Vitis AI Library User Guide

UG1354 (v1.0) December 2, 2019

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/02/2019 Version 1.0	
Entire document	Updated the content for Vitis™ AI v1.0. Removed support for the Ultra96 board.
08/13/2019 Version 2.0	
Entire document	Updated framework figure. Added “About this document” and “Release Notes” in Chapter 1. Updated Installation in Chapter 2. Added Programming Examples chapter. Added Application demos chapter. Added Resnet18, face landmark and ReID model. Updated Performance data for ZCU102, ZCU104,Ultra96. Removed the original Chapter 3: Installation. Removed the original Chapter 4: Cross-Compiling Removed the original Chapter 6: Libraries Advanced Application. Removed Roadline_deephi Model.
05/31/2019 Version 1.2	
Chapter 3: Libraries and Samples	Added Inception_V4, YOLOV2, Roadline_deephi model. Removed RefineDet_640x480 model.
Chapter 7: Performance	Updated Performance data for ZCU102, ZCU104, Ultra96.
05/24/2019 Version 1.1	
Entire document	Editorial updates.
04/29/2019 Version 1.0	
Initial release.	N/A

Table of Contents

Introduction

About this Document

Related Version

The following AI Library version is related to this document.

Table 1: **Vitis AI Library Packet List**

No	Packet Name	Version
1	vitis_ai_library_r1.0_video.tar.gz	r1.0
2	vitis_ai_library_2019.2-r1.0.deb	r1.0
3	vitis_ai_model_ZCU102_2019.2-r1.0.deb	r1.0
4	vitis_ai_model_ZCU104_2019.2-r1.0.deb	r1.0

Intended Audience

The users of Vitis AI libraries are as follows:

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

Note: If the users have custom models that are completely different from the model supported by the AI Library or has a special post-processing part, they can also use our samples and libraries implementation for reference.

Document Navigation

This document describes how to install, use, and develop with the AI Library.

- Chapter 1 is an introduction to the AI Library. This chapter provides a clear understanding of the AI Library in general, its framework, supported networks, supported hardware platforms and so on.

- Chapter 2 describes how to install the AI Library and run the example. The information in this chapter will help quickly set up the host and target environments, compile and execute the AI Library related examples.
- Chapter 3 describes, in detail, each model library supported by the AI Library. This chapter provides an understanding of the model libraries supported by the AI Library, the purpose of each library, how to test the library with images or videos, and how to test the performance of the library.
- Chapter 4 describes, in detail, how to develop applications with AI Library. This chapter provides an understanding of the following:
  - Development using Vitis API
  - Development using your models
  - Customizing pre-processing
  - Using the configuration file as pre-processing and post-processing parameters
  - Using the post-processing library in AI Library
  - Implementing your post-processing code
- Chapter 5 describes how to set up a test environment and run the application demos. There are two application demos provided with the Vitis AI Library.
- Chapter 6 describes how to find the programming APIs.
- Chapter 7 describes, in detail, the performance of the Vitis AI library on different boards.

Overview

The Vitis AI Library is a set of high-level libraries and APIs built for efficient AI inference with Deep-Learning Processor Unit (DPU). It is built based on the Vitis AI Runtime with unified APIs, and it fully supports XRT 2019.2.

The Vitis AI Library provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks. This simplifies the use of deep-learning neural networks, even for users without knowledge of deep-learning or FPGAs. The Vitis AI Library allows users to focus more on the development of their applications, rather than the underlying hardware.

For the intended audience for the AI Library, please refer to the About this Document section.

Block Diagram

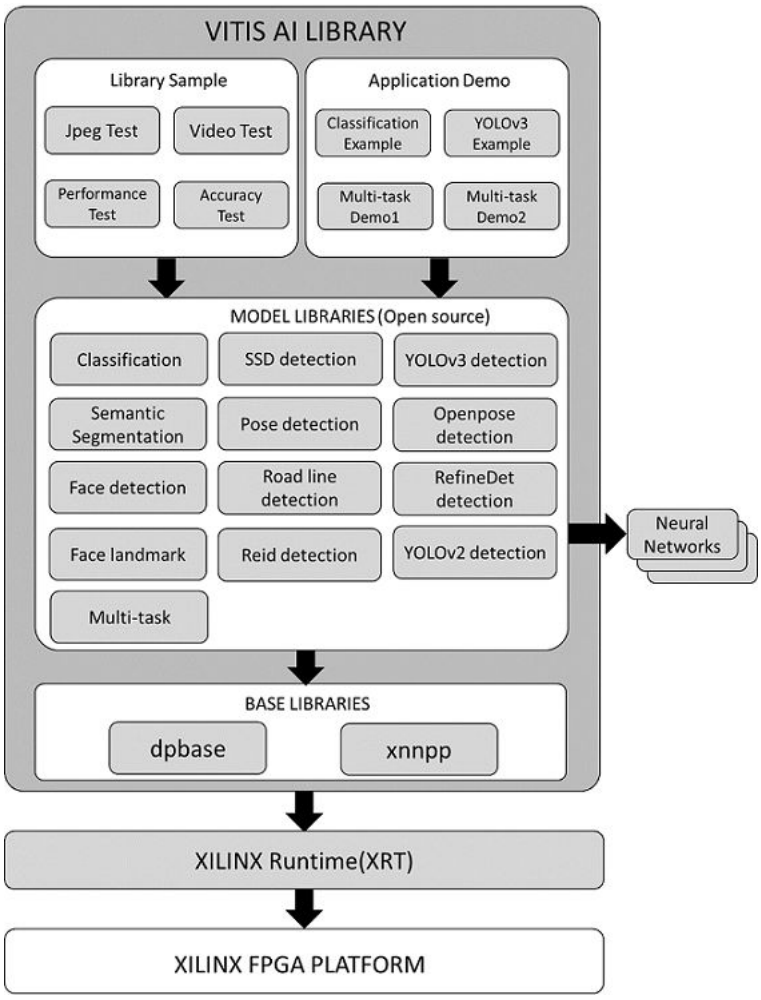
The Vitis AI Library contains four parts: the base libraries, the model libraries, the library test samples, and the application demos.

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

***Note*:** The *xnnpp* library is a closed source.

The model libraries implement most of the neural network deployment which are open source. They include common types of networks, such as classification, detection, segmentation, and so on. These libraries provide an easy-to-use and fast development method with a unified interface, which are applicable to the Xilinx models or custom models. The library test samples are used to quickly test and evaluate the model libraries. The application demos show users how to use AI Library to develop applications. The Vitis AI Library block diagram is shown in the following figure.

Figure 1: **Vitis AI Library Block Diagram**



Features

The Vitis AI Library features include:

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

Vitis AI Library 1.0 Release Notes

This section contains information regarding the features and updates of the Vitis AI Library 1.0 release. This release is the successor of last Xilinx AI SDK v2.0 release.

The Vitis AI Library is a set of high-level libraries and APIs built for efficient AI inference with Deep-Learning Processor Unit (DPU). It provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks.

Key Features And Enhancements

This AI Library release includes the following key features and enhancements.

- **Support for new Vitis AI Runtime:** Vitis AI Library is updated to be based on the new Vitis AI Runtime with unified APIs. It also fully supports XRT 2019.2.
- **New DPU support:** Besides DPUv2 for edge devices, new AI Library will support new cloud based DPU IPs using same codes (runtime and models for cloud DPU will not be included in this release).
- **New Tensorflow model support:** There are up to 21 tensorflow models supported, which are from official Tensorflow repository. The pre-compiled models for edge devices are included, while original models are released by updated Model Zoo.
- **New Libraries and Demos:** There are two new libraries `libdpmultitask` and `libdptfssd` which supports multi-task models and SSD models from official tensor repository.

  There is an updated classification demo that shows how to uses unified APIs in Vitis AI runtime.
- **New Open Source Library:** The `libdpbase` library is open source in this release, which shows how to use unified APIs in Vitis AI runtime to construct high-level libraries.
- **New Installation Method:** The host side environment adopts docker image installation, which simplifies and unifies the installation process.

Compatibility

- Vitis AI Library 1.0 has been tested with the following images.
  - xilinx-zcu102-dpu-v2019.2.img
  - xilinx-zcu104-dpu-v2019.2.img
- For existing Xilinx AI SDK v2.0 users, the library interface remains consistent and the application can be directly ported to the new Vitis AI Library.

