <xilinx/yolov3/yolov3.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "../process_result.hpp"
using namespace std;
```

```
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::yolov3::YOLOv3::create(Xilinx::yolov3::ADAS_512x256); },
        process_result);
}
```

1. Before running the YOLOv3 detection example, 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 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 24 for reference).

For video, the calling method is as follows:

```
#include <xilinx/yolov3/yolov3.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::yolov3::YOLOv3::create(xilinx::yolov3::ADAS_512x256); },
        process_result);
}
```

5. Run the video example.

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

6. If you want to test model performance, you can refer to the following call example. A set of default images for model performance testing has been prepared and is stored in the `./images` directory.

```
#include <xilinx/yolov3/yolov3.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::yolov3 ::YOLOv3
                ::create(xilinx::yolov3 ::ADAS_512x256);
        }
    });
}
```

7. Before running the YOLOv3 detection performance example, 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`

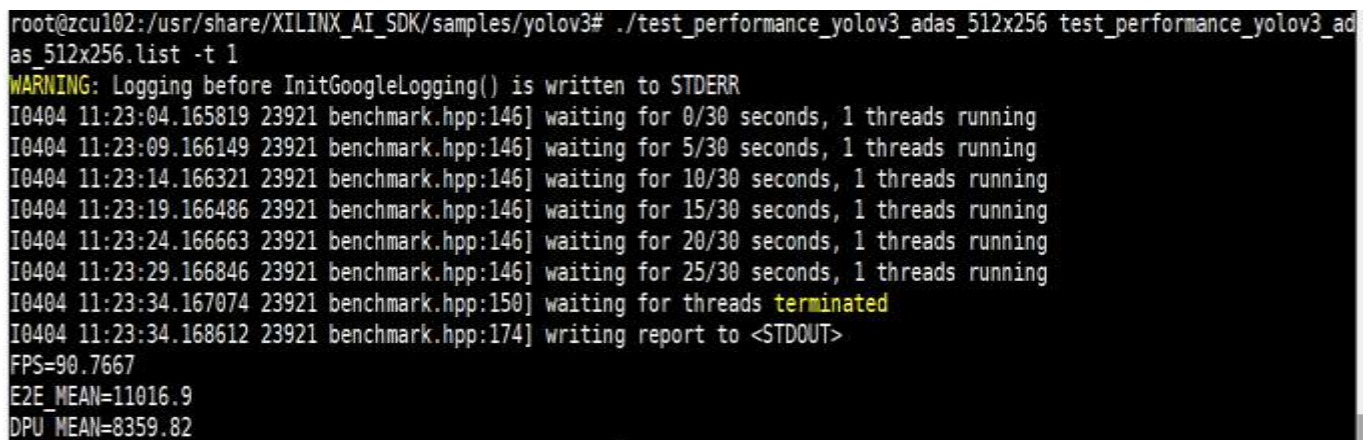
8. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

9. Run the example.

```
#./test_performance_yolov3_adas_512x256
test_performance_yolov3_adas_512x256.list -t 1
```

10. You will see the printing result on the terminal (see Figure 25).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/yolov3# ./test_performance_yolov3_adas_512x256 test_performance_yolov3_adas_512x256.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 11:23:04.165819 23921 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 11:23:09.166149 23921 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 11:23:14.166321 23921 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 11:23:19.166486 23921 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 11:23:24.166663 23921 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 11:23:29.166846 23921 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 11:23:34.167074 23921 benchmark.hpp:150] waiting for threads terminated
I0404 11:23:34.168612 23921 benchmark.hpp:174] writing report to <STDOUT>
FPS=90.7667
E2E_MEAN=11016.9
DPU_MEAN=8359.82
```

**Figure 25: YOLOv3 Detection Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/yolov3` directory in the SDK.

## Notes

YOLOv3 supports the types of models included below, where its enumeration definition is shown as well. Refer to `/usr/include/xilinx/yolov3/yolov3.hpp` for more details.

```
/**
 * @brief YOLOV3 Network Type
 */
enum Type {
    /// VOC detect model, input size is 416x416
    VOC_416x416,
    /// ADAS vehicle detect model, 4 classes, input size is 512x256
    ADAS_512x256,
    /// ADAS vehicle detect model, 10 classes, no pruned.
    ADAS_512x288,
    /// VOC detect model, train by tensorflow, input size is 416x416
    VOC_416x416_TF,

    NUM_OF_TYPE
};
```

---

## YOLOv2 Detection

### Introduction

YOLOv2 performs the same task as YOLOv3, which is an upgraded version of YOLOv2.

### Sample

1. Before running the YOLOv2 detection example, choose one of the following executable programs to run:

- `test_jpeg_yolov2_baseline`
- `test_jpeg_yolov2_compress22g`
- `test_jpeg_yolov2_compress24g`
- `test_jpeg_yolov2_compress26g`

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

```
#sh build.sh
```

5. Run the image example.

```
#./test_jpeg_yolov2_baseline sample_yolov2_voc.jpg
```

6. You will see the print result on the terminal. Also, you can view the output image: `sample_yolov2_voc_result.jpg` (see Figure 26).





