Vitis AI Library User Guide

UG1354 (v1.3) February 3, 2021

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
02/03/2021 Version 1.3	
Entire document	Updated links. Updated the performance data in the Chapter 7: Performance chapter.
12/17/2020 Version 1.3	
Chapter 1: Introduction	Added the Vitis AI Library 1.3 Release Notes. Updated the block diagram in the Overview section
Chapter 2: Installation	Update the whole chapter.
Chapter 3: Libraries and Samples	Added eight new libraries incuding Retinaface, Face Quality, Hourglass, and Pointpillars.
Chapter 7: Performance	Updated the performance data for all the platform.
Chapter 8: API Reference	Updated the API References.
07/21/2020 Version 1.2	
Entire document	Minor changes
07/13/2020 Version 1.2	
Chapter 1: Introduction	Added Vitis AI Library 1.2 Release Notes Updated the block diagram in the Overview section
Chapter 2: Installation	Update the whole chapter.
Chapter 3: Libraries and Samples	Added manipulation methods for multiple elf models. Added Face Recognition, Plate Detection, Plate Recognition, and Medical Segmentation.
Chapter 6: Programming APIs	Updated this chapter and all the APIs are introduced as appendices in this document.
Chapter 7: Performance	Added the performance data of U280 and U50LV Updated the performance data of U50, ZCU102, and ZCU104
03/23/2020 Version 1.1	
Chapter 1: Introduction	Added Vitis AI Library 1.1 Release Notes
Chapter 2: Installation	Added content for Cloud operation. Updated the Setting Up the Host section.
Chapter 7: Performance	Added the performance data of U50

Table of Contents

XILINX

Chapter 1

Introduction

About this Document

Related Libraries

The following Vitis™ AI Libraries are related to this document.

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

No	Package Name	Version
1	vitis_ai_library_r1.3.0_video.tar.gz	r1.3.0
2	vitis_ai_library_r1.3.0_image.tar.gz	r1.3.0
3	vitis-ai-runtime-1.3.0.tar.gz	r1.3.0
4	vitis_ai_2020.2-r1.3.0.tar.gz	r1.3.0
6	alveo_xclbin-1.3.0.tar.gz	r1.3.0
7	sdk-2020.2.0.0.sh	2020.2

Intended Audience

The target users of Vitis AI libraries are as follows:

- Users who want to use Xilinx 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 Vitis AI's post processing library.

***Note*:** If you have custom models that are completely different from the model supported by the Vitis 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 Vitis AI Library.

- Chapter 1: Introduction is an introduction to the Vitis AI Library. This chapter provides a clear understanding of the Vitis AI Library in general, its framework, supported networks, supported hardware platforms and so on.
- Chapter 2: Installation describes how to install the Vitis AI Library and run the examples. The information in this chapter will help quickly set up the host and target environments, compile and execute the Vitis AI Library related examples.
- Chapter 3: Libraries and Samples describes each model library supported by the Vitis AI Library. This chapter provides an understanding of the model libraries supported by the Vitis 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: Programming Examples describes how to develop applications with Vitis 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 Vitis AI Library
  - Implementing your post-processing code
- Chapter 5: Application Demos 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: Programming APIs describes how to find the programming APIs.
- Chapter 7: Performance describes the performance of the Vitis AI library on different boards.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - Chapter 2: Installation
  - Chapter 4: Programming Examples
  - Chapter 5: Application Demos

- **Host Software Development:** Developing the application code, accelerator development, including library, XRT, and Graph API use. Topics in this document that apply to this design process include:
  - Chapter 3: Libraries and Samples
  - Chapter 6: Programming APIs
  - Chapter 8: API Reference
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Chapter 4: Programming Examples
- **System Integration and Validation:** Integrating and validating the system functional performance, including timing, resource use, and power closure. Topics in this document that apply to this design process include:
  - Chapter 7: Performance

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 2020.2.

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 you to focus more on the development of your applications, rather than the underlying hardware.

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

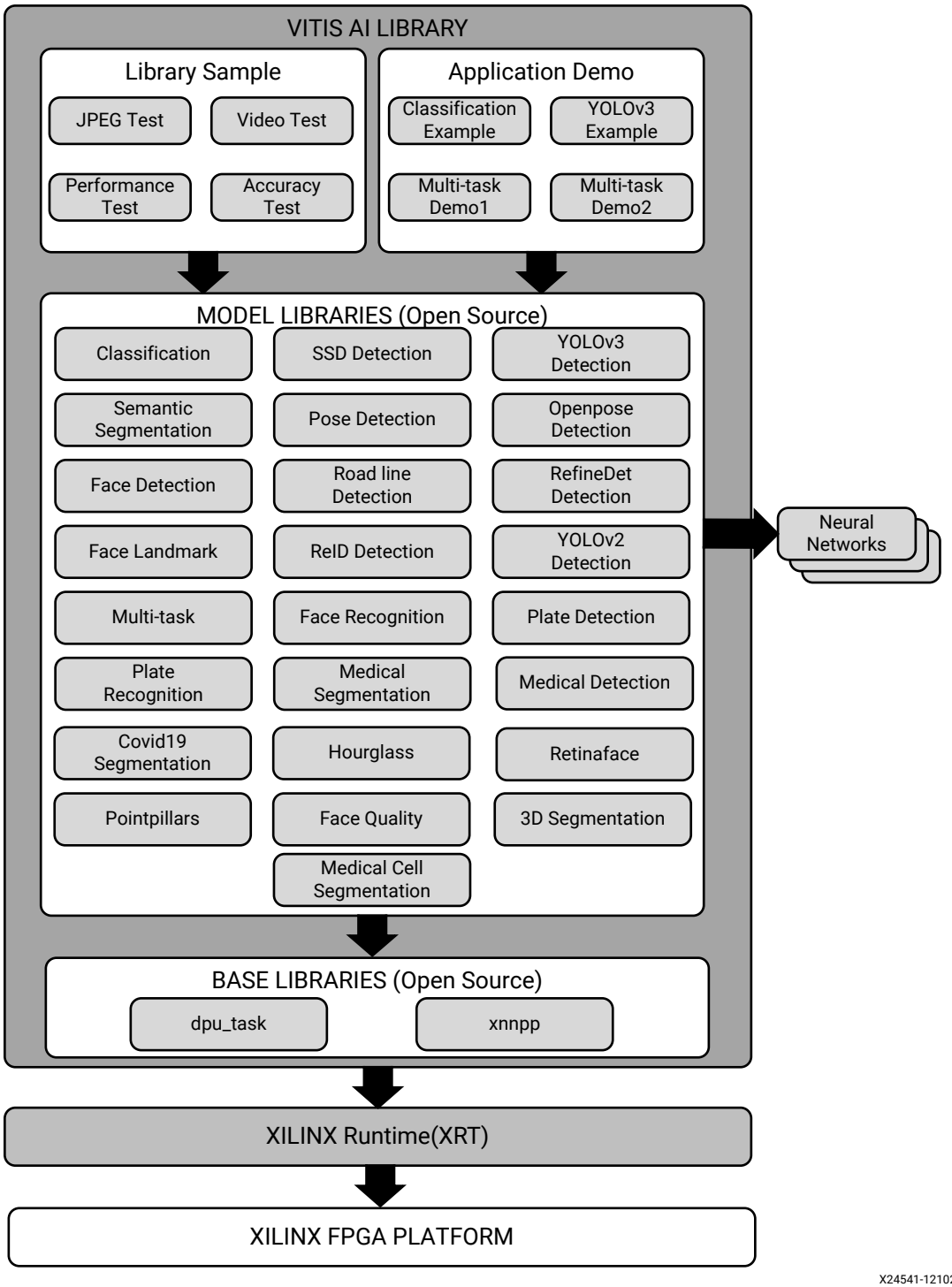
Block Diagram

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. *dpu_task* is the interface library for DPU operations. *xnnpp* is the post-processing library of each model, with built-in modules such as optimization and acceleration.

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 you how to use the Vitis 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 bottom
- Optimized pre-processing 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 1.3 Release Notes

This section contains information regarding the features and updates of the Vitis™ AI Library 1.3 release.

Key Features And Enhancements

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

- **New Board Support:** Versal Series: VCK190 is supported in this release.
- **Xmodel Support for Edge:** For Edge, the xmodel is used, which is consistent with the xmodel used on Cloud.
- **PyTorch Framework Support for Edge:** In VAI1.3, PyTorch framework is supported for Edge.
- **New DPU Support:**

  Enable new DPU target DPUCAHX8L designed for Alveo U50, U50LV, and U280 devices.
- **New Model Libraries:** The following new model libraries are supported.
  - `pointpillars detection`
  - `covid19 segmentation`
  - `medical detection`
  - `medical cell segmentation`
  - `hourglass pose detection`
  - `retinaface detection`

  - face quality
  - 3D segmentation
- **New Model Support:**
  - Added 13 new PyTorch models
  - Added 17 new TensorFlow models, including five TensorFlow2 models.
  - Added six new Caffe models

Changes

- For Edge, xmodel is used and elf model is no longer supported.

Compatibility

The Vitis™ AI Library 1.3 is tested with the following images.

- xilinx-zcu102-dpu-v2020.2-v1.3.0.img.gz
- xilinx-zcu104-dpu-v2020.2-v1.3.0.img.gz

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	ZCU102/ ZCU104	VCK190	U50/U50LV/ U280	U50-V3ME/ U50LV-V3ME/ U280-V3ME	Application
1	inception_resnet_v2_tf	Y	Y	Y	Y	Image Classification
2	inception_v1_tf	Y	Y	Y	Y	
3	inception_v3_tf	Y	Y	Y	Y	
4	inception_v4_2016_09_09_tf	Y	Y	Y	Y	
5	mobilenet_v1_0_25_128_tf	Y	N/A	N/A	N/A	
6	mobilenet_v1_0_5_160_tf	Y	N/A	N/A	Y	
7	mobilenet_v1_1_0_224_tf	Y	N/A	N/A	Y	
8	mobilenet_v2_1_0_224_tf	Y	N/A	N/A	Y	
9	mobilenet_v2_1_4_224_tf	Y	N/A	N/A	Y	
10	resnet_v1_101_tf	Y	Y	Y	Y	
11	resnet_v1_152_tf	Y	Y	Y	Y	
12	resnet_v1_50_tf	Y	Y	Y	Y	
13	vgg_16_tf	Y	Y	Y	Y	
14	vgg_19_tf	Y	Y	Y	Y	
15	ssd_mobilenet_v1_coco_tf	Y	N/A	N/A	Y	Object Detection
16	ssd_mobilenet_v2_coco_tf	Y	N/A	N/A	Y	
17	ssd_resnet_50_fpn_coco_tf	Y	Y	Y	N/A	
18	yolov3_voc_tf	Y	Y	Y	N/A	
19	mlperf_ssd_resnet34_tf	Y	Y	Y	Y	
20	resnet50	Y	Y	Y	Y	Image Classification
21	resnet18	Y	Y	Y	Y	
22	inception_v1	Y	Y	Y	Y	
23	inception_v2	Y	Y	Y	Y	
24	inception_v3	Y	Y	Y	Y	
25	inception_v4	Y	Y	Y	Y	
26	mobilenet_v2	Y	N/A	N/A	Y	
27	squeezenet	Y	Y	Y	Y	
28	ssd_pedestrian_pruned_0_97	Y	Y	Y	Y	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Y	Y	Y	Y	Traffic Detection
30	ssd_adas_pruned_0_95	Y	Y	Y	Y	ADAS Vehicle Detection

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

No.	Neural Network	ZCU102/ ZCU104	VCK190	U50/U50LV/ U280	U50-V3ME/ U50LV-V3ME/ U280-V3ME	Application
31	ssd_mobilenet_v2	Y	N/A	N/A	Y	Object Detection
32	refinedet_pruned_0_8	Y	Y	Y	Y	
33	refinedet_pruned_0_92	Y	Y	Y	Y	
34	refinedet_pruned_0_96	Y	Y	Y	Y	
35	vpgnet_pruned_0_99	Y	Y	Y	Y	ADAS Lane Detection
36	fpn	Y	Y	Y	Y	ADAS Segmentation
37	sp_net	Y	Y	Y	Y	Pose Estimation
38	openpose_pruned_0_3	Y	Y	Y	Y	
39	densebox_320_320	Y	Y	Y	Y	Face Detection
40	densebox_640_360	Y	Y	Y	Y	
41	face_landmark	Y	Y	Y	Y	Face Detection and Recognition
42	reid	Y	Y	Y	Y	Object tracking
43	multi_task	Y	Y	Y	Y	ADAS
44	yolov3_adas_pruned_0_9	Y	Y	Y	N/A	Object Detection
45	yolov3_voc	Y	Y	Y	N/A	
46	yolov3_bdd	Y	Y	Y	N/A	
47	yolov2_voc	Y	Y	Y	N/A	
48	yolov2_voc_pruned_0_66	Y	Y	Y	N/A	
49	yolov2_voc_pruned_0_71	Y	Y	Y	N/A	
50	yolov2_voc_pruned_0_77	Y	Y	Y	N/A	
51	facerec_resnet20	Y	Y	Y	Y	Face Recognition
52	facerec_resnet64	Y	Y	Y	Y	
53	plate_detection	Y	Y	Y	Y	Plate Recognition
54	plate_recognition	Y	Y	Y	N/A	
55	FPN_Res18_Medical_segmentation	Y	Y	Y	Y	Medical Segmentation
56	refinedet_baseline	Y	Y	Y	Y	Object Detection
57	resnet50_pt	Y	Y	Y	Y	Image Classification
58	squeezenet_pt	Y	Y	Y	Y	
59	inception_v3_pt	Y	Y	Y	Y	
60	personreid-res50_pt	Y	Y	Y	Y	Object Tracking
61	facereid-large_pt	Y	Y	Y	N/A	
62	facereid-small_pt	Y	Y	Y	Y	

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

No.	Neural Network	ZCU102/ ZCU104	VCK190	U50/U50LV/ U280	U50-V3ME/ U50LV-V3ME/ U280-V3ME	Application
63	SemanticFPN_cityscapes_pt	Y	Y	Y	Y	Segmentation
64	facerec-resnet20_mixed_pt	Y	Y	Y	Y	Face Recognition
65	face-quality_pt	Y	Y	Y	Y	
66	MT-resnet18_mixed_pt	Y	N/A	N/A	N/A	ADAS
67	salsanext_pt	Y	Y	Y	Y	Point Cloud
68	pointpillars_kitti_12000_0_pt pointpillars_kitti_12000_1_pt	Y	N/A	N/A	N/A	
69	unet_chaos-CT_pt	Y	Y	Y	N/A	CT Segmentation
70	FPN-resnet18_covid19-seg_pt	Y	Y	Y	Y	Covid-19 Segmentation
71	ENet_cityscapes_pt	Y	Y	Y	Y	Segmentation
72	personreid-res18_pt	Y	Y	Y	N/A	Object Tracking
73	yolov4_leaky_spp_m	Y	Y	Y	N/A	Object Detection
74	hourglass-pe_mpii	Y	N/A	N/A	N/A	Pose Estimation
75	retinaface	Y	N/A	N/A	N/A	Face Detection
76	FPN-resnet18_Endov	Y	N/A	N/A	N/A	Robot Instrument Segmentation
77	tiny_yolov3_vmss	Y	Y	Y	N/A	Object Detection
78	face-quality	Y	Y	Y	Y	Face Recognition
79	ssdlite_mobilenet_v2_coco_tf	Y	N/A	N/A	Y	Object Detection
80	ssd_inception_v2_coco_tf	Y	N/A	N/A	N/A	
81	MLPerf_resnet50_v1.5_tf	Y	Y	Y	Y	Image Classification
82	mobilenet_edge_1_0_tf	Y	N/A	N/A	N/A	
83	mobilenet_edge_0_75_tf	Y	N/A	N/A	N/A	
84	refinedet_VOC_tf	Y	Y	Y	Y	Object Detection
85	RefineDet-Medical_EDD_tf	Y	Y	Y	Y	Medical Detection
86	resnet_v2_50_tf	Y	N/A	N/A	N/A	Image Classification
87	resnet_v2_101_tf	Y	N/A	N/A	N/A	
88	resnet_v2_152_tf	Y	N/A	N/A	N/A	
89	mobilenet_v2_cityscapes_tf	Y	N/A	N/A	N/A	Segmentation

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

No.	Neural Network	ZCU102/ ZCU104	VCK190	U50/U50LV/ U280	U50-V3ME/ U50LV-V3ME/ U280-V3ME	Application
90	inception_v2_tf	Y	N/A	N/A	Y	Image Classification
91	resnet50_tf2	Y	Y	Y	Y	
92	mobilenet_1_0_224_tf2	Y	N/A	N/A	Y	
93	inception_v3_tf2	Y	Y	Y	Y	
94	medical_seg_cell_tf2	Y	Y	Y	Y	Medical Segmentation
95	semantic_seg_citys_tf2	Y	Y	Y	Y	Segmentation

Notes:

1. Networks with the suffix "_tf" or "_tf2" were trained on TensorFlow.
2. Networks with the suffix "_pt" were trained on PyTorch.
3. Networks with no suffix were trained on Caffe.
4. The column of U50-V3ME/U50LV-V3ME/U280-V3ME is for DPUCAHX8L.

Device Support

The following platforms and evaluation boards (EVB) are supported by the Vitis™ AI Library 1.3.

Table 3: **Edge Device Support**

Platform	EVB	Version
Zynq UltraScale+ MPSoC ZU9EG	Xilinx ZCU102	V1.1
Zynq® UltraScale+™ MPSoC ZU7EV	Xilinx ZCU104	V1.0
Versal AI Core series VC1902	Xilinx VCK190	V1.0

Table 4: **Cloud Board Support**

Accelerator Cards
Xilinx Alveo U50
Xilinx Alveo U50LV
Xilinx Alveo U200
Xilinx Alveo U250
Xilinx Alveo U280

Limitations

- Some neural networks with mobilenet as the backbone are not supported on U50, U50LV, U280, and VCK190.

- Due to limitations of the Docker environment, Multi-Task demos cannot run in the DRM mode on Cloud boards.

Vitis AI Library 1.2 Release Notes

This section contains information regarding the features and updates of the Vitis™ AI Library 1.2 release.

Key Features And Enhancements

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

- **New Cloud Board Support:** Alveo™ U50LV and U280 cards are now supported in this release.
- **New Model Libraries:** The following new model libraries are supported.
  - `face recognition`
  - `plate detection`
  - `plate recognition`
  - `medical segmentation`
- **PyTorch Model Support:** Three PyTorch models are supported.
- **Support for new Caffe Models:** Six new Caffe models are supported.

Changes

- The installation mode of the target for the Edge is changed and the RPM format package is used.
- `meta.json` file in the model has been deprecated.

Compatibility

The Vitis™ AI Library 1.2 is tested with the following images.

- `xilinx-zcu102-dpu-v2020.1-v1.2.0.img.gz`
- `xilinx-zcu104-dpu-v2020.1-v1.2.0.img.gz`

Model Support

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

Table 5: **Model Supported by the AI Library**

No.	Neural Network	ZCU102/ZCU104	U50/U50LV/ U280	Application
1	inception_resnet_v2_tf	Y	Y	Image Classification
2	inception_v1_tf	Y	Y	
3	inception_v3_tf	Y	Y	
4	inception_v4_2016_09_09_tf	Y	Y	
5	mobilenet_v1_0_25_128_tf	Y	N/A	
6	mobilenet_v1_0_5_160_tf	Y	N/A	
7	mobilenet_v1_1_0_224_tf	Y	N/A	
8	mobilenet_v2_1_0_224_tf	Y	N/A	
9	mobilenet_v2_1_4_224_tf	Y	N/A	
10	resnet_v1_101_tf	Y	Y	
11	resnet_v1_152_tf	Y	Y	
12	resnet_v1_50_tf	Y	Y	
13	vgg_16_tf	Y	Y	
14	vgg_19_tf	Y	Y	
15	ssd_mobilenet_v1_coco_tf	Y	N/A	Object Detection
16	ssd_mobilenet_v2_coco_tf	Y	N/A	
17	ssd_resnet_50_fpn_coco_tf	Y	Y	
18	yolov3_voc_tf	Y	Y	
19	mlperf_ssd_resnet34_tf	Y	N/A	
20	resnet50	Y	Y	Image Classification
21	resnet18	Y	Y	
22	inception_v1	Y	Y	
23	inception_v2	Y	Y	
24	inception_v3	Y	Y	
25	inception_v4	Y	Y	
26	mobilenet_v2	Y	N/A	
27	squeezenet	Y	Y	
28	ssd_pedestrian_pruned_0_97	Y	Y	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Y	Y	Traffic Detection
30	ssd_adas_pruned_0_95	Y	Y	ADAS Vehicle Detection
31	ssd_mobilenet_v2	Y	N/A	Object Detection
32	refinedet_pruned_0_8	Y	Y	
33	refinedet_pruned_0_92	Y	Y	
34	refinedet_pruned_0_96	Y	Y	
35	vpgnet_pruned_0_99	Y	Y	ADAS Lane Detection
36	fpn	Y	Y	ADAS Segmentation
37	sp_net	Y	Y	Pose Estimation

Table 5: **Model Supported by the AI Library** *(cont'd)*

No.	Neural Network	ZCU102/ZCU104	U50/U50LV/ U280	Application
38	openpose_pruned_0_3	Y	Y	
39	densebox_320_320	Y	Y	Face Detection
40	densebox_640_360	Y	Y	
41	face_landmark	Y	Y	Face Detection and Recognition
42	reid	Y	Y	Object tracking
43	multi_task	Y	Y	ADAS
44	yolov3_adas_pruned_0_9	Y	Y	Object Detection
45	yolov3_voc	Y	Y	
46	yolov3_bdd	Y	Y	
47	yolov2_voc	Y	Y	
48	yolov2_voc_pruned_0_66	Y	Y	
49	yolov2_voc_pruned_0_71	Y	Y	
50	yolov2_voc_pruned_0_77	Y	Y	
51	facerec_resnet20	Y	Y	Face Recognition
52	facerec_resnet64	Y	Y	
53	plate_detection	Y	Y	Plate Recognition
54	plate_recognition	Y	Y	
55	FPN_Res18_Medical_segmentation	Y	Y	Medical Segmentation
56	refinedet_baseline	Y	Y	Object Detection
57	resnet50_pt	N/A	Y	Image Classification
58	squeezenet_pt	N/A	Y	
59	inception_v3_pt	N/A	Y	

Notes:

1. No1-No19 neural network models are trained based on the TensorFlow framework.
2. No20-No56 neural network models are trained based on the Caffe framework.
3. No57-No59 neural network models are trained based on the PyTorch framework.

Device Support

The following platforms and EVBs are supported by the Vitis™ AI Library 1.2.

Table 6: **Edge Device Support**

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

Table 7: **Cloud Device Support**

Accelerator Cards
Xilinx Alveo U50
Xilinx Alveo U50lv
Xilinx Alveo U280

Limitations

- Some neural networks with mobilenet as the backbone are not supported on U50, U50lv, and U280.
- PyTorch models are not supported for edge devices.
- Due to limitations of the Docker environment, Multi-Task demos cannot run in the DRM mode on Cloud devices.

Vitis AI Library 1.1 Release Notes

This section contains information regarding the features and updates of the Vitis™ AI Library 1.1 release.

Key Features And Enhancements

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

- **Support for the Cloud:** The Alveo U50 card is supported with this release.
- **New DPU support:** DPU is supported which can be used for the cloud.
- **New Open Source Library:** The `xnnpp` library is open source in this release, which shows how to do the pre-processing and post-processing for the neural networks.
- **Model Library Update:** The new model library unifies the interface between the cloud and edge.

Changes

The installation mode of the host for the Edge is changed, and the original Docker installation mode is no longer used.

Compatibility

The Vitis™ AI Library 1.1 is tested with the following images.

- `xilinx-zcu102-dpu-v2019.2-v2.img`

- `xilinx-zcu104-dpu-v2019.2-v2.img`

Model Support

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

Table 8: **Model Supported by the Vitis AI Library**

No.	Neural Network	ZCU102/ZCU104	U50	Application
1	inception_resnet_v2_tf	Y	Y	Image Classification
2	inception_v1_tf	Y	Y	
3	inception_v3_tf	Y	Y	
4	inception_v4_2016_09_09_tf	Y	Y	
5	mobilenet_v1_0_25_128_tf	Y	N/A	
6	mobilenet_v1_0_5_160_tf	Y	N/A	
7	mobilenet_v1_1_0_224_tf	Y	N/A	
8	mobilenet_v2_1_0_224_tf	Y	N/A	
9	mobilenet_v2_1_4_224_tf	Y	N/A	
10	resnet_v1_101_tf	Y	Y	
11	resnet_v1_152_tf	Y	Y	
12	resnet_v1_50_tf	Y	Y	
13	vgg_16_tf	Y	Y	
14	vgg_19_tf	Y	Y	
15	ssd_mobilenet_v1_coco_tf	Y	N/A	Object Detection
16	ssd_mobilenet_v2_coco_tf	Y	N/A	
17	ssd_resnet_50_fpn_coco_tf	Y	Y	
18	yolov3_voc_tf	Y	Y	
19	mlperf_ssd_resnet34_tf	Y	N/A	
20	resnet50	Y	Y	Image Classification
21	resnet18	Y	Y	
22	inception_v1	Y	Y	
23	inception_v2	Y	Y	
24	inception_v3	Y	Y	
25	inception_v4	Y	Y	
26	mobilenet_v2	Y	N/A	
27	squeezenet	Y	Y	
28	ssd_pedestrian_pruned_0_97	Y	Y	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Y	Y	Traffic Detection
30	ssd_adas_pruned_0_95	Y	Y	ADAS Vehicle Detection

Table 8: **Model Supported by the Vitis AI Library** *(cont'd)*

No.	Neural Network	ZCU102/ZCU104	U50	Application
31	ssd_mobilenet_v2	Y	N/A	Object Detection
32	refinedet_pruned_0_8	Y	Y	
33	refinedet_pruned_0_92	Y	Y	
34	refinedet_pruned_0_96	Y	Y	
35	vpgnet_pruned_0_99	Y	Y	ADAS Lane Detection
36	fpn	Y	Y	ADAS Segmentation
37	sp_net	Y	Y	Pose Estimation
38	openpose_pruned_0_3	Y	Y	
39	densebox_320_320	Y	Y	Face Detection
40	densebox_640_360	Y	Y	
41	face_landmark	Y	Y	Face Detection and Recognition
42	reid	Y	Y	Object tracking
43	multi_task	Y	Y	ADAS
44	yolov3_adas_pruned_0_9	Y	Y	Object Detection
45	yolov3_voc	Y	Y	
46	yolov3_bdd	Y	Y	
47	yolov2_voc	Y	Y	
48	yolov2_voc_pruned_0_66	Y	Y	
49	yolov2_voc_pruned_0_71	Y	Y	
50	yolov2_voc_pruned_0_77	Y	Y	

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 Library 1.1.

Table 9: **Edge Device Support**

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

Table 10: **Cloud Device Support**

Accelerator Cards
Xilinx Alveo U50

Limitations

Some neural networks with mobilenet as the backbone are not supported on U50.

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 DPU. It provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks.

Key Features And Enhancements

This Vitis AI Library release includes the following key features and enhancements:

- **Support for new Vitis AI runtime:** The 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 DPU for Edge devices, the new AI Library will support new cloud based DPU IPs using the 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 the official TensorFlow repository. The pre-compiled models for edge devices are included, while original models are released by the 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 Tensorflow repository. There is an updated classification demo that shows the usage of the unified APIs in Vitis AI Library 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 11: **Model Supported by the Vitis 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_pedestrian_pruned_0_97	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Traffic Detection
30	ssd_adas_pruned_0_95	ADAS Vehicle Detection

Table 11: **Model Supported by the Vitis AI Library** *(cont'd)*

No.	Neural Network	Application
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
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 Library 1.0.

Table 12: **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 the low-level unified APIs in Vitis AI run time.
- **Removed pre-compiled models:** Six TensorFlow models have been removed in this release but provided in the previous Xilinx AI v2.0 release, to keep sync with the updated Model Zoo. Models that are removed can be replaced by similar models in updated the 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 downloaded for free from the Vitis AI repository on GitHub.

> **RECOMMENDED:** *Use an Alveo™ card or an evaluation board that supports the Vitis AI Library to allow you to become familiar with the product. See the AI Developer Hub for more details about evaluation boards that support the Vitis AI Library. See the Alveo Accelerator Cards product page for more details about Alveo cards.*

This release supports the following evaluation boards:

- Xilinx® ZCU102
- Xilinx ZCU104
- Xilinx VCK190

This release supports the following Alveo cards:

- Alveo U50
- Alveo U50LV
- Alveo U200
- Alveo U250
- Alveo U280

Setting Up the Host

For Edge

Use the following steps to set up the host for Edge device development.

1. Download the `sdk-2020.2.0.0.sh` from here.

2. Install the cross-compilation system environment.

```
./sdk-2020.2.0.0.sh
```

3. Follow the prompts to install.

   Note: The `~/petalinux_sdk` path is recommended for the installation. Regardless of the path you choose for the installation, make sure the path has read-write permissions. In this section, it is installed in `~/petalinux_sdk`.