Model Support

The following models are supported by this version of the Vitis AI Library.

Table 2: **Model Supported by the AI Library**

No.	Neural Network	Application
1	inception_resnet_v2_tf	Image Classification
2	inception_v1_tf	
3	inception_v3_tf	
4	inception_v4_2016_09_09_tf	
5	mobilenet_v1_0_25_128_tf	
6	mobilenet_v1_0_5_160_tf	
7	mobilenet_v1_1_0_224_tf	
8	mobilenet_v2_1_0_224_tf	
9	mobilenet_v2_1_4_224_tf	
10	resnet_v1_101_tf	
11	resnet_v1_152_tf	
12	resnet_v1_50_tf	
13	vgg_16_tf	
14	vgg_19_tf	
15	ssd_mobilenet_v1_coco_tf	Object Detection
16	ssd_mobilenet_v2_coco_tf	
17	ssd_resnet_50_fpn_coco_tf	
18	yolov3_voc_tf	
19	mlperf_ssd_resnet34_tf	
20	resnet50	Image Classification
21	resnet18	
22	inception_v1	
23	inception_v2	
24	inception_v3	
25	inception_v4	
26	mobilenet_v2	
27	squeezenet	
28	ssd_pedestrain_pruned_0_97	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Traffic Detection
30	ssd_adas_pruned_0_95	ADAS Vehicle Detection
31	ssd_mobilenet_v2	Object Detection
32	refinedet_pruned_0_8	
33	refinedet_pruned_0_92	
34	refinedet_pruned_0_96	
35	vpgnet_pruned_0_99	ADAS Lane Detection

Table 2: **Model Supported by the AI Library** *(cont'd)*

No.	Neural Network	Application
36	fpn	ADAS Segmentation
37	sp_net	Pose Estimation
38	openpose_pruned_0_3	
39	densebox_320_320	Face Detection
40	densebox_640_360	
41	face_landmark	Face Detection and Recognition
42	reid	Object tracking
43	multi_task	ADAS
44	yolov3_adas_pruned_0_9	Object Detection
45	yolov3_voc	
46	yolov3_bdd	
47	yolov2_voc	
48	yolov2_voc_pruned_0_66	
49	yolov2_voc_pruned_0_71	
50	yolov2_voc_pruned_0_77	

Notes:

1. No1-No19 neural network models are trained based on the Tensorflow framework.
2. No20-No50 neural network models are trained based on the Caffe framework.

Device Support

The following platforms and EVBs are supported by the Vitis AI Library1.0.

Table 3: **Device Support**

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

Limitations

Because of the complicated configuration for SSD models from the official Tensorflow repository, there is a new `libdptfssd` library that is different from the original `libdpssd` library for caffe models. These two libraries may be merged in future releases.

Deprecated Features

The following features are deprecated in Vitis AI Library 1.0.

- Removed demos.

The squeezenet and SSD demos have been removed. Because we highly encourage customers to use high-level APIs from AI Library for applications and solutions, we only provide one classification demo in this release to show how to use low-level unified APIs in Vitis AI runtime.

- Removed pre-compiled models.

  We removed six Tensorflow models in this release but provided in the previous Xilinx AI v2.0 release, to keep sync with updated Model Zoo. Models that are removed can be replaced by similar models in updated Model Zoo which come from Tensorflow slim models. The models are:

  . resnet_50_tf

  . inception_v1_tf

  . resnet_18_tf

  . mobilenet_v1_tf

  . mobilenet_v2_tf

  . ssd_voc_tf

Installation

Downloading the Vitis AI Library

The Vitis AI Library package can be freely downloaded after registration on the Xilinx website.

Xilinx recommends that you use a Vitis AI Library-supported evaluation board 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 Vitis AI Library-supported evaluation boards.

The evaluation boards supported for this release are:

- Xilinx ZCU102
- Xilinx ZCU104

Setting Up the Host

The host side development environment is setting up by docker image.

1. Download the `vitis-ai-docker-runtime` image from https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html
2. Set up the docker runtime system according to the docker installation document.

   ```
   $sh docker_run.sh
   ```

   After the docker image is installed, the cross compiler tools are stored in `/opt/vitis_ai/petalinux_sdk/`.

   Note: A `workspace` folder will be created by the docker runtime system. And it will be mounted in `/workspace` of the docker runtime system.

3. Place the program, data and other files to be developed in the `workspace` folder. After the docker system starts, you will find them in the `/workspace` of the docker system.

   Do not put the files in any other path of the docker system. They will be lost after you exit the docker system.

4. Using Git, clone the corresponding AI Library from https://github.com/Xilinx/Vitis-AI.

```
$cd /workspace
$git clone https://github.com/Xilinx/Vitis-AI.git
```

5. Cross compile the demo in the AI Library, using yolov3 as example,

```
$cd /workspace/Vitis-AI/Vitis-AI-Library/demo/demo_yolov3
$sh -x build.sh
```

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

```
$sh build.sh
```

If the compilation process does not report any error and the executable file `demo_yolov3` is generated, the host environment is installed correctly.

6. To compile the library sample in the AI Library, take `classification` for example, execute the following command.

```
$cd /workspace/Vitis-AI/Vitis-AI-Library/samples/classification
$sh -x build.sh
```

The executable program is now produced.

7. To modify the library source code, view and modify them under `/workspace/Vitis-AI/Vitis-AI-Library/libsrc`.

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

```
$cd /workspace/Vitis-AI/Vitis-AI-Library/libsrc/libdpclassification
$sh -x build.sh
```

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

Note: All the source code, samples, demos, and head files can be found in `/workspace/Vitis-AI/Vitis-AI-Library`.

AI Library File Locations

The following table shows the AI Library file location after the installation is complete.

Table 4: **AI Library File Location List**

Files	Location
Sour code of the libraries	`/workspace/Vitis-AI/Vitis-AI-Library/libsrc`
Library files	`/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib`
Header files	`/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/include/xilinx/ai`

Table 4: **AI Library File Location List** *(cont'd)*

Files	Location
Samples	`/workspace/Vitis-AI/Vitis-AI-Library/samples`
Demos	`/workspace/Vitis-AI/Vitis-AI-Library/demo`
Test	`/workspace/Vitis-AI/Vitis-AI-Library/libsrc/[model]/test`

Notes:
The following symbols/abbreviations are used.

- `/workspace/` is the path to extract the AI Library compressed package in the docker system.
- `/opt/vitis_ai` is the docker system's path.
- “Samples” is used for rapid application construction and evaluation, and it is for users.
- “Demos” provides more practical examples for user development, and it is for users.
- “Test” is a test example for each model library which is for library developers.

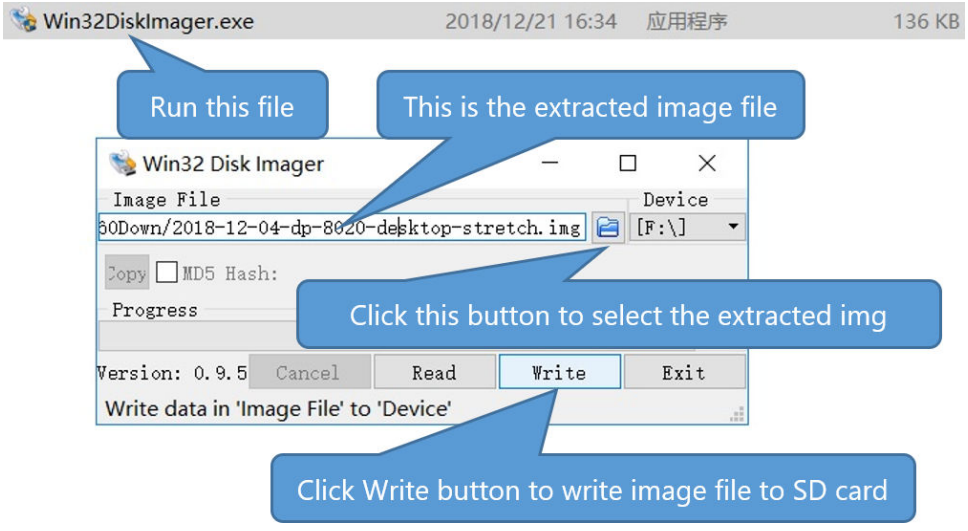
Setting Up the Target

To set up the target, you should follow three steps. The first step is to install the board image,the second step is to install the AI model packet, and the third step is to install the AI Library packet.