**Figure 26: YOLOv2 Detection Example**

7. Run the video example.

```
#./test_video_yolov2_baseline video_input.mp4 -t 8
```

8. To run the performance example, choose one of the following executable programs:

- test\_performance\_yolov2\_baseline
- test\_performance\_yolov2\_compress22g
- test\_performance\_yolov2\_compress24g
- test\_performance\_yolov2\_compress26g

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

```
#sh build.sh
```

10. Run the performance example.

```
#./test_performance_yolov2_baseline test_performance_yolov2_baseline.list -t 8
```

11. You will see the printing result on the terminal (see Figure 27).



```

root@zcu102:/usr/share/XILINX_AI_SDK/samples/yolov2# ./test_performance_yolov2_baseline test_performance_yolov2_baseline.list -t
8
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0531 12:42:14.417320 30964 benchmark.hpp:198] writing report to <STDOUT>
I0531 12:42:14.437685 30964 benchmark.hpp:219] waiting for 0/30 seconds, 8 threads running
I0531 12:42:24.437873 30964 benchmark.hpp:219] waiting for 10/30 seconds, 8 threads running
I0531 12:42:34.438081 30964 benchmark.hpp:219] waiting for 20/30 seconds, 8 threads running
I0531 12:42:44.438347 30964 benchmark.hpp:227] waiting for threads terminated
FPS=81.2667
E2E_MEAN=98623.9
DPU_MEAN=92847.1

```

**Figure 27: YOLOv2 Detection Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/yolov2` directory in the SDK.

## Notes

YOLOv2 supports the types of models listed below; its enumeration definition is also shown. Refer to `/usr/include/xilinx/yolov2/yolov2.hpp` for more details.

- BASELINE
- COMPRESS22G
- COMPRESS24G
- COMPRESS26G

---

## Openpose Detection

### Introduction

The library is used to draw a given person's posture. It is represented by the line between a first point and a second point, stored as pairs. Every pair of points represents a connection, which results in sets of pairs that are then stored as vectors.

The image on the following page shows the result of Openpose detection.



**Figure 28: Openpose Detection Example**

## Sample

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

For images, the calling method is as follows:

```
#include <xilinx/openpose/openpose.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;

int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return xilinx::openpose::OpenPose::create(); },
        process_result);
}
```

1. Before running the Openpose detection example, make sure the following executable program exists:

```
test_jpeg_openpose
```

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

```
#sh build.sh
```

3. Run the example.

```
#./test_jpeg_openpose 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 28 for reference).

For video, the calling method is as follows:

```
#include <xilinx/openpose/openpose.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "./process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return xilinx::openpose::OpenPose::create(); },
        process_result);
}
```

5. Run the video example.

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

6. To test model performance, refer to the following call example. A set of default images for model performance testing are prepared and stored in the ./image directory.

```
#include <xilinx/openpose/openpose.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::openpose ::OpenPose
                ::create();
        }
    });
}
```

7. Before running the Openpose detection performance example, make sure the following executable program exists:

```
test_performance_openpose
```

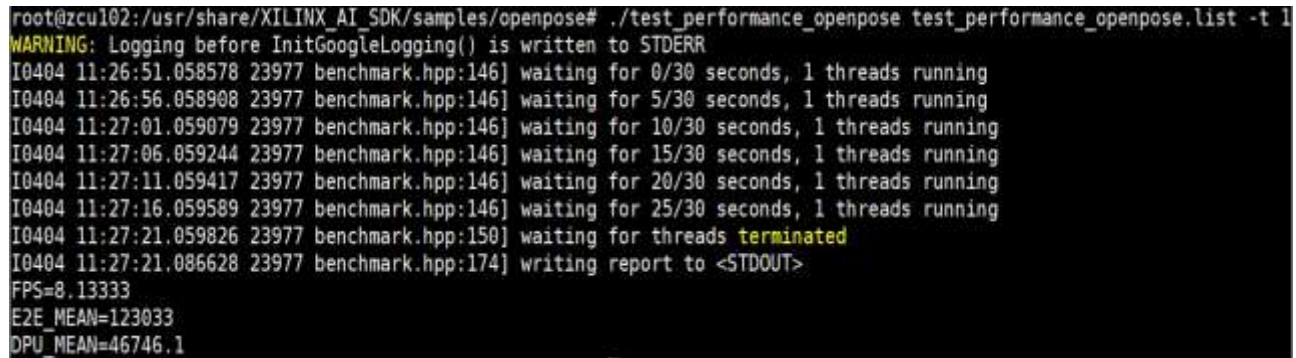
8. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

9. Run the example.