4. When the installation is complete, follow the prompts and execute the following command:

```
source ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
```

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

5. Download the `vitis_ai_2020.2-r1.3.x.tar.gz` from here and install it to the PetaLinux system.

```
tar -xzvf vitis_ai_2020.2-r1.3.x.tar.gz -C ~/petalinux_sdk/sysroots/
aarch64-xilinx-linux
```

6. Clone the Vitis AI repository:

```
cd ~
git clone --recurse-submodules https://github.com/Xilinx/Vitis-AI
```

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

```
cd ~/Vitis-AI/demo/Vitis-AI-Library/samples/classification
bash -x build.sh
```

   The executable program is now produced.

8. To modify the library source code, view and modify them under `~/Vitis-AI/tools/Vitis-AI-Library`.

   Before compiling the Vitis AI libraries, confirm the compiled output path. The default output path is `$HOME/build`.

   To change the default output path, modify the `build_dir_default` in `cmake.sh`. For example, you can change from `build_dir_default=$HOME/build/build.${target_info}/${project_name}` to `build_dir_default=/workspace/build/build.${target_info}/${project_name}`.

   Note: If you want to modify the `build_dir_default`, it is suggested to modify `$HOME` only.

9. Build the libraries all at once by executing the following command.

```
cd ~/Vitis-AI/tools/Vitis-AI-Library
./cmake.sh --clean
```

After compiling, you can find the generated AI libraries under `build_dir_default`. If you want to change the compilation rules, check and change the `cmake.sh` in the library's directory.

Note: All the source codes, samples, demos, and head files can be found under `~/Vitis-AI/tools/Vitis-AI-Library`.

For Cloud (U50/U50LV/U280)

Set up the host on the Cloud by running the docker image.

1. Clone the Vitis AI repository.

```
git clone --recurse-submodules https://github.com/Xilinx/Vitis-AI
cd Vitis-AI
```

2. Run Docker container according to the instructions in the docker installation guide.

```
./docker_run.sh -X xilinx/vitis-ai-cpu:<x.y.z>
```

Note: A `workspace` folder is created by the docker runtime system and is 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, locate 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. Select the model for your platform. You can find the latest models download links in the model's yaml files under `Vitis-AI/models/AI-Model-Zoo`.

   - If the `/usr/share/vitis_ai_library/model` folder does not exist, create it first.

     ```
     sudo mkdir -p /usr/share/vitis_ai_library/models
     ```

   - For `DPUCAHX8H` of U50, take `resnet_v1_50_tf` as an example.

     ```
     wget https://www.xilinx.com/bin/public/openDownload?
     filename=resnet_v1_50_tf-u50-r1.3.0.tar.gz -O resnet_v1_50_tf-u50-
     r1.3.0.tar.gz
     tar -xzvf resnet_v1_50_tf-u50-r1.3.0.tar.gz
     sudo cp resnet_v1_50_tf /usr/share/vitis_ai_library/models -r
     ```

   - For `DPUCAHX8L` of U50LV, take `resnet_v1_50_tf` as an example.

     ```
     wget https://www.xilinx.com/bin/public/openDownload?
     filename=resnet_v1_50_tf-u50-u50lv-u280-v3me-r1.3.0.tar.gz -O
     resnet_v1_50_tf-u50-u50lv-u280-v3me-r1.3.0.tar.gz
     tar -xzvf resnet_v1_50_tf-u50-u50lv-u280-v3me-r1.3.0.tar.gz
     sudo cp resnet_v1_50_tf /usr/share/vitis_ai_library/models -r
     ```

5. Download the cloud xclbin package from here. Untar it, select the Alveo card, and install it. Take `U50` as an example.

```
tar -xzvf alveo_xclbin-1.3.0.tar.gz
cd alveo_xclbin-1.3.0/U50/6E300M
sudo cp dpu.xclbin hbm_address_assignment.txt /usr/lib
```

   For `DPUCAHX8L`, take `U50lv` as an example.

```
tar -xzvf alveo_xclbin-1.3.0.tar.gz
cd alveo_xclbin-1.3.0/U50lv-V3ME/1E250M
sudo cp dpu.xclbin /opt/xilinx/overlaybins/
export XLNX_VART_FIRMWARE=/opt/xilinx/overlaybins/dpu.xclbin
```

6. If there is more than one card installed on the server and you want to specify some cards to run the program, you can set `XLNX_ENABLE_DEVICES` to achieve this function. The following is the usage of `XLNX_ENABLE_DEVICES`:

   - `export XLNX_ENABLE_DEVICES=0` --only use device 0 for DPU
   - `export XLNX_ENABLE_DEVICES=0,1,2` --select device 0, device 1 and device 2 to be used for DPU
   - If you do not set this environment variable, use all devices for DPU by default.

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

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

   The executable program is now produced.

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

   Before compiling the AI libraries, confirm the compiled output path. The default output path is: `$HOME/build`.

   If you want to change the default output path, modify the `build_dir_default` in `cmake.sh`. Such as, change from `build_dir_default=$HOME/build/build.${target_info}/${project_name}` to `build_dir_default=/workspace/build/build.${target_info}/${project_name}`.

   Note: If you want to modify the `build_dir_default`, modify `$HOME` only.

   Execute the following command to build the libraries all at once:

```
cd /workspace/tools/Vitis-AI-Library
./cmake.sh --clean
```

   After compiling, you can find the generated AI libraries under `build_dir_default`. If you want to change the compilation rules, check and change the `cmake.sh` in the library's directory.

Scaling Down the Frequency of the DPU

Due to the power limitation of the card, all CNN models on each Alveo card cannot run at the highest frequencies. Sometimes frequency scaling-down operation is necessary.

The DPU core clock is generated from an internal DCM module driven by the platform Clock_1 with the default value of 100 MHz, and the core clock is always linearly proportional to Clock_1. For example, in U50LV-10E275M overlay, the 275 MHz core clock is driven by the 100 MHz clock source. So, to set the core clock of this overlay to 220 MHz, set the frequency of Clock_1 to (220/275)*100 = 80 MHz.

You could use XRT xbutil tools to scale down the running frequency of the DPU overlay before you run the VART/Library examples. Before the frequency scaling-down operation, the overlays should be programmed into the FPGA first. Refer to the following example commands to program the FPGA and scale down the frequency. These commands will set the Clock_1 to 80 MHz and can be run at host or in the docker.

```
/opt/xilinx/xrt/bin/xbutil program -p /usr/lib/dpu.xclbin
/opt/xilinx/xrt/bin/xbutil clock -d0 -g 80
```

`d0` is the Alveo card device number. For more information about `xbutil` tool, see the XRT documents.

For Cloud (U200/U250)

Set up the host on the cloud by running the docker image.

1. Clone the Vitis AI repository.

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

2. Run Docker container according to the instructions in the docker installation guide.

   ```
   $./docker_run.sh -X xilinx/vitis-ai-cpu:<x.y.z>
   ```

   Note: A `workspace` folder is created by the docker runtime system, and is mounted in `/workspace` of the docker runtime system.

3. Activate the conda environment.

   ```
   $conda activate vitis-ai-caffe
   ```

4. To modify the library source code, view and modify them under `/workspace/Vitis-AI-Library`. Before compiling the AI libraries, confirm the compiled output path. The default output path is `$HOME/build`. If you want to change the default output path, modify the `` `build_dir_default` `` in `cmake.sh`.

   ```
   build_dir_default=$HOME/build/build.${target_info}/${project_name}
   to build_dir_default=/workspace/build/build.${target_info}/$
   {project_name}
   ```

Note: If you want to modify the `build_dir_default`, it is suggested to modify `$HOME`.

5. Execute the following command to build all DPUCADX8G supported examples in the AI Library.

```
$cd /workspace/Vitis-AI-Library
$./cmake.sh --clean --type=release --cmake-options=-DCMAKE_PREFIX_PATH=
$CONDA_PREFIX --cmake-options=-DENABLE_DPUCADX8G_RUNNER=ON
```

After successful building, you can find the generated AI libraries and executables under `build_dir_default`.

Note: If you want to change the compilation rules, check and change the `cmake.sh` in the library's directory.

AI Library File Locations

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

Table 13: **AI Library File Location List**

Files	Location
Source code of the libraries	`/workspace/tools/Vitis-AI-Library`
Samples	`/workspace/demo/Vitis-AI-Library/samples`
Apps	`/workspace/demo/Vitis-AI-Library/apps`
Test	`/workspace/tools/Vitis-AI-Library/[model]/test`

Notes:
The following symbols/abbreviations are used.

- `/workspace/` is the path to extract the AI Library compressed package in the docker system.
- “Samples” is used for rapid application construction and evaluation, and it is for users.
- “Apps” 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

There are three steps to set up the target. 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 Vitis AI Library packet.

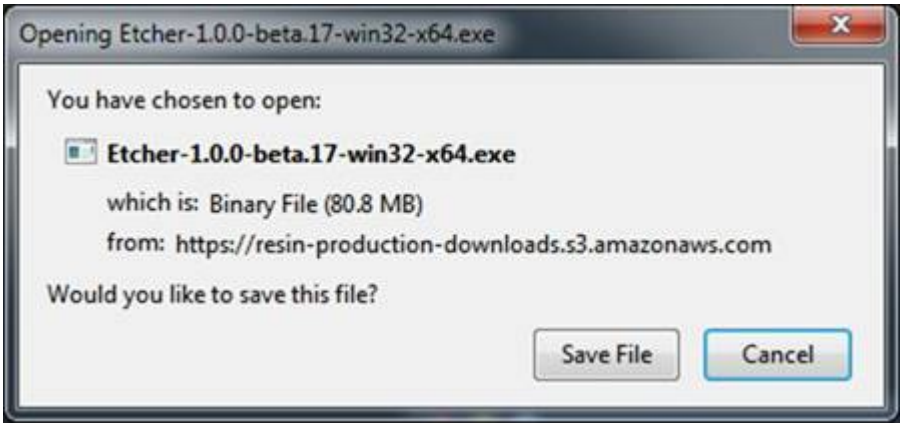
To improve the user experience, the Vitis AI Runtime packages, Vitis-AI-Library samples, and models are built into the board image. Therefore, you do not have to install Vitis AI Runtime packages and model package on the board separately. However, you can still install the model or Vitis AI Runtime on your own image or on the official image by following these steps.

Note: The version of the board image should be 2020.2 or above.

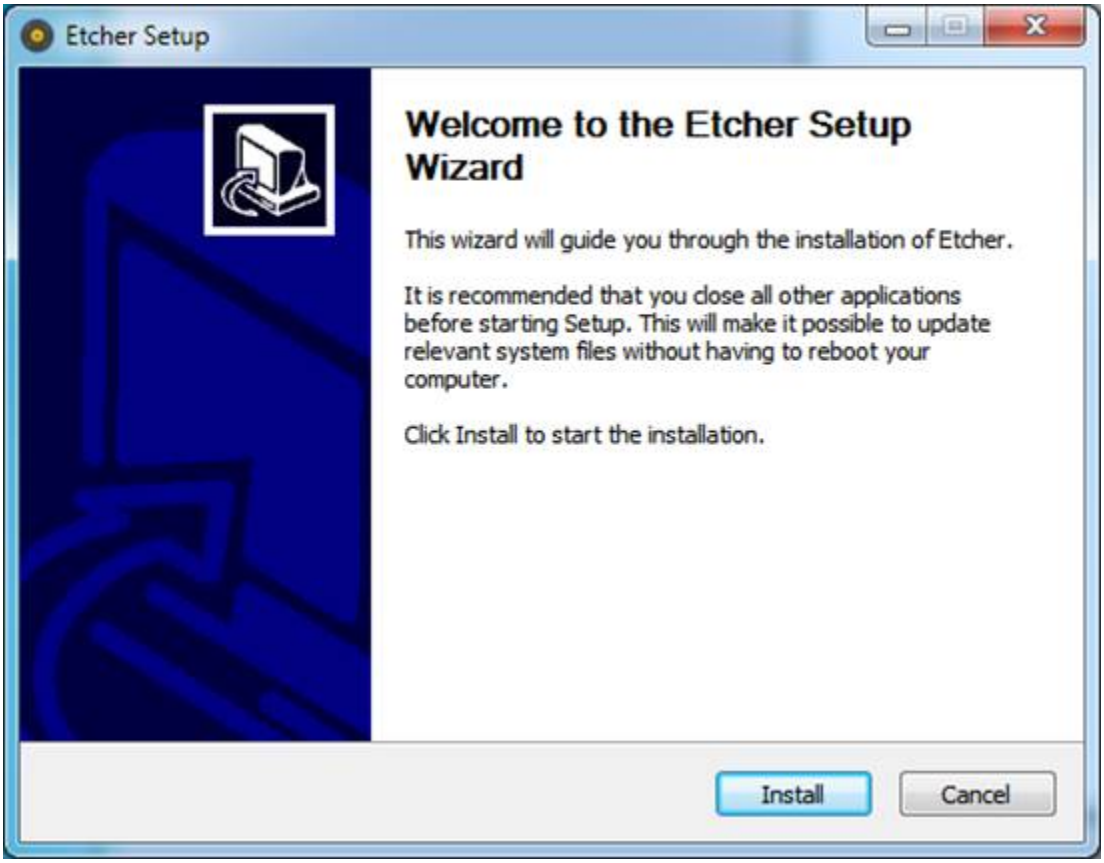
Step 1: Installing a Board Image

For ZCU102, the system images can be downloaded from here; for ZCU104, it can be downloaded from here. One suggested software application for flashing the SD card is Etcher. It is a cross-platform tool for flashing OS images to SD cards, available for Windows, Linux, and Mac systems. The following example uses Windows.

1. Download Etcher from: https://etcher.io/ and save the file as shown in the following figure.

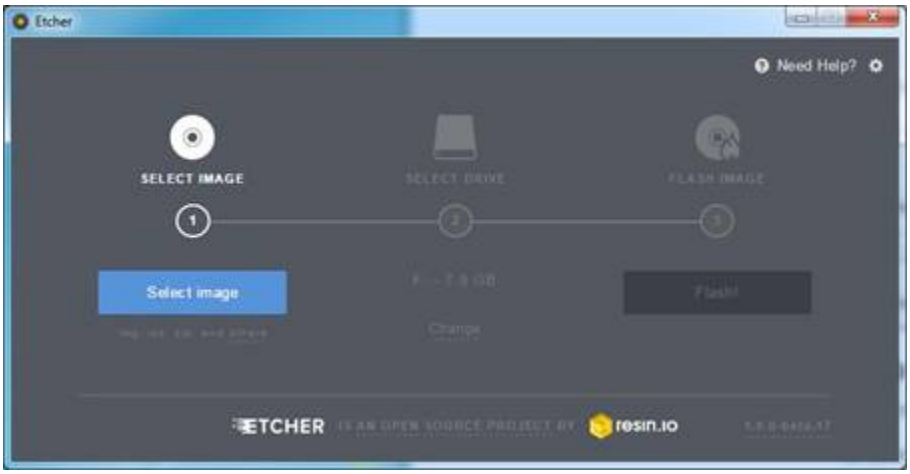


2. Install Etcher, as shown in the following figure.

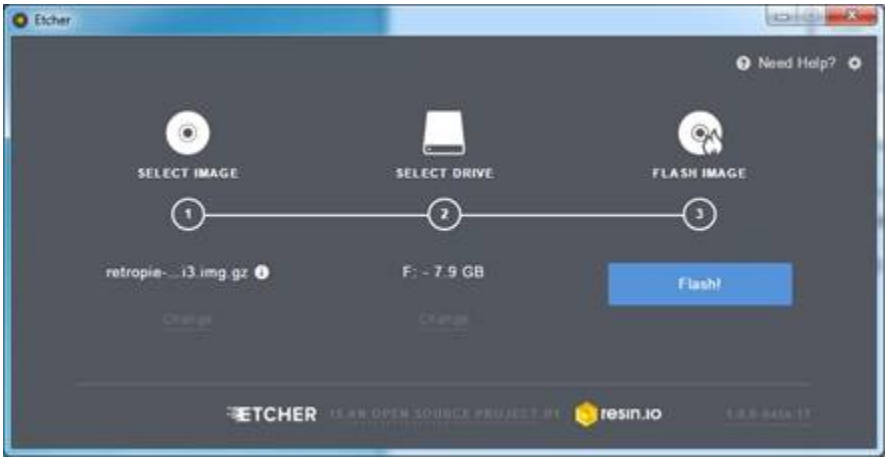


3. Eject any external storage devices such as USB flash drives and backup hard disks. This makes it easier to identify the SD card. Then, insert the SD card into the slot on your computer, or into the reader.
4. Run the Etcher program by double clicking on the Etcher icon shown in the following figure, or select it from the Start menu.

Etcher launches, as shown in the following figure.



5. Select the image file by clicking **Select Image**. You can select a **.zip** or **.gz** compressed file.
6. Etcher tries to detect the SD drive. Verify the drive designation and the image size.
7. Click **Flash!**.



8. Insert the SD card with the image into the destination board.
9. Plug in the power and boot the board using the serial port to operate on the system.
10. 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

The Vitis AI Runtime packages, Vitis-AI-Library samples and models are built into the board image. Therefore, you do not have to install Vitis AI Runtime packages and model package on the board separately. However, you can still install the model on your own image or on the official image by following these steps:

1. For each model, there is a yaml file which is used for describe all the details about the model. In the yaml, you will find the model's download links for different platforms. Choose the corresponding model and download it.
2. Copy the downloaded file to the board using `scp` with the following command.

   ```
   scp <model>.tar.gz root@IP_OF_BOARD:~/
   ```

3. Log in to the board (using ssh or serial port) and install the model package.
4. If the `/usr/share/vitis_ai_library/model` folder does not exist, create it first.

   ```
   mkdir -p /usr/share/vitis_ai_library/models
   ```

5. Install the model on the target side.

   ```
   tar -xzvf <model>.tar.gz
   cp -r <model> /usr/share/vitis_ai_library/models
   ```

   By default, the models are located in the `/usr/share/vitis_ai_library/models` directory on the target side.

Step 3: Installing AI Library Package

The Vitis AI Runtime packages, Vitis-AI-Library samples and models are built into the board image. Therefore, you do not have to install Vitis AI Runtime packages and model package on the board separately. However, you can still install the Vitis AI Runtime on your own image or on the official image by following these steps:

1. Download the `vitis-ai-runtime-1.3.x.tar.gz` from here. Untar it and copy the following files to the board using scp.

   ```
   tar -xzvf vitis-ai-runtime-1.3.x.tar.gz
   scp -r vitis-ai-runtime-1.3.x/aarch64/centos root@IP_OF_BOARD:~/
   ```

   ***Note*:** You can take the RPM 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 independently, while the others are common libraries. The operation command is as follows:

   ```
   rpm2cpio libvitis_ai_library-1.3.0-r<x>.aarch64.rpm | cpio -idmv
   ```

2. Log in to the board using ssh.

You can also use the serial port to log in.

3. Run the `zynqmp_dpu_optimize.sh` script.

```
cd ~/dpu_sw_optimize/zynqmp/
./zynqmp_dpu_optimize.sh
```

4. Install the Vitis AI Library.

```
cd ~/centos
bash setup.sh
```

You can also execute the following command to install the library one by one.

```
cd ~/centos
rpm -ivh --force libunilog-1.3.0-r<x>.aarch64.rpm
rpm -ivh --force libxir-1.3.0-r<x>.aarch64.rpm
rpm -ivh --force libtarget-factory-1.3.0-r<x>.aarch64.rpm
rpm -ivh --force libvart-1.3.0-r<x>.aarch64.rpm
rpm -ivh --force libvitis_ai_library-1.3.0-r<x>.aarch64.rpm
```

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/vitis/ai`

Running Vitis AI Library Examples

Before running the Vitis AI Library examples on Edge or on Cloud, download the vitis_ai_library_r1.3.x_images.tar.gz and vitis_ai_library_r1.3.x_video.tar.gz. The images or videos used in the following example can be found in both packages.

For Edge

The Vitis AI Runtime packages, Vitis AI Library samples and models are built into the board image. You can run the examples directly. If you have a new program, compile it on the host side and copy the executable program to the target.

1. Copy `vitis_ai_library_r1.3.x_images.tar.gz` and `vitis_ai_library_r1.3.x_video.tar.gz` from host to the target using scp with the following command:

```
[Host]$scp vitis_ai_library_r1.3.x_images.tar.gz root@IP_OF_BOARD:~/
[Host]$scp vitis_ai_library_r1.3.x_video.tar.gz root@IP_OF_BOARD:~/
```

2. Untar the image and video packages on the target.

```
cd ~
tar -xzvf vitis_ai_library_r1.3*_images.tar.gz -C Vitis-AI/demo/Vitis-AI-Library
tar -xzvf vitis_ai_library_r1.3*_video.tar.gz -C Vitis-AI/demo/Vitis-AI-Library
```

3. Enter the extracted directory of example in target board and then compile the example. Take `facedetect` as an example.

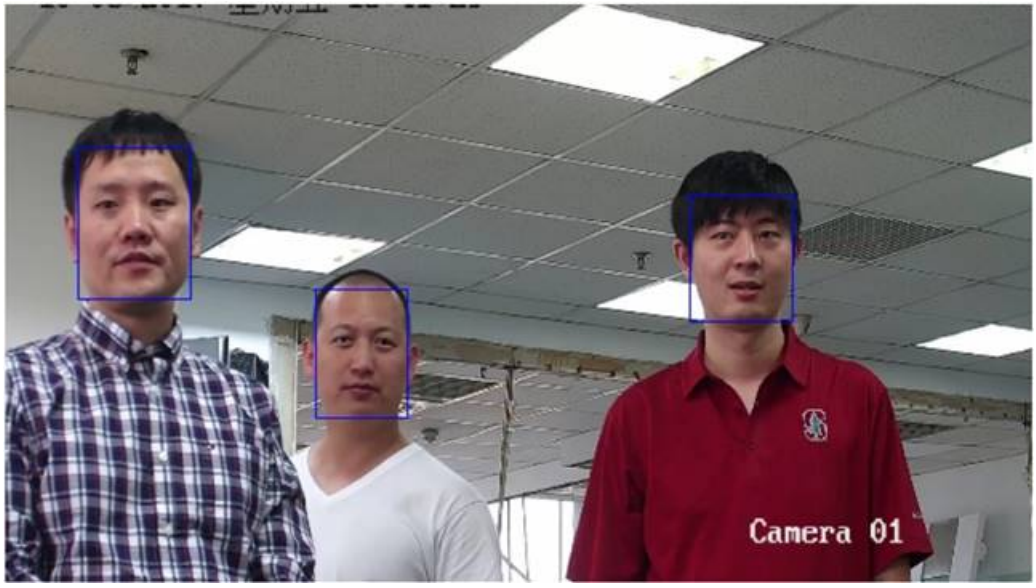
```
cd ~/Vitis-AI/demo/Vitis-AI-Library/samples/facedetect
```

4. Run the example.

```
./test_jpeg_facedetect densebox_320_320 sample_facedetect.jpg
```

5. View the running results.

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



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

```
./test_video_facedetect densebox_320_320 video_input.webm -t 8
```

   where, `video_input.webm` is the name of the video file for input and `-t` is the <num_of_threads>. You must prepare the video file yourself.

   Note:

   - The official system image only supports video file input in `webm` or `raw` format. If you want to use video file in other format as the input, you have to install the relevant packages on the system, such as `ffmpeg` package.

- Due to the limitation of video playback and display in the base platform system, it could only be displayed according to the frame rate of display standard, which could not reflect the real processing performance. But you can check the actual video processing performance, especially with the multithreading, with the following commands:

```
env DISPLAY=:0.0 DEBUG_DEMO=1 ./test_video_facedetect \
densebox_320_320 'multifilesrc location=~/video_input.webm \
! decodebin  !  videoconvert ! appsink sync=false' -t 2
```

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

```
./test_video_facedetect densebox_320_320 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
```

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

```
./test_performance_facedetect densebox_320_320
test_performance_facedetect.list -t 8 -s 60
```

-t: <num_of_threads>

-s: <num_of_seconds>

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

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

For Cloud (U50/U50LV/U280)

If you have downloaded `Vitis-AI`, enter to the `Vitis-AI` directory, and then start Docker.

1. Enter the directory of the sample and then compile it. Take `facedetect` as an example.

```
cd /workspace/demo/Vitis-AI-Library/samples/facedetect
bash -x build.sh
```

2. Run the sample.

```
./test_jpeg_facedetect densebox_320_320 sample_facedetect.jpg
```

3. If you want to run the program in batch mode, which means that the DPU processes multiple images at once to prompt for processing performance, you have to compile the entire Vitis AI Library according to Setting Up the Host section. Then the batch program will be generated under `build_dir_default`. Enter `build_dir_default`, take facedetect as an example, execute the following command.

```
./test_facedetect_batch densebox_320_320 <img1_url> [<img2_url> ...]
```

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

```
./test_video_facedetect densebox_320_320 <video_input.mp4> -t 8
```

`video_input.mp4`: The name of the video file for input. You need to prepare the video file.

-t: <num_of_threads>

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

```
./test_performance_facedetect densebox_320_320
test_performance_facedetect.list -t 8 -s 60
```

- -t: <num_of_threads>
- -s: <num_of_seconds>

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

***Note*:** The performance test program is automatically run in batch mode.

For Cloud (U200/U250)

1. Load and run the Docker Container.

```
$./docker_run.sh -X xilinx/vitis-ai-cpu:<x.y.z>
```

2. Download and untar the model directory [vai_lib_u200_u250_models.tar.gz] package.

```
$cd /workspace/Vitis-AI-Library
$wget -O vai_lib_u200_u250_models.tar.gz https://www.xilinx.com/bin/
public/openDownload?filename=vai_lib_u200_u250_models.tar.gz
$sudo tar -xvf vai_lib_u200_u250_models.tar.gz --absolute-names
```

***Note*:** All models will download to `/usr/share/vitis_ai_library/models` directory. Currently supported networks are classification, facedetect, facelandmark, reid, and yolov3.

3. To download a minimal validation set for Imagenet2012 using Collective Knowledge (CK), refer to the Alveo examples.

4. Setup the environment.

```
$source /workspace/alveo/overlaybins/setup.sh
$export LD_LIBRARY_PATH=$HOME/.local/${taget_info}/lib/:$LD_LIBRARY_PATH
```

5. Make sure to compile the entire Vitis AI Library according to the For Cloud (U50/U50LV/U280) section. Run the classification image test example.

```
$HOME/build/build.${taget_info}/${project_name}/test_classification
<model_dir> <img_path>
```

For example:

```
$~/build/build.Ubuntu.18.04.x86_64.Release/Vitis-AI-Library/
classification/test_classification inception_v1 <img_path>
```

6. Run the classification accuracy test example.

```
$HOME/build/build..${taget_info}/${project_name}/
test_classification_accuracy <model_dir> <img_dir_path> <output_file>
```

For example:

```
$~/build/build.Ubuntu.18.04.x86_64.Release/Vitis-AI-Library/
classification/test_classification_accuracy inception_v1 <img_dir_path>
<output_file>
```

Support

You can visit the Vitis AI Library community forum on the Xilinx website 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
- Face recognition
- Plate detection
- Plate recognition
- Medical segmentation

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

- Classification
- SSD detection
- YOLOv3 detection
- Medical detection

And, the Vitis™ AI supports the following type of neural network libraries based on PyTorch framework.

- Classification
- ReID detection
- Face recognition
- Semantic segmentation
- Point cloud
- Medical segmentation
- 3D segmentation

The related libraries are open source and can be modified as needed. The open source codes are available on Github.

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. Decoding by software libraries on Arm® CPU may have inconsistent decoding time, which may affect the accuracy of evaluation.

***Note*:** For Edge, 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].xmodel`
- `[model_name].prototxt`

***Note*:** The elf model is not supported by the Vitis AI Library in VAI 1.3.

Take the "inception_v1" model as an example. `inception_v1.xmodel` is the model data. `inception_v1.prototxt` is the parameter of the model.

***Note*:** The name of the model directory should be the same with the model name.

Model Type

Classification

The Classification 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 Vitis AI Library integrates networks including, but not limited to, ResNet18, ResNet50, Inception_v1, Inception_v2, Inception_v3, Inception_v4, Vgg, mobilenet_v1, mobilenet_v2, and Squeezenet into Xilinx libraries. The input is a picture with an object and the output is the top-K most probable category.

Figure 2: **Classification Example**

The following table lists the classification models supported by the Vitis AI Library.

Table 14: **Classification Models**

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	
15	mobilenet_edge_1_0_tf	
16	mobilenet_edge_0_75_tf	
17	inception_v2_tf	
18	MLPerf_resnet50_v1.5_tf	
19	resnet50_tf2	
20	mobilenet_1_0_224_tf2	
21	inception_v3_tf2	
22	resnet_v2_50_tf	
23	resnet_v2_101_tf	
24	resnet_v2_152_tf	
25	resnet50	Caffe
26	resnet18	
27	inception_v1	
28	inception_v2	
29	inception_v3	
30	inception_v4	
31	mobilenet_v2	
32	squeezenet	
33	resnet50_pt	PyTorch
34	squeezenet_pt	
35	inception_v3_pt	

Face Detection

The Face Detection library uses the DenseBox neuron network to detect human faces. The input is a picture with the faces you want to detect and the output is a vector of the result structure containing the information of each detection box. The following image shows the result of face detection.