***Note*:** The version of the board image should be 2019.2 or above.

Step 1: Installing a Board Image

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



3. Insert the SD card with the image into the destination board.

4. Plug in the power and boot the board using the serial port to operate on the system.
5. Set up the IP information of the board using the serial port.

   You can now operate on the board using SSH.

Step 2: Installing AI Model Package

1. Download the `vitis_ai_model_ZCU102_2019.2-r1.0.deb` packet.
2. Copy the downloaded file to the board using `scp` with the following command.

   ```
   $scp vitis_ai_model_ZCU102_2019.2-r1.0.deb root@IP_OF_BOARD:~/
   ```

   Note: The deb package can be taken as a normal archive, and you can extract the contents on the host side if you only need part of the models. The operation command is as follows.

   ```
   $mkdir extract
   $dpkg -X vitis_ai_model_ZCU102_2019.2-r1.0.deb extract
   ```

3. Log in to the board (usong ssh or serial port) and install the model package.
4. Run the following command.

   ```
   #dpkg -i vitis_ai_model_ZCU102_2019.2-r1.0.deb
   ```

   After the installation is complete, the model files are stored in `/usr/share/vitis_ai_library/models` on the target side.

Step 3: Installing AI Library Package

1. Download the `vitis_ai_library_2019.2-r1.0.deb` packet.
2. Copy the downloaded file to the board using `scp` with the following command.

   ```
   $scp vitis_ai_library_2019.2-r1.0.deb root@IP_OF_BOARD:~/
   ```

   Note: You can take the deb package as a normal archive, and extract the contents on the host side, if you only need some of the libraries. Only model libraries can be separated dependently, while the others are common libraries. The operation command is as follows.

   ```
   $mkdir extract
   $dpkg -X vitis_ai_library_2019.2-r1.0.deb extract
   ```

3. Log in to the board using ssh.

   You can also use the serial port to login.
4. Install the Vitis AI Library.

   Execute the following command.

   ```
   #dpkg -i vitis_ai_library_2019.2-r1.0.deb
   ```

After the installation is complete, the directories are as follows.

- Library files are stored in `/usr/lib`
- The header files are stored in `/usr/include/xilinx/ai`
- Samples are stored in `/usr/share/vitis_ai_library/samples`
- Demos are stored in `/usr/share/vitis_ai_library/demo`

Running Vitis AI Library Examples

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

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

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

2. Run the example.

   ```
   #./test_jpeg_facedetect densebox_640_360 sample_facedetect.jpg
   ```

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

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

3. View the running results.

   There are two ways to view the results. One is to view the results by printing information, while the other is to view images by downloading the `sample_facedetect_result.jpg` image as shown in the following image.

Figure 2: **Face Detection Example**

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

```
#./test_video_facedetect densebox_640_360 video_input.mp4 -t 8
Video_input.mp4: The video file's name for input.The user needs to
prepare the videofile.
-t: <num_of_threads>
```

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

```
#./test_video_facedetect densebox_640_360 0 -t 8
0: The first USB camera device node. If you have multiple
        USB camera, the value might be 1,2,3 etc.
-t: <num_of_threads>
```

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

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

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

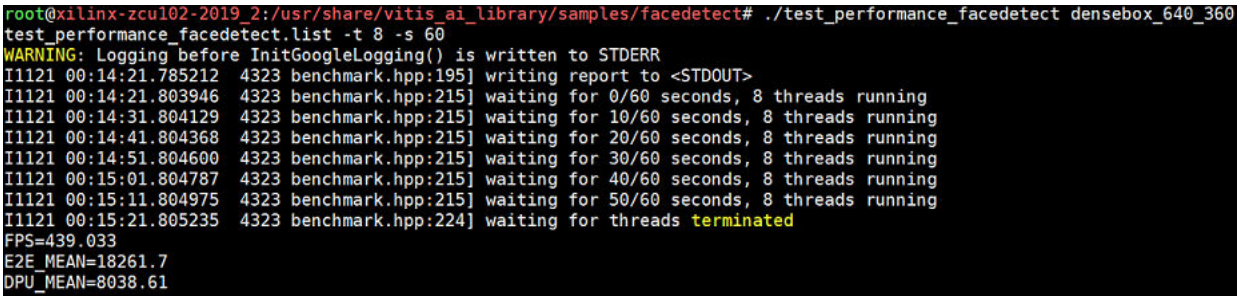
```
#./test_performance_facedetect densebox_640_360
test_performance_facedetect.list -t 8 -s 60

-t: <num_of_threads>

-s: <num_of_seconds>
```

For more parameter information, enter `-h` for viewing. The following image shows the result of performance testing in 8 threads.

Figure 3: **Face Detection Performance Test Result**



7. To check the version of Vitis AI Library, run the following command:

```
#vitis_ai
```

8. To run the demo, refer to Chapter 5: Application Demos.

Support

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

Libraries and Samples

The Vitis AI Library contains the following types of neural network libraries based on Caffe framework:

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

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

- Classification
- SSD detection
- YOLOv3 detection

The related libraries are open source and can be modified as needed. The open source codes are stored in the `/workspace/Vitis-AI/Vitis-AI-Library/libsrc` directory in the docker system.

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

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

Model Library

After the model packet is installed on the target, all the models are stored under `/usr/share/vitis_ai_library/models/`. Each model is stored in a separate folder, which is composed of the following files by default.

- [model_name].elf
- [model_name].prototxt
- meta.json

Take the "inception_v1" model as an example. "inception_v1.elf" is the model data. "inception_v1.prototxt" is the parameter of the model. `meta.json` is the configuration file of the model. The application will get the model info through this configuration file. The following table is detail description of `meta.json`.

Table 5: **The content of the "meta.json"**

Element	Instruction
target	The type of DPU, such as DPUv2 and DPUv3E
lib	The driver of DPU, such as "libvart_dpu.so"
filename	The name of model file
kernel	The kernel name of the model
config_file	The parameter file name of the model

Note that the `meta.json` file should be under the same directory with the model file and the name of the model directory should be the same with the model name.

Model Type

Classification

This library is used to classify images. Such neural networks are trained on ImageNet for ILSVRC and they can identify the objects from its 1000 classification. The AI Library r1.0 integrated Resnet18, Resnet50, Inception_v1, Inception_v2, Inception_v3, Inception_v4, Vgg, mobilenet_v1, mobilenet_v2 and Squeezenet into our library. The input is a picture with an object and the output is the top-K most probable category.

Figure 4: **Classification Example**

The following table shows the classification model supported by the AI Library.

Table 6: **The Classification Model List**

No	Model Name	Framework
1	inception_resnet_v2_tf	Tensorflow
2	inception_v1_tf	
3	inception_v3_tf	
4	inception_v4_2016_09_09_tf	
5	mobilenet_v1_0_25_128_tf	
6	mobilenet_v1_0_5_160_tf	
7	mobilenet_v1_1_0_224_tf	
8	mobilenet_v2_1_0_224_tf	
9	mobilenet_v2_1_4_224_tf	
10	resnet_v1_101_tf	
11	resnet_v1_152_tf	
12	resnet_v1_50_tf	
13	vgg_16_tf	
14	vgg_19_tf	

Table 6: **The Classification Model List** *(cont'd)*

No	Model Name	Framework
15	resnet50	Caffe
16	resnet18	
17	inception_v1	
18	inception_v2	
19	inception_v3	
20	inception_v4	
21	mobilenet_v2	
22	squeezenet	

Face Detection

This library uses DenseBox neuron network to detect human face. Input is a picture with some faces you would like to detect. Output is a vector of the result structure containing each box's information.

The following image show the result of face detection.

Figure 5: **Face Detection Example**

The following table shows the face detection model supported by the AI Library.

Table 7: **The face detection model list**

No	Model Name	Framework
1	densebox_320_320	Caffe
2	densebox_640_360	

Face Landmark Detection

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

Figure 6: **Face Landmark Detection Example**

The following table shows the face landmark model supported by the AI Library.