```
#./test_performance_openpose test_performance_openpose.list -t 1
```

10. You will see the print result on the terminal (see Figure 29 for reference).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/openpose# ./test_performance_openpose test_performance_openpose.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 11:26:51.058578 23977 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 11:26:56.058908 23977 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 11:27:01.059079 23977 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 11:27:06.059244 23977 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 11:27:11.059417 23977 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 11:27:16.059589 23977 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 11:27:21.059826 23977 benchmark.hpp:150] waiting for threads terminated
I0404 11:27:21.086628 23977 benchmark.hpp:174] writing report to <STDOUT>
FPS=8.13333
E2E_MEAN=123033
DPU_MEAN=46746.1
```

**Figure 29: Openpose Detection Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/openpose` directory in the SDK.

## Notes

The input of the image is fixed at 368 x 368, so all images of other sizes need to be resized.

# RefineDet Detection

## Introduction

This library is in common use within the RefineDet neuron network, which is used to detect human bodies. Input will be a picture with some individuals to detect, whereas output is a vector of the result struct containing each box's information.

The image on the following page shows the result of RefineDet detection:



**Figure 30: RefineDet Detection Example**

## Sample

The Xilinx AI SDK provides RefineDet detection samples based on either images or videos. A sample performance test of RefineDet detection is included in the kit.

For images, the calling method is as follows:

```
#include <xilinx/refinedet/refinedet.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;

int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_jpeg_demo(
        argc, argv, [] { return
xilinx::refinedet::RefineDet::create(Xilinx::refinedet::REFINEDET_480x360);
},
        process_result);
}
```

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

- `test_jpeg_refinedet_refinedet_480x360`
- `test_jpeg_refinedet_refinedet_480x360_5g`
- `test_jpeg_refinedet_refinedet_480x360_10g`

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

```
#sh build.sh
```

2. Run the example.

```
#./test_jpeg_refinedet_refinedet_480x360 sample_refinedet.jpg
```

3. You will see the printing result on the terminal. Also, you can view the output image: `sample_refinedet_result.jpg` (see Figure 30 for reference).

For video, the calling method is as follows:

```
#include <xilinx/refinedet/refinedet.hpp>
#include <iostream>
#include <memory>
#include <glog/logging.h>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xilinx/demo/demo.hpp>
#include "process_result.hpp"
using namespace std;
int main(int argc, char *argv[]) {
    return xilinx::demo::main_for_video_demo(
        argc, argv, [] {
            return
xilinx::refinedet::RefineDet::create(xilinx::refinedet::REFINEDET_480x360);
        }, process_result);
}
```

4. Run the video example.

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

5. To test model performance, refer to the call example on the following page. By default, a set of default images for model performance testing has been prepared and stored in the `./images` directory.



```
#include <xilinx/refinedet/refinedet.hpp>
#include <xilinx/benchmark/benchmark.hpp>
int main(int argc, char *argv[]) {
    return xilinx::benchmark::main_for_performance(argc, argv, [] {
        {
            return xilinx::refinedet ::Refinedet
                ::create(xilinx::refinedet ::REFINEDET_480x360);
        }
    });
}
```

6. Before running the Refinedet detection performance example, you choose one of the following executable program to run:

- test\_performance\_refinedet\_refinedet\_480x360
- test\_performance\_refinedet\_refinedet\_480x360\_5g
- test\_performance\_refinedet\_refinedet\_480x360\_10g

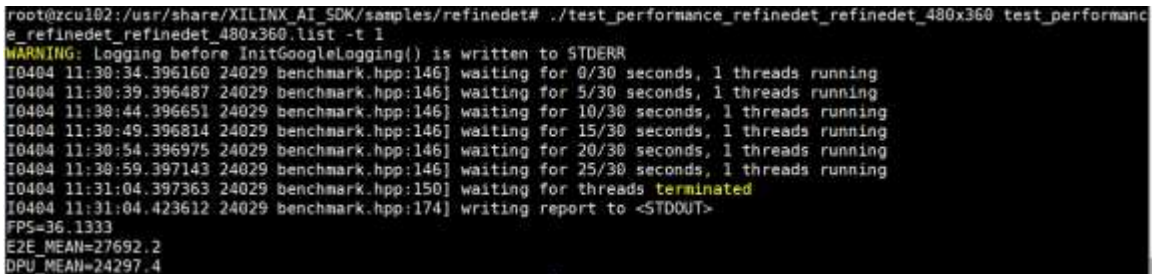
7. If the executable program does not exist, it can be compiled and generated as follows:

```
#sh build.sh
```

8. Run the example.

```
#./test_performance_refinedet_refinedet_480x360
test_performance_refinedet_refinedet_480x360.list -t 1
```