Figure 3: **Face Detection Example**

The following table lists the face detection models supported by the AI Library.

Table 15: **Face Detection Models**

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 on a human face. The five points include the left eye, the right eye, the nose, the left corner of the lips, and the right corner of the lips. This network is used to correct face direction (what this means is if a face is not directly facing the camera (e.g., tilted 20 degrees left or right), it is "adjusted" to face the camera directly) before face feature extraction. The input image should be a face which is detected by the face detection network. The output of the network is the five key points. The five key points are normalized. The following image shows the result of face detection.

Figure 4: **Face Landmark Detection Example**

The following table lists the face landmark models supported by the AI Library.

Table 16: **Face Landmark Models**

No	Model Name	Framework
1	face_landmark	Caffe

SSD Detection

The SSD Detection library is commonly used with the SSD neuron network. SSD is a neural network which is used to detect objects. The input is a picture with some objects you want to detect. The output is a vector of the result structure containing the information of each detection box. The following image shows the result of SSD detection.

Figure 5: **SSD Detection Example**

The following table lists the SSD detection models supported by the Vitis AI Library.

Table 17: **SSD Models**

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	ssdlite_mobilenet_v2_coco_tf	
6	ssd_inception_v2_coco_tf	
7	ssd_pedestrian_pruned_0_97	Caffe
8	ssd_traffic_pruned_0_9	
9	ssd_adas_pruned_0_95	
10	ssd_mobilenet_v2	

Pose Detection

The Pose Detection library is used to detect the posture of the human body. This library includes a neural network which can identify 14 key points on 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 the coordinates of each point. The following image shows the result of pose detection.

Figure 6: **Pose Detection Example**

The following table lists the pose detection models supported by the Vitis AI Library.

Table 18: **Pose Detection Models**

No	Model Name	Framework
1	sp_net	Caffe

***Note*:** If the input image is arbitrary and you do not know the exact location of the person, perform the SSD detection first. See the `test_jpeg_posedetect_with_ssd.cpp` file. The input for test_jpeg_posedetect_ssd can be any image with or without a person in it. If there is a person in the image, it will first detect the person with SSD, then send the position of the person as the input for posedetect. If the SSD detection does not identify any person in the image, then posedetect does not run. As test_jpeg_posedetect only performs posedetect, so the input image must have atleast one person. If you input an image without a person for test_jpeg_posedetect, it will throw an error. See the `test_jpeg_posedetect.cpp` file.

Semantic Segmentation

Semantic segmentation assigns a semantic category to each pixel in the input image, that is, it identifies pixels as part of an object, say, a car, a road, a tree, a horse, etc. Libsegmentation is a segmentation library which can be used in ADAS applications. It offers simple interfaces for a developer to deploy segmentation tasks on a Xilinx® FPGA.

The following is an example of semantic segmentation, where "blue gray" denotes the sky, "green" denotes trees, "red" denotes people, "dark blue" denotes cars, "plum" denotes the road, and "gray" denotes structures.

Figure 7: **Semantic Segmentation Example**

The following table lists the semantic segmentation models supported by the Vitis AI Library.

Table 19: **Semantic Segmentation Models**

No	Model Name	Framework
1	fpn	Caffe
2	FPN-resnet18_Endov	
3	semantic_seg_citys_tf2	TensorFlow
4	mobilenet_v2_cityscapes_tf	
5	SemanticFPN_cityscapes_pt	PyTorch
6	ENet_cityscapes_pt	
7	unet_chaos-CT_pt	

Road Line Detection

The Road Line Detection library is used to draw lane lines in ADAS applications. Each lane line is represented by a number representing the category. A vector<Point> is 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, and the polygon interface `cv::polylines()` of OpenCV is used to draw the lane line. The following image shows the result of road line detection.

Figure 8: **Road Line Detection Example**

The following table lists the road line detection models supported by the Vitis AI Library.

Table 20: **Road Line Detection Models**

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

YOLO is a neural network which is used to detect objects. The current version is v3. The input is a picture with one or more objects and the 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 9: **YOLOv3 Detection Example**

The following table lists the YOLOv3 detection models supported by the Vitis AI Library.

Table 21: **YOLOv3 Detection Models**

No	Model Name	Framework
1	yolov3_voc_tf	TensorFlow
2	yolov3_adas_pruned_0_9	Caffe
3	yolov3_voc	
4	yolov3_bdd	
5	yolov4_leaky_spp_m	
6	tiny_yolov3_vmss	

YOLOv2 Detection

YOLOv2 does the same thing as YOLOv3, which is an upgraded version of YOLOv2. The following table lists the YOLOv2 detection models supported by the Vitis AI Library.

Table 22: **YOLOv2 Detection Models**

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

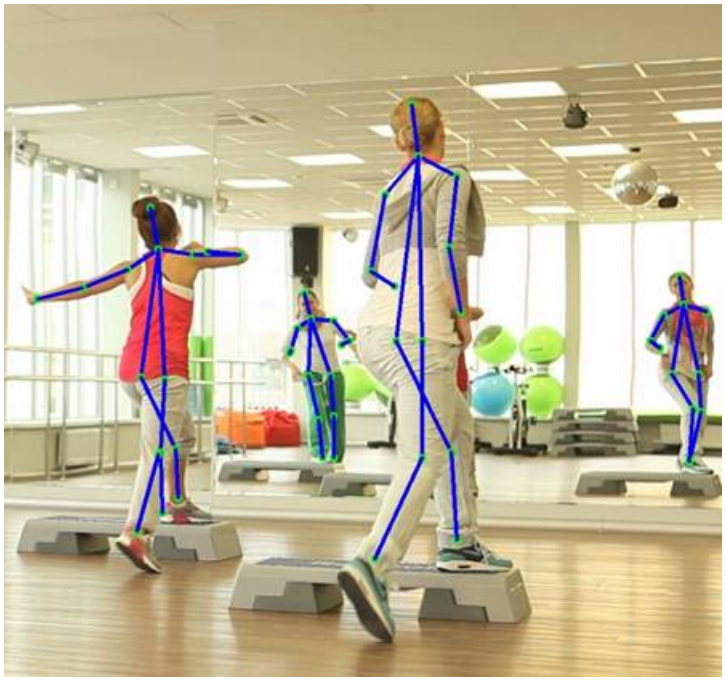
The Openpose Detection library is used to detect posture of the human body. The posture is represented by an array of 14 key points as shown below:

```
0: head, 1: neck, 2: L_shoulder, 3:L_elbow, 4: L_wrist, 5: R_shoulder,
6: R_elbow, 7: R_wrist, 8: L_hip, 9: L_knee, 10: L_ankle, 11: R_hip,
12: R_knee, 13: R_ankle
```

The input of the network is 368x368. The following image shows the result of openpose detection.

RECOMMENDED: *Use a square picture for input. If you need to detect pictures of other size ratios, use a network with the same input size ratio.*

Figure 10: **Openpose Detection Example**



The following table lists the Openpose detection models supported by the Vitis AI Library.

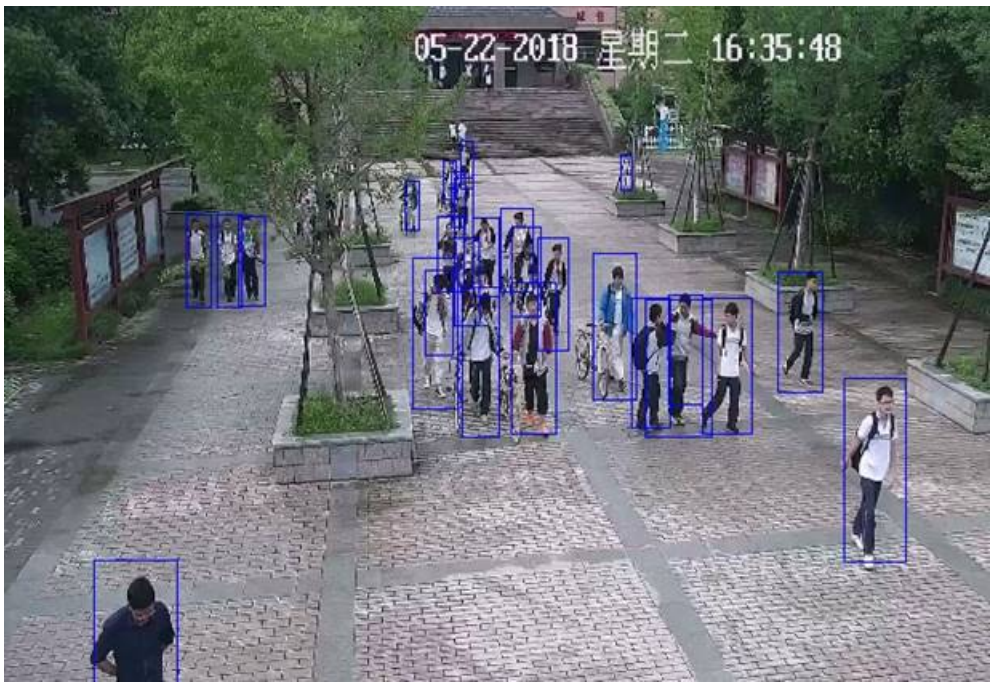
Table 23: **Openpose Detection Models**

No	Model Name	Framework
1	openpose_pruned_0_3	Caffe

RefineDet Detection

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 11: **RefineDet Detection Example**



The following table lists the RefineDet detection models supported by the Vitis AI Library.

Table 24: **RefineDet Detection Models**

No	Model Name	Framework
1	refinedet_pruned_0_8	Caffe
2	refinedet_pruned_0_92	
3	refinedet_pruned_0_96	
4	refinedet_baseline	
5	refinedet_VOC_tf	TensorFlow

ReID Detection

The task of person re-identification is to identify a person of interest at any time or place. This is done by extracting the image feature and comparing the features. Images of the same person should have similar features and have small feature distance, while images of different persons have large feature distance. Given a queried image and a pile of candidate images, the image that has the smallest feature distance is identified as the same person as the queried image. The following table lists the ReID detection models supported by the Vitis AI Library.

Table 25: **ReID Detection Models**

Number	Model Name	Framework
1	reid	Caffe
2	personreid-res18_pt	PyTorch
3	personreid-res50_pt	
4	facereid-large_pt	
5	facereid-small_pt	

Multi-task

The multi-task library is appropriate for a model that has multiple sub-tasks. The Multi-task model in the Vitis AI Library has two sub-tasks: semantic segmentation and SSD detection. The following table listss the multi-task models supported by the Vitis AI Library.

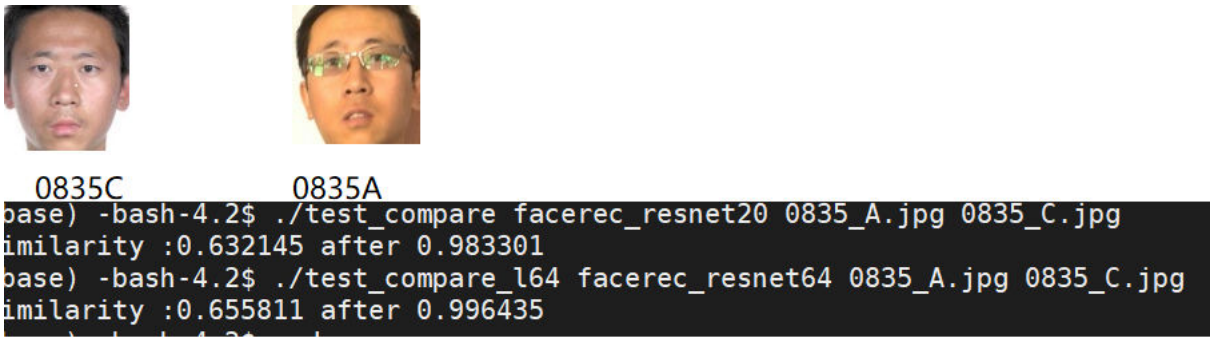
Table 26: **Multi-task Models**

Number	Model Name	Framework
1	multi_task	Caffe
2	MT-resnet18_mixed_pt	PyTorch

Face Recognition

The models of face feature are used for face recognition. They can extract the features of a person's face. The output of these models are 512 features. If you have two different images and you want to know if they are of the same person, use these models to extract features of the two images, and then use calculation functions and mapped functions to get the similarity of the two images.

Figure 12: **Face Recognition Example**



The following table listss the face recognition models supported by the Vitis AI Library.

Table 27: **Face Recognition Models**

No	Model Name	Framework
1	facerec_resnet20	Caffe
2	facerec_resnet64	
3	facerec-resnet20_mixed_pt	PyTorch

Plate Detection

The Plate Detection library uses the DenseBox neuron network to detect license plates. The input is a picture of the vehicle that is detected by the SSD and the output is a structure containing the plate location information. The following image shows the result of the plate detection.

Figure 13: **Plate Detection Example**

The following table lists the plate detection models supported by the Vitis AI Library.

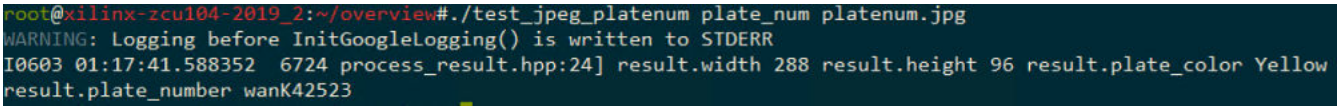
Table 28: **Plate Detection Models**

No	Model Name	Framework
1	plate_detect	Caffe

Plate Recognition

The Plate Recognition library uses a classification network to recognize license plate number (Chinese license plates only). The input is a picture of the license plate that is detected by plate detect. The output is a structure containing license plate number information. The following image shows the result of the plate recognition.

Figure 14: **Plate Recognition Example**



The following table lists the plate recognition models supported by the Vitis AI Library.

Table 29: **Plate Recognition Models**

No	Model Name	Framework
1	plate_num	Caffe

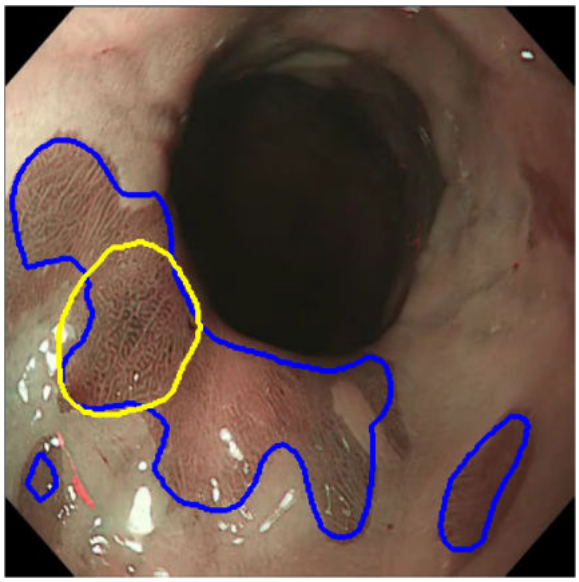
Medical Segmentation

Endoscopy is a common clinical procedure for the early detection of cancers in hollow-organs such as nasopharyngeal cancer, esophageal adenocarcinoma, gastric cancer, colorectal cancer, and bladder cancer. Accurate and temporally consistent localization and segmentation of diseased region-of-interests enable precise quantification and mapping of lesions from clinical endoscopy videos, which is critical for monitoring and surgical planning.

The medical segmentation model is used to classify diseased region-of-interests in the input image. It can be classified into many categories, including BE, cancer, HGD, polyp, and suspicious.

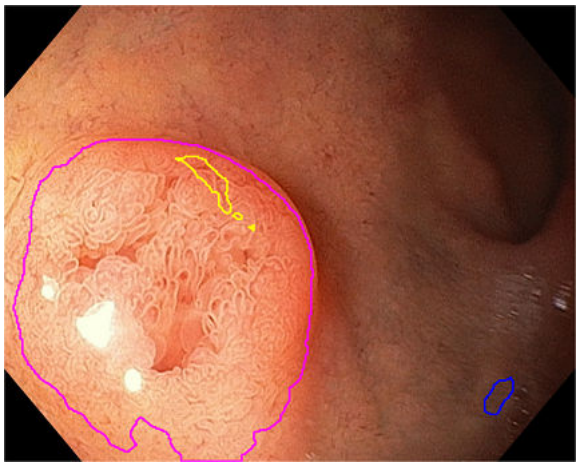
Libmedicalsegmentation is a segmentation library which can be used in segmentation of multi-class diseases in endoscopy. It offers simple interfaces for developers to deploy segmentation tasks on Xilinx FPGAs. The following is an example of medical segmentation, where the goal is to mark the diseased region.

Figure 15: **Marking the Diseased Region**



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

Figure 16: **Medical Segmentation Example**



The following table lists the medical segmentation models supported by the Vitis AI Library.

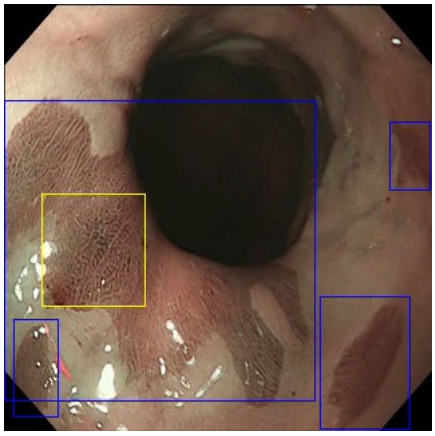
Table 30: **Semantic Segmentation Models**

No	Model Name	Framework
1	FPN_Res18_Medical_segmentation	Caffe

Medical Detection

The RefineDet model is based on vgg16. It is used for medical detection and can detect five types of diseases, namely, BE, cancer, HGD, polyp, and suspicious from an input endoscopy image like the Endoscopy Disease Detection and Segmentation database (EDD2020).

Figure 17: **Medical Detection Example**



The following table lists the medical detection models supported by the Vitis AI Library.

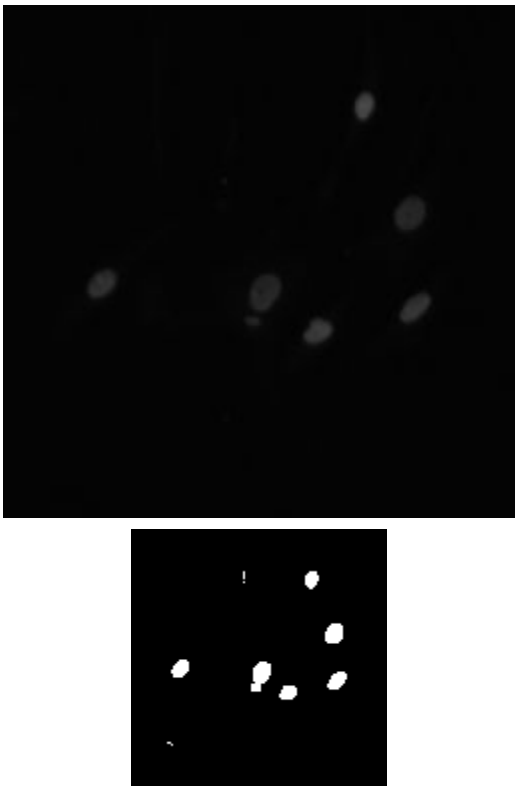
Table 31: **Semantic Detection Models**

No	Model Name	Framework
1	RefineDet-Medical_EDD_tf	TensorFlow

Medical Cell Segmentation

The nucleus is an organelle present within all eukaryotic cells, including human cells. Abberant nuclear shape can be used to identify cancer cells, for example, pap smear tests for the diagnosis of cervical cancer. Medical segmentation cell models offer nuclear segmentation in digital microscopic tissue images which can enable extraction of high quality features for nuclear morphometric and other analyses in computational pathology. The following images show the results of cell segmentation.

Figure 18: **Medical Cell Segmentation Examples**



The following table lists the Medical Cell Segmentation models supported by the Vitis AI Library.

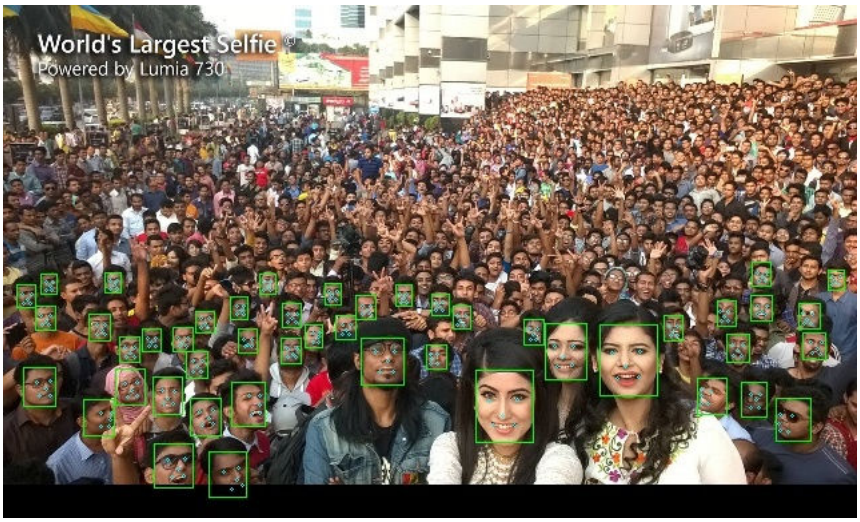
Table 32: **Medical Cell Segmentation Models**

No	Model Name	Framework
1	medical_seg_cell_tf2	TensorFlow

Retinaface

This retinaface network is used to detect human face and face landmark. The input is a picture with some faces you would like to detect and the output contains face positions, scores, and landmarks of faces.

Figure 19: **Retinaface Detection Example**



The following table lists the retinaface detection models supported by the Vitis AI Library.

Table 33: **Retinaface Detection Models**

No	Model Name	Framework
1	retinaface	Caffe

Face Quality

Th Face Quality library uses the face quality network to detect the quality score of a face. If a face is clear and a front face, the score is high. On the contrary, a blurry or side face will get a low score. The score range from 0 to 1. It also provide face landmark positions. The input is a face which is detected by face detect network and the output contains quality score and five landmark key points.

Figure 20: **Face Quality Example**

```
root@xilinx-zcu102-2020_1:/home/test#  ./test_jpeg_facequality5pt face-quality ./sample_facequality5pt.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1104 07:07:03.790011  6291 process_result.hpp:26] score : 0.746494 points
I1104 07:07:03.790123  6291 process_result.hpp:29] 0.333333 0.35
I1104 07:07:03.790196  6291 process_result.hpp:29] 0.8 0.3625
I1104 07:07:03.790225  6291 process_result.hpp:29] 0.6 0.5
I1104 07:07:03.790252  6291 process_result.hpp:29] 0.366667 0.7375
I1104 07:07:03.790280  6291 process_result.hpp:29] 0.75 0.75
```

The following table lists the face quality models supported by the Vitis AI Library.

Table 34: **Face Quality Models List**

No	Model Name	Framework
1	face-quality	Caffe
2	face-quality_pt	PyTorch

Hourglass

The Hourglass library is used to detect posture of the human body. It is represented by an array of 16 joint points. Joint points are arranged in order:

```
0 - r ankle, 1 - r knee, 2 - r hip, 3 - l hip, 4 - l knee, 5 - l ankle,
6 - pelvis, 7 - thorax, 8 - upper neck, 9 - head top, 10 - r wrist,
11 - r elbow, 12 - r shoulder, 13 - l shoulder, 14 - l elbow, 15 - l wrist
```

This network can detect the posture of only one person in the input image. The input of the network is 256x256. The following image shows the result of hourglass detection.

RECOMMENDED: *Use a square picture for input. If you need to detect pictures of other size ratios, use a network with the same input size ratio.*

The following table lists the hourglass models supported by the Vitis AI Library.

Table 35: **Hourglass Models**

No	Model Name	Framework
1	hourglass-pe_mpii	Caffe

Pointpillars

Object detection in point clouds is an important aspect of many robotics applications such as autonomous driving. The pointpillars model is a novel deep network and encoder that can be trained end-to-end on LiDAR point clouds. It offers the best architecture for 3D object detection from LiDAR. The following image shows the result of a pointpillar test.

Figure 21: **Pointpillars Test Example**

The following table lists the pointpillars models supported by the Vitis AI Library.

Table 36: **Pointpillar Models**

No	Model Name	Framework
1	pointpillars_kitti_12000_0_pt	PyTorch
2	pointpillars_kitti_12000_1_pt	PyTorch

3D Segmentation

The 3D segmentation library can support the SalsaNext model, which is used for the uncertainty-aware semantic segmentation of a full 3D LiDAR point cloud in real-time. SalsaNext is the next version of SalsaNet which has an encoder-decoder architecture, where the encoder unit has a set of ResNet blocks and the decoder unit combines upsampled features from the residual blocks.

The following table lists the3D segmentation models supported by the Vitis AI Library.

Table 37: **3D Segmentation Models**

No	Model Name	Framework
1	salsanext_pt	PyTorch

Covid19 Segmentation

The Covid19 segmentation library can support the COVID-Net model which is a deep convolutional neural network design tailored for the detection of COVID-19 cases from chest X-ray (CXR) images.

The following table lists the Covid19 segmentation models supported by the Vitis AI Library.