Table 8: **The face landmark model list**

No	Model Name	Framework
1	face_landmark	Caffe

SSD Detection

This library is in common use to SSD neuron network. SSD is a neural network which is used to detect objects. Input is a picture with some objects you'd like to detect. Output is a vector of the result structure containing each box's information. The following image shows the result of SSD detection.

Figure 7: **SSD Detection Example**

The following table shows the SSD detection model supported by the AI Library.

Table 9: **The SSD Model List**

No	Model Name	Framework
1	ssd_mobilenet_v1_coco_tf	Tensorflow
2	ssd_mobilenet_v2_coco_tf	
3	ssd_resnet_50_fpn_coco_tf	
4	mlperf_ssd_resnet34_tf	
5	ssd_pedestrain_pruned_0_97	Caffe
6	ssd_traffic_pruned_0_9	
7	ssd_adas_pruned_0_95	
8	ssd_mobilenet_v2	

Pose Detection

This library is used to detect posture of the human body. This library includes a neural network which can mark 14 key points of the human body (you can use our SSD detection library). The input is a picture that is detected by the pedestrian detection neural network. The output is a structure containing coordinates of each point .

The following image shows the result of pose detection.

Figure 8: **Pose Detection Example**

The following table shows the pose detection model supported by the AI Library.

Table 10: **The pose detection model list**

No	Model Name	Framework
1	sp_net	Caffe

***Note*:** If the input image is arbitrary and the user does not know the exact location of the person, we must perform the SSD detection first. See the `test_jpeg_posedetect_with_ssd.cpp` file. If the input picture is the picture of the person who has been cut out, you can only perform pose detection. See the `test_jpeg_posedetect.cpp` file.

Semantic Segmentation

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

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

Figure 9: **Semantic Segmentation Example**

The following table shows the semantic segmentation model supported by the AI Library.

Table 11: **The semantic segmentation model list**

No	Model Name	Framework
1	fpn	Caffe

Road Line Detection

The library is used to draw lane lines in the adas library and each lane line is represented by a number type representing the category and a vector<Point> used to draw the lane line. In the test code, color map is used. Different types of lane lines are represented by different colors. The point is stored in the container vector, and the polygon interface cv::polylines() of OpenCv is used to draw the lane line. The following image show the result of road line detection.

Figure 10: **Road Line Detection Example**

The following table shows the road line detection model supported by the AI Library.

Table 12: **The road line detection model list**

No	Model Name	Framework
1	vpgnet_pruned_0_99	Caffe

***Note*:** The input of the image is fixed at 480x640 and images of other sizes need to be resized.

YOLOV3 Detection

This lib is in common use to YOLO neuron network. YOLO is a neural network which is used to detect objects. Now its version is v3. Input is a picture with one or more objects. Output is a vector of the result struct which is composed of the detected information. The following image shows the result of YOLOv3 detection.

Figure 11: **YOLOv3 Detection Example**

The following table shows the YOLOv3 detection model supported by the AI Library.

Table 13: **The YOLOv3 detection model list**

No	Model Name	Framework
1	yolov3_voc_tf	Tensorflow
2	yolov3_adas_pruned_0_9	Caffe
3	yolov3_voc	
4	yolov3_bdd	

YOLOV2 Detection

YOLOV2 does the same thing as YOLOV3, which is an upgraded version of YOLOV2. The following table shows the YOLOv2 detection model supported by the AI Library.

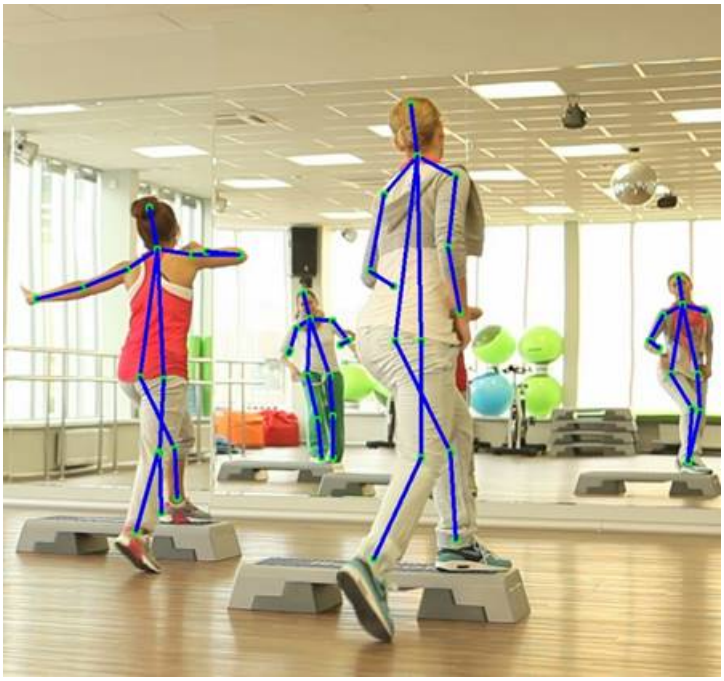
Table 14: **The YOLOv2 detection model list**

No	Model Name	Framework
1	yolov2_voc	Caffe
2	yolov2_voc_pruned_0_66	
3	yolov2_voc_pruned_0_71	
4	yolov2_voc_pruned_0_77	

Openpose Detection

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

Figure 12: **Openpose Detection Example**



The following table shows the Openpose detection model supported by the AI Library.

Table 15: **The Openpose detection model list**

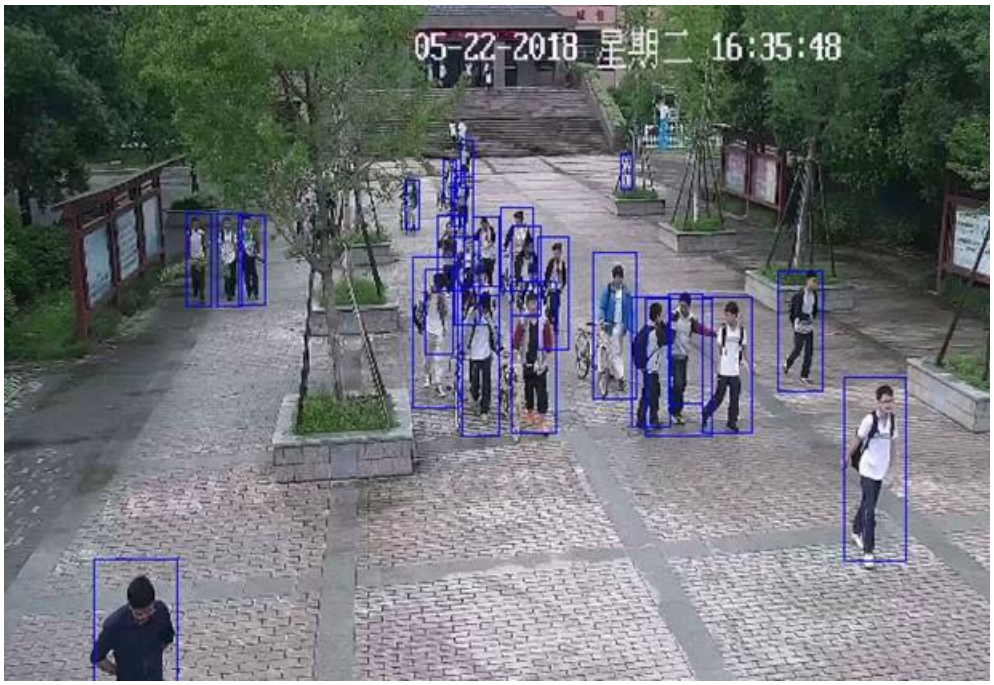
No	Model Name	Framework
1	openpose_pruned_0_3	Caffe

RefineDet Detection

This library is commonly used to RefineDet neuron network. RefineDet is a neural network that is used to detect human bodies. The input is a picture with some individuals that you would like to detect. The output is a vector of the result structure that contain each box's information.

The following image shows the result of RefineDet detection:

Figure 13: **RefineDet Detection Example**



The following table shows the RefineDet detection model supported by the AI Library.

Table 16: **The RefineDet detection model list**

No	Model Name	Framework
1	refinedet_pruned_0_8	Caffe
2	refinedet_pruned_0_92	
3	refinedet_pruned_0_96	

ReID Detection

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

Table 17: **The ReID detection model list**

No	Model Name	Framework
1	reid	Caffe

Multi-task

The multi-task library is appropriate for a model that has multiple sub-tasks. The Multi-task model in AI Library has two sub-tasks. One is Semantic Segmentation and the other is SSD Detection. The following table shows the Multi-task model supported by the AI Library.