9. You will see the print result on the terminal (see Figure 31 for reference).



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/refinedet# ./test_performance_refinedet_refinedet_480x360 test_performanc
e_refinedet_refinedet_480x360.list -t 1
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0404 11:30:34.396160 24029 benchmark.hpp:146] waiting for 0/30 seconds, 1 threads running
I0404 11:30:39.396487 24029 benchmark.hpp:146] waiting for 5/30 seconds, 1 threads running
I0404 11:30:44.396651 24029 benchmark.hpp:146] waiting for 10/30 seconds, 1 threads running
I0404 11:30:49.396814 24029 benchmark.hpp:146] waiting for 15/30 seconds, 1 threads running
I0404 11:30:54.396975 24029 benchmark.hpp:146] waiting for 20/30 seconds, 1 threads running
I0404 11:30:59.397143 24029 benchmark.hpp:146] waiting for 25/30 seconds, 1 threads running
I0404 11:31:04.397363 24029 benchmark.hpp:150] waiting for threads terminated
I0404 11:31:04.423612 24029 benchmark.hpp:174] writing report to <STDOUT>
FPS=36.1333
E2E_MEAN=27692.2
DPU_MEAN=24297.4
```

**Figure 31: Refinedet Detection Performance Test Result**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/refinedet` directory in the SDK.

## Notes

Refinedet detection supports the types of models detailed on the following page and its enumeration definition is also shown. Refer to `/usr/include/xilinx/refinedet/refinedet.hpp` for more details.

```
/**
 * @brief RefineDet Network Type
 */
enum Type {
    /// Input size is 480x360
    REFINEDET_480x360,
    /// Input size is 480x360, OPS is 10g
    REFINEDET_480x360_10G,
    /// Input size is 480x360, OPS is 5g
    REFINEDET_480x360_5G
};
```

---

## DNNDK Sample

### Introduction

The sample will show how to run a neural network by directly calling DNNDK. For details about the DNNDK APIs, refer to the DNNDK User Guide at

[https://www.xilinx.com/support/documentation/user\\_guides/ug1327-dnndk-user-guide.pdf](https://www.xilinx.com/support/documentation/user_guides/ug1327-dnndk-user-guide.pdf).

To develop deep-learning applications on the DPU, four types of work must be done:

1. Train a neural network and compile a dpumodel file.  
Refer to Chapter 6 for the method to compile the dpumodel file.
2. Use DNNDK APIs to manage DPU kernels.  
DPU kernel management requires the following elements:
  - DPU kernel creation and destruction
  - DPU task creation
  - Managing input and output tensors
3. Implement kernels not supported by the DPU on the CPU.
4. Add pre-processing and post-processing routines to read in data or calculate results.

### Sample

An example involving the Resnet-50 neural network is provided below.

```
#include <opencv2/opencv.hpp>
#include <dnndk/dnndk.h>
using namespace cv;
using namespace std;
#define KERNEL_CONV "resnet_50"
#define CONV_INPUT_NODE "conv1"
#define CONV_OUTPUT_NODE "fc1000"
int main(int argc, char** argv) {
    cv::Mat image = imread(argv[1]);
```



```

    setenv("DPU_COMPILATIONMODE","1",1);
DPUKernel *kernelConv;
//Open the DPU device
    dpuOpen();
    // Create the kernel for Resnet_50
    kernelConv = dpuLoadKernel(KERNEL_CONV);
    // Create DPU Tasks for Resnet_50 from DPU Kernel.
    DPUTask *taskconv = dpuCreateTask(kernelConv, 0);
    // Run conv kernels for ResNet-50
    run_resnet_50(taskconv, image);
    // Destroy DPU Tasks & free resources
    dpuDestroyTask(taskconv);
    // Destroy the kernel of Resnet_50 after classification
    dpuDestroyKernel(kernelConv);
// Close DPU
    dpuClose();
return 0;
}

```

The main operations perform the following tasks:

- Fetches an image using the OpenCV function `imread()`
- Sets the environment `DPU_COMPILATIONMODE = 1` to use dpu model file
- Calls `dpuOpen()` to open the DPU device
- Calls `dpuLoadKernel()` to load the DPU kernel Resnet-50
- Calls `dpuCreateTask()` to create tasks for each DPU kernel
- Calls to implement kernels not supported by the DPU on the CPU, as well as pre-processing and post-processing routines to read in data or calculate results
- Calls `dpuDestroyKernel()` and `dpuDestroyTask()` to destroy DPU kernels and tasks
- Calls `dpuClose()` to close the DPU device

```

void run_resnet_50(DPUTask *taskConv, const Mat &img) {
    assert(taskConv );
    // Get the number of category in Resnet_50
    int channel = dpuGetOutputTensorChannel(taskConv, CONV_OUTPUT_NODE);
    // Get the scale of classification result
    float scale = dpuGetOutputTensorScale(taskConv, CONV_OUTPUT_NODE);
    vector<float> smRes (channel);
    int8_t* fcRes;
    // Set the input image to the DPU task
    dpuSetInputImage2(taskConv, CONV_INPUT_NODE, img);
    // Processing the classification in DPU
    dpuRunTask(taskConv);
    // Get the output tensor from DPU in DPU INT8 format
    DPUTensor* dpuOutTensorInt8 = dpuGetOutputTensorInHWCInt8(taskConv,
CONV_OUTPUT_NODE);
    // Get the data pointer from the output tensor
    fcRes = dpuGetTensorAddress(dpuOutTensorInt8);
    // Processing softmax in DPU with batchsize=1