Table 38: **Covid19 Segmentation Models**

No	Model Name	Framework
1	FPN-resnet18_covid19-seg_pt	PyTorch

Model Samples

Currently, there are 27 model samples that are located in `~/Vitis-AI/demo/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 models to run:
   a. yolov3_bdd

b. yolov3_voc
c. yolov3_voc_tf

2. Ensure that the following test programs exists:
   a. test_jpeg_yolov3
   b. test_video_yolov3
   c. test_performance_yolov3
   d. test_accuracy_yolov3_bdd
   e. test_accuracy_yolov3_adas_pruned_0_9
   f. test_accuracy_yolov3_voc
   g. test_accuracy_yolov3_voc_tf

   If the executable program does not exist, you have to cross compile it on the host and then copy the executable program to the target.
3. To test the image data, execute the following command:

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

   The result is printed 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
   ```

   The result is printed on the terminal.
6. To test the model accuracy, prepare your own image dataset, image list file and the ground truth of the images. Then execute the following command:

   ```
   #./test_accuracy_yolov3_bdd [image_list_file] [output_file]
   ```

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

Programming Examples

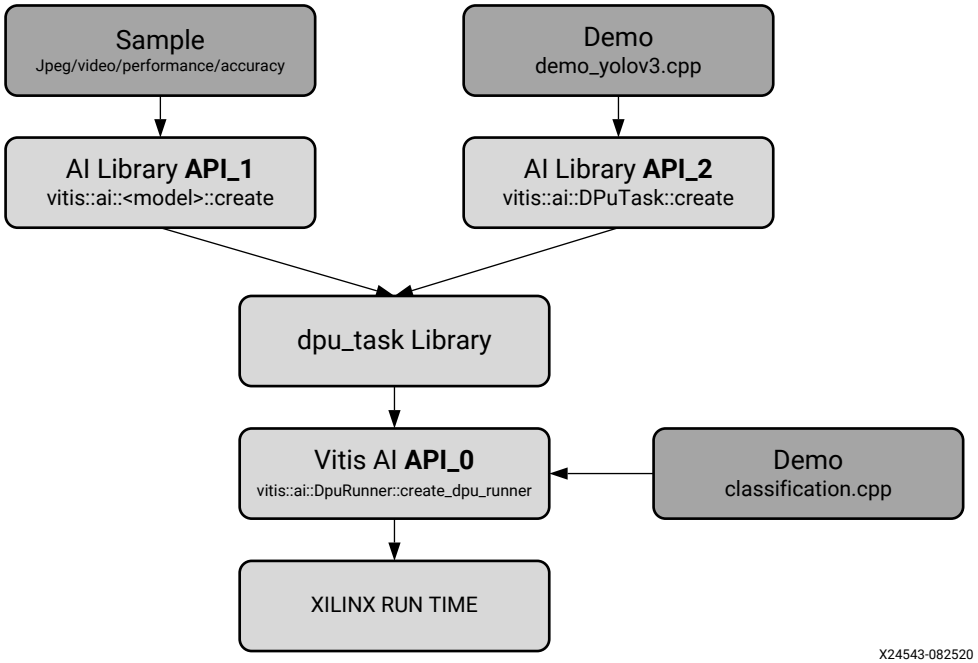
You can have many different application requirements, but all of them can be categorized into three categories. The first is to use the ready-made models provided by the Xilinx® Vitis™ AI Library to quickly build their own application; the second is to use your own custom models which are similar to the models in the Vitis AI Library; and the last is to use new models that are totally different from the models in the Vitis AI Library. This chapter describes the detailed development steps for the first two cases. For the third case, you can also use the Vitis 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 Vitis AI Library's post-processing library
- How to implement user post-processing code

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

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

Figure 22: **The Diagram of AI Library API**



Developing With Vitis AI API_0

1. Install the cross-compilation system on the host side, refer to Chapter 2: Installation.
2. Download the model that you want to use, such as `resnet50`, and copy it to the board using scp.
3. Install the model on the target side.

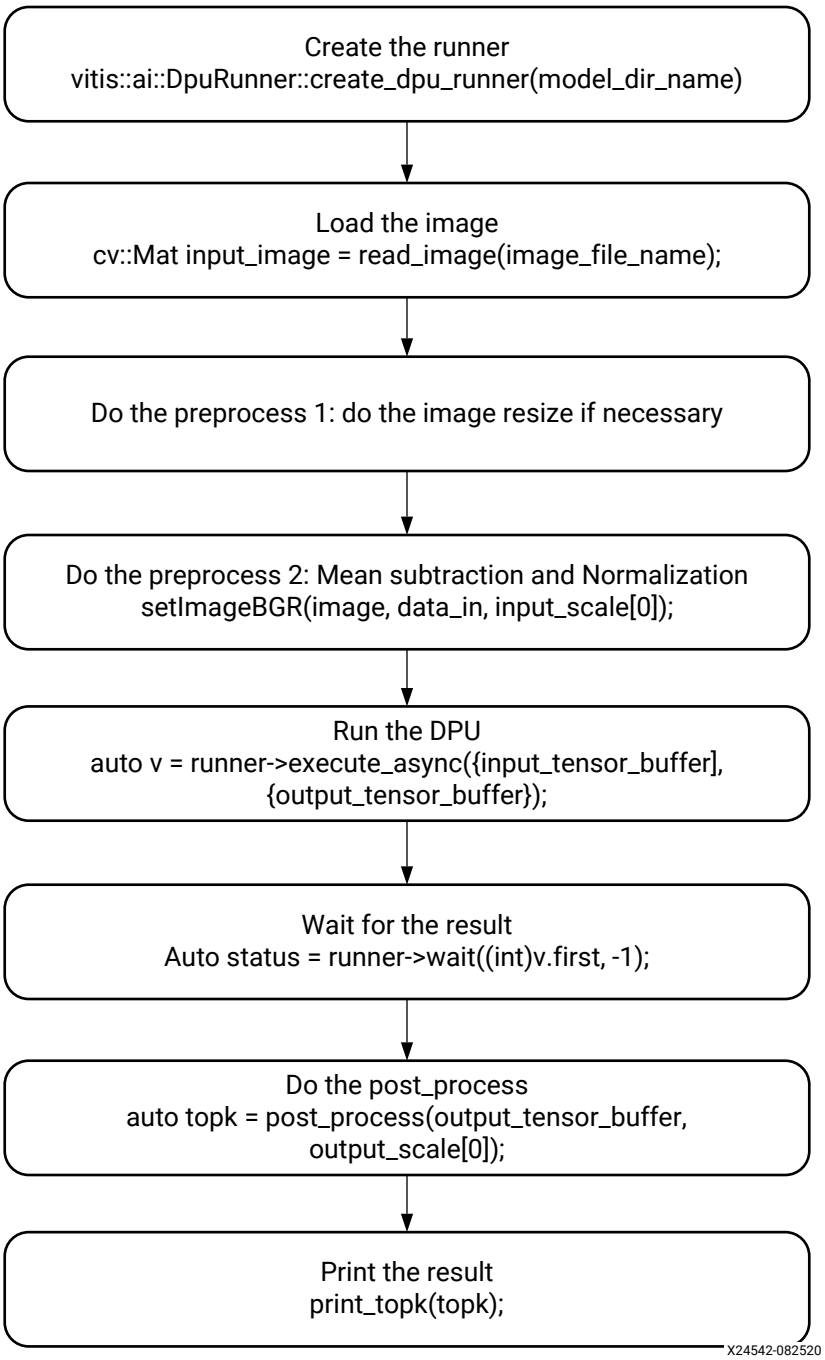
   ```
   tar -xzvf <model>.tar.gz
   cp -r <model> /usr/share/vitis_ai_library/models
   ```

   By default, we put the models under `/usr/share/vitis_ai_library/models` directory on the target side.

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

   ```
   mkdir classification
   ```
6. Create the `demo_classification.cpp` source file. The main flow is shown in the following figure. See `Vitis-AI/demo/Vitis-AI-Library/samples/dpu_task/classification/demo_classification.cpp` for a complete code example.

Figure 23: **Main Program Flow Chart**



7. Create a `build.sh` file as shown below, or copy one from the Vitis 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 -lvitis_ai_library-dpu_task -
lvitis_ai_library-model_config -lvart-runner
```

8. Cross compile the program.

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

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

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

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

```
./demo_classification /usr/share/vitis_ai_library/models/resnet50/
resnet50.xmodel resnet50_0 demo_classification.jpg
```

***Note*:**

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

Developing with User Model and AI Library API_2

When you use your own models, it is important to note that your 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 docker image from https://github.com/Xilinx/Vitis-AI.
2. Load and run the docker.
3. Create a folder and place the float model under it on the host side, then use `AI Quantizer` tool to do the quantization. For more details, see *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).
4. Use `AI Compiler` tool to do the model compiling to get the xmodel file, such as `yolov3_custom.xmodel`. For more information, see *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431).

5. 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
    layer_name: "59"
    layer_name: "67"
    layer_name: "75"
    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` is effective only 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. See the `Vitis-AI/demo/Vitis-AI-Library/samples/dpu_task/yolov3/demo_yolov3.cpp` for details.

6. Create the `demo_yolov3.cpp` file. See `Vitis-AI/demo/Vitis-AI-Library/samples/dpu_task/yolov3/demo_yolov3.cpp` for reference.

7. Create a `build.sh` file as shown below, or copy one from the Vitis 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-xnnpp -lvitis_ai_library-model_config -
lprotobuf -lvitis_ai_library-dpu_task
```

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

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

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

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

Note: `/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.

11. Copy the `yolov3_custom.xmodel and yolov3_custom.prototxt` to the target and put them under `/usr/share/vitis_ai_library/models/yolov3_custom`.

```
scp yolov3_custom.xmodel  yolov3_custom.prototxt root@IP_OF_BOARD:/usr/
share/vitis_ai_library/models/yolov3_custom
```

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

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

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

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

Customizing Pre-Processing

Before convolution neural network processing, image data generally needs to be pre-processed. 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

Call the `setMeanScaleBGR` function to implement the Mean subtraction and normalization, as shown in the following figure. See `Vitis-AI/tools/Vitis-AI-Library/dpu_task/include/vitis/ai/dpu_task.hpp` for details.

Figure 24: **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});
```

Call the `cv::resize` function to scale the image, as shown in the following figure.

Figure 25: **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;
}
```

Using the Configuration File

The 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`.

```
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 39: **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, 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 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
    layer_name: "59"
    layer_name: "67"
    layer_name: "75"
    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
  }
```

The parameters for the YOLOv3 model are listed in the following table. You can modify them as your model requires.

Table 40: **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
layer_name	The kernel's output layer names. When your model's output is more than one, you need to use this parameter to ensure a certain sequence. Such name should be same as these in kernel's. (If you fill it as an invalid name, the model creator will use the kernel default order.)
conf_threshold	The threshold of the boxes' confidence, which can be modified to fit your practical application
nms_threshold	The threshold of NMS
biases	These parameters are the 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
    }
}
```

Following 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 41: **SSD Model Parameters**

Parameter Type	Description
num_classes	The actual number of the model's detection categories
anchorCnt	The number of this model's anchor
conf_threshold	The threshold of the boxes' confidence. Each category can 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.

Table 42: **PriorBox Parameters**

Parameter Type	Description
layer_width/layer_height	The input width/height of this layer. Such numbers can be computed from the net structure.
ariances	These numbers are used for boxes regression, only to fill them as the 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; other 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 the 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 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].

Example Code

The following is the example code.

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

To create the YOLOv3 object, use `create`.

```
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. For more details about the example, refer to `~/Vitis-AI/Vitis-AI-Library/yolov3/test/test_yolov3.cpp`.

Implementing User Post-Processing Code

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

Figure 26: **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 more details, see `~/Vitis-AI/demo/Vitis-AI-Library/samples/dpu_task/classification/demo_classification.cpp`.

Using 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 `xnnpp` post-processing library is provided in Vitis AI Library to facilitate user calls. It is 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
- Face recognition
- Plate detection
- Plate recognition
- Medical segmentation
- Medical detection
- Face quality
- Hourglass
- Retinaface

There are two ways to call `xnnpp`:

- One is an automatic call, through `vitis::ai::<model>::create`, to create the task, such as `vitis::ai::YOLOv3::create("yolov3_bdd", true)`. After <model>->run finished, `xnnpp` is automatically processed. You can modify the parameters through the model configuration file.
- One is a manual call, through `vitis::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 configuration and set the correlating data to control post-process.

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

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

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

  3. Fill the other parameters.

     ```
     fillconfig(config);
     ```

4. Create an object of SSDPostProcess.

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

5. Run the post-process.

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

***Note*:** For more details about the post processing examples, see the `~/Vitis-AI/demo/Vitis-AI-Library/samples/dpu_task/yolov3/demo_yolov3.cpp` and `~/Vitis-AI/tools/Vitis-AI-Library/yolov3/test/test_yolov3.cpp` files in the host system.

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 Vitis™ AI Library. Here, we take ZCU102 board as the test platform.

Demo Overview

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

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

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

Note: To achieve the best performance, the demos use the DRM (Direct Render Manager) for video display. 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

- **Hardware:**
  - 1 x ZCU102 Prod Silicon https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
  - 1 x Windows 7/10 laptop
  - 1 x 16 GB SD card

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

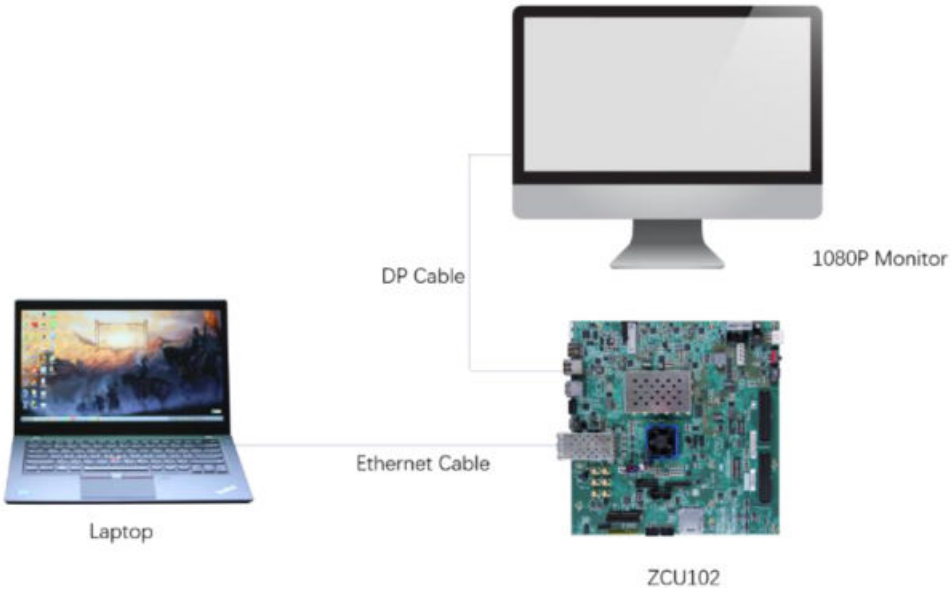
- **Software:**
  - ZCU102 board image https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge
  - Vitis AI Library
  - Images and video files
  - Terminal software like MobaXterm, Putty

DPU Configuration & Dev Tool Used

- 3xB4096 @281MHz
- Vivado 2020.2, Vitis AI Library r1.3

Demo Setup Illustration

Figure 27: **Demo Setup**



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

Target Application

ADAS/AD

AI Model, Performance, and Power

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

Build and Run the Demo

Build the demo in the host and copy the program to the target board.

```
cd Vitis-AI/demo/Vitis-AI-Library/apps/segs_and_roadline_detect
bash -x build.sh
scp segs_and_roadline_detect_x segs_and_roadline_detect_drm
root@IP_OF_BOARD:~/
```

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:

```
./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*:**

1. The video files are in the `vitis_ai_library_r1.3.0_video.tar.gz`. Download the package from here.
2. Due to the limitation of Docker environment, the Multi-Task demos cannot run in DRM mode on the cloud devices.

Demo Picture

Figure 28: **Segmentation and Roadline Detection Demo Picture**

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

Target Application

- ADAS/AD
- Smartcity

AI Model, Performance, and Power

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

Build and Run the Demo

Build the demo in the host and copy the program to the target board.

```
cd Vitis-AI/demo/Vitis-AI-Library/apps/seg_and_pose_detect
bash -x build.sh
scp seg_and_pose_detect_x seg_and_pose_detect_drm root@IP_OF_BOARD:~/
```

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 using SSH, and run the following command:

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

***Note*:**

1. The video files are in the `vitis_ai_library_r1.3.0_video.tar.gz`. Download the package from here.
2. Due to the limitation of Docker environment, the Multi-Task demos cannot run in DRM mode on the cloud devices.

Demo Picture

Figure 29: **Segmentation and Pose Detection Demo Picture**

Programming APIs

To use the library, you need to prepare the development board and cross-compilation environment. Pay attention to the header files, library files, and the model library files.

***Note*:** The files in the development environment must match the version provided in the Vitis™ Unified Software Development Environment. These libraries can be executed on ZCU102, ZCU104, VCK190 and Xilinx Alveo U50, U50LV, and U280 data center accelerator cards.

1. Select an image. For example, `cv::Mat`.
2. Call the `create` method provided by the corresponding library to get a class instance. If you set the `need_preprocess` variable to `false`, the model will not decrease its mean and scale.
3. Call the `getInputWidth()` and the `getInputHeight()` functions to get the network needed column and row values of the input image.
4. Resize image to `inputWidth x inputHeight`.
5. Call `run()` to get the result of the network.

For details about the Programming APIs, see the Chapter 8: API Reference.

Also, for the Vitis AI APIs, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431). You can download it from the Xilinx website.

Performance

This chapter describes in detail the performance of the Vitis AI Library on the following different evaluation boards and data center accelerator cards.

- ZCU102 (0432055-05)
- ZCU104
- VCK190
- Alveo™ U50
- Alveo U50LV
- Alveo U280

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). Because the 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.

***Note*:** The DPU on the ZCU102 has hardware softmax acceleration module. Due to the limitation of hardware softmax module, the software softmax is faster when the number of categories reaches 1000. Set `XLNX_ENABLE_C_SOFTMAX=1` to enable the software softmax: `softmax_c`. The default value of `XLNX_ENABLE_C_SOFTMAX` is 0, which means the softmax method is selected according to the following priorities.

1. Neon Acceleration
2. Hardware Softmax
3. Software Softmax_c

For ZCU102, use the following command to test the performance of classification.

```
env XLNX_ENABLE_C_SOFTMAX=1 ./test_performance_classification resnet50
test_performance_classification.list -t 8 -s 60
```

Table 43: **ZCU102 (0432055-05) Performance**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	428.9	1638.6
2	densebox_640_360	360x640	1.1	220.1	889.6
3	ENet_cityscapes_pt	512x1024	8.6	9.1	38.5
4	face_landmark	96x72	0.14	846.1	1560.7
5	face-quality	80x60	0.06	2133.2	6681.4
6	face-quality_pt	80x60	0.06	2117.3	7410.8
7	facerec_resnet20	112x96	3.5	163.5	329
8	facerec-resnet20_mixed_pt	112x96	3.5	165.1	330.7
9	facerec_resnet64	112x96	11	71.2	177.5
10	facereid-large_pt	96x96	0.5	844.3	2177.7
11	facereid-small_pt	80x80	0.09	1869.9	5994.9
12	fpn	256x512	8.9	33.3	148.4
13	FPN_Res18_Medical_segmentation	320x320	45.3	12.3	44.9
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	36	107
15	FPN-resnet18_Endov	240x320	13.75	33.6	160.2
16	hourglass-pe_mpii	256x256	10.2	18.2	76.1
17	inception_resnet_v2_tf	299x299	26.4	22.2	50
18	inception_v1	224x224	3.2	169.1	429.3
19	inception_v1_tf	224x224	3	172.4	429.6
20	inception_v2	224x224	4	125.2	290
21	inception_v2_tf	224x224	3.88	88.1	222.7
22	inception_v3	299x299	11.4	58.5	135.7
23	inception_v3_pt	299x299	5.7	58.6	136.1
24	inception_v3_tf	299x299	11.5	58	134.1
25	inception_v3_tf2	299x299	11.5	57.1	133.7
26	inception_v4	299x299	24.5	28.6	68.8
27	inception_v4_2016_09_09_tf	299x299	24.6	28.5	68.7
28	medical_seg_cell_tf2	128x128	5.3	154.1	392.6
29	MLPerf_resnet50_v1.5_tf	224x224	8.19	71.2	167.9
30	mlperf_ssd_resnet34_tf	1200x1200	433	2	7
31	mobilenet_1_0_224_tf2	224x224	1.1	292.6	918.9
32	mobilenet_edge_0_75_tf	224x224	0.62	237.6	699.4

Table 43: **ZCU102 (0432055-05) Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
33	mobilenet_edge_1_0_tf	224x224	0.99	198.3	544.3
34	mobilenet_v1_0_25_128_tf	128x128	0.027	1127	3929.4
35	mobilenet_v1_0_5_160_tf	160x160	0.15	714.6	2692.3
36	mobilenet_v1_1_0_224_tf	224x224	1.1	297.3	931.4
37	mobilenet_v2	224x224	0.6	250	729.2
38	mobilenet_v2_1_0_224_tf	224x224	0.6	245	689.9
39	mobilenet_v2_1_4_224_tf	224x224	1.2	179.7	467.8
40	mobilenet_v2_cityscapes_tf	1024x2048	132.74	1.6	4.6
41	MT-resnet18_mixed_pt	512x320	13.65	29.3	100.2
42	multi_task	288x512	14.8	35.2	125.2
43	openpose_pruned_0_3	368x368	49.9	3.5	15
44	personreid-res18_pt	176x80	1.1	350	682.2
45	personreid-res50_pt	256x128	5.4	96.9	227.3
46	plate_detection	320x320	0.49	505.9	2025.7
47	plate_num	96x288	1.75	149.8	441.1
48	pointpillars_kitti_12000_0_pt pointpillars_kitti_12000_1_pt	12000x100	10.8	19.9	49.9
49	refinedet_baseline	480x360	123	8.5	24.7
50	RefineDet-Medical_EDD_tf	320x320	9.8	66.1	225
51	refinedet_pruned_0_8	360x480	25	32.6	100.3
52	refinedet_pruned_0_92	360x480	10.1	63.1	201.8
53	refinedet_pruned_0_96	360x480	5.1	87.4	288.2
54	refinedet_VOC_tf	320x320	81.9	11.3	34.4
55	reid	80x160	0.95	351	689.8
56	resnet18	224x224	3.7	183.2	476.4
57	resnet50	224x224	7.7	72.7	173.1
58	resnet50_pt	224x224	4.1	68.8	165.8
59	resnet50_tf2	224x224	7.7	70.7	170.5
60	resnet_v1_101_tf	224x224	14.4	42.6	106.2
61	resnet_v1_152_tf	224x224	21.8	29.1	73.9
62	resnet_v1_50_tf	224x224	7	79.6	184.8
63	resnet_v2_101_tf	299x299	26.78	20.5	53.8
64	resnet_v2_152_tf	299x299	40.47	14.5	37.2
65	resnet_v2_50_tf	299x299	13.1	35.1	93.8
66	retinaface	360x640	1.11	126.2	578.2
67	salsanext_pt	64x2048	20.4	5.3	19.7
68	SemanticFPN_cityscapes_pt	256x512	10	33.2	161.3

Table 43: **ZCU102 (0432055-05) Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
69	semantic_seg_citys_tf2	512x1024	54	6.8	23.9
70	sp_net	128x224	0.55	485.9	1408.8
71	squeezenet	227x227	0.76	270.6	1119.8
72	squeezenet_pt	224x224	0.82	221.7	847.2
73	ssd_adas_pruned_0_95	360x480	6.3	87	298.1
74	ssd_inception_v2_coco_tf	300x300	9.6	38.5	101
75	ssdlite_mobilenet_v2_coco_tf	300x300	1.5	101.7	303.9
76	ssd_mobilenet_v1_coco_tf	300x300	2.5	107.9	324.4
77	ssd_mobilenet_v2	360x480	6.6	39.1	115.9
78	ssd_mobilenet_v2_coco_tf	300x300	3.8	79.7	210.9
79	ssd_pedestrian_pruned_0_97	360x360	5.9	76.3	279.2
80	ssd_resnet_50_fpn_coco_tf	640x640	178.4	2.9	5.2
81	ssd_traffic_pruned_0_9	360x480	11.6	54.9	200.2
82	tiny_yolov3_vmss	416x416	5.46	117.7	390.1
83	unet_chaos-CT_pt	512x512	23.3	21.2	68.1
84	vgg_16_tf	224x224	31	20.1	41
85	vgg_19_tf	224x224	39.3	17.3	36.5
86	vpgnet_pruned_0_99	480x640	2.5	96.9	360.3
87	yolov2_voc	448x448	34	26.4	69.6
88	yolov2_voc_pruned_0_66	448x448	11.6	62.8	189.1
89	yolov2_voc_pruned_0_71	448x448	9.9	72.2	220.6
90	yolov2_voc_pruned_0_77	448x448	7.8	84.3	265
91	yolov3_adas_pruned_0_9	256x512	5.5	88.3	260.7
92	yolov3_bdd	288x512	53.7	12.4	32.8
93	yolov3_voc	416x416	65.4	12.7	33.4
94	yolov3_voc_tf	416x416	65.6	13.2	34.3
95	yolov4_leaky_spp_m	416x416	60.1	13.1	33.5

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 44: **ZCU104 Performance**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	434.7	1560.6
2	densebox_640_360	360x640	1.1	225.4	784.1
3	ENet_cityscapes_pt	512x1024	8.6	9.3	39.8
4	face_landmark	96x72	0.14	891.3	1601.7
5	face-quality	80x60	0.06	2187.5	6350.5
6	face-quality_pt	80x60	0.06	2176.5	6612.1
7	facerec_resnet20	112x96	3.5	174.2	308.5
8	facerec-resnet20_mixed_pt	112x96	3.5	175.9	311.1
9	facerec_resnet64	112x96	11	75.8	146
10	facereid-large_pt	96x96	0.5	884.6	1995.9
11	facereid-small_pt	80x80	0.09	1914.5	5617.9
12	fpn	256x512	8.9	34.1	128.6
13	FPN_Res18_Medical_segmentation	320x320	45.3	12.9	33.6
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	38.1	80.7
15	FPN-resnet18_Endov	240x320	13.75	34.7	133.8
16	hourglass-pe_mpii	256x256	10.2	18.6	76
17	inception_resnet_v2_tf	299x299	26.4	23.7	44.2
18	inception_v1	224x224	3.2	178.6	371.3
19	inception_v1_tf	224x224	3	182.1	377.2
20	inception_v2	224x224	4	132.9	263.4
21	inception_v2_tf	224x224	3.88	93.7	188.2
22	inception_v3	299x299	11.4	62.1	121.4
23	inception_v3_pt	299x299	5.7	62.2	121.7
24	inception_v3_tf	299x299	11.5	61.5	120.3
25	inception_v3_tf2	299x299	11.5	60.6	118.9
26	inception_v4	299x299	24.5	30.4	59
27	inception_v4_2016_09_09_tf	299x299	24.6	30.4	59
28	medical_seg_cell_tf2	128x128	5.3	163.2	341.5
29	MLPerf_resnet50_v1.5_tf	224x224	8.19	75.6	146.2
30	mlperf_ssd_resnet34_tf	1200x1200	433	1.8	5.1
31	mobilenet_1_0_224_tf2	224x224	1.1	306.4	739.8
32	mobilenet_edge_0_75_tf	224x224	0.62	249.8	569.6
33	mobilenet_edge_1_0_tf	224x224	0.99	208.6	454.5
34	mobilenet_v1_0_25_128_tf	128x128	0.027	1147.4	3973.5

Table 44: **ZCU104 Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
35	mobilenet_v1_0_5_160_tf	160x160	0.15	736.7	2132.6
36	mobilenet_v1_1_0_224_tf	224x224	1.1	311.1	750.2
37	mobilenet_v2	224x224	0.6	262.3	599.2
38	mobilenet_v2_1_0_224_tf	224x224	0.6	256.8	578.4
39	mobilenet_v2_1_4_224_tf	224x224	1.2	189.2	399
40	mobilenet_v2_cityscapes_tf	1024x2048	132.74	1.6	4.6
41	MT-resnet18_mixed_pt	512x320	13.65	30.5	90
42	multi_task	288x512	14.8	36.8	108.2
43	openpose_pruned_0_3	368x368	49.9	3.7	11
44	personreid-res18_pt	176x80	1.1	371.8	684.5
45	personreid-res50_pt	256x128	5.4	102.8	198.2
46	plate_detection	320x320	0.49	513.8	1909.6
47	plate_num	96x288	1.75	161.9	446
48	pointpillars_kitti_12000_0_pt pointpillars_kitti_12000_1_pt	12000x100	10.8	20.2	48.8
49	refinedet_baseline	480x360	123	9.1	18.4
50	RefineDet-Medical_EDD_tf	320x320	9.8	69.6	169.6
51	refinedet_pruned_0_8	360x480	25	34.6	74.5
52	refinedet_pruned_0_92	360x480	10.1	66.6	154.7
53	refinedet_pruned_0_96	360x480	5.1	91.9	227.2
54	refinedet_VOC_tf	320x320	81.9	10.8	25.8
55	reid	80x160	0.95	372.9	698.6
56	resnet18	224x224	3.7	193.4	414.6
57	resnet50	224x224	7.7	77.2	149.7
58	resnet50_pt	224x224	4.1	73.1	142.4
59	resnet50_tf2	224x224	7.7	75.1	146.2
60	resnet_v1_101_tf	224x224	14.4	45.3	87.9
61	resnet_v1_152_tf	224x224	21.8	31	60.2
62	resnet_v1_50_tf	224x224	7	84.5	162.7
63	resnet_v2_101_tf	299x299	26.78	21.8	45.1
64	resnet_v2_152_tf	299x299	40.47	15.4	31
65	resnet_v2_50_tf	299x299	13.1	37	81.6
66	retinaface	360x640	1.11	129.5	480.8
67	salsanext_pt	64x2048	20.4	5.3	19.8
68	SemanticFPN_cityscapes_pt	256x512	10	34	132.6
69	semantic_seg_citys_tf2	512x1024	54	7.1	24.2
70	sp_net	128x224	0.55	507.9	1134

Table 44: **ZCU104 Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
71	squeezenet	227x227	0.76	275.6	980
72	squeezenet_pt	224x224	0.82	227.9	737.4
73	ssd_adas_pruned_0_95	360x480	6.3	91	237.9
74	ssd_inception_v2_coco_tf	300x300	9.6	40.8	85.8
75	ssdlite_mobilenet_v2_coco_tf	300x300	1.5	104.9	265.7
76	ssd_mobilenet_v1_coco_tf	300x300	2.5	112.8	295.3
77	ssd_mobilenet_v2	360x480	6.6	25.6	100.3
78	ssd_mobilenet_v2_coco_tf	300x300	3.8	80.6	190.3
79	ssd_pedestrian_pruned_0_97	360x360	5.9	79.7	214.9
80	ssd_resnet_50_fpn_coco_tf	640x640	178.4	2.9	5.2
81	ssd_traffic_pruned_0_9	360x480	11.6	57.5	150
82	tiny_yolov3_vmss	416x416	5.46	123.2	328.8
83	unet_chaos-CT_pt	512x512	23.3	21.9	59.1
84	vgg_16_tf	224x224	31	21.4	37.1
85	vgg_19_tf	224x224	39.3	18.5	32.7
86	vpgnet_pruned_0_99	480x640	2.5	100.5	311.4
87	yolov2_voc	448x448	34	28	57.4
88	yolov2_voc_pruned_0_66	448x448	11.6	66.4	153.3
89	yolov2_voc_pruned_0_71	448x448	9.9	76.2	180.7
90	yolov2_voc_pruned_0_77	448x448	7.8	88.6	216.8
91	yolov3_adas_pruned_0_9	256x512	5.5	92.8	231.9
92	yolov3_bdd	288x512	53.7	13.1	26.7
93	yolov3_voc	416x416	65.4	13.5	27.2
94	yolov3_voc_tf	416x416	65.6	14.1	28.2
95	yolov4_leaky_spp_m	416x416	60.1	13.9	28.5

VCK190 Performance

VCK190 is the first Versal AI Core series evaluation kit, enabling designers to develop solutions using AI and DSP engines capable of delivering over 100X greater compute performance compared to current server class CPUs. For this release, a B8192F DPU core with batch=3 is implemented using AI Engine.

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on VCK190 with DPU running at 1333 MHz.