Table 18: **The Multi-task model list**

No	Model Name	Framework
1	multi_task	Caffe

Model Samples

There are up to 13 model samples that are loacated in `/usr/share/vitis_ai_library/samples`. Each sample has the following four kinds of test samples.

- test_jpeg_[model type]
- test_video_[model type]
- test_performance_[model type]
- test_accuracy_[model type]

Take yolov3 as an example.

1. Before you run the yolov3 detection example, you can choose one of the following yolov3 model to run.
   a. yolov3_bdd
   b. yolov3_voc
   c. yolov3_voc_tf
2. Ensure that the following test program exists.
   a. test_jpeg_yolov3
   b. test_video_yolov3
   c. test_performance_yolov3
   d. test_accuracy_yolov3

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

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

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

   ```
   #./test_jpeg_yolov3 yolov3_bdd sample_yolov3.jpg
   ```

You will see the printing result on the terminal. Also, you can view the output image: `sample_yolov3_result.jpg`.

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

```
    #./test_video_yolov3 yolov3_bdd video_input.mp4 -t 8
```

5. To test the model performance, execute the following command.

```
    #./test_performance_yolov3 yolov3_bdd test_performance_yolov3.list -
t 8
```

   You will see the printing result on the terminal.

6. To test the model accurary, users need to prepare their own image dataset, image list file and the ground truth of the images. Then execute the following command.

```
    #./test_accuracy_yolov3 yolov3_bdd [image_list_file] [output_file]
```

After the output_file is generate, a script file is needed to automatically compare the results. Finally, the accuracy result can be obtained.

Programming Examples

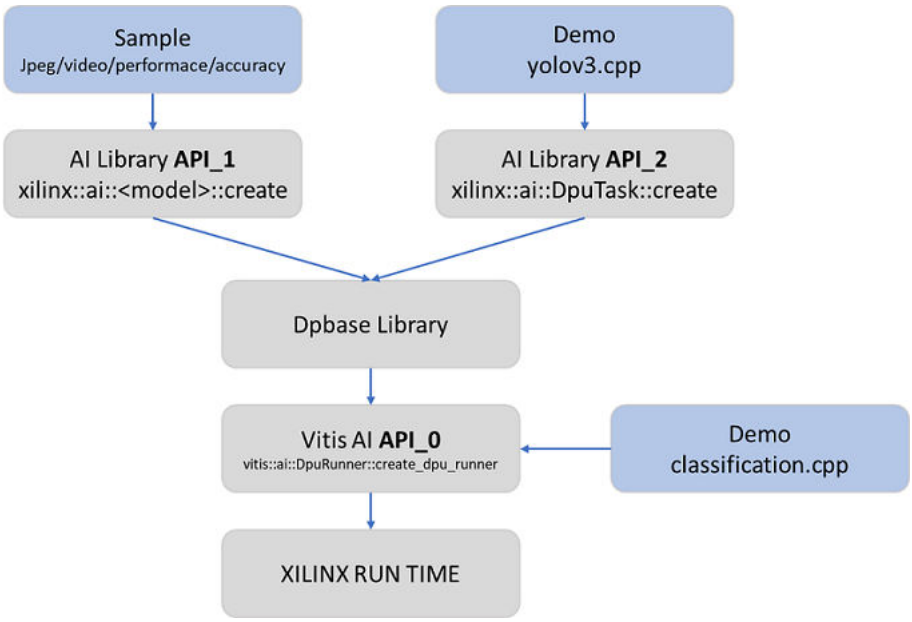
In practice, users have many different application requirements, but it basically falls into three categories. The first is to use the ready-made models provided by Vitis AI Library to quickly build their own applications, and the second is to use users' own custom models which are similar to the models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

- How to customize pre-processing
- How to use the configuration file as pre-processing and post-processing parameter
- How to use the AI Library's post-processing library
- How to implement user post-processing code

The following figure shows the relationships of the various AI Library APIs and their corresponding example. And there are three kinds of APIs in this release.

- Vitis AI *API_0*
- AI Library *API_1*
- AI Library *API_2*

Figure 14: **The Diagram of AI Library API**



Developing With Vitis AI API_0

1. Install the `vitis-ai-docker-runtime` image on the host side, refer to Chapter 2: Installation.
2. After the setting up the docker system, the libraries can be found in the `/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib` directory.
3. Download the `vitis_ai_model_ZCU102_2019.2-r1.0.deb` packet, and copy it to the board via scp.
4. Install the Xilinx Model Package on the target side.

   ```
   #dpkg -i vitis_ai_model_ZCU102_2019.2-r1.0.deb
   ```

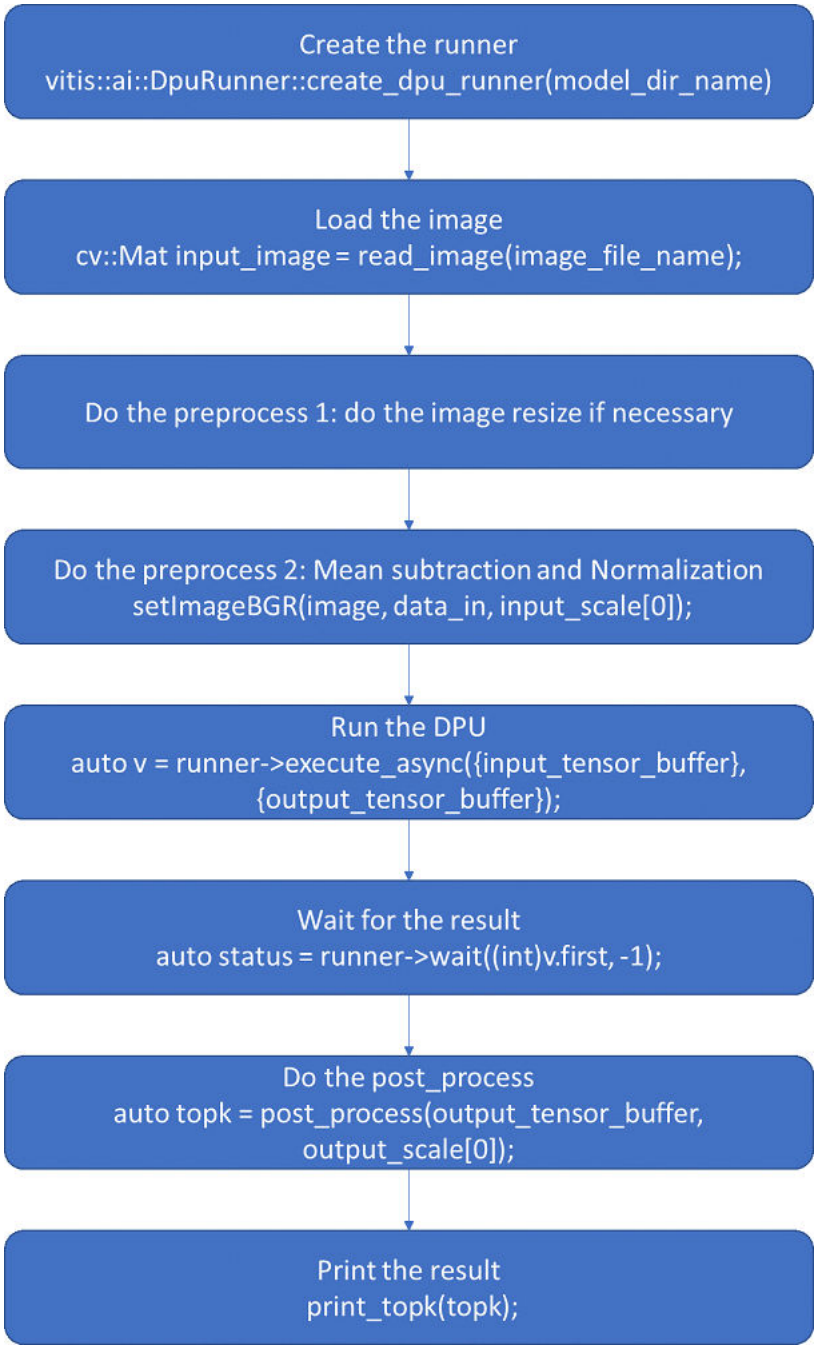
   After the installation, the models can be found in the `/usr/share/vitis_ai_library/models` directory on the target side.