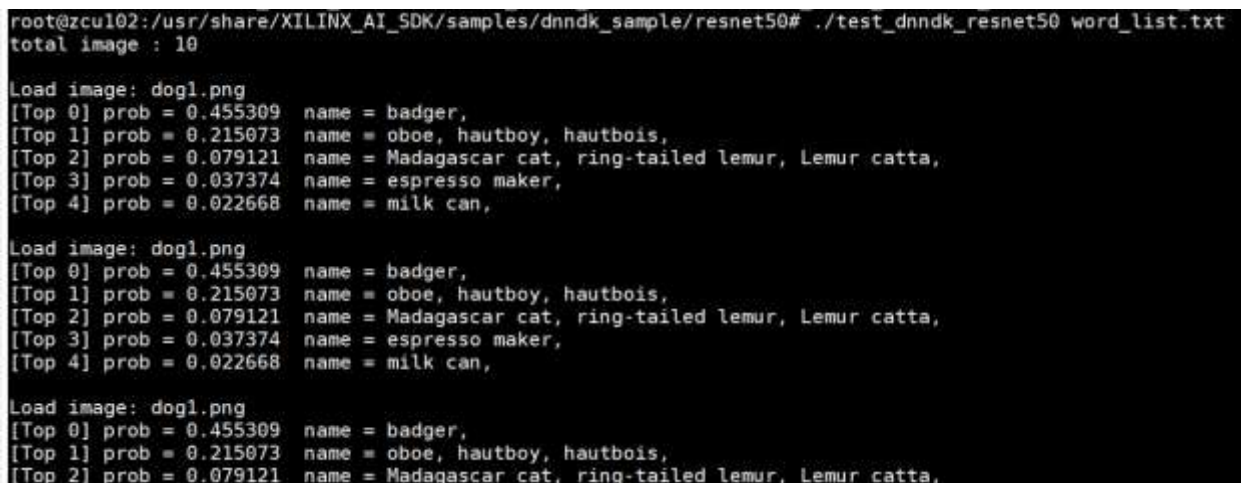
```

```
dpuRunSoftmax(fcRes, smRes.data(), channel, 1, scale);
// Show the top 5 classification results with their label and probability
TopK(smRes.data(), channel, 5);
}
```

The `run_resnet_50()` operators perform the following tasks:

- Sets the image as the input to the DPU kernel `resnet_50` by calling the `dpuSetInputImage2()` API
- Calls `dpuRunTask()` to run the `taskConv` convolution operation in the Resnet-50 network model
- Performs softmax on the CPU using the output of the fully-connected operation as input
- Outputs the top-5 classification category and the corresponding probability

The following image shows the result of Resnet-50.



```
root@zcu102:/usr/share/XILINX_AI_SDK/samples/dnndk_sample/resnet50# ./test_dnndk_resnet50 word_list.txt
total image : 10

Load image: dog1.png
[Top 0] prob = 0.455309 name = badger,
[Top 1] prob = 0.215073 name = oboe, hautboy, hautbois,
[Top 2] prob = 0.079121 name = Madagascar cat, ring-tailed lemur, Lemur catta,
[Top 3] prob = 0.037374 name = espresso maker,
[Top 4] prob = 0.022668 name = milk can,

Load image: dog1.png
[Top 0] prob = 0.455309 name = badger,
[Top 1] prob = 0.215073 name = oboe, hautboy, hautbois,
[Top 2] prob = 0.079121 name = Madagascar cat, ring-tailed lemur, Lemur catta,
[Top 3] prob = 0.037374 name = espresso maker,
[Top 4] prob = 0.022668 name = milk can,

Load image: dog1.png
[Top 0] prob = 0.455309 name = badger,
[Top 1] prob = 0.215073 name = oboe, hautboy, hautbois,
[Top 2] prob = 0.079121 name = Madagascar cat, ring-tailed lemur, Lemur catta,
```

**Figure 32: DNNDK Resnet-50 Example**

For more details about the sample, refer to `/usr/share/XILINX_AI_SDK/samples/dnndk_sample` directory in the SDK.

## Notes

1. The `.elf` needs to be renamed to `dpumodelXXX.so` use `g++`. Please refer to Chapter 6 for the correct naming conventions.
2. `setenv("DPU_COMPILATIONMODE","1",1)`

## Chapter 6: Libraries Advanced Application

This chapter describes, in detail, the advanced applications of the YOLOv3 and SSD libraries. With this configuration, users can quickly develop their own applications based on the libraries.

### Introduction

The `libdpyolov3` and `libdpssd` can be used to modify some parameters to fit user demands. Users can compile their models as a ".so" file, fill kernel names, create input/output node names, and insert other parameters into a configuration file to control such models (see Table 1 for specific elements).

### How to Compile

You can compile an "elf" file by "dnnc". Then use the following command to get the "so" file.