Table 45: **VCK190 Performance**

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	1333	2243.8
2	densebox_640_360	360x640	1.1	1333	1067
3	ENet_cityscapes_pt	512x1024	8.6	1333	48.9
4	face_landmark	96x72	0.14	1333	9458.9
5	face-quality	80x60	0.06	1333	19703.5
6	face-quality_pt	80x60	0.06	1333	19783.5
7	facerec_resnet20	112x96	3.5	1333	2276.3
8	facerec-resnet20_mixed_pt	112x96	3.5	1333	2277.1
9	facerec_resnet64	112x96	11	1333	1143.7
10	facereid-large_pt	96x96	0.5	1333	11714
11	facereid-small_pt	80x80	0.09	1333	17743.8
12	fpn	256x512	8.9	1333	196.7
13	FPN_Res18_Medical_segmentation	320x320	45.3	1333	186.6
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	1333	540.3
15	inception_resnet_v2_tf	299x299	26.4	1333	299.3
16	inception_v1	224x224	3.2	1333	1553
17	inception_v1_tf	224x224	3	1333	1577.6
18	inception_v2	224x224	4	1333	1137.8
19	inception_v3	299x299	11.4	1333	591
20	inception_v3_pt	299x299	5.7	1333	590.7
21	inception_v3_tf	299x299	11.5	1333	596.2
22	inception_v3_tf2	299x299	11.5	1333	622.1
23	inception_v4	299x299	24.5	1333	297.6
24	inception_v4_2016_09_09_tf	299x299	24.6	1333	297.5
25	medical_seg_cell_tf2	128x128	5.3	1333	1976.3
26	MLPerf_resnet50_v1.5_tf	224x224	8.19	1333	1304
27	mlperf_ssd_resnet34_tf	1200x1200	433	1333	26.2
28	multi_task	288x512	14.8	1333	210
29	openpose_pruned_0_3	368x368	49.9	1333	43.4
30	personreid-res18_pt	176x80	1.1	1333	4195.5
31	personreid-res50_pt	256x128	5.4	1333	1634.6
32	plate_detection	320x320	0.49	1333	2563.3
33	plate_num	96x288	1.75	1333	904
34	refinedet_baseline	480x360	123	1333	161.1
35	RefineDet-Medical_EDD_tf	320x320	9.8	1333	1020.8
36	refinedet_pruned_0_8	360x480	25	1333	516
37	refinedet_pruned_0_92	360x480	10.1	1333	737.4

Table 45: **VCK190 Performance** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
38	refinedet_pruned_0_96	360x480	5.1	1333	890.8
39	refinedet_VOC_tf	320x320	81.9	1333	194.4
40	reid	80x160	0.95	1333	4317.1
41	resnet18	224x224	3.7	1333	2814
42	resnet50	224x224	7.7	1333	1310.2
43	resnet50_pt	224x224	4.1	1333	1282
44	resnet50_tf2	224x224	7.7	1333	1287.4
45	resnet_v1_101_tf	224x224	14.4	1333	794.7
46	resnet_v1_152_tf	224x224	21.8	1333	559.1
47	resnet_v1_50_tf	224x224	7	1333	1392.4
48	salsanext_pt	64x2048	20.4	1333	21.5
49	SemanticFPN_cityscapes_pt	256x512	10	1333	197.3
50	semantic_seg_citys_tf2	512x1024	54	1333	46.5
51	sp_net	128x224	0.55	1333	3724.2
52	squeezenet	227x227	0.76	1333	1773.9
53	squeezenet_pt	224x224	0.82	1333	1891.9
54	ssd_adas_pruned_0_95	360x480	6.3	1333	848.2
55	ssd_pedestrian_pruned_0_97	360x360	5.9	1333	740.1
56	ssd_resnet_50_fpn_coco_tf	640x640	178.4	1333	13.1
57	ssd_traffic_pruned_0_9	360x480	11.6	1333	681.2
58	tiny_yolov3_vmss	416x416	5.46	1333	1389.5
59	unet_chaos-CT_pt	512x512	23.3	1333	212.4
60	vgg_16_tf	224x224	31	1333	336.9
61	vgg_19_tf	224x224	39.3	1333	300.4
62	vpgnet_pruned_0_99	480x640	2.5	1333	703.1
63	yolov2_voc	448x448	34	1333	477.3
64	yolov2_voc_pruned_0_66	448x448	11.6	1333	943.2
65	yolov2_voc_pruned_0_71	448x448	9.9	1333	1058.2
66	yolov2_voc_pruned_0_77	448x448	7.8	1333	1190.8
67	yolov3_adas_pruned_0_9	256x512	5.5	1333	1083.7
68	yolov3_bdd	288x512	53.7	1333	221.4
69	yolov3_voc	416x416	65.4	1333	217
70	yolov3_voc_tf	416x416	65.6	1333	216.8
71	yolov4_leaky_spp_m	416x416	60.1	1333	181.1

U50/U50LV Performance

The Xilinx® Alveo U50 Data Center accelerator cards are peripheral component interconnect express (PCIe®) Gen3x16 compliant and Gen4x8 compatible cards featuring the Xilinx 16 nm UltraScale+ technology. In this release, DPU is implemented in program logic for deep learning inference acceleration.

Note: Some models cannot run at the highest frequency of DPU and need DPU frequency reduction. See For Edge for DPU frequency reduction operation.

U50 Performance with 6E300 MHz DPUCAHX8H

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on U50 Gen3x4 with DPUCAHX8H running at 6E@300 MHz.

Table 46: **U50 Performance with 6E300 MHz DPUCAHX8H**

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	300	2004.5
2	densebox_640_360	360x640	1.1	300	893.2
3	ENet_cityscapes_pt	512x1024	8.6	300x0.9	77.9
4	face_landmark	96x72	0.14	300	10042.2
5	face-quality	80x60	0.06	300x0.9	16522.1
6	face-quality_pt	80x60	0.06	300x0.9	16525.5
7	facerec_resnet20	112x96	3.5	300	1355.7
8	facerec-resnet20_mixed_pt	112x96	3.5	300x0.9	1235.2
9	facerec_resnet64	112x96	11	300	507.9
10	facereid-large_pt	96x96	0.5	300x0.9	7430.5
11	facereid-small_pt	80x80	0.09	300x0.9	18563
12	fpn	256x512	8.9	300	448.6
13	FPN_Res18_Medical_segmentation	320x320	45.3	300	103.5
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	300x0.9	212.9
15	inception_resnet_v2_tf	299x299	26.4	300	186.1
16	inception_v1	224x224	3.2	300	1265.4
17	inception_v1_tf	224x224	3	300	1282.1
18	inception_v2	224x224	4	300	1004.8
19	inception_v3	299x299	11.4	300	419.1
20	inception_v3_pt	299x299	5.7	300	419
21	inception_v3_tf	299x299	11.5	300	418.8
22	inception_v3_tf2	299x299	11.5	300x0.9	377

Table 46: **U50 Performance with 6E300 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
23	inception_v4	299x299	24.5	300	194.2
24	inception_v4_2016_09_09_tf	299x299	24.6	300	194.3
25	medical_seg_cell_tf2	128x128	5.3	300x0.9	1063.9
26	MLPerf_resnet50_v1.5_tf	224x224	8.19	300x0.9	563.4
27	mlperf_ssd_resnet34_tf	1200x1200	433	300x0.9	14.1
28	multi_task	288x512	14.8	300	351.9
29	openpose_pruned_0_3	368x368	49.9	300x0.9	29
30	personreid-res18_pt	176x80	1.1	300x0.9	3542.2
31	personreid-res50_pt	256x128	5.4	300x0.9	915.8
32	plate_detection	320x320	0.49	300	5081.7
33	plate_num	96x288	1.75	300x0.9	1031.1
34	refinedet_baseline	480x360	123	300x0.9	50
35	RefineDet-Medical_EDD_tf	320x320	9.8	300x0.9	423.3
36	refinedet_pruned_0_8	360x480	25	300x0.9	198.3
37	refinedet_pruned_0_92	360x480	10.1	300x0.9	413.4
38	refinedet_pruned_0_96	360x480	5.1	300x0.9	608.3
39	refinedet_VOC_tf	320x320	81.9	300x0.9	70.9
40	reid	80x160	0.95	300	4000
41	resnet18	224x224	3.7	300	1480.9
42	resnet50	224x224	7.7	300	647
43	resnet50_pt	224x224	4.1	300	624.2
44	resnet50_tf2	224x224	7.7	300x0.9	583.8
45	resnet_v1_101_tf	224x224	14.4	300	367.8
46	resnet_v1_152_tf	224x224	21.8	300	246.1
47	resnet_v1_50_tf	224x224	7	300	712.4
48	salsanext_pt	64x2048	20.4	300x0.9	132.7
49	SemanticFPN_cityscapes_pt	256x512	10	300x0.9	454.5
50	semantic_seg_citys_tf2	512x1024	54	300x0.9	51.9
51	sp_net	128x224	0.55	300	1951.6
52	squeezenet	227x227	0.76	300	3538.5
53	squeezenet_pt	224x224	0.82	300	2235.9
54	ssd_adas_pruned_0_95	360x480	6.3	300	666.2
55	ssd_pedestrian_pruned_0_97	360x360	5.9	300x0.9	566.8
56	ssd_resnet_50_fpn_coco_tf	640x640	178.4	300x0.9	32.7
57	ssd_traffic_pruned_0_9	360x480	11.6	300	435.7
58	tiny_yolov3_vmss	416x416	5.46	300x0.9	872.3
59	unet_chaos-CT_pt	512x512	23.3	300x0.9	59.6

Table 46: **U50 Performance with 6E300 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
60	vgg_16_tf	224x224	31	300	161.7
61	vgg_19_tf	224x224	39.3	300	134.9
62	vpgnet_pruned_0_99	480x640	2.5	300x0.9	471.2
63	yolov2_voc	448x448	34	300x0.9	164
64	yolov2_voc_pruned_0_66	448x448	11.6	300x0.9	413.3
65	yolov2_voc_pruned_0_71	448x448	9.9	300x0.9	484.7
66	yolov2_voc_pruned_0_77	448x448	7.8	300x0.9	589.3
67	yolov3_adas_pruned_0_9	256x512	5.5	300x0.9	681.3
68	yolov3_bdd	288x512	53.7	300x0.9	78.3
69	yolov3_voc	416x416	65.4	300x0.9	80.7
70	yolov3_voc_tf	416x416	65.6	300x0.9	80.7
71	yolov4_leaky_spp_m	416x416	60.1	300x0.9	84.5

Model End-to-End Performance on U50 333 MHz Single Core DPUCAHX8L

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on U50 Gen3x4 with DPUCAHX8L running at 1E@333 MHz.

Table 47: **Model End-to-End Performance on U50 333 MHz Single Core DPUCAHX8L**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	222.2	373.0
2	densebox_640_360	360x640	1.1	106.6	169.1
3	ENet_cityscapes_pt	512x1024	8.6	4.5	5.2
4	face_landmark	96x72	0.14	1955.1	5019.7
5	face-quality	80x60	0.06	2543.3	7674.0
6	face-quality_pt	80x60	0.06	2597.0	7729.3
7	facerec_resnet20	112x96	3.5	266.4	309.7
8	facerec-resnet20_mixed_pt	112x96	3.5	268.3	311.9
9	facerec_resnet64	112x96	11	132.2	145.6
10	facereid-small_pt	80x80	0.09	1773.0	4935.3
11	fpn	256x512	8.9	27.3	33.1
12	FPN_Res18_Medical_segmentation	320x320	45.3	13.7	14.5
13	FPN-resnet18_covid19-seg_pt	352x352	22.7	77.8	90.5
14	inception_resnet_v2_tf	299x299	26.4	31.8	33.7

Table 47: **Model End-to-End Performance on U50 333 MHz Single Core DPUCAHX8L** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
15	inception_v1	224x224	3.2	248.0	330.3
16	inception_v1_tf	224x224	3	238.6	335.4
17	inception_v2	224x224	3.88	166.3	200.6
18	inception_v3	299x299	11.4	93.6	109.0
19	inception_v3_pt	299x299	5.7	92.2	111.3
20	inception_v3_tf	299x299	11.5	92.4	110.9
21	inception_v3_tf2	299x299	11.5	87.0	101.4
22	inception_v4	299x299	24.5	47.2	51.0
23	inception_v4_2016_09_09_tf	299x299	24.6	46.6	51.0
24	medical_seg_cell_tf2	128x128	5.3	75.0	76.6
25	MLPerf_resnet50_v1.5_tf	224x224	8.19	80.4	89.4
26	mlperf_ssd_resnet34_tf	1200x1200	433	4.4	7.0
27	mobilenet_1_0_224_tf2	224x224	1.1	649.5	2014.0
28	mobilenet_v1_0_5_160_tf	160x160	0.15	1147.7	5413.3
29	mobilenet_v1_1_0_224_tf	224x224	1.1	654.7	2085.3
30	mobilenet_v2	224x224	0.6	580.0	1156.2
31	mobilenet_v2_1_0_224_tf	224x224	0.6	528.0	1152.4
32	mobilenet_v2_1_4_224_tf	224x224	1.2	447.8	860.6
33	multi_task	288x512	14.8	14.4	17.9
34	openpose_pruned_0_3	368x368	49.9	10.4	17.2
35	personreid-res50_pt	256x128	5.4	97.6	107.3
36	plate_detection	320x320	0.49	462.5	1220.6
37	refinedet_baseline	480x360	123	27.4	30.7
38	RefineDet-Medical_EDD_tf	320x320	9.8	93.8	157.7
39	refinedet_pruned_0_8	360x480	25	56.9	71.2
40	refinedet_pruned_0_92	360x480	10.1	62.7	76.6
41	refinedet_pruned_0_96	360x480	5.1	72.4	94.8
42	refinedet_VOC_tf	320x320	81.9	30.8	48.2
43	reid	80x160	0.95	444.5	597.7
44	resnet18	224x224	3.7	229.0	290.4
45	resnet50	224x224	7.7	80.9	87.9
46	resnet50_pt	224x224	4.1	80.8	88.3
47	resnet50_tf2	224x224	7.7	81.0	87.2
48	resnet_v1_101_tf	224x224	14.4	53.3	55.5
49	resnet_v1_152_tf	224x224	21.8	37.1	38.6
50	resnet_v1_50_tf	224x224	7	92.3	102.8

Table 47: **Model End-to-End Performance on U50 333 MHz Single Core DPUCAHX8L** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
51	retinaface	360x640	1.11	96.8	195.1
52	salsanext_pt	64x2048	20.4	7.2	10.5
53	SemanticFPN_cityscapes_pt	256x512	10	27.5	35.3
54	semantic_seg_citys_tf2	512x1024	54	4.6	5.4
55	sp_net	128x224	0.55	1476.5	7606.3
56	squeezenet	227x227	0.76	441.9	756.3
57	squeezenet_pt	224x224	0.82	311.0	466.6
58	ssd_adas_pruned_0_95	360x480	6.3	75.8	125.0
59	ssdlite_mobilenet_v2_coco_tf	300x300	1.5	273.8	609.8
60	ssd_mobilenet_v1_coco_tf	300x300	2.5	338.4	944.6
61	ssd_mobilenet_v2	360x480	6.6	41.7	158.6
62	ssd_mobilenet_v2_coco_tf	300x300	3.8	158.1	238.4
63	ssd_pedestrian_pruned_0_97	360x360	5.9	25.9	30.9
64	ssd_traffic_pruned_0_9	360x480	11.6	63.0	114.4
65	vgg_16_tf	224x224	31	53.6	57.6
66	vgg_19_tf	224x224	39.3	48.4	51.3
67	vpgnet_pruned_0_99	480x640	2.5	19.1	20.0

U50LV Performance with 9E275 MHz DPUCAHX8H

The following table shows the throughput performance (in frames/sec or fps) for various neural network samples on U50LV Gen3x4 with DPUCAHX8H running at 9E@275 MHz.

Table 48: **U50LV Performance with 9E275 MHz DPUCAHX8H**

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	275	2304.9
2	densebox_640_360	360x640	1.1	275	1025.6
3	ENet_cityscapes_pt	512x1024	8.6	275x0.9	85.5
4	face_landmark	96x72	0.14	275	13366.1
5	face-quality	80x60	0.06	275x0.9	18244.3
6	face-quality_pt	80x60	0.06	275x0.9	18103
7	facerec_resnet20	112x96	3.5	275	1831
8	facerec-resnet20_mixed_pt	112x96	3.5	275x0.9	1667.5
9	facerec_resnet64	112x96	11	275	676.7

Table 48: **U50LV Performance with 9E275 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
10	facereid-large_pt	96x96	0.5	275x0.9	9524.1
11	facereid-small_pt	80x80	0.09	275x0.9	19880
12	fpn	256x512	8.9	275x0.9	566.5
13	FPN_Res18_Medical_segmentation	320x320	45.3	275	140.5
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	275x0.9	287.4
15	inception_resnet_v2_tf	299x299	26.4	275	238.3
16	inception_v1	224x224	3.2	275	1730.8
17	inception_v1_tf	224x224	3	275	1749.8
18	inception_v2	224x224	4	275	1377
19	inception_v3	299x299	11.4	275	575.4
20	inception_v3_pt	299x299	5.7	275	575.4
21	inception_v3_tf	299x299	11.5	275	575.3
22	inception_v3_tf2	299x299	11.5	275x0.9	517.8
23	inception_v4	299x299	24.5	275	264.8
24	inception_v4_2016_09_09_tf	299x299	24.6	275	264.8
25	medical_seg_cell_tf2	128x128	5.3	275x0.9	1457.8
26	MLPerf_resnet50_v1.5_tf	224x224	8.19	275x0.9	689.4
27	mlperf_ssd_resnet34_tf	1200x1200	433	275x0.9	18.9
28	multi_task	288x512	14.8	275x0.9	427.1
29	openpose_pruned_0_3	368x368	49.9	275	44.1
30	personreid-res18_pt	176x80	1.1	275x0.9	4635.6
31	personreid-res50_pt	256x128	5.4	275x0.9	1108.6
32	plate_detection	320x320	0.49	275	5163.1
33	plate_num	96x288	1.75	275x0.9	1282.8
34	refinedet_baseline	480x360	123	275	75.9
35	RefineDet-Medical_EDD_tf	320x320	9.8	275x0.9	581.6
36	refinedet_pruned_0_8	360x480	25	275	289.1
37	refinedet_pruned_0_92	360x480	10.1	275	622.5
38	refinedet_pruned_0_96	360x480	5.1	275	903.3
39	refinedet_VOC_tf	320x320	81.9	275x0.9	97.4
40	reid	80x160	0.95	275	5219.5
41	resnet18	224x224	3.7	275	1948.1
42	resnet50	224x224	7.7	275	789.1
43	resnet50_pt	224x224	4.1	275	764.2
44	resnet50_tf2	224x224	7.7	275x0.9	711.7
45	resnet_v1_101_tf	224x224	14.4	275	460
46	resnet_v1_152_tf	224x224	21.8	275	306.7

Table 48: **U50LV Performance with 9E275 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
47	resnet_v1_50_tf	224x224	7	275	885.6
48	salsanext_pt	64x2048	20.4	275x0.9	139.2
49	SemanticFPN_cityscapes_pt	256x512	10	275x0.9	585.7
50	semantic_seg_citys_tf2	512x1024	54	275x0.9	67.8
51	sp_net	128x224	0.55	275	2590.4
52	squeezenet	227x227	0.76	275	4432.2
53	squeezenet_pt	224x224	0.82	275	2817
54	ssd_adas_pruned_0_95	360x480	6.3	275	904
55	ssd_pedestrian_pruned_0_97	360x360	5.9	275x0.9	768.2
56	ssd_resnet_50_fpn_coco_tf	640x640	178.4	275x0.9	43.2
57	ssd_traffic_pruned_0_9	360x480	11.6	275	595.9
58	tiny_yolov3_vmss	416x416	5.46	275x0.9	1193.1
59	unet_chaos-CT_pt	512x512	23.3	275x0.9	81.2
60	vgg_16_tf	224x224	31	275	221.8
61	vgg_19_tf	224x224	39.3	275	184.8
62	vpgnet_pruned_0_99	480x640	2.5	275x0.9	620.7
63	yolov2_voc	448x448	34	275x0.9	225.3
64	yolov2_voc_pruned_0_66	448x448	11.6	275x0.9	567.2
65	yolov2_voc_pruned_0_71	448x448	9.9	275x0.9	665.3
66	yolov2_voc_pruned_0_77	448x448	7.8	275x0.9	807.3
67	yolov3_adas_pruned_0_9	256x512	5.5	275x0.9	869.6
68	yolov3_bdd	288x512	53.7	275x0.9	104.5
69	yolov3_voc	416x416	65.4	275x0.9	106
70	yolov3_voc_tf	416x416	65.6	275x0.9	106.3
71	yolov4_leaky_spp_m	416x416	60.1	275x0.9	112.5

U50LV Performance with 10E275 MHz DPUCAHX8H

The following table shows the throughput performance (in frames/sec or fps) for various neural network samples on U50LV Gen3x4 with DPUCAHX8H running at 10E@275 MHz.

Table 49: **U50LV Performance with 10E275 MHz DPUCAHX8H**

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	275	2455.1

Table 49: **U50LV Performance with 10E275 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
2	densebox_640_360	360x640	1.1	275	1100.6
3	ENet_cityscapes_pt	512x1024	8.6	275x0.9	98.5
4	face_landmark	96x72	0.14	275	14684.5
5	face-quality	80x60	0.06	275x0.9	19370.2
6	face-quality_pt	80x60	0.06	275x0.9	19431.5
7	facerec_resnet20	112x96	3.5	275	2029.5
8	facerec-resnet20_mixed_pt	112x96	3.5	275x0.9	1851.5
9	facerec_resnet64	112x96	11	275	751.4
10	facereid-large_pt	96x96	0.5	275x0.9	10469.8
11	facereid-small_pt	80x80	0.09	275x0.9	21452.8
12	fpn	256x512	8.9	275x0.9	561.6
13	FPN_Res18_Medical_segmentation	320x320	45.3	275x0.9	140.1
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	275x0.9	315.4
15	inception_resnet_v2_tf	299x299	26.4	275x0.8	209.4
16	inception_v1	224x224	3.2	275x0.9	1648.4
17	inception_v1_tf	224x224	3	275x0.9	1669.9
18	inception_v2	224x224	4	275x0.9	1319.4
19	inception_v3	299x299	11.4	275x0.8	493.6
20	inception_v3_pt	299x299	5.7	275x0.8	493.7
21	inception_v3_tf	299x299	11.5	275x0.8	493.6
22	inception_v3_tf2	299x299	11.5	275x0.9	561.4
23	inception_v4	299x299	24.5	275x0.8	226.4
24	inception_v4_2016_09_09_tf	299x299	24.6	275x0.8	226.4
25	medical_seg_cell_tf2	128x128	5.3	275x0.9	1582.5
26	MLPerf_resnet50_v1.5_tf	224x224	8.19	275x0.9	765.5
27	mlperf_ssd_resnet34_tf	1200x1200	433	275x0.9	20.7
28	multi_task	288x512	14.8	275x0.9	456.2
29	openpose_pruned_0_3	368x368	49.9	275x0.7	34.6
30	personreid-res18_pt	176x80	1.1	275x0.9	5134.6
31	personreid-res50_pt	256x128	5.4	275x0.9	1230.1
32	plate_detection	320x320	0.49	275	5252.2
33	plate_num	96x288	1.75	275x0.9	1341
34	refinedet_baseline	480x360	123	275x0.7	26
35	RefineDet-Medical_EDD_tf	320x320	9.8	275x0.9	638.1
36	refinedet_pruned_0_8	360x480	25	275x0.9	287.8
37	refinedet_pruned_0_92	360x480	10.1	275x0.9	610
38	refinedet_pruned_0_96	360x480	5.1	275x0.9	871.6

Table 49: **U50LV Performance with 10E275 MHz DPUCAHX8H** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
39	refinedet_VOC_tf	320x320	81.9	275x0.9	107.7
40	reid	80x160	0.95	275	5782.5
41	resnet18	224x224	3.7	275x0.9	1936.7
42	resnet50	224x224	7.7	275x0.9	790.2
43	resnet50_pt	224x224	4.1	275x0.9	765.4
44	resnet50_tf2	224x224	7.7	275x0.9	790.3
45	resnet_v1_101_tf	224x224	14.4	275x0.9	460.6
46	resnet_v1_152_tf	224x224	21.8	275x0.9	307
47	resnet_v1_50_tf	224x224	7	275x0.9	887.2
48	salsanext_pt	64x2048	20.4	275x0.9	132.7
49	SemanticFPN_cityscapes_pt	256x512	10	275x0.9	621.8
50	semantic_seg_citys_tf2	512x1024	54	275x0.9	72.9
51	sp_net	128x224	0.55	275	2745.2
52	squeezenet	227x227	0.76	275x0.9	4184.6
53	squeezenet_pt	224x224	0.82	275x0.9	2628.3
54	ssd_adas_pruned_0_95	360x480	6.3	275x0.8	807.6
55	ssd_pedestrian_pruned_0_97	360x360	5.9	275x0.9	827.9
56	ssd_resnet_50_fpn_coco_tf	640x640	178.4	275x0.8	41.9
57	ssd_traffic_pruned_0_9	360x480	11.6	275	644.9
58	tiny_yolov3_vmss	416x416	5.46	275x0.9	1309.9
59	unet_chaos-CT_pt	512x512	23.3	275x0.9	89.6
60	vgg_16_tf	224x224	31	275x0.9	223.2
61	vgg_19_tf	224x224	39.3	275x0.9	186.2
62	vpgnet_pruned_0_99	480x640	2.5	275x0.9	643
63	yolov2_voc	448x448	34	275x0.8	220.8
64	yolov2_voc_pruned_0_66	448x448	11.6	275x0.8	557.8
65	yolov2_voc_pruned_0_71	448x448	9.9	275x0.8	653.5
66	yolov2_voc_pruned_0_77	448x448	7.8	275x0.8	795.3
67	yolov3_adas_pruned_0_9	256x512	5.5	275x0.8	815.6
68	yolov3_bdd	288x512	53.7	275x0.8	101.6
69	yolov3_voc	416x416	65.4	275x0.8	102.9
70	yolov3_voc_tf	416x416	65.6	275x0.8	103.4
71	yolov4_leaky_spp_m	416x416	60.1	275x0.8	109.1

Model end-to-end Performance on U50LV 300 MHz Single Core DPUCAHX8L

The following table shows the throughput performance (in frames/sec or fps) for various neural network samples on U50LV Gen3x4 with single DPUCAHX8L running at 300 MHz.