   Note that users do not need to install the Xilinx model packet if they want to use their own model.
5. Git clone the corresponding AI Library from https://github.com/Xilinx/Vitis-AI.
6. Create a folder under your workspace, using classification as an example.

   ```
   $mkdir classification
   ```

7. Create the `demo_classification.cpp` source file. The main flow is shown below. See `/workspace/Vitis-AI/Vitis-AI-Library/demo/classification/demo_classification.cpp` for a complete code example.

Figure 15: **Main program Flow Chart**



8. Create a `build.sh` file as shown below, or copy one from the AI Library's demo and modify it.

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

9. Cross compile the program.

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

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

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

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

```
#./demo_classification resnet50 input_image.jpg
```

***Note*:**

- `demo_classification.cpp` uses user-defined pre-processing parameter as input.
- `demo_classification.cpp` uses user post-processing code. And if you want to use the AI Library's post-processing library, please check How to Use the AI Library's Post-Processing Library

For more details about the demo, refer to `/workspace/Vitis-AI/Vitis-AI-Library/demo` in the docker runtime system.

Developing with User Model and AI Library API_2

When users use their own models, it is important to note that the user's model framework should be within the scope supported by the Vitis AI Library. The following is an introduction of how to deploy a retrained YOLOv3 Caffe model to ZCU102 platform based on Vitis AI Library step by step.

1. Download the corresponding `vitis-ai-docker-tools` and `vitis-ai-docker-runtime` images from https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.
2. Start the `vitis-ai-docker-tools` system.
3. Create a folder and place the float model under it on the host side, then use `AI Quantizer` tool to do the quantization. Please refer to UG1414-vitis-ai-user-guide.pdf for the details.
4. Use `AI Compiler` tool to do the model compiling to get the elf file, such as `yolov3_custom.elf`. For more information, see *Vitis AI User Guide* (UG1414).

5. Create the meta.json file, as shown in the following

```
{
  "target": "DPUv2",
  "lib": "libvart_dpu.so",
  "filename": "yolov3_custom.elf",
  "kernel": [ "yolov3_custom" ],
  "config_file": "yolov3_custom.prototxt"
}
```

6. Create the yolov3_custom.prototxt, as shown in the following.

```
model {
  name: "yolov3_custom"
  kernel {
     name: "yolov3_custom"
     mean: 0.0
     mean: 0.0
     mean: 0.0
     scale: 0.00390625
     scale: 0.00390625
     scale: 0.00390625
  }
  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
  }
}
```

Note: The `<model_name>.prototxt` only take effect when you use AI Library API_1.

When you use AI Library API_2, the parameter of the model needs to be loaded and read manually by the program. Refer to `/workspace/Vitis-AI/Vitis-AI-Library/demo/yolov3/demo_yolov3.cpp` for details.

7. Create the `demo_yolov3.cpp` file. See `/workspace/Vitis-AI/Vitis-AI-Library/demo/yolov3/demo_yolov3.cpp` for reference.

8. Create a `build.sh` file as shown below, or copy one from the AI Library demo and modify it.

```
#/bin/sh
CXX=${CXX:-g++}
$CXX -std=c++11 -O3 -I. -o demo_yolov3 demo_yolov3.cpp -lopencv_core -
lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -
lopencv_highgui -lglog -lxnnpp -ldpproto -lprotobuf -ldpbase
```

9. Exit the docker tool system and start the docker runtime system.
10. Cross compile the program, generate executable file `demo_yolov3`.

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

11. Create model folder under `/usr/share/vitis_ai_library/models` on the target side.

```
#mkdir yolov3_custom /usr/share/vitis_ai_library/models
```

    Note that `/usr/share/vitis_ai_library/models` is the default location for the program to read the model. You can also place the model folder in the same directory as the executable program.

12. Copy the `yolov3_custom.elf, yolov3_custom.prototxt and meta.json` to the target and put them under `/usr/share/vitis_ai_library/models/yolov3_custom`.

```
$scp yolov3_custom.elf  yolov3_custom.prototxt  meta.json
root@IP_OF_BOARD:/usr/share/vitis_ai_library/models/yolov3_custom
```

13. Copy the executable program to the target board using `scp`.

```
$scp demo_yolov3 root@IP_OF_BOARD:~/
```

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

```
#./demo_yolov3 yolov3_custom sample.jpg
```

How to Customize Pre-Processing

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

- Mean subtraction
- Normalization
- PCA and Whitening

User calls the setMeanScaleBGR function to implement the Mean subtraction and normalization, as shown in the figure below. See `/workspace/Vitis-AI/Vitis-AI-Library/include/Xilinx/ai/dpu_task.hpp` for details in the docker runtime system.

Figure 16: **setMeanScaleBGR Example**

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

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

Figure 17: **cv::resize Example**

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

How to Use the Configuration File

Vitis AI Library provides a way to read model parameters by reading the configuration file. It facilitates uniform configuration management of model parameters. The configuration file is located in `/usr/share/vitis_ai_library/models/[model_name]/[model_name].prototxt`.

Note that if you are developing on the host side, the configuration files are installed in `/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/share/vitis_ai_library/models/[model_name]/[model_name].prototxt` in the docker runtime system.

```
model
{
  name: "yolov3_voc"
  kernel {
     name: "yolov3_voc"
     mean: 0.0
     mean: 0.0
     mean: 0.0
     scale: 0.00390625
     scale: 0.00390625
     scale: 0.00390625
```

```
  }
  model_type : YOLOv3
  yolo_v3_param {
    …
  }
  is_tf: false
}
```

Table 19: **Compiling Model and Kernel Parameters**

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

yolo_v3_param

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

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

Table 20: **YOLOv3 Model Parameters**

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

SSD_param

```
model_type : SSD
ssd_param :
{
    num_classes : 4
    nms_threshold : 0.4
    conf_threshold : 0.0
    conf_threshold : 0.6
    conf_threshold : 0.4
    conf_threshold : 0.3
    keep_top_k : 200
    top_k : 400
    prior_box_param {
    layer_width : 60,
    layer_height: 45,
    variances: 0.1
    variances: 0.1
    variances: 0.2
    variances: 0.2
    min_sizes: 21.0
    max_sizes: 45.0
    aspect_ratios: 2.0
    offset: 0.5
    step_width: 8.0
    step_height: 8.0
    flip: true
    clip: false
    }
}
```

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

Table 21: **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

Table 21: **SSD Model Parameters** *(cont'd)*

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

Table 22: **PriorBox Parameters**

Parameter Type	Description
layer_width/layer_height	The input width/height of this layer. Such numbers could be computed from the net structure.
ariances	These numbers are used for boxes regression, just only to fill them as original model. There should be four variances.
min_sizes/max_size	Filled as the "deploy.prototxt", but each number should be written in a separate line.
aspect_ratios	The ratio's number (each one should be written in a separate line). Default has 1.0 as its first ratio. If you set a new number here, there will be two ratios created when the opposite is true. One is a filled number; another is its reciprocal. For example, this parameter has only one set element, "ratios: 2.0". The ratio vector has three numbers: 1.0, 2.0. 0.5
offset	Normally, the PriorBox is created by each central point of the feature map, so that offset is 0.5.
step_width/step_height	Copy from the original file. If there are no such numbers there, you can use the following formula to compute them: step_width = img_width ÷ layer_width step_height = img_height ÷ layer_height
offset	Normally, PriorBox is created by each central point of the feature map, so that the offset is 0.5.
flip	Control whether rotate the PriorBox and change the ratio of length/width.
clip	Set as false. If true, it will let the detection boxes' coordinates keep at [0, 1].

Example Code

The following is the example code.

```
Mat img = cv::imread(argv[1]);
auto yolo = xilinx::ai::YOLOv3::create("yolov3_voc",   true);
auto results   = yolo->run(img);
for(auto   &box : results.bboxes){
    int label = box.label;
    float xmin = box.x * img.cols + 1;
    float ymin = box.y * img.rows + 1;
    float xmax = xmin + box.width * img.cols;
    float ymax = ymin + box.height * img.rows;
    if(xmin < 0.) xmin = 1.;
    if(ymin < 0.) ymin = 1.;
    if(xmax > img.cols) xmax = img.cols;
    if(ymax > img.rows) ymax = img.rows;
    float confidence = box.score;
    cout << "RESULT: "   << label <<   "\t" << xmin << "\t"   << ymin <<
"\t"
            << xmax <<   "\t"   << ymax << "\t" << confidence   << "\n";
    rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0),
1, 1, 0);
}
imshow("", img);
waitKey(0);
```

You should use the “create” to create the YOLOv3 object.