```
g++ \
    -nostdlib \
    -fPIC \
    -shared \
    ${DIR}/${MODEL_NAME}/dpu_${MODEL_NAME}*.elf \
    -o \
    ${DIR}/libdpumodel${MODEL_NAME}.so || touch ${DIR}/libdpumodel${MODEL_NAME}.so
```

Users should be aware of the following parameters:

- If compiling on the host side, replace `g++` with `aarch64-linux-gnu-g++`. This is because `g++` is target side compile toolchain
- `${DIR}` is the path which stores the elf
- `${MODEL_NAME}` is the elf name
- Such shared libraries should have a prefix named "libdpumodel" and the postfix ".so"

### How to Fill a Configuration File

The configuration files are all located in `/etc/XILINX_AI_SDK.conf.d`. When compiling the model to a ".so", the configuration file should have the same name as the `${MODEL_NAME}.prototxt`.

When developing on the host side, the configuration files are located in `/home/johnlin/sdk/sysroots/x86_64-petalinux-linux/etc/XILINX_AI_SDK.conf.d`

```

model {
  name: "yolov3_voc_416"
  kernel {
    name: "yolov3_voc_416"
    input: "layer0_conv"
    output: "layer105_conv"
    output: "layer93_conv"
    output: "layer81_conv"
    mean: 0.0
    mean: 0.0
    mean: 0.0
    scale: 0.00390625
    scale: 0.00390625
    scale: 0.00390625
  }
  model_type : YOLOv3
  yolo_v3_param {
    ...
  }
}

```

**Table 1: 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. (Example: inception_v1_0)
	input	This is the input-node name. If the model has more than one input-node, fill them with the same number's lines.
	output	This is the output-node name. If the model has more than one output-node, fill them with the same number's lines.
	mean	Normally there are three lines, each 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.

## 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, which can be modified them as required.

Table 2: 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 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.

## 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 3: 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 <ul style="list-style-type: none"> <li>Note that each category could have a different threshold, but its amount must be equal to <code>num_classes</code>.</li> </ul>
nms_threshold	the threshold of NMS
biases	These parameters are same as the model's. <ul style="list-style-type: none"> <li>Each bias need writes in a separate line. (Biases amount) = <code>anchorCnt * (output-node amount) * 2</code>. Set correct lines in the prototxt.</li> </ul>

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 (which is the first category)
prior_box_param	There is more than one PriorBox, which can be found in the original model (deploy.prototxt) for corresponding each different scale. - These PriorBoxes should oppose each other. - See Table 4 for Prior Box Parameters

**Table 4: 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 actually 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
flip	control whether or not to 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].

## How to Use the Code

The following is the example code.