Table 50: **Model end-to-end Performance on U50LV 300 MHz Single Core DPUCAHX8L**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	187.2	365.5
2	densebox_640_360	360x640	1.1	89.2	164.7
3	ENet_cityscapes_pt	512x1024	8.6	4.3	5.1
4	face_landmark	96x72	0.14	1678.7	4507.5
5	face-quality	80x60	0.06	1983.3	6735.3
6	face-quality_pt	80x60	0.06	2048.9	6828.4
7	facerec_resnet20	112x96	3.5	234.2	288.3
8	facerec-resnet20_mixed_pt	112x96	3.5	234.9	288.1
9	facerec_resnet64	112x96	11	114.0	127.4
10	facereid-small_pt	80x80	0.09	1492.9	4365.4
11	fpn	256x512	8.9	23.1	33.7
12	FPN_Res18_Medical_segmentation	320x320	45.3	12.3	13.2
13	FPN-resnet18_covid19-seg_pt	352x352	22.7	64.8	80.5
14	inception_resnet_v2_tf	299x299	26.4	27.1	30.1
15	inception_v1	224x224	3.2	202.1	284.8
16	inception_v1_tf	224x224	3	182.1	295.9
17	inception_v2	224x224	3.88	137.1	180.4
18	inception_v3	299x299	11.4	80.1	97.1
19	inception_v3_pt	299x299	5.7	72.3	99.1
20	inception_v3_tf	299x299	11.5	72.0	98.2
21	inception_v3_tf2	299x299	11.5	68.7	88.6
22	inception_v4	299x299	24.5	41.1	45.9
23	inception_v4_2016_09_09_tf	299x299	24.6	38.9	45.8
24	medical_seg_cell_tf2	128x128	5.3	70.7	75.8
25	MLPerf_resnet50_v1.5_tf	224x224	8.19	66.6	77.8
26	mlperf_ssd_resnet34_tf	1200x1200	433	3.8	6.0
27	mobilenet_1_0_224_tf2	224x224	1.1	465.0	1613.2
28	mobilenet_v1_0_5_160_tf	160x160	0.15	918.1	4718.0
29	mobilenet_v1_1_0_224_tf	224x224	1.1	473.7	1658.4
30	mobilenet_v2	224x224	0.6	431.4	984.5
31	mobilenet_v2_1_0_224_tf	224x224	0.6	377.0	955.3
32	mobilenet_v2_1_4_224_tf	224x224	1.2	306.7	702.3

Table 50: **Model end-to-end Performance on U50LV 300 MHz Single Core DPUCAHX8L** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
33	multi_task	288x512	14.8	13.1	17.7
34	openpose_pruned_0_3	368x368	49.9	8.7	14.2
35	personreid-res50_pt	256x128	5.4	84.1	97.4
36	plate_detection	320x320	0.49	369.9	1091.1
37	refinedet_baseline	480x360	123	22.2	25.7
38	RefineDet-Medical_EDD_tf	320x320	9.8	75.1	140.1
39	refinedet_pruned_0_8	360x480	25	46.4	64.6
40	refinedet_pruned_0_92	360x480	10.1	50.9	72.5
41	refinedet_pruned_0_96	360x480	5.1	59.0	89.1
42	refinedet_VOC_tf	320x320	81.9	23.3	39.0
43	reid	80x160	0.95	378.8	542.2
44	resnet18	224x224	3.7	190.3	263.1
45	resnet50	224x224	7.7	69.5	77.2
46	resnet50_pt	224x224	4.1	66.9	78.6
47	resnet50_tf2	224x224	7.7	69.7	78.6
48	resnet_v1_101_tf	224x224	14.4	44.9	50.6
49	resnet_v1_152_tf	224x224	21.8	31.6	33.2
50	resnet_v1_50_tf	224x224	7	74.8	90.3
51	retinaface	360x640	1.11	82.5	183.9
52	salsanext_pt	64x2048	20.4	6.7	9.8
53	SemanticFPN_cityscapes_pt	256x512	10	22.9	34.0
54	semantic_seg_citys_tf2	512x1024	54	4.4	5.3
55	sp_net	128x224	0.55	1153.4	8313.6
56	squeezenet	227x227	0.76	325.0	680.3
57	squeezenet_pt	224x224	0.82	246.0	420.3
58	ssd_adas_pruned_0_95	360x480	6.3	60.7	116.7
59	ssdlite_mobilenet_v2_coco_tf	300x300	1.5	189.6	487.7
60	ssd_mobilenet_v1_coco_tf	300x300	2.5	223.3	769.8
61	ssd_mobilenet_v2	360x480	6.6	34.0	135.5
62	ssd_mobilenet_v2_coco_tf	300x300	3.8	111.5	203.1
63	ssd_pedestrian_pruned_0_97	360x360	5.9	22.8	30.2
64	ssd_traffic_pruned_0_9	360x480	11.6	49.8	106.2
65	vgg_16_tf	224x224	31	46.0	48.9
66	vgg_19_tf	224x224	39.3	41.1	44.4
67	vpgnet_pruned_0_99	480x640	2.5	17.9	20.1

U280 Performance

The Xilinx® Alveo U280 Data Center accelerator cards are peripheral component interconnect express (PCIe®) Gen3x16 compliant and Gen4x8 compatible cards featuring the Xilinx 16 nm UltraScale+ technology. In this release, DPU is implemented in program logic for deep learning inference acceleration.

Note: Some models cannot run at the highest frequency of DPU and need DPU frequency reduction. See the For Edge for DPU frequency reduction operation.

U280 Performance with 14E300 MHz DPU

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on U280 Gen3x16 with DPUCAHX8H running at 14E@300 MHz.

Table 51: **U280 Performance with 14E300 MHz DPU**

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	300x0.5	2999
2	densebox_640_360	360x640	1.1	300x0.5	1338.8
3	ENet_cityscapes_pt	512x1024	8.6	300x0.5	89.3
4	face_landmark	96x72	0.14	300x0.5	12384.7
5	face-quality	80x60	0.06	300x0.5	23735.6
6	face-quality_pt	80x60	0.06	300x0.5	23559.9
7	facerec_resnet20	112x96	3.5	300x0.5	1602.1
8	facerec-resnet20_mixed_pt	112x96	3.5	300x0.5	1602.2
9	facerec_resnet64	112x96	11	300x0.5	580.5
10	facereid-large_pt	96x96	0.5	300x0.5	9374
11	facereid-small_pt	80x80	0.09	300x0.5	20726
12	fpn	256x512	8.9	300x0.5	416.8
13	FPN_Res18_Medical_segmentation	320x320	45.3	300x0.5	116.3
14	FPN-resnet18_covid19-seg_pt	352x352	22.7	300x0.5	263.2
15	inception_resnet_v2_tf	299x299	26.4	300x0.5	191
16	inception_v1	224x224	3.2	300x0.5	1367.4
17	inception_v1_tf	224x224	3	300x0.5	1415.4
18	inception_v2	224x224	4	300x0.5	1096.3
19	inception_v3	299x299	11.4	300x0.5	442.2
20	inception_v3_pt	299x299	5.7	300x0.5	442.4
21	inception_v3_tf	299x299	11.5	300x0.5	442.2
22	inception_v3_tf2	299x299	11.5	300x0.5	451.7

Table 51: **U280 Performance with 14E300 MHz DPU** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
23	inception_v4	299x299	24.5	300x0.5	209.2
24	inception_v4_2016_09_09_tf	299x299	24.6	300x0.5	209.3
25	medical_seg_cell_tf2	128x128	5.3	300x0.5	1275.5
26	MLPerf_resnet50_v1.5_tf	224x224	8.19	300x0.5	647.3
27	mlperf_ssd_resnet34_tf	1200x1200	433	300x0.5	13.9
28	multi_task	288x512	14.8	300x0.5	344.2
29	openpose_pruned_0_3	368x368	49.9	300x0.5	37.3
30	personreid-res18_pt	176x80	1.1	300x0.5	4529.3
31	personreid-res50_pt	256x128	5.4	300x0.5	1044.6
32	plate_detection	320x320	0.49	300x0.5	4146.4
33	plate_num	96x288	1.75	300x0.5	1201.5
34	refinedet_baseline	480x360	123	300x0.5	55.8
35	RefineDet-Medical_EDD_tf	320x320	9.8	300x0.5	512.9
36	refinedet_pruned_0_8	360x480	25	300x0.5	183.9
37	refinedet_pruned_0_92	360x480	10.1	300x0.5	455
38	refinedet_pruned_0_96	360x480	5.1	300x0.5	637.5
39	refinedet_VOC_tf	320x320	81.9	300x0.5	87.8
40	reid	80x160	0.95	300x0.5	4769.6
41	resnet18	224x224	3.7	300x0.5	1660.3
42	resnet50	224x224	7.7	300x0.6	801.1
43	resnet50_pt	224x224	4.1	300x0.6	776
44	resnet50_tf2	224x224	7.7	300x0.5	668
45	resnet_v1_101_tf	224x224	14.4	300x0.5	390.1
46	resnet_v1_152_tf	224x224	21.8	300x0.5	260
47	resnet_v1_50_tf	224x224	7	300x0.5	751.9
48	salsanext_pt	64x2048	20.4	300x0.9	108.5
49	SemanticFPN_cityscapes_pt	256x512	10	300x0.5	429.2
50	semantic_seg_citys_tf2	512x1024	54	300x0.5	65.5
51	sp_net	128x224	0.55	300x0.5	2939.3
52	squeezenet	227x227	0.76	300x0.5	3264.5
53	squeezenet_pt	224x224	0.82	300x0.5	2140.1
54	ssd_adas_pruned_0_95	360x480	6.3	300x0.5	641.6
55	ssd_pedestrian_pruned_0_97	360x360	5.9	300x0.5	544.6
56	ssd_resnet_50_fpn_coco_tf	640x640	178.4	300x0.5	36.9
57	ssd_traffic_pruned_0_9	360x480	11.6	300x0.5	432.8
58	tiny_yolov3_vmss	416x416	5.46	300x0.5	962.9
59	unet_chaos-CT_pt	512x512	23.3	300x0.5	129.8

Table 51: **U280 Performance with 14E300 MHz DPU** *(cont'd)*

No	Neural Network	Input Size	GOPS	DPU Frequency (MHz)	Performance (fps) (Multiple thread)
60	vgg_16_tf	224x224	31	300x0.5	188.6
61	vgg_19_tf	224x224	39.3	300x0.5	157.2
62	vpgnet_pruned_0_99	480x640	2.5	300x0.5	634.9
63	yolov2_voc	448x448	34	300x0.5	202.4
64	yolov2_voc_pruned_0_66	448x448	11.6	300x0.5	499.3
65	yolov2_voc_pruned_0_71	448x448	9.9	300x0.5	583.4
66	yolov2_voc_pruned_0_77	448x448	7.8	300x0.5	695.3
67	yolov3_adas_pruned_0_9	256x512	5.5	300x0.5	810.9
68	yolov3_bdd	288x512	53.7	300x0.5	93.7
69	yolov3_voc	416x416	65.4	300x0.5	96.9
70	yolov3_voc_tf	416x416	65.6	300x0.5	97
71	yolov4_leaky_spp_m	416x416	60.1	300x0.5	99.4

Model End-to-End Performance on U280 250 MHz dual cores DPUCAHX8L

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on U280 Gen3x16 with dual DPUCAHX8L running at 300 MHz.

Table 52: **Model End-to-End Performance on U280 250 MHz Dual Cores DPUCAHX8L**

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	densebox_320_320	320x320	0.49	190.6	464.1
2	densebox_640_360	360x640	1.1	92.4	209.5
3	ENet_cityscapes_pt	512x1024	8.6	4.5	8.6
4	face_landmark	96x72	0.14	1848.4	6682.2
5	face-quality	80x60	0.06	2131.1	9672.5
6	face-quality_pt	80x60	0.06	2311.7	9949.8
7	facerec_resnet20	112x96	3.5	255.8	430.6
8	facerec-resnet20_mixed_pt	112x96	3.5	255.8	446.8
9	facerec_resnet64	112x96	11	127.0	247.9
10	facereid-small_pt	80x80	0.09	1756.7	6408.1
11	fpn	256x512	8.9	24.5	55.3
12	FPN_Res18_Medical_segmentation	320x320	45.3	12.9	20.4
13	FPN-resnet18_covid19-seg_pt	352x352	22.7	69.6	143.8
14	inception_resnet_v2_tf	299x299	26.4	29.8	60.5

Table 52: **Model End-to-End Performance on U280 250 MHz Dual Cores DPUCAHX8L** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
15	inception_v1	224x224	3.2	214.9	504.8
16	inception_v1_tf	224x224	3	201.0	491.4
17	inception_v2	224x224	3.88	148.0	315.7
18	inception_v3	299x299	11.4	85.5	183.0
19	inception_v3_pt	299x299	5.7	80.9	180.4
20	inception_v3_tf	299x299	11.5	80.8	179.4
21	inception_v3_tf2	299x299	11.5	77.6	171.1
22	inception_v4	299x299	24.5	44.4	91.2
23	inception_v4_2016_09_09_tf	299x299	24.6	43.2	88.5
24	medical_seg_cell_tf2	128x128	5.3	72.8	102.1
25	MLPerf_resnet50_v1.5_tf	224x224	8.19	74.1	153.5
26	mlperf_ssd_resnet34_tf	1200x1200	433	4.1	10.7
27	mobilenet_1_0_224_tf2	224x224	1.1	558.0	2328.6
28	mobilenet_v1_0_5_160_tf	160x160	0.15	1067.3	6310.9
29	mobilenet_v1_1_0_224_tf	224x224	1.1	558.7	2520.8
30	mobilenet_v2	224x224	0.6	480.1	1466.8
31	mobilenet_v2_1_0_224_tf	224x224	0.6	436.4	1500.6
32	mobilenet_v2_1_4_224_tf	224x224	1.2	370.4	1141.7
33	multi_task	288x512	14.8	13.4	31.0
34	openpose_pruned_0_3	368x368	49.9	9.6	25.4
35	personreid-res50_pt	256x128	5.4	92.4	187.4
36	plate_detection	320x320	0.49	390.2	1527.8
37	refinedet_baseline	480x360	123	25.1	51.2
38	RefineDet-Medical_EDD_tf	320x320	9.8	81.3	216.8
39	refinedet_pruned_0_8	360x480	25	49.3	108.8
40	refinedet_pruned_0_92	360x480	10.1	53.5	112.7
41	refinedet_pruned_0_96	360x480	5.1	61.7	133.8
42	refinedet_VOC_tf	320x320	81.9	24.8	69.0
43	reid	80x160	0.95	407.4	796.8
44	resnet18	224x224	3.7	203.2	396.2
45	resnet50	224x224	7.7	65.7	126.3
46	resnet50_pt	224x224	4.1	64.4	131.7
47	resnet50_tf2	224x224	7.7	76.0	152.9
48	resnet_v1_101_tf	224x224	14.4	49.8	99.5
49	resnet_v1_152_tf	224x224	21.8	34.9	63.5
50	resnet_v1_50_tf	224x224	7	84.0	168.5

Table 52: **Model End-to-End Performance on U280 250 MHz Dual Cores DPUCAHX8L** *(cont'd)*

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
51	retinaface	360x640	1.11	91.8	207.0
52	salsanext_pt	64x2048	20.4	7.0	16.1
53	SemanticFPN_cityscapes_pt	256x512	10	24.5	44.6
54	semantic_seg_citys_tf2	512x1024	54	4.6	9.2
55	sp_net	128x224	0.55	330.6	7747.6
56	squeezenet	227x227	0.76	361.8	1043.1
57	squeezenet_pt	224x224	0.82	266.5	688.4
58	ssd_adas_pruned_0_95	360x480	6.3	63.4	160.3
59	ssdlite_mobilenet_v2_coco_tf	300x300	1.5	209.8	714.3
60	ssd_mobilenet_v1_coco_tf	300x300	2.5	268.1	1095.0
61	ssd_mobilenet_v2	360x480	6.6	38.7	169.3
62	ssd_mobilenet_v2_coco_tf	300x300	3.8	128.7	303.7
63	ssd_pedestrian_pruned_0_97	360x360	5.9	23.0	35.1
64	ssd_traffic_pruned_0_9	360x480	11.6	51.6	158.0
65	vgg_16_tf	224x224	31	50.3	99.0
66	vgg_19_tf	224x224	39.3	45.4	89.6
67	vpgnet_pruned_0_99	480x640	2.5	18.0	20.5

API Reference

vitis::ai::Classification

Base class for detecting objects in the input image (cv::Mat).

Input is an image (cv::Mat).

Output is index and score of objects in the input image.

Sample code:

```
auto image = cv::imread("sample_classification.jpg");
auto network = vitis::ai::Classification::create(
                 "resnet50");
auto result = network->run(image);
for (const auto &r : result.scores) {
   auto score = r.score;
   auto index = result.lookup(r.index);
}
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Classification` class:

Table 53: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Classification` >	create	const std::string & model_name bool need_preprocess
`vitis::ai::ClassificationResult`	run	const cv::Mat & image
std::vector< `vitis::ai::ClassificationResult` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void

Table 53: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `Classification`.

Prototype

```
std::unique_ptr< Classification > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 54: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name.
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Classification` class.

run

Function to get running results of the classification neuron network.

Prototype

```
vitis::ai::ClassificationResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 55: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`ClassificationResult`.

run

Function to get running results of the classification neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::ClassificationResult > run(const std::vector<
cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 56: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of batch input images (vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `ClassificationResult`.

getInputWidth

Function to get InputWidth of the classification network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

Input width of the classification network

getInputHeight

Function to get InputHeight of the classification network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

Input height of the classification network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** The batch size of different DPU core may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::Covid19Segmentation

Base class for `Covid19Segmentation`.

Declaration `Covid19Segmentation` Network Branch positive detection: label0-negative, label1-positive Branch Infected area detection: label0-negative, label1-consolidation, label2-GGO Input is an image (cv:Mat).

Output is result of running the `Covid19Segmentation` network.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Covid19Segmentation` class:

Table 57: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Covid19Segmentation` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void

Table 57: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
int	getInputHeight	void
size_t	get_input_batch	void
Covid19SegmentationResult	run_8UC1	const cv::Mat & image
std::vector< Covid19SegmentationResult >	run_8UC1	const std::vector< cv::Mat > & images
Covid19SegmentationResult	run_8UC3	const cv::Mat & image
std::vector< Covid19SegmentationResult >	run_8UC3	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Covid19Segmentation`.

Prototype

```
std::unique_ptr< Covid19Segmentation > create(const std::string
&model_name, bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 58: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Covid19Segmentation` class.

getInputWidth

Function to get InputWidth of the covid19segmentation network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the covid19segmentation network.

getInputHeight

Function to get InputHeight of the covid19segmentation network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the covid19segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run_8UC1

Function to get running result of the covid19segmentation network.

***Note*:** The type of CV_8UC1 of the covid19segmentation result.

Prototype

```
Covid19SegmentationResult run_8UC1(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 59: **run_8UC1 Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

Covid19segmentation output data.

run_8UC1

Function to get running results of the covid19segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC1 of the covid19segmentation result.

Prototype

```
std::vector< Covid19SegmentationResult > run_8UC1(const std::vector<
cv::Mat > &images)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 60: **run_8UC1 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of Covid19segmentationResult.

run_8UC3

Function to get running result of the covid19segmentation network.

***Note*:** The type of CV_8UC3 of the covid19segmentation result.

Prototype

```
Covid19SegmentationResult run_8UC3(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 61: **run_8UC3 Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

Covid19segmentation image and shape.

run_8UC3

Function to get running results of the covid19segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC3 of the Result's covid19segmentation.

Prototype

```
std::vector< Covid19SegmentationResult > run_8UC3(const std::vector<
cv::Mat > &images)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 62: **run_8UC3 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of Covid19SegmentationResult.

vitis::ai::Covid19Segmentation8UC1

The Class of `Covid19Segmentation8UC1`, this class run function return a cv::Mat with the type is cv_8UC1.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Covid19Segmentation8UC1` class:

Table 63: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Covid19Segmentation8UC1` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
Covid19SegmentationResult	run	const cv::Mat & image
std::vector< Covid19SegmentationResult >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Covid19Segmentation8UC1`.

Prototype

```
std::unique_ptr< Covid19Segmentation8UC1 > create(const std::string
&model_name, bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 64: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Covid19Segmentation8UC1` class.

getInputWidth

Function to get InputWidth of the covid19segmentation network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the covid19segmentation network.

getInputHeight

Function to get InputHeight of the covid19segmentation network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputHeight of the covid19segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be differnt. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function to get running result of the covid19segmentation network.

***Note*:** The result cv::Mat of the type is CV_8UC1.

Prototype

```
Covid19SegmentationResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 65: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of the image (cv::Mat)

Returns

The result of covid19segmentation network.

run

Function to get running results of the covid19segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC1 of the Result's covid19segmentation.

Prototype

```
std::vector< Covid19SegmentationResult > run(const std::vector< cv::Mat >
&images);
```

Parameters

The following table lists the `run` function arguments.

Table 66: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of Covid19SegmentationResult.

vitis::ai::Covid19Segmentation8UC3

The Class of `Covid19Segmentation8UC3`, this class run function return a cv::Mat with the type is cv_8UC3.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Covid19Segmentation8UC3` class:

Table 67: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Covid19Segmentation8UC3` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
Covid19SegmentationResult	run	const cv::Mat & image
std::vector< Covid19SegmentationResult >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Covid19Segmentation8UC3`.

Prototype

```
std::unique_ptr< Covid19Segmentation8UC3 > create(const std::string
&model_name, bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 68: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Covid19Segmentation8UC3` class.

getInputWidth

Function to get InputWidth of the covid19segmentation network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the covid19segmentation network.

getInputHeight

Function to get InputWidth of the covid19segmentation network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputWidth of the covid19segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function to get running result of the covid19segmentation network.

***Note*:** The result cv::Mat of the type is CV_8UC3.

Prototype

```
Covid19SegmentationResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 69: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of the image (cv::Mat)

Returns

Covid19SegmentationResult The result of covid19segmentation network.

run

Function to get running results of the covid19segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC3 of the Result's covid19segmentation.

Prototype

```
std::vector< Covid19SegmentationResult > run(const std::vector< cv::Mat >
&images);
```

Parameters

The following table lists the `run` function arguments.

Table 70: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of Covid19SegmentationResult.

vitis::ai::FaceDetect

Base class for detecting the position of faces in the input image (cv::Mat).

Input is an image (cv::Mat).

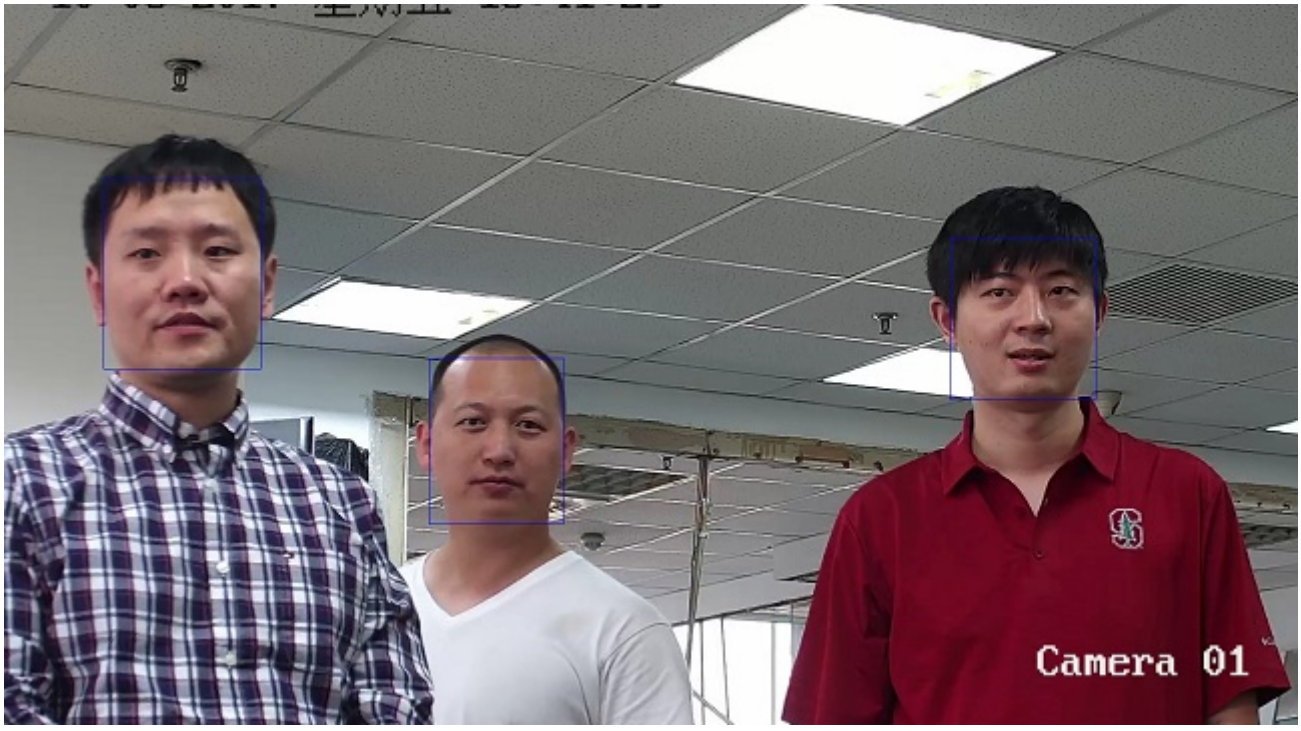
Output is a vector of position and score for faces in the input image.

Sample code:

```
auto image = cv::imread("sample_facedetect.jpg");
auto network = vitis::ai::FaceDetect::create(
               "densebox_640_360",
               true);
auto result = network->run(image);
for (const auto &r : result.rects) {
   auto score = r.score;
   auto x = r.x * image.cols;
   auto y = r.y * image.rows;
   auto width = r.width * image.cols;
   auto height = r.height * image.rows;
}
```

Display of the model results:

Figure 30: **result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::FaceDetect` class:

Table 71: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `FaceDetect` >	create	const std::string & model_name bool need_preprocess
std::unique_ptr< `FaceDetect` >	create	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
float	getThreshold	void
void	setThreshold	float threshold
`FaceDetectResult`	run	const cv::Mat & img
std::vector< `FaceDetectResult` >	run	const std::vector< cv::Mat > & imgs

Functions

create

Factory function to get instance of derived classes of class `FaceDetect`.

Prototype

```
std::unique_ptr< FaceDetect > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 72: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `FaceDetect` class.

create

Prototype

```
std::unique_ptr< FaceDetect > create(const std::string &model_name,
xir::Attrs *attrs, bool need_preprocess=true);
```

getInputWidth

Function to get InputWidth of the facedetect network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the facedetect network

getInputHeight

Function to get InputHeight of the facedetect network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the facedetect network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

getThreshold

Function to get detect threshold.

Prototype

```
float getThreshold() const =0;
```

Returns

The detect threshold. The value ranges from 0 to 1.

setThreshold

Function of update detect threshold.

***Note*:** The detection results will filter by detect threshold (score >= threshold).

Prototype

```
void setThreshold(float threshold)=0;
```

Parameters

The following table lists the `setThreshold` function arguments.

Table 73: **setThreshold Arguments**

Type	Name	Description
float	threshold	The detect threshold. The value ranges from 0 to 1.

Returns

run

Function to get running result of the facedetect network.

Prototype

```
FaceDetectResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 74: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input Data ,input image (cv::Mat) need to be resized to InputWidth and InputHeight required by the network.

Returns

The detection result of the face detect network, filtered by score >= det_threshold

run

Function to get running results of the facedetect neuron network in batch mode.

Prototype

```
std::vector< FaceDetectResult > run(const std::vector< cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 75: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of `FaceDetectResult`.

vitis::ai::FaceFeature

Base class for getting the features of a face image (cv::Mat).

Input is a face image (cv::Mat).

Output is the features of a face in the input image.

Float sample code :

Note: Two interfaces are provided to get the float features or fixed features. They return `FaceFeatureFloatResult` or `FaceFeatureFixedResult`.

```
cv:Mat image = cv::imread("test_face.jpg");
auto network  = vitis::ai::FaceFeature::create("facerec_resnet20", true);
auto result = network->run(image);
auto features = result.feature;
```

Fixed sample code :

```
cv:Mat image = cv::imread("test_face.jpg");
auto network  = vitis::ai::FaceFeature::create("facerec_resnet20", true);
auto result = network->run_fixed(image);
auto features = result.feature;
```

Similarity calculation formula :

Calaculate the similarity of two images:

```
auto result_fixed = network->run_fixed(image);
auto result_fixed2 = network->run_fixed(image2);
auto similarity_original = feature_compare(result_fixed.feature->data(),
                                           result_fixed2.feature->data());
float similarity_mapped = score_map(similarity_original);
```

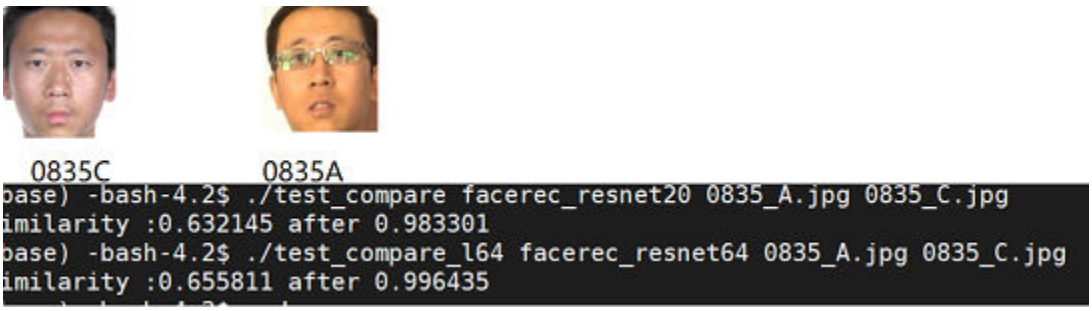
Fixed compare code :

```
 float feature_norm(const int8_t *feature) {
    int sum = 0;
    for (int i = 0; i < 512; ++i) {
        sum += feature[i] * feature[i];
    }
    return 1.f / sqrt(sum);
 }

/// This function is used for computing dot product of two vector
static float feature_dot(const int8_t *f1, const int8_t *f2) {
   int dot = 0;
   for (int i = 0; i < 512; ++i) {
      dot += f1[i] * f2[i];
   }
   return (float)dot;
}

float feature_compare(const int8_t *feature, const int8_t *feature_lib){
   float norm = feature_norm(feature);
   float feature_norm_lib = feature_norm(feature_lib);
   return feature_dot(feature, feature_lib) * norm * feature_norm_lib;
}

/// This function is used for model "facerec_resnet20"
float score_map_l20(float score) { return 1.0 / (1 + exp(-12.4 * score
```

```
+ 3.763)); }

 /// This function is used for type "facerec_resnet64"
 float score_map_l64(float score) { return 1.0 / (1 + exp(-17.0836 * score
+ 5.5707)); }
```

Display of the compare result with a set of images:

Figure 31: **facecompare result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::FaceFeature` class:

Table 76: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `FaceFeature` >	create	const std::string & model_name bool need_preprocess
std::unique_ptr< `FaceFeature` >	create	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`FaceFeatureFloatResult`	run	const cv::Mat & img
`FaceFeatureFixedResult`	run_fixed	const cv::Mat & img
std::vector< `FaceFeatureFloatResult` >	run	const std::vector< cv::Mat > & imgs
std::vector< `FaceFeatureFixedResult` >	run_fixed	const std::vector< cv::Mat > & imgs

Functions

create

Factory function to get an instance of derived classes of class `FaceFeature`.

Prototype

```
std::unique_ptr< FaceFeature > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 77: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `FaceFeature` class.

create

Prototype

```
std::unique_ptr< FaceFeature > create(const std::string &model_name,
xir::Attrs *attrs, bool need_preprocess=true);
```

getInputWidth

Function to get InputWidth of the feature network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the feature network.

getInputHeight

Function to get InputHeight of the feature network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the feature network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function of get running result of the feature network.