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

Note: The model_name is same as the prototxt's. For more details about the example, refer to `/workspace/Vitis-AI/Vitis-AI-Library/libsrc/libdpyolov3/test/test_yolov3.cpp`.

How to Implement User Post-Processing Code

Users can also call their own post-processing functions on their own request. Take `demo_yolov3.cpp` and `demo_classification.cpp` as an example. Use `xilinx::ai::DpuTask::create` or `vitis::ai::DpuRunner::create_dpu_runner` to create the task, and after DPU processing is complete, the user's post-processing function can be invoked. The `post_process` function in the following figure is a user post-processing code.

Figure 18: **User Post-Processing Code Example**

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

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

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

For the definition of OutputTensor. See the `/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/include/xilinx/ai/tensor.hpp` header file for details in the docker runtime system.

Refer to `/workspace/vitis_ai_library/demo/classification/demo_classification.cpp` for more details in the docker runtime system.

How to Use the AI Library's Post-Processing Library

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

- Classification
- Face detection

- Face landmark detection
- SSD detection
- Pose detection
- Semantic segmentation
- Road line detection
- YOLOV3 detection
- YOLOV2 detection
- Openpose detection
- RefineDet detection
- ReId detection
- Multi-task

There are two ways to call xnnpp.

- One is automatic call, through `xilinx::ai::<model>::create` create the task, such as `xilinx::ai::YOLOv3::create("yolov3_bdd", true)`. After <model>->run finished, xnnpp will be automatically processed, users can modify the parameters through the model configuration file.
- One is manual call, through `xilinx::ai::DpuTask::create` to create the task. Then, create the object of the post-process and run the post-process. Take SSD post-processing as an example, the specific steps are as follows.
  1. Create a config and set the correlating data to control post-process.

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

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

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

  3. Fill other parameters.

     ```
     fillconfig(config);
     ```

  4. Create an object of SSDPostProcess.

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

  5. Run the post-process.

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

Note:

- The header files of the `libxnnpp.so` are stored in `/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/include/xilinx/ai/nnpp/` in the docker runtime syetem.
- The `libxnnpp.so` is stored in `/opt/vitis_ai/petalinux_sdk/sysroots/aarch64-xilinx-linux/usr/lib` in the docker runtime syetem.
- For more details about the post processing examples, refer to `/workspace/vitis_ai_library/demo/yolov3/yolov3.cpp` and `/workspace/vitis_ai_library/libsrc/libdpyolov3/test/test_yolov3.cpp` in the docker runtime syetem.

Application Demos

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

Demo Overview

There are two application demos provided within the Vitis AI Libray. They use the AI Library to build their applications. The codes and video files are stored in `/workspace/Vitis-AI/Vitis-AI-Library/demo/segs_and_roadline_detect` and `/workspace/Vitis-AI/Vitis-AI-Library/demo/seg_and_pose_detect` in the docker system.

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

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

***Note*:** To achieve the best performance, the demos use the DRM (Direct Render Manager) for video display. Please Log in the board using `ssh` or serial port and run the demo remotely. If you do not want to use DRM for video display, set “USE_DRM=0” in the compile option.

Demo Platform and Setup

Demo Platform

- **HW:**
  - 1 x ZCU102 Prod Silicon https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
  - 1 x Win7/10 laptop
  - 1 x 16GB SD card

- 1 x Ethernet cables
- 1 x DP 1080P compatible monitor
- 1 x DP cable

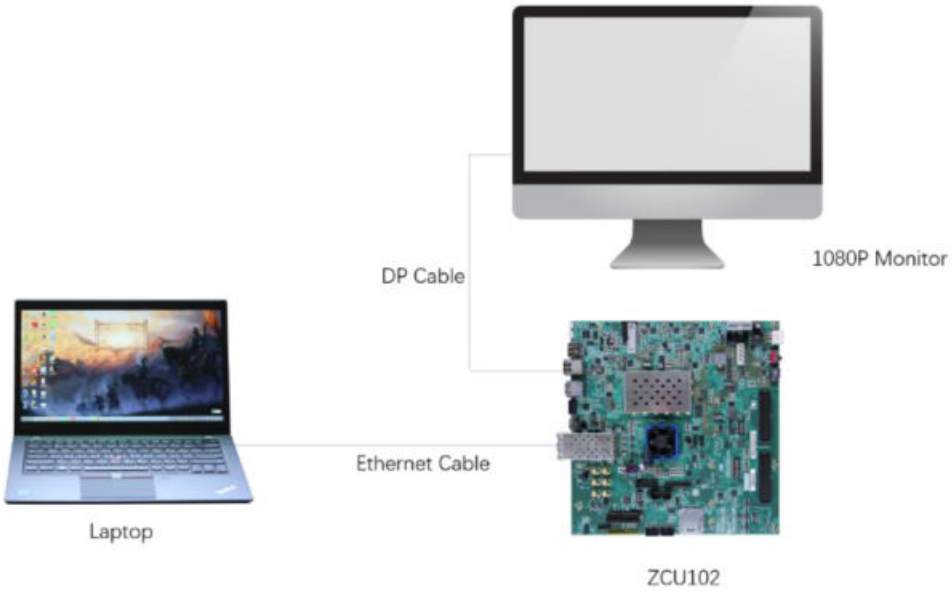
- **SW:**
  - ZCU102 board imagehttps://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge
  - Vitis-AI-Library
  - Images and video files
  - Terminal software like MobaXterm, Putty

DPU Configuration & Dev Tool Used

- 3xB4096 @281MHz
- Vivado 2019.2, AI Library r1.0

Demo Setup Illustration

Figure 19: **Demo Setup**



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

Target Application

- ADAS/AD

AI Model & Performance & Power

- FPN
  - 512x288, 4ch, 20fps
- VPGNET
  - 640x480, 1ch, 56fps
- 20W @ ZU9EG

Build and Run the Demo

```
#cd /usr/share/vitis_ai_library/demo/segs_and_roadline_detect
#sh -x build.sh
```

To use OpenCV display, run the following command.

```
#./segs_and_roadline_detect_x seg_512_288.avi seg_512_288.avi
seg_512_288.aviseg_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3
>/dev/null 2>&1
```

If you want to use DRM display, please connect to the board using SSH and run the following command.

```
#/etc/init.d/weston stop
#./segs_and_roadline_detect_drm seg_512_288.avi seg_512_288.avi
seg_512_288.avi
seg_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3 >/dev/null 2>&1
```

Note that the video files are in the `vitis_ai_library_r1.0_video.tar.gz`. Please download the package from https://www.xilinx.com/products/design-tools/ai-inference/aideveloper-hub.html#edge .

Demo Picture

Figure 20: **Segmentation and Roadline Detection Demo Picture**

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

Target Application

- ADAS/AD
- Smartcity

AI Model & Performance & Power

- FPN
  - 960x540, 1ch, 30fps
- Openpose
  - 960x540, 1ch, 30fps
- 20W @ ZU9EG

Build and Run the Demo

```
#cd /usr/share/vitis_ai_library/demo/seg_and_pose_detect
#sh -x build.sh
```

To use OpenCV display, run the following command.

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

If you want to use DRM display, please connect to the board via SSH. And run the following command.

```
#/etc/init.d/weston stop
#./seg_and_pose_detect_drm seg_960_540.avi pose_960_540.avi -t 4 -t 4 >/dev/
null 2>&1
```

Note that the video files are in the `vitis_ai_library_r1.0_video.tar.gz`. Please download the package from https://www.xilinx.com/products/design-tools/ai-inference/aideveloper-hub.html#edge .

Demo Picture

Figure 21: **Segmentation and Pose Detection Demo Picture**

Programming APIs

For details about the Programming APIs, refer to *Vitis AI Library Programming Guide (UG1355)*. You can download it from the Xilinx websitehttps://www.xilinx.com/products/design-tools/ai-inference/aideveloper-hub.html#edge.