```
Mat img = cv::imread(argv[1]);
auto yolo = xilinx::yolov3::YOLOv3::create_ex("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);
```

Use the "create\_ex" to create the YOLOv3 object.

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

## Notes

1. The model\_name is same as the prototxt's name.
2. For more details about the sample, refer to  
 /usr/share/XILINX\_AI\_SDK/samples/yolov3/test\_customer\_provided\_model\_yolov3.cpp  
 and /usr/share/XILINX\_AI\_SDK/samples/ssd/test\_customer\_provided\_model\_ssd.cpp.



## *Chapter 7: 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 Xilinx website at:

[https://www.xilinx.com/support/documentation/user\\_guides/ug1355-xilinx-ai-sdk-programming-guide.pdf](https://www.xilinx.com/support/documentation/user_guides/ug1355-xilinx-ai-sdk-programming-guide.pdf)

## Chapter 8: Performance

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

More neural network features are supported by the DPU released in this version, but the power limit of the evaluation boards has not increased. Because of this, the actual running frequency of the DPU is lower than the previous 2.0.8 release. Thus, the performance of some networks is slightly less efficient than that of the previous release, resulting in slower output.

### 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 1 is the performance of ZCU102 (0432055-04) and Table 2 is the performance of ZCU102 (0432055-05). In both boards, triple B4096F DPU cores are implemented in program logic.

Refer to Table 5 for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-04) with DPU running at 294Mhz.

**Table 5: ZCU102 (0432055-04) Performance**

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single Thread)	Performance(fps) (Multiple Thread)
ResNet-50	224x224	7.7	64.5	168.2
Inception-v1	224x224	3.2	157.8	460.2
Inception-v2	224x224	4.0	114.8	302.1
Inception-v3	299x299	11.4	59.3	159.6
Inception-v4	299x299	24.5	29.7	81.2

MobileNet-v2	224x224	0.6	200.2	573.3
ResNet-50_TF <sup>1</sup>	224x224	7.0	68.9	179.4
Inception-v1_TF <sup>1</sup>	224x224	3.0	131.8	356.1
MobileNet-v2_TF1 <sup>1</sup>	224x224	1.2	150	408.2
SSD_ADAS_VEHICLE	480x360	6.3	82	271.6
SSD_ADAS_PEDESTRIAN	640x360	5.9	58	220.7
SSD_TRAFFIC	480x360	11.6	49.8	186.3
SSD_MobileNet	480x360	5.7	47.9	156.7
SSD_MobileNet-v2	480x360	6.6	40.1	137.9
SSD_VOC_TF <sup>1</sup>	300x300	62.8	14.7	46.4
DenseBox (face detect)	320x320	0.49	296.4	1250.9
DenseBox_640x360	640x360	1.1	151.5	682.2
YOLOv3_ADAS	512x256	5.5	84.6	235.4
YOLOv3_VOC	416x416	65.4	14.4	41.4
YOLOv3_VOC_TF <sup>1</sup>	416x416	65.4	14.3	41.1
YOLOv2_BASELINE	448x448	34	25.0	83.1
YOLOv2_COMPRESS22G	448x448	11.6	52.4	203.3
YOLOv2_COMPRESS24G	448x448	9.9	58.4	231.4
YOLOv2_COMPRESS26G	448x448	7.8	65.8	270.6
RefineDet	480x360	25	33.1	103.5
RefineDet_10G	480x360	10.1	60.3	193.1
RefineDet_5G	480x360	5.1	80.5	262.8
FPN (segmentation)	512x256	8.9	54.5	189.4
VPGnet (roadline detection)	640x480	18.6	71.4	202.4

Roadline_deephi	512x128	4.9	95.8	301.7
Sp-net (pose detection)	128x224	0.55	415.6	1275.6
Openpose	368x368	189	3.8	16.3

<sup>1</sup>These neural network models are trained based on the Tensorflow framework.

Refer to Table 6 for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-05) with DPU running at 294Mhz.

**Table 6: ZCU102 (0432055-05) Performance**

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
Resnet-50	224x224	7.7	63.3	157.2
Inception-v1	224x224	3.2	156.1	440.8
Inception-v2	224x224	4.0	113.1	283.4
Inception-v3	299x299	11.4	58.8	151.6
Inception-v4	299x299	24.5	29.5	77.7
MobileNet-v2	224x224	0.6	195.7	533.8
Resnet-50_TF <sup>1</sup>	224x224	7.0	67.6	167
Inception-v1_TF <sup>1</sup>	224x224	3.0	130.2	336.5
MobileNet-v2_TF <sup>1</sup>	224x224	1.2	146.9	381.1
SSD_ADAS_VEHICLE	480x360	6.3	81.8	265.9
SSD_ADAS_PEDESTRIAN	640x360	5.9	57.8	218.7
SSD_TRAFFIC	480x360	11.6	49.8	185.3
SSD_MobileNet	480x360	5.7	47.3	145.5
SSD_MobileNet-v2	480x360	6.6	39.7	122.9
SSD_VOC_TF <sup>1</sup>	300x300	62.8	14.7	46.3
DenseBox (face detect)	320x320	0.49	292.6	1141.8
DenseBox_640x360	640x360	1.1	150	603.8

YOLOv3_ADAS	512x256	5.5	84.1	216
YOLOv3_VOC	416x416	65.4	14.4	40.5
YOLOv3_VOC_TF <sup>1</sup>	416x416	65.4	14.3	40.1
YOLOv2_BASELINE	448x448	34	25.0	81.7