Prototype

```
FaceFeatureFloatResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 78: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data for image (cv::Mat) detected by the facedetect network and then rotated and aligned.

Returns

`FaceFeatureFloatResult`

run_fixed

Function of get running result of the feature network.

Prototype

```
FaceFeatureFixedResult run_fixed(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run_fixed` function arguments.

Table 79: **run_fixed Arguments**

Type	Name	Description
const cv::Mat &	img	Input data for image (cv::Mat) detected by the facedetect network and then rotated and aligned.

Returns

`FaceFeatureFixedResult`

run

Function of get running result of the feature network in batch mode.

Prototype

```
std::vector< FaceFeatureFloatResult > run(const std::vector< cv::Mat >
&imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 80: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of batch input images (vector<cv::Mat>) detected by the facedetect network and then rotated and aligned. The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `FaceFeatureFloatResult`.

run_fixed

Function of get running result of the feature network in batch mode.

Prototype

```
std::vector< FaceFeatureFixedResult > run_fixed(const std::vector< cv::Mat
> &imgs)=0;
```

Parameters

The following table lists the `run_fixed` function arguments.

Table 81: **run_fixed Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of batch input images (vector<cv::Mat>) detected by the facedetect network and then rotated and aligned. The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `FaceFeatureFixedResult`.

vitis::ai::FaceLandmark

Base class for detecting five key points, and the score from a face image (cv::Mat).

Input a face image (cv::Mat).

Output score, five key points of the face.

Sample code:

***Note*:** Usually the input image contains only one face. When it contains multiple faces, the functions returns the highest score.

```
cv:Mat image = cv::imread("sample_facelandmark.jpg");
auto landmark  = vitis::ai::FaceLandmark::create("face_landmark");
auto result = landmark->run(image);
float score = result.score;
auto points = result.points;
for(int i = 0; i< 5 ; ++i){
    auto x = points[i].frist  * image.cols;
    auto y = points[i].second * image.rows;
}
```

Display of the model results:

Figure 32: **result image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::FaceLandmark` class:

Table 82: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `FaceLandmark` >	create	const std::string & model_name bool need_preprocess
std::unique_ptr< `FaceLandmark` >	create	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`FaceLandmarkResult`	run	const cv::Mat & input_image
std::vector< `FaceLandmarkResult` >	run	const std::vector< cv::Mat > & input_images

Functions

create

Factory function to get an instance of derived classes of class `FaceLandmark`.

Prototype

```
std::unique_ptr< FaceLandmark > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 83: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `FaceLandmark` class.

create

Prototype

```
std::unique_ptr< FaceLandmark > create(const std::string &model_name,
xir::Attrs *attrs, bool need_preprocess=true);
```

getInputWidth

Function to get InputWidth of the landmark network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the face landmark network.

getInputHeight

Function to get InputHeight of the landmark network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the face landmark network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function to get running result of the face landmark network.

Set data of a face(e.g. data of cv::Mat) and get the five key points.

Prototype

```
FaceLandmarkResult run(const cv::Mat &input_image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 84: **run Arguments**

Type	Name	Description
const cv::Mat &	input_image	Input data of input image (cv::Mat) of detected by the facedetect network and resized as inputwidth and inputheight.

Returns

The struct of `FaceLandmarkResult`

run

Function to get running results of the face landmark neuron network in batch mode.

Prototype

```
std::vector< FaceLandmarkResult > run(const std::vector< cv::Mat >
&input_images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 85: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	input_images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of `FaceLandmarkResult`.

vitis::ai::FaceQuality5pt

Base class for evaluating the quality and five key points coordinate of a face image (cv::Mat).

Input is a face image (cv::Mat).

Output is the quality and five key points coordinate of a face in the input image.

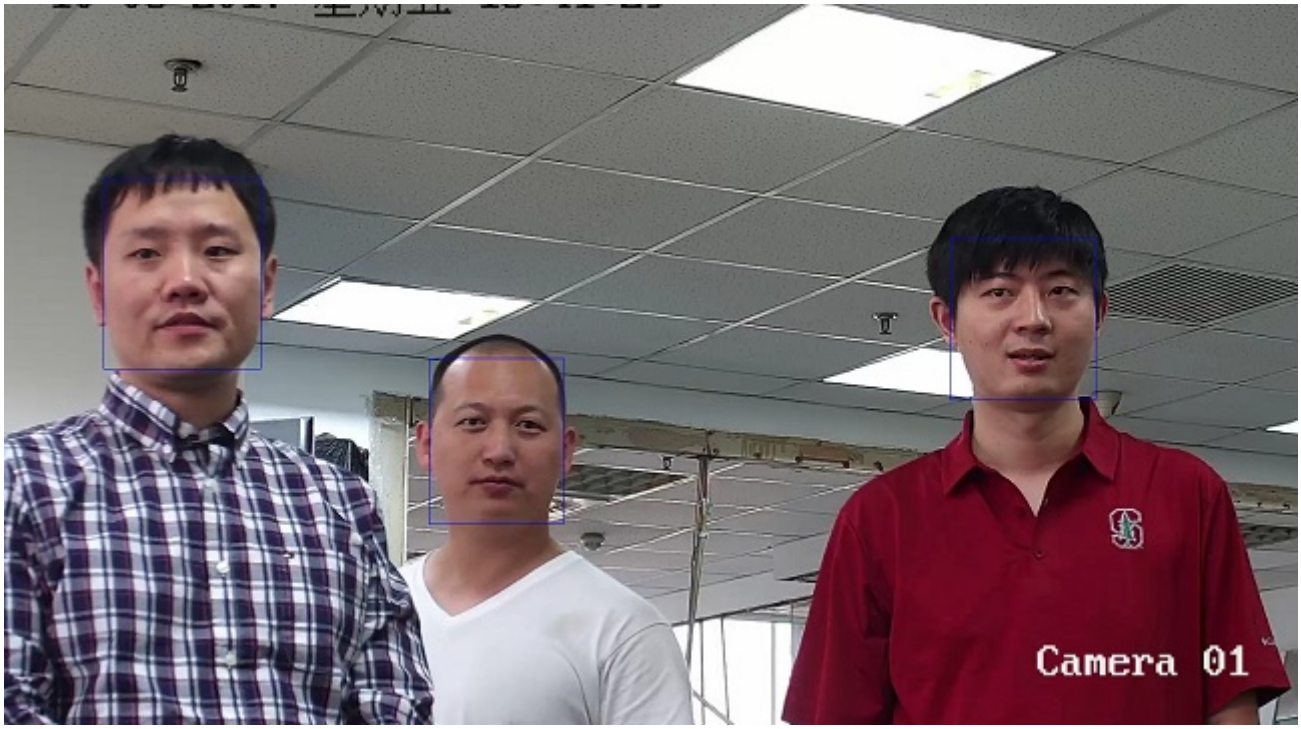
Sample code : Display of the `FaceQuality5pt` model results:

```
cv:Mat image = cv::imread("sample_facequality5pt.jpg");
auto network =
      vitis::ai::FaceQuality5pt::create("face-quality", true);
auto result = network->run(image);
auto quality = result.score;
auto points = result.points;
for(int i = 0; i< 5 ; ++i){
    auto x = points[i].frist  * image.cols;
    auto y = points[j].second * image.rows;
}
```

***Note*:** Default mode is day, if day night switch network is used and the background of the input image is night, please use API setMode

```
network->setMode(vitis::ai::FaceQuality5pt::Mode::NIGHT);
```

Figure 33: **result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::FaceQuality5pt` class:

Table 86: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `FaceQuality5pt` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`Mode`	getMode	void

Table 86: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	setMode	`Mode` mode
`FaceQuality5ptResult`	run	const cv::Mat & img
std::vector< `FaceQuality5ptResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `FaceQuality5pt`.

Prototype

```
std::unique_ptr< FaceQuality5pt > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 87: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `FaceQuality5pt` class.

getInputWidth

Function to get InputWidth of the facequality5pt network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the facequality5pt network.

getInputHeight

Function to get InputHeight of the facequality5pt network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of facequality5pt network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

getMode

Function to get Mode.

Prototype

```
            Mode getMode()=0;
```

setMode

Function to set Mode.

Prototype

```
void setMode(Mode mode)=0;
```

Parameters

The following table lists the `setMode` function arguments.

Table 88: **setMode Arguments**

Type	Name	Description
`Mode`	mode	Type::Mode

Returns

mode

run

Function of get running result of the facequality5pt network.

Prototype

```
FaceQuality5ptResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 89: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat) of detected counterpart and resized to InputWidth and InputHeight required by the network.

Returns

The result of the facequality5pt network.

run

Function of get running results of the facequality5pt network in batch mode.

Prototype

```
std::vector< FaceQuality5ptResult > run(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 90: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std::vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of the `FaceQuality5ptResult`.

Enumerations

Enumeration Mode

Scene of sending image.

Table 91: **Enumeration Mode Values**

Value	Description
DAY	Use DAY when the background of the image is daytime.
NIGHT	Use NIGHT when the background of the image is night.

vitis::ai::Hourglass

`Hourglass` model, input size is 256x256.

Base class for detecting poses of people.

Input is an image (cv:Mat).

Output is `HourglassResult`.

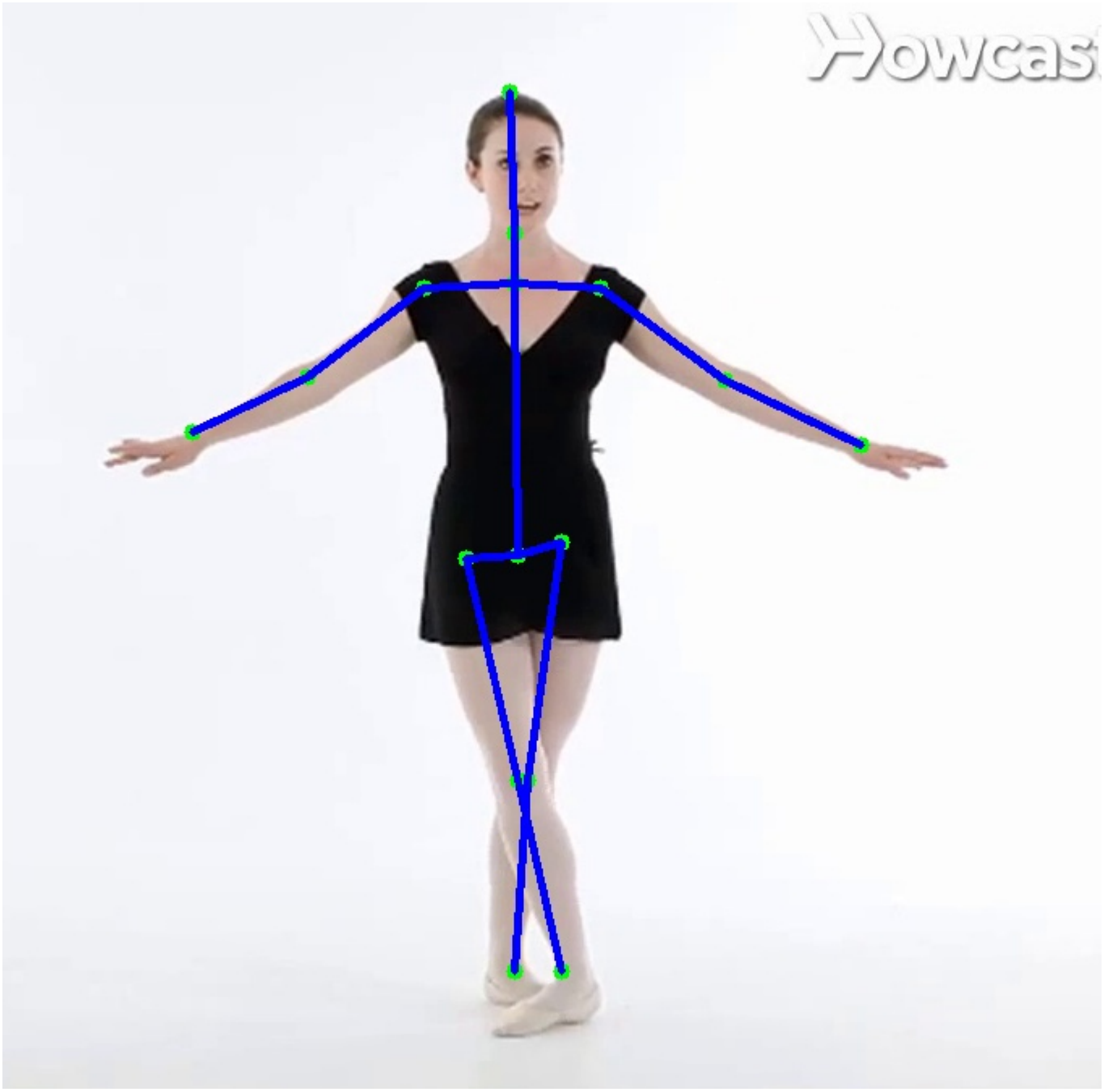
Sample code:

```
auto image = cv::imread(argv[2]);
if (image.empty()) {
  std::cerr << "cannot load " << argv[2] << std::endl;
  abort();
}
auto det = vitis::ai::Hourglass::create(argv[1]);
vector<vector<int>> limbSeq = {{0, 1},  {1, 2},   {2, 6},  {3, 6},  {3, 4},
{4, 5},
                               {6, 7},   {7, 8},  {8, 9}, {7, 12},
                               {12, 11}, {11, 10}, {7, 13}, {13, 14}, {14,
15}};
```

```
auto results = det->run(image.clone());
for (size_t i = 0; i < results.poses.size(); ++i) {
  cout<< results.poses[i].point<<endl;
  if (results.poses[i].type == 1) {
    cv::circle(image, results.poses[i].point, 5, cv::Scalar(0, 255, 0),
               -1);
  }
}
for (size_t i = 0; i < limbSeq.size(); ++i) {
  Result a = results.poses[limbSeq[i][0]];
  Result b = results.poses[limbSeq[i][1]];
  if (a.type == 1 && b.type == 1) {
    cv::line(image, a.point, b.point, cv::Scalar(255, 0, 0), 3, 4);
  }
}
```

Display of the hourglass model results:

Figure 34: **hourglass result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Hourglass` class:

Table 92: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Hourglass` >	create	const std::string & model_name bool need_preprocess
`HourglassResult`	run	const cv::Mat & image
std::vector< `HourglassResult` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `Hourglass`.

Prototype

```
std::unique_ptr< Hourglass > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 93: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Hourglass` class.

run

Function to get running result of the hourglass neuron network.

Prototype

```
HourglassResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 94: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`HourglassResult`.

run

Function to get running results of the hourglass neuron network in batch mode.

Prototype

```
std::vector< HourglassResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 95: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of batch input images (vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `HourglassResult`.

getInputWidth

Function to get InputWidth of the hourglass network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the hourglass network

getInputHeight

Function to get InputHeight of the hourglass network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the hourglass network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::MedicalDetection

Base class for detecting five objects of Endoscopy Disease Detection and `Segmentation` database (EDD2020) .

Input is an image (cv:Mat).

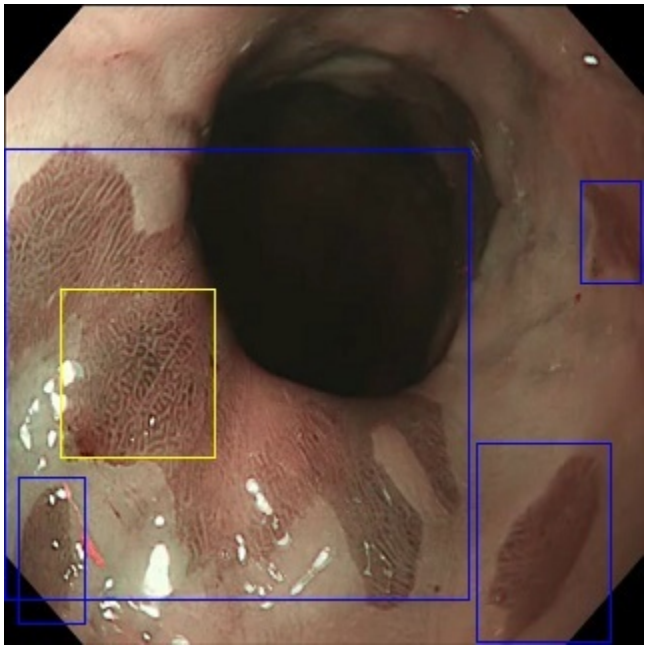
Output is a struct of detection results, named `MedicalDetectionResult`.

Sample code :

```
Mat img = cv::imread("sample_medicaldetection.jpg");
auto medicaldetection = vitis::ai::MedicalDetection::create("RefineDet-
Medical_EDD_tf",true);
auto results = medicaldetection->run(img);
for(const auto &r : results.bboxes){
   auto label = r.label;
   auto x = r.x * img.cols;
   auto y = r.y * img.rows;
   auto width = r.width * img.cols;
   auto height = r.height * img.rows;
   auto score = r.score;
   std::cout << "RESULT: " << label << "\t" << x << "\t" << y << "\t" <<
width
      << "\t" << height << "\t" << score << std::endl;
}
```

Display of the model results:

Figure 35: **detection result**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MedicalDetection` class:

Table 96: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MedicalDetection` >	create	const std::string & model_name bool need_preprocess
`vitis::ai::MedicalDetectionResult`	run	const cv::Mat & img
std::vector< `vitis::ai::MedicalDetectionResult` >	run	const std::vector< cv::Mat > & imgs
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `MedicalDetection`.

Prototype

```
std::unique_ptr< MedicalDetection > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 97: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `MedicalDetection` class.

run

Function of get result of the `MedicalDetection` neuron network.

Prototype

```
vitis::ai::MedicalDetectionResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 98: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat).

Returns

`MedicalDetectionResult`.

run

Function to get running results of the `MedicalDetection` neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::MedicalDetectionResult > run(const std::vector<
cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 99: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MedicalDetectionResult`.

getInputWidth

Function to get InputWidth of the `MedicalDetection` network (input image cols).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `MedicalDetection` network.

getInputHeight

Function to get InputHeight of the `MedicalDetection` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `MedicalDetection` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::MedicalDetectionPostProcess

Class of the `MedicalDetection` post-process. It initializes the parameters once instead of computing them every time when the program execute.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MedicalDetectionPostProcess` class:

Table 100: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MedicalDetectionPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
`MedicalDetectionResult`	medicaldetection_post_process	void
std::vector< `MedicalDetectionResult` >	medicaldetection_post_process	void

Functions

create

Create an `MedicalDetectionPostProcess` object.

Prototype

```
std::unique_ptr< MedicalDetectionPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 101: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The dpu model configuration information.

Returns

An unique printer of `MedicalDetectionPostProcess`.

medicaldetection_post_process

The post-processing function of the `MedicalDetection` network.

Prototype

```
            MedicalDetectionResult medicaldetection_post_process(unsigned
int idx)=0;
```

Returns

The struct of `MedicalDetectionResult`.

medicaldetection_post_process

The batch mode post-processing function of the `MedicalDetection` network.

Prototype

```
std::vector< MedicalDetectionResult > medicaldetection_post_process()=0;
```

Returns

The vector of struct of `MedicalDetectionResult`.

vitis::ai::MedicalSegcell

Base class for segmenting nuclei from images of cells.

Input is an image (cv:Mat).

Output is a struct of detection results, named `MedicalSegcellResult`.

Sample code :

```
Mat img = cv::imread("sample_medicalsegcell.jpg");
auto medicalsegcell =
vitis::ai::MedicalSegcell::create("medical_seg_cell_tf2",true);
auto results = medicalsegcell->run(img);
// results is structure holding cv::Mat.
// please check test samples for detail usage.
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MedicalSegcell` class:

Table 102: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MedicalSegcell` >	create	const std::string & model_name bool need_preprocess
`vitis::ai::MedicalSegcellResult`	run	const cv::Mat & img
std::vector< `vitis::ai::MedicalSegcellResult` >	run	const std::vector< cv::Mat > & imgs
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `MedicalSegcell`.

Prototype

```
std::unique_ptr< MedicalSegcell > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 103: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `MedicalSegcell` class.

run

Function of get result of the `MedicalSegcell` neuron network.

Prototype

```
vitis::ai::MedicalSegcellResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 104: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat).

Returns

`MedicalSegcellResult`.

run

Function to get running results of the `MedicalSegcell` neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::MedicalSegcellResult > run(const std::vector<
cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 105: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MedicalSegcellResult`.

getInputWidth

Function to get InputWidth of the `MedicalSegcell` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `MedicalSegcell` network.

getInputHeight

Function to get InputHeight of the `MedicalSegcell` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `MedicalSegcell` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::MedicalSegmentation

Base class for segment five objects of Endoscopy Disease Detection and `Segmentation` database (EDD2020).

Input is an image (cv:Mat).

Output is a struct of detection results, named `MedicalSegmentationResult`.

Sample code :

```
Mat img = cv::imread("sample_medicalsegmentation.jpg");
auto medicalsegmentation =
vitis::ai::MedicalSegmentation::create("FPN_Res18_Medical_segmentation",true
);
auto results = medicalsegmentation->run(img);
// results is  std::vector<cv::Mat>(5) for 5 classes.
// please check test samples for detail usage.
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MedicalSegmentation` class:

Table 106: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MedicalSegmentation` >	create	const std::string & model_name bool need_preprocess
`vitis::ai::MedicalSegmentationResult`	run	const cv::Mat & img
std::vector< `vitis::ai::MedicalSegmentationResult` >	run	const std::vector< cv::Mat > & imgs
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `MedicalSegmentation`.

Prototype

```
std::unique_ptr< MedicalSegmentation > create(const std::string
&model_name, bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 107: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `MedicalSegmentation` class.

run

Function of get result of the `MedicalSegmentation` neuron network.

Prototype

```
vitis::ai::MedicalSegmentationResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 108: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat).

Returns

`MedicalSegmentationResult`.

run

Function to get running results of the `MedicalSegmentation` neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::MedicalSegmentationResult > run(const std::vector<
cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 109: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MedicalSegmentationResult`.

getInputWidth

Function to get InputWidth of the `MedicalSegmentation` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `MedicalSegmentation` network.

getInputHeight

Function to get InputHeight of the `MedicalSegmentation` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `MedicalSegmentation` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::MedicalSegmentationPostProcess

Class of the `MedicalSegmentation` post-process. It will initialize the parameters once instead of computing them every time when the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MedicalSegmentationPostProcess` class:

Table 110: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MedicalSegmentationPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
`MedicalSegmentationResult`	medicalsegmentation_post_process	void
std::vector< `MedicalSegmentationResult` >	medicalsegmentation_post_process	void

Functions

create

Create an `MedicalSegmentationPostProcess` object.

Prototype

```
std::unique_ptr< MedicalSegmentationPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 111: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique printer of `MedicalSegmentationPostProcess`.

medicalsegmentation_post_process

The post-processing function of the `MedicalSegmentation` network.

Prototype

```
            MedicalSegmentationResult
medicalsegmentation_post_process(unsigned int idx)=0;
```

Returns

The struct of `MedicalSegmentationResult`.

medicalsegmentation_post_process

The batch mode post-processing function of the `MedicalSegmentation` network.

Prototype

```
std::vector< MedicalSegmentationResult >
medicalsegmentation_post_process()=0;
```

Returns

The vector of struct of `MedicalSegmentationResult`.

vitis::ai::MultiTask

Multitask Network Type , declaration multitask Network.

number of classes label: 0 name: "background" label: 1 name: "person" label: 2 name: "car" label: 3 name: "truck" label: 4 name: "bus" label: 5 name: "bike" label: 6 name: "sign" label: 7 name: "light" Base class for ADAS MuiltTask from an image (cv::Mat).

Input an image (cv::Mat).

Output is a struct of `MultiTaskResult` include segmentation results, detection results and vehicle towards;

Sample code:

```
auto det = vitis::ai::MultiTask::create("multi_task");
auto image = cv::imread("sample_multitask.jpg");
auto result = det->run_8UC3(image);
cv::imwrite("sample_multitask_result.jpg",result.segmentation);
```

Display of the model results:

Figure 36: **result image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MultiTask` class:

Table 112: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MultiTask` >	create	const std::string & model_name bool need_preprocess

Table 112: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`MultiTaskResult`	run_8UC1	const cv::Mat & image
std::vector< `MultiTaskResult` >	run_8UC1	const std::vector< cv::Mat > & images
`MultiTaskResult`	run_8UC3	const cv::Mat & image
std::vector< `MultiTaskResult` >	run_8UC3	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class Multitask.

Prototype

```
std::unique_ptr< MultiTask > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 113: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of Multitask class.

getInputWidth

Function to get InputWidth of the multitask network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHeight of the multitask network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the multitask network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run_8UC1

Function of get running result from the `MultiTask` network.

***Note*:** The type is CV_8UC1 of the `MultiTaskResult.segmentation`.

Prototype

```
MultiTaskResult run_8UC1(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 114: **run_8UC1 Arguments**

Type	Name	Description
const cv::Mat &	image	Input image

Returns

The struct of `MultiTaskResult`

run_8UC1

Function to get running results of the `MultiTask` neuron network in batch mode.

***Note*:** The type is CV_8UC1 of the `MultiTaskResult.segmentation`.

Prototype

```
std::vector< MultiTaskResult > run_8UC1(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 115: **run_8UC1 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MultiTaskResult`.

run_8UC3

Function to get running result from the `MultiTask` network.

***Note*:** The type is CV_8UC3 of the `MultiTaskResult.segmentation`.

Prototype

```
MultiTaskResult run_8UC3(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 116: **run_8UC3 Arguments**

Type	Name	Description
const cv::Mat &	image	Input image;

Returns

The struct of `MultiTaskResult`

run_8UC3

Function to get running results of the `MultiTask` neuron network in batch mode.

***Note*:** The type is CV_8UC3 of the `MultiTaskResult.segmentation`.

Prototype

```
std::vector< MultiTaskResult > run_8UC3(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 117: **run_8UC3 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MultiTaskResult`.

vitis::ai::MultiTask8UC1

Base class for ADAS MuiltTask8UC1 from an image (cv::Mat).

Input is an image (cv::Mat).

Output is struct `MultiTaskResult` include segmentation results, detection results and vehicle towards; The result cv::Mat type is CV_8UC1

Sample code:

```
auto det = vitis::ai::MultiTask8UC1::create(vitis::ai::MULTITASK);
auto image = cv::imread("sample_multitask.jpg");
auto result = det->run(image);
cv::imwrite("res.jpg",result.segmentation);
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MultiTask8UC1` class:

Table 118: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MultiTask8UC1` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`MultiTaskResult`	run	const cv::Mat & image
std::vector< `MultiTaskResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `MultiTask8UC1`.

Prototype

```
std::unique_ptr< MultiTask8UC1 > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 119: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `MultiTask8UC1` class.

getInputWidth

Function to get InputWidth of the multitask network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHeight of the multitask network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputHeight of the multitask network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be differnt. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function of get running result from the `MultiTask` network.

***Note*:** The type is CV_8UC1 of the `MultiTaskResult.segmentation`.

Prototype

```
MultiTaskResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 120: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input image

Returns

The struct of `MultiTaskResult`

run

Function to get running results of the `MultiTask` neuron network in batch mode.

***Note*:** The type is CV_8UC1 of the `MultiTaskResult.segmentation`.

Prototype

```
std::vector< MultiTaskResult > run(const std::vector< cv::Mat > &images);
```

Parameters

The following table lists the `run` function arguments.

Table 121: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MultiTaskResult`.

vitis::ai::MultiTask8UC3

Base class for ADAS MuiltTask8UC3 from an image (cv::Mat).

Input is an image (cv::Mat).

Output is struct `MultiTaskResult` include segmentation results, detection results and vehicle orientation; The result cv::Mat type is CV_8UC3

Sample code:

```
auto det = vitis::ai::MultiTask8UC3::create(vitis::ai::MULITASK);
auto image = cv::imread("sample_multitask.jpg");
auto result = det->run(image);
cv::imwrite("res.jpg",result.segmentation);
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MultiTask8UC3` class:

Table 122: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MultiTask8UC3` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`MultiTaskResult`	run	const cv::Mat & image
std::vector< `MultiTaskResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `MultiTask8UC3`.

Prototype

```
std::unique_ptr< MultiTask8UC3 > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 123: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `MultiTask8UC3` class.

getInputWidth

Function to get InputWidth of the multitask network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHeight of the multitask network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputHeight of the multitask network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

Note: Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function of get running result from the `MultiTask` network.