Also, for the Vitis AI APIs, refer to the *Vitis AI User Guide (UG1414)*. You can download it from the Xilinx website https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

Performance

This chapter describes in detail the performance of the Vitis AI Library on the following different boards.

- ZCU102 (0432055-05)
- ZCU104

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 Vitis AI Library varies between the two hardware versions (because of different DDR performance). Since 0432055-04 version of ZCU102 has been discontinued, the following table only shows the performance of ZCU102 (0432055-05). In ZCU102 board, triple B4096F DPU cores are implemented in program logic.

Refer to the following table for throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-05) with DPU running at 281 MHz.

Table 23: **ZCU102 (0432055-05) Performance**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	inception_resnet_v2_tf	299x299	26.4	22.9	49.1
2	inception_v1_tf	224x224	3.0	167.2	434.8
3	inception_v3_tf	299x299	11.5	54.7	129.7
4	inception_v4_2016_09_09_tf	299x299	24.6	27.7	67.5
5	mobilenet_v1_0_25_128_tf	128x128	0.027	836.1	2270.7
6	mobilenet_v1_0_5_160_tf	160x160	0.15	566.7	1816.9
7	mobilenet_v1_1_0_224_tf	224x224	1.1	256.1	763.7
8	mobilenet_v2_1_0_224_tf	224x224	0.60	213.6	575.2
9	mobilenet_v2_1_4_224_tf	224x224	1.2	158.7	395.4
10	resnet_v1_101_tf	224x224	14.4	42.5	90.7

Table 23: **ZCU102 (0432055-05) Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
11	resnet_v1_152_tf	224x224	21.8	29.3	63.3
12	resnet_v1_50_tf	224x224	7.0	76.4	159.4
13	vgg_16_tf	224x224	31.0	19	41.8
14	vgg_19_tf	224x224	39.3	16.5	37.8
15	ssd_mobilenet_v1_coco_tf	300x300	2.5	90	320.6
16	ssd_mobilenet_v2_coco_tf	300x300	3.8	61.8	196.6
17	ssd_resnet_50_fpn_coco_tf	640x640	178.4	1.3	5.9
18	yolov3_voc_tf	416x416	65.6	13.4	37.8
19	mlperf_ssd_resnet34_tf	1200x1200	433	1.9	7.7
20	resnet50	224x224	7.7	71.1	150.2
21	resnet18	224x224	3.7	171.1	437.6
22	inception_v1	224x224	3.2	162.2	422.4
23	inception_v2	224x224	4.0	133	321.4
24	inception_v3	299x299	11.4	54.8	131.2
25	inception_v4	299x299	24.5	27.7	67.4
26	mobilenet_v2	224x224	0.6	210.1	557.4
27	squeezenet	227x227	0.76	264.5	1121.6
28	ssd_pedestrain_pruned_0_97	360x360	5.9	76	306.1
29	ssd_traffic_pruned_0_9	360x480	11.6	55.4	214.2
30	ssd_adas_pruned_0_95	360x480	6.3	82.6	299
31	ssd_mobilenet_v2	360x480	6.6	38.7	117.8
32	refinedet_pruned_0_8	360x480	25	31.6	106
33	refinedet_pruned_0_92	360x480	10.1	59.6	206.2
34	refinedet_pruned_0_96	360x480	5.1	82.3	292.6
35	vpgnet_pruned_0_99	480x640	2.5	101.8	401.1
36	fpn	256x512	8.9	58.6	186.7
37	sp_net	128x224	0.55	511.6	1386.4
38	openpose_pruned_0_3	368x368	49.9	3.5	15.6
39	densebox_320_320	320x320	0.49	383	1363.7
40	densebox_640_360	360x640	1.1	190.7	637.8
41	face_landmark	96x72	0.14	779.6	1348
42	reid	80x160	0.95	343.3	659.4
43	multi_task	288x512	14.8	35.5	133.2
44	yolov3_adas_pruned_0_9	256x512	5.5	82	227.3
45	yolov3_voc	416x416	65.4	13.5	38.2
46	yolov3_bdd	288x512	53.7	12.9	37.5
47	yolov2_voc	448x448	34	24.7	76.2

Table 23: **ZCU102 (0432055-05) Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
48	yolov2_voc_pruned_0_66	448x448	11.6	53.1	203.7
49	yolov2_voc_pruned_0_71	448x448	9.9	59.7	235.9
50	yolov2_voc_pruned_0_77	448x448	7.8	67.8	281.6

ZCU104 Performance

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

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU104 with DPU running at 300 MHz.

Table 24: **ZCU104 Performance**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	inception_resnet_v2_tf	299x299	26.4	25.1	45.4
2	inception_v1_tf	224x224	3.0	192.5	383.8
3	inception_v3_tf	299x299	11.5	59.3	112.7
4	inception_v4_2016_09_09_tf	299x299	24.6	29.9	57.7
5	mobilenet_v1_0_25_128_tf	128x128	0.027	1233	3863.9
6	mobilenet_v1_0_5_160_tf	160x160	0.15	739.9	1929.3
7	mobilenet_v1_1_0_224_tf	224x224	1.1	304.4	672.3
8	mobilenet_v2_1_0_224_tf	224x224	0.60	245.3	519.3
9	mobilenet_v2_1_4_224_tf	224x224	1.2	180.8	369.1
10	resnet_v1_101_tf	224x224	14.4	46.8	85.6
11	resnet_v1_152_tf	224x224	21.8	32.2	59.2
12	resnet_v1_50_tf	224x224	7.0	84.9	152.2
13	vgg_16_tf	224x224	31.0	20.8	37.1
14	vgg_19_tf	224x224	39.3	18.1	33
15	ssd_mobilenet_v1_coco_tf	300x300	2.5	92.8	315.8
16	ssd_mobilenet_v2_coco_tf	300x300	3.8	65.3	177.6
17	ssd_resnet_50_fpn_coco_tf	640x640	178.4	1.4	6.1
18	yolov3_voc_tf	416x416	65.6	14.1	29.3
19	mlperf_ssd_resnet34_tf	1200x1200	433	1.9	5.6

Table 24: **ZCU104 Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
20	resnet50	224x224	7.7	79.1	142.7
21	resnet18	224x224	3.7	193	382.1
22	inception_v1	224x224	3.2	184.7	371.4
23	inception_v2	224x224	4.0	149.7	285
24	inception_v3	299x299	11.4	59.5	113.6
25	inception_v4	299x299	24.5	29.9	57.6
26	mobilenet_v2	224x224	0.6	244.2	510.5
27	squeezenet	227x227	0.76	270.6	1060.4
28	ssd_pedestrain_pruned_0_97	360x360	5.9	78.1	192.8
29	ssd_traffic_pruned_0_9	360x480	11.6	57.2	133.1
30	ssd_adas_pruned_0_95	360x480	6.3	84.5	197.5
31	ssd_mobilenet_v2	360x480	6.6	25.3	108.4
32	refinedet_pruned_0_8	360x480	25	32.4	75
33	refinedet_pruned_0_92	360x480	10.1	60.9	137.8
34	refinedet_pruned_0_96	360x480	5.1	83.1	193.2
35	vpgnet_pruned_0_99	480x640	2.5	104.9	351.3
36	fpn	256x512	8.9	61	162.7
37	sp_net	128x224	0.55	534.9	1147.4
38	openpose_pruned_0_3	368x368	49.9	3.7	11.1
39	densebox_320_320	320x320	0.49	389.5	1342.9
40	densebox_640_360	360x640	1.1	196.7	661.5
41	face_landmark	96x72	0.14	837.2	1171.7
42	reid	80x160	0.95	365.3	619.2
43	multi_task	288x512	14.8	36	107.3
44	yolov3_adas_pruned_0_9	256x512	5.5	83.2	208.8
45	yolov3_voc	416x416	65.4	14.2	29.6
46	yolov3_bdd	288x512	53.7	13.5	28.7
47	yolov2_voc	448x448	34	26.1	58.5
48	yolov2_voc_pruned_0_66	448x448	11.6	55.4	144.2
49	yolov2_voc_pruned_0_71	448x448	9.9	62.3	169.3
50	yolov2_voc_pruned_0_77	448x448	7.8	70.4	208.7

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

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