YOLOv2_COMPRESS22G	448x448	11.6	52.3	202.7
YOLOv2_COMPRESS24G	448x448	9.9	58.3	229.0
YOLOv2_COMPRESS26G	448x448	7.8	65.7	266.9
RefineDet	480x360	25	33	102.8
RefineDet_10G	480x360	10.1	60.1	189.9
RefineDet_5G	480x360	5.1	80.3	254.3
FPN (segmentation)	512x256	8.9	54.2	181.2
VPGnet (roadline detection)	640x480	18.6	70.9	190.8
Roadline_deepphi	512x128	4.9	94.9	292.1
Sp-net (pose detection)	128x224	0.55	408.2	1219.6
Openpose	368x368	189	3.8	16.2

<sup>1</sup>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 deliver 2.4 TOPS INT8 peak performance for deep-learning inference acceleration.

Refer to Table 7 (on the following page) for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU104 with DPU running at 302Mhz.

**Table 7: ZCU104 Performance**


Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
Resnet-50	224x224	7.7	66.5	124.2
Inception-v1	224x224	3.2	162.1	340.2
Inception-v2	224x224	4.0	116	223
Inception-v3	299x299	11.4	60	118.2
Inception-v4	299x299	24.5	30.1	57.9
MobileNet-v2	224x224	0.6	203.3	433.6
Resnet-50_TF <sup>1</sup>	224x224	7.0	70.6	132.2
Inception-v1_TF <sup>1</sup>	224x224	3.0	132.7	263.1
MobileNet-v2_TF <sup>1</sup>	224x224	1.2	152.4	306.2
SSD_ADAS_VEHICLE	480x360	6.3	74.5	213.5
SSD_ADAS_PEDESTRIAN	640x360	5.9	59.8	183.3
SSD_TRAFFIC	480x360	11.6	50.9	147.3
SSD_MobileNet	480x360	5.7	30.2	124.8
SSD_MobileNet-v2	480x360	6.6	22.3	109.3
SSD_VOC_TF <sup>1</sup>	300x300	62.8	12.6	32.4
DenseBox (face detect)	320x320	0.49	304.5	1056.9
DenseBox_640x360	640x360	1.1	155.6	564
YOLOv3_ADAS	512x256	5.5	85.7	197.1
YOLOv3_VOC	416x416	65.4	14.8	29.5
YOLOv3_VOC_TF <sup>1</sup>	416x416	65.4	14.7	29.3
YOLOv2_BASELINE	448x448	34	25.6	58.4
YOLOv2_COMPRESS22G	448x448	11.6	53.7	153.5
YOLOv2_COMPRESS24G	448x448	9.9	59.8	180.1

YOLOv2_COMPRESS26G	448x448	7.8	67.3	214.9
RefineDet	480x360	25	34	75.3
RefineDet_10G	480x360	10.1	62	149.9
RefineDet_5G	480x360	5.1	82.9	213.3
FPN (segmentation)	512x256	8.9	55.4	147.9
VPNet (roadline detection)	640x480	18.6	72.5	180.1
Roadline_deepHi	512x128	4.9	97.7	236.2
Sp-net (pose detection)	128x224	0.55	424.6	908
Openpose	368x368	189	3.9	11.4

<sup>1</sup>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.

 The B2304F DPU core is implemented in program logic of Ultra96 and delivers 587 GOPS INT8 peak performance for deep-learning inference acceleration.

Refer to Table 8 for the throughput performance (in frames/sec or fps) for various neural network samples on Ultra96 with DPU running at 255Mhz.

**Table 8: Ultra96 Performance**

Neural Network	Input Size	MAC(GOPS)	Performance(fps) (Single thread)	Performance(fps) (Multiple thread)
Resnet-50	224x224	7.7	31.8	32.1
Inception-v1	224x224	3.2	79.1	80.7
Inception-v2	224x224	4.0	55.2	56

Inception-v3	299x299	11.4	27.2	27.7
Inception-v4	299x299	24.5	13.7	14
MobileNet-v2	224x224	0.6	124.1	130.1
Resnet-50_TF <sup>1</sup>	224x224	7.0	34.6	34.9
Inception-v1_TF <sup>1</sup>	224x224	3.0	65	66
MobileNet-v2_TF <sup>1</sup>	224x224	1.2	89.3	91.1
SSD_ADAS_VEHICLE	480x360	6.3	39.4	47.9
SSD_ADAS_PEDESTRIAN	640x360	5.9	36.6	48.4
SSD_TRAFFIC	480x360	11.6	28	33.8
SSD_MobileNet	480x360	5.7	21.5	36.8
SSD_MobileNet-v2	480x360	6.6	16.4	32.4
SSD_VOC_TF <sup>1</sup>	300x300	62.8	6.3	7.0
DenseBox (face detect)	320x320	0.49	204.7	294.7
DenseBox_640x360	640x360	1.1	104.6	162.5
YOLOv3_ADAS	512x256	5.5	44.4	48.4
YOLOv3_VOC	416x416	65.4	6.6	6.9
YOLOv3_VOC_TF <sup>1</sup>	416x416	65.4	6.5	6.8
YOLOv2_BASELINE	448x448	34	11.5	12
YOLOv2_COMPRESS22G	448x448	11.6	31.4	36.7
YOLOv2_COMPRESS24G	448x448	9.9	35.6	43
YOLOv2_COMPRESS26G	448x448	7.8	40.9	51.5
RefineDet	480x360	25	16.2	17.2
RefineDet_10G	480x360	10.1	34.2	38.2
RefineDet_5G	480x360	5.1	49	57.7
FPN (segmentation)	512x256	8.9	30.8	35.2



VPGnet (roadline detection)	640x480	18.6	40.3	47.2
Roadline_deephi	512x128	4.9	51.2	57.5
Sp-net (pose detection)	128x224	0.55	230.5	242.7
Openpose	368x368	189	2.1	2.8

<sup>1</sup>These neural network models are trained based on the Tensorflow framework.

## Appendix A: Legal Notices

---

### Please Read: Important Legal Notices

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

#### **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..

© Copyright 2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.