Note: The type is CV_8UC3 of the `MultiTaskResult.segmentation`.

Prototype

```
MultiTaskResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 124: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input image

Returns

The struct of `MultiTaskResult`

run

Function to get running results of the `MultiTask` neuron network in batch mode.

Note: The type is CV_8UC3 of the `MultiTaskResult.segmentation`.

Prototype

```
std::vector< MultiTaskResult > run(const std::vector< cv::Mat > &images);
```

Parameters

The following table lists the `run` function arguments.

Table 125: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `MultiTaskResult`.

vitis::ai::MultiTaskPostProcess

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::MultiTaskPostProcess` class:

Table 126: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< MultiTaskPostProcess >	create	const std::vector< std::vector< vitis::ai::library::InputTensor >> & input_tensors const std::vector< std::vector< vitis::ai::library::OutputTensor >> & output_tensors const vitis::ai::proto::DpuModelParam & config
std::vector< `MultiTaskResult` >	post_process_seg	void
std::vector< `MultiTaskResult` >	post_process_seg_visualization	void

Functions

create

Factory function to get an instance of derived classes of MultiTaskPostProcess.

Prototype

```
std::unique_ptr< MultiTaskPostProcess > create(const std::vector<
std::vector< vitis::ai::library::InputTensor >> &input_tensors, const
std::vector< std::vector< vitis::ai::library::OutputTensor >>
&output_tensors, const vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 127: **create Arguments**

Type	Name	Description
const std::vector< std::vector< vitis::ai::library::InputTensor >> &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[kernel_index][input_tensor_index].
const std::vector< std::vector< vitis::ai::library::OutputTensor >> &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[kernel_index][output_index].
const vitis::ai::proto::DpuModelParam &	config	The dpu model configuration information.

Returns

The struct of `MultiTaskResult`.

post_process_seg

The post-processing function of the multitask which stored the original segmentation classes.

Prototype

```
std::vector< MultiTaskResult > post_process_seg()=0;
```

Returns

The struct of `SegmentationResult`.

post_process_seg_visualization

The post-processing function of the multitask which return a result include segmentation image mapped to color.

Prototype

```
std::vector< MultiTaskResult > post_process_seg_visualization()=0;
```

Returns

The struct of `SegmentationResult`.

vitis::ai::OpenPose

openpose model, input size is 368x368.

Base class for detecting poses of people.

Input is an image (cv:Mat).

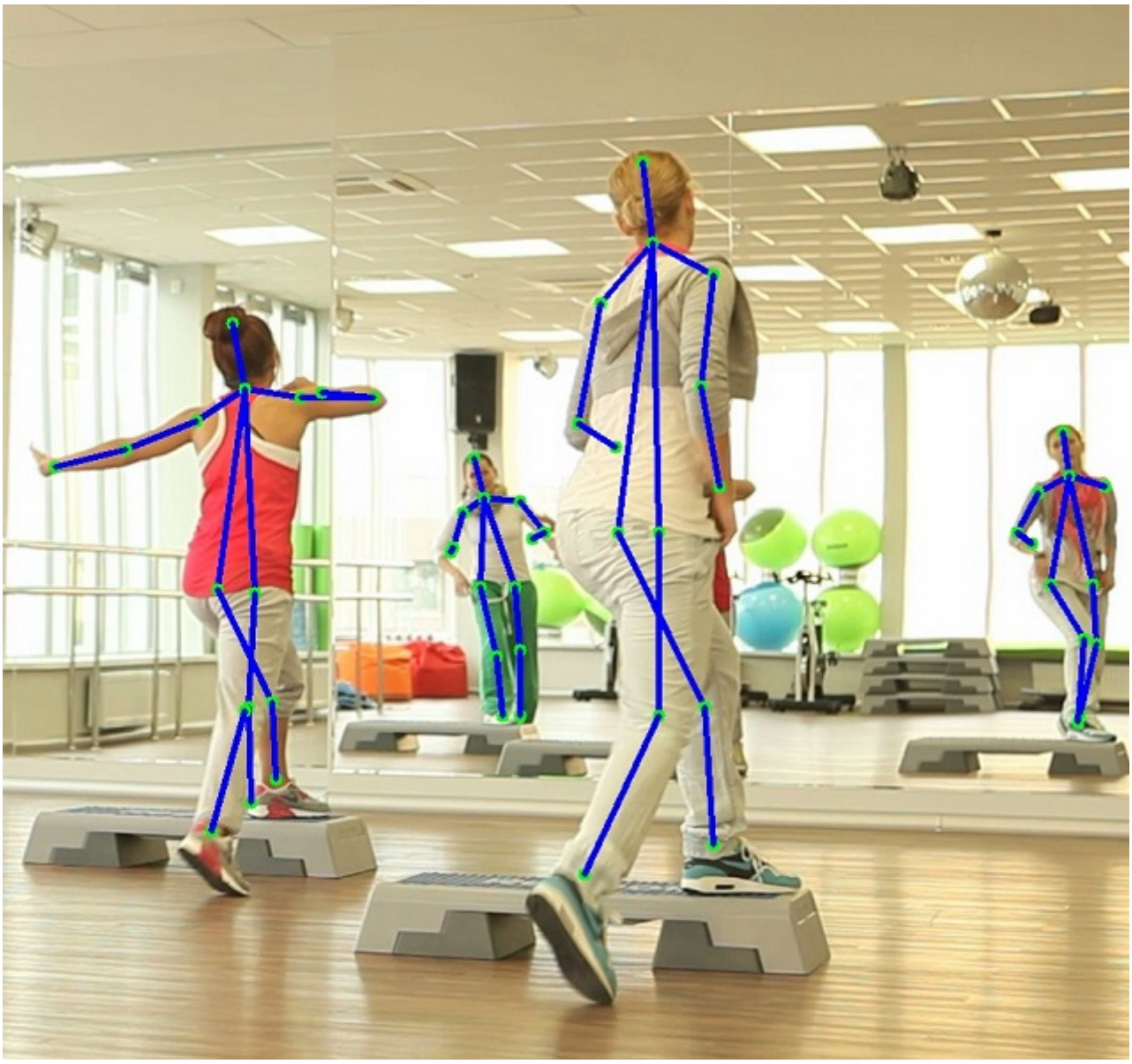
Output is a `OpenPoseResult`.

Sample code :

```
 auto image = cv::imread("sample_openpose.jpg");
 if (image.empty()) {
   std::cerr << "cannot load image" << std::endl;
   abort();
 }
 auto det = vitis::ai::OpenPose::create("openpose_pruned_0_3");
 int width = det->getInputWidth();
 int height = det->getInputHeight();
 vector<vector<int>> limbSeq = {{0,1}, {1,2}, {2,3}, {3,4}, {1,5}, {5,6},
{6,7}, {1,8}, \ {8,9}, {9,10}, {1,11}, {11,12}, {12,13}}; float scale_x =
float(image.cols) / float(width); float scale_y = float(image.rows) /
float(height); auto results = det->run(image); for(size_t k = 1; k <
results.poses.size(); ++k){ for(size_t i = 0; i < results.poses[k].size();
++i){ if(results.poses[k][i].type == 1){ results.poses[k][i].point.x *=
scale_x; results.poses[k][i].point.y *= scale_y; cv::circle(image,
results.poses[k][i].point, 5, cv::Scalar(0, 255, 0), -1);
        }
    }
    for(size_t i = 0; i < limbSeq.size(); ++i){
        Result a = results.poses[k][limbSeq[i][0]];
        Result b = results.poses[k][limbSeq[i][1]];
        if(a.type == 1 && b.type == 1){
            cv::line(image, a.point, b.point, cv::Scalar(255, 0, 0), 3, 4);
        }
    }
 }
```

Display of the openpose model results:

Figure 37: **openpose result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::OpenPose` class:

Table 128: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `OpenPose` >	create	const std::string & model_name bool need_preprocess

Table 128: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
`OpenPoseResult`	run	const cv::Mat & image
std::vector< `OpenPoseResult` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `OpenPose`.

Prototype

```
std::unique_ptr< OpenPose > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 129: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `OpenPose` class.

run

Function to get running result of the openpose neuron network.

Prototype

```
OpenPoseResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 130: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`OpenPoseResult`.

run

Function to get running results of the openpose neuron network in batch mode.

Prototype

```
std::vector< OpenPoseResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 131: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of batch input images (vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `OpenPoseResult`.

getInputWidth

Function to get InputWidth of the openpose network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the openpose network

getInputHeight

Function to get InputHeight of the openpose network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the openpose network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::PlateDetect

Base class for detecting the position of plate in a vehicle image (cv::Mat).

Input is a vehicle image (cv::Mat).

Output is position and score of plate in the input image.

Sample code:

```
cv::Mat image = cv::imread("car.jpg");
auto network = vitis::ai::PlateDetect::create(true);
auto r = network->run(image);
auto score = r.box.score.
auto x = r.box.x * image.cols;
auto y = r.box.y * image.rows;
auto witdh = r.box.width * image.cols;
auto height = r.box.height * image.rows;
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::PlateDetect` class:

Table 132: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `PlateDetect` >	create	bool need_mean_scale_process
std::unique_ptr< `PlateDetect` >	create	void
	PlateDetect	void
	PlateDetect	void
`PlateDetect` &	operator=	void
	~PlateDetect	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`PlateDetectResult`	run	const cv::Mat & image
std::vector< `PlateDetectResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class platedetect.

Prototype

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

Parameters

The following table lists the `create` function arguments.

Table 133: **create Arguments**

Type	Name	Description
bool	need_mean_scale_process	Normalize with mean/scale or not, true by default.

Returns

An instance of the `PlateDetect` class.

create

Prototype

```
std::unique_ptr< PlateDetect > create(const std::string &model_name,
xir::Attrs *attrs, bool need_mean_scale_process=true);
```

PlateDetect

Prototype

```
PlateDetect();
```

PlateDetect

Prototype

```
PlateDetect(const PlateDetect &)=delete;
```

operator=

Prototype

```
PlateDetect & operator=(const PlateDetect &)=delete;
```

~PlateDetect

Prototype

```
~PlateDetect();
```

getInputWidth

Function to get InputWidth of the platedetect network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the platedetect network.

getInputHeight

Function to get InputHeight of the platedetect network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the platedetect network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function of get running result of the platedetect network.

Prototype

```
            PlateDetectResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 134: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat) of detected counterpart and resized as inputwidth an outputheight.

Returns

Plate position and plate score.

run

Function to get running results of the platedetect neuron network in batch mode.

Prototype

```
std::vector< PlateDetectResult > run(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 135: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of PLateDetectResult.

vitis::ai::PlateNum

Base class for recognizing plate from an image (cv::Mat).

Input is a plate image (cv::Mat).

Output is the number and color of plate in the input image.

sample code:

***Note*:** Only China plate Only edge platform supported

```
cv::Mat image = cv::imread("plate.jpg");
auto network = vitis::ai::PlateNum::create(true);
auto r = network->run(image);
auto plate_number = r.plate_number;
auto plate_color = r.plate_color;
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::PlateNum` class:

Table 136: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `PlateNum` >	create	bool need_mean_scale_process
std::unique_ptr< `PlateNum` >	create	void
	PlateNum	void
	PlateNum	void

Table 136: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
`PlateNum` &	operator=	void
	~PlateNum	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`PlateNumResult`	run	const cv::Mat & img
std::vector< `PlateNumResult` >	run	const std::vector< cv::Mat > & imgs

Functions

create

Factory function to get an instance of derived classes of class `PlateNum`.

Prototype

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

Parameters

The following table lists the `create` function arguments.

Table 137: **create Arguments**

Type	Name	Description
bool	need_mean_scale_process	normalize with mean/scale or not, true by default.

Returns

An instance of `PlateNum` class.

create

Prototype

```
std::unique_ptr< PlateNum > create(const std::string &model_name,
xir::Attrs *attrs, bool need_mean_scale_process=true);
```

PlateNum

Prototype

```
PlateNum();
```

PlateNum

Prototype

```
PlateNum(const PlateNum &)=delete;
```

operator=

Prototype

```
PlateNum & operator=(const PlateNum &)=delete;
```

~PlateNum

Prototype

```
~PlateNum();
```

getInputWidth

Function to get InputWidth of the platenum network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the platenum network.

getInputHeight

Function to get InputHeight of the platenum network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the platenum network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function of get running result of platenum network.

Prototype

```
            PlateNumResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 138: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat) and resized as InputWidth and IntputHeight.

Returns

The plate number and plate color.

run

Function to get running results of the platenum neuron network in batch mode.

Prototype

```
std::vector< PlateNumResult > run(const std::vector< cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 139: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of PLateNumResult.

vitis::ai::PointPillars

Base class for pointpillars .

Input is points data.

Output is a struct of detection results, named `PointPillarsResult`.

Sample code :

```
  ...
  auto net =
vitis::ai::PointPillars::create("pointpillars_kitti_12000_0_pt",
"pointpillars_kitti_12000_1_pt", true);
  V1F PointCloud ;
  int len = getfloatfilelen( lidar_path);
  PointCloud.resize( len );
  myreadfile(PointCloud.data(), len, lidar_path);
  auto res = net->run(PointCloud);
  ...
please see the test sample for detail.
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::PointPillars` class:

Table 140: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `PointPillars` >	create	const std::string & model_name const std::string & model_name1
`vitis::ai::PointPillarsResult`	run	const V1F & v1f
size_t	get_input_batch	void
void	do_pointpillar_display	`PointPillarsResult` & res int flag `DISPLAY_PARAM` & dispp cv::Mat & rgb_map cv::Mat & bev_map int width int height `ANNORET` & annoret

Functions

create

Factory function to get an instance of derived classes of class `PointPillars`.

Prototype

```
std::unique_ptr< PointPillars > create(const std::string &model_name, const
std::string &model_name1);
```

Parameters

The following table lists the `create` function arguments.

Table 141: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name for PointNet
const std::string &	model_name1	Model name for RPN

Returns

An instance of `PointPillars` class.

run

Function of get result of the `PointPillars` neuron network.

Prototype

```
vitis::ai::PointPillarsResult run(const V1F &v1f)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 142: **run Arguments**

Type	Name	Description
const V1F &	v1f	point data in vector<float>

Returns

`PointPillarsResult`.

get_input_batch

Function to get input batch of the `PointPillars` network.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Input batch of the `PointPillars` network.

do_pointpillar_display

Function to produce the visible result from `PointPillarsResult` after calling `run()`. This is a helper function which can be ignored if you wants to process the `PointPillarsResult` using another method.

Prototype

```
void do_pointpillar_display(PointPillarsResult &res, int flag,
DISPLAY_PARAM &dispp, cv::Mat &rgb_map, cv::Mat &bev_map, int width, int
height, ANNORET &annoret)=0;
```

Parameters

The following table lists the `do_pointpillar_display` function arguments.

Table 143: **do_pointpillar_display Arguments**

Type	Name	Description
`PointPillarsResult` &	res	[input] `PointPillarsResult` from `run()`.
int	flag	[input] which visible result to produce. can be assigned to below values: E_BEV : only produce BEV picture E_RGB : only produce RGB picture E_BEV|E_RGB : produce both pictures
`DISPLAY_PARAM` &	dispp	[input] display parameter for the Points data. Refer to the readme in the overview for more detail.
cv::Mat &	rgb_map	[input|output] : original rgb picture for drawing detect result. It can be blank (cv::Mat{}), if only BEV is required
cv::Mat &	bev_map	[input|output] original bev picture for drawing detect result. It can be blank (cv::Mat{}), if only RGB required
int	width	[input] original rgb picture width.
int	height	[input] original rgb picture height.
`ANNORET` &	annoret	[output] return the annoret variable for accuracy calculation.

vitis::ai::PoseDetect

Base class for detecting a pose from an input image (cv::Mat).

Input an image (cv::Mat).

***Note*:** Support detect a single pose.

Output is a struct of `PoseDetectResult`, include 14 point.

Sample code:

```
auto det = vitis::ai::PoseDetect::create("sp_net");
auto image = cv::imread("sample_posedetect.jpg");
auto results = det->run(image);
for(auto result: results.pose14pt) {
    std::cout << result << std::endl;
}
```

Display of the posedetect model results:

Figure 38: **pose detect image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::PoseDetect` class:

Table 144: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `PoseDetect` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`PoseDetectResult`	run	const cv::Mat & image
std::vector< `PoseDetectResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `PoseDetect`.

Prototype

```
std::unique_ptr< PoseDetect > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 145: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name .
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `PoseDetect` class.

getInputWidth

Function to get InputWidth of the `PoseDetect` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `PoseDetect` network.

getInputHeight

Function to get InputHeight of the `PoseDetect` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `PoseDetect` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function to get running results of the posedetect neuron network.

Prototype

```
            PoseDetectResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 146: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`PoseDetectResult`.

run

Function to get running results of the posedetect neuron network in batch mode.

Prototype

```
std::vector< PoseDetectResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 147: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `PoseDetectResult`.

vitis::ai::RefineDet

Base class for detecting pedestrians in the input image (cv::Mat).

Input is an image (cv::Mat).

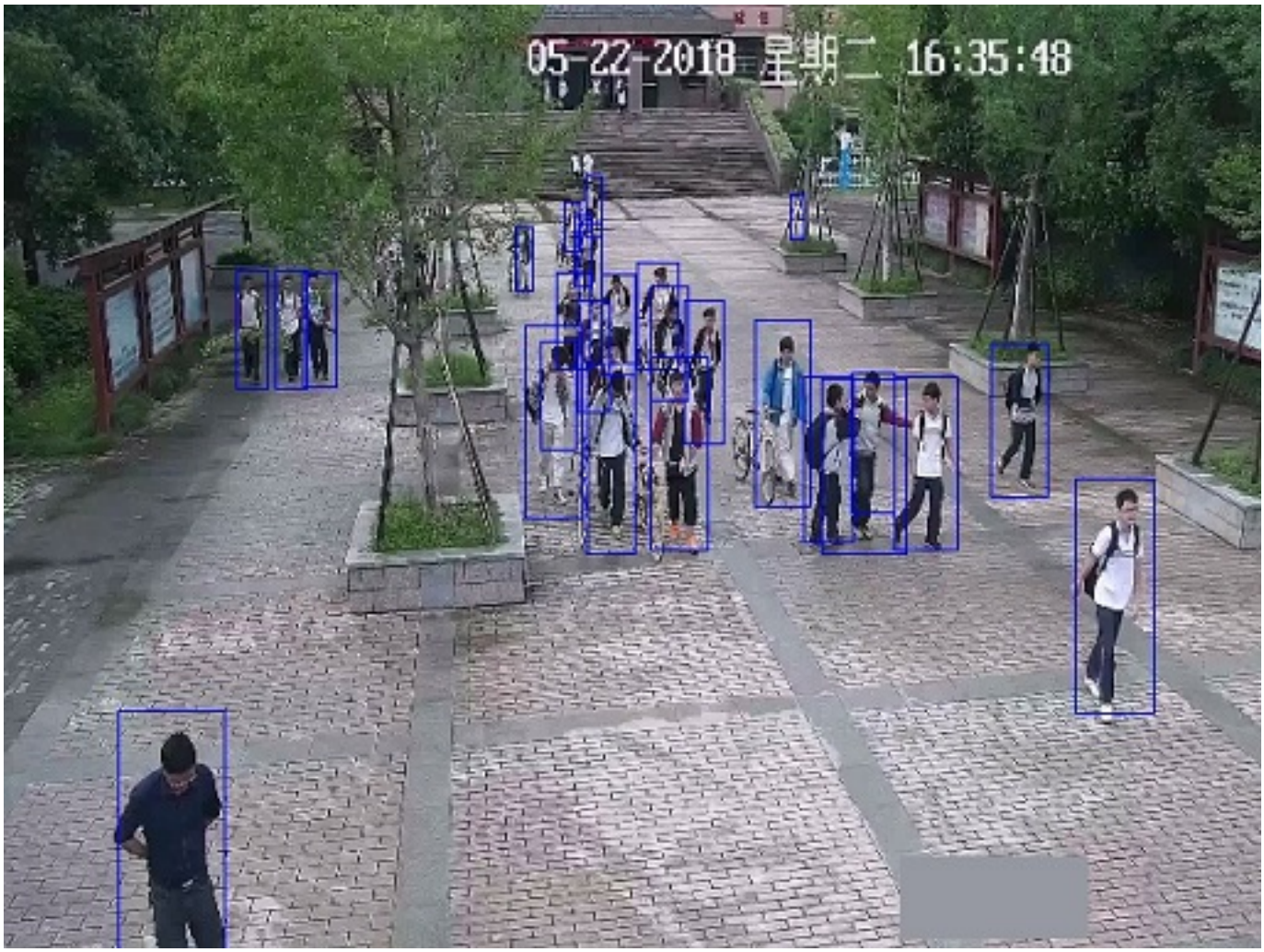
Output is the position and score of the pedestrians in the input image.

Sample code:

```
auto det = vitis::ai::RefineDet::create("refinedet_pruned_0_8");
auto image = cv::imread("sample_refinedet.jpg");
cout << "load image" << endl;
if (image.empty()) {
  cerr << "cannot load " << argv[1] << endl;
  abort();
}

auto results = det->run(image);

auto img = image.clone();
for (auto &box : results.bboxes) {
    float x = box.x * (img.cols);
    float y = box.y * (img.rows);
    int xmin = x;
    int ymin = y;
    int xmax = x + (box.width) * (img.cols);
    int ymax = y + (box.height) * (img.rows);
    float score = box.score;
    xmin = std::min(std::max(xmin, 0), img.cols);
    xmax = std::min(std::max(xmax, 0), img.cols);
    ymin = std::min(std::max(ymin, 0), img.rows);
    ymax = std::min(std::max(ymax, 0), img.rows);

    cv::rectangle(img, cv::Point(xmin, ymin), cv::Point(xmax, ymax),
                  cv::Scalar(0, 255, 0), 1, 1, 0);
}
auto out = "sample_refinedet_result.jpg";
LOG(INFO) << "write result to " << out;
cv::imwrite(out, img);
```

Display of the model results:

Figure 39: **result image**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RefineDet` class:

Table 148: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RefineDet` >	create	const std::string & model_name bool need_preprocess
`RefineDetResult`	run	const cv::Mat & image
std::vector< `RefineDetResult` >	run	const std::vector< cv::Mat > & images

Table 148: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `RefineDet`.

Prototype

```
std::unique_ptr< RefineDet > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 149: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `RefineDet` class.

run

Function to get running result of the `RefineDet` neuron network.

Prototype

```
RefineDetResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 150: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

A Struct of `RefineDetResult`.

run

Function to get running result of the `RefineDet` neuron network in batch mode.

Prototype

```
std::vector< RefineDetResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 151: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (vector<cv::Mat>).

Returns

vector of Struct of `RefineDetResult`.

getInputWidth

Function to get InputWidth of the refinedet network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the refinedet network

getInputHeight

Function to get InputHeight of the refinedet network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the refinedet network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::RefineDetPostProcess

Class of the refinedet post-process. It initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RefineDetPostProcess` class:

Table 152: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RefineDetPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config

Table 152: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
`RefineDetResult`	refine_det_post_process	void
std::vector< `RefineDetResult` >	refine_det_post_process	void

Functions

create

Create an `RefineDetPostProcess` object.

Prototype

```
std::unique_ptr< RefineDetPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 153: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique printer of `RefineDetPostProcess`.

refine_det_post_process

Run refinedet post-process.

Prototype

```
RefineDetResult refine_det_post_process(unsigned int idx)=0;
```

Returns

The struct of `RefineDetResult`.

refine_det_post_process

Run batch mode of refinedet post-process.

Prototype

```
std::vector< RefineDetResult > refine_det_post_process()=0;
```

Returns

The vector of struct of `RefineDetResult`.

vitis::ai::Reid

Base class for detecting roadline from an image (cv::Mat).

Input is an image (cv::Mat).

Output road line type and points maked road line.

Sample code:

***Note*:** The input image size is 640x480

```
if(argc < 3){
    cerr<<"need two images"<<endl;
    return -1;
}
Mat imgx = imread(argv[1]);
if(imgx.empty()){
    cerr<<"can't load image! "<<argv[1]<<endl;
    return -1;
}
Mat imgy = imread(argv[2]);
if(imgy.empty()){
    cerr<<"can't load image! "<<argv[2]<<endl;
    return -1;
}
```

```
auto det = vitis::ai::Reid::create("reid");
Mat featx = det->run(imgx).feat;
Mat featy = det->run(imgy).feat;
double dismat= cosine_distance(featx, featy);
printf("dismat : %.3lf \n", dismat);
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Reid` class:

Table 154: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Reid` >	create	const std::string & model_name bool need_preprocess
`ReidResult`	run	const cv::Mat & image
std::vector< `ReidResult` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `Reid`.

Prototype

```
std::unique_ptr< Reid > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 155: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name

Table 155: **create Arguments** *(cont'd)*

Type	Name	Description
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Reid` class.

run

Function to get running result of the ReID neuron network.

Prototype

```
ReidResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 156: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`ReidResult`.

run

Function to get running result of the ReID neuron network in batch mode.

Prototype

```
std::vector< ReidResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 157: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (vector<cv::Mat>).

Returns

vector of `ReidResult`.

getInputWidth

Function to get InputWidth of the ReID network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the ReID network

getInputHeight

Function to get InputHeight of the ReID network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the ReID network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::RetinaFace

Base class for detecting the position,score and landmark of faces in the input image (cv::Mat).

Input is an image (cv::Mat).

Output is a vector of position and score for faces in the input image.

Sample code:

```
auto image = cv::imread("sample_retinaface.jpg");
auto network = vitis::ai::RetinaFace::create(
               "retinaface",
               true);
auto result = network->run(image);
for (auto i = 0u; i < result.bboxes.size(); ++i) {
   auto score = result.bboxes[i].score;
   auto x = result.bboxes[i].x * image.cols;
   auto y = result.bboxes[i].y * image.rows;
   auto width = result.bboxes[i].width * image.cols;
   auto height = result.bboxes[i].height * image.rows;
   auto landmark = results.landmarks[i];
   for (auto j = 0; j < 5; ++j) {
     auto px = landmark[j].first * image.cols;
     auto py = landmark[j].second * image.rows;
   }
}
```

Display of the model results:

Figure 40: **result image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RetinaFace` class:

Table 158: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RetinaFace` >	create	const std::string & model_name bool need_preprocess
std::unique_ptr< `RetinaFace` >	create	void
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`RetinaFaceResult`	run	const cv::Mat & img

Table 158: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
std::vector< `RetinaFaceResult` >	run	const std::vector< cv::Mat > & imgs

Functions

create

Factory function to get an instance of derived classes of class `RetinaFace`.

Prototype

```
std::unique_ptr< RetinaFace > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 159: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `RetinaFace` class.

create

Prototype

```
std::unique_ptr< RetinaFace > create(const std::string &model_name,
xir::Attrs *attrs, bool need_preprocess=true);
```

getInputWidth

Function to get InputWidth of the retinaface network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the retinaface network

getInputHeight

Function to get InputHeight of the retinaface network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the retinaface network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function to get running result of the retinaface network.

Prototype

```
RetinaFaceResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 160: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input Data ,input image (cv::Mat) need to be resized to InputWidth and InputHeight required by the network.

Returns

The detection result of the face detect network , filtered by score >= det_threshold

run

Function to get running results of the retinaface neuron network in batch mode.

Prototype

```
std::vector< RetinaFaceResult > run(const std::vector< cv::Mat > &imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 161: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch. The input images need to be resized to InputWidth and InputHeight required by the network.

Returns

The vector of `RetinaFaceResult`.

vitis::ai::RetinaFacePostProcess

Class of the retinaface post-process. It initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RetinaFacePostProcess` class:

Table 162: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RetinaFacePostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
std::vector< `RetinaFaceResult` >	retinaface_post_process	void

Functions

create

Create a `RetinaFacePostProcess` object.

Prototype

```
std::unique_ptr< RetinaFacePostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 163: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique pointer of `RetinaFacePostProcess`.

retinaface_post_process

The batch mode post-processing function of the retinaface network.

Prototype

```
std::vector< RetinaFaceResult > retinaface_post_process(size_t
batch_size)=0;
```

Returns

The vector of struct of `RetinaFaceResult`.

vitis::ai::RoadLine

Base class for detecting lanedetect from an image (cv::Mat).

Input is an image (cv::Mat).

Output road line type and points marked road line.

Sample code:

***Note*:** The input image size is 640x480

```
auto det = vitis::ai::RoadLine::create("vpgnet_pruned_0_99");
auto image = cv::imread("sample_lanedetect.jpg");
//    Mat image;
//    resize(img, image, Size(640, 480));
if (image.empty()) {
  cerr << "cannot load " << argv[1] << endl;
  abort();
}

vector<int> color1 = {0, 255, 0, 0, 100, 255};
vector<int> color2 = {0, 0, 255, 0, 100, 255};
vector<int> color3 = {0, 0, 0, 255, 100, 255};

RoadLineResult results = det->run(image);
for (auto &line : results.lines) {
  vector<Point> points_poly = line.points_cluster;
  // for (auto &p : points_poly) {
  //  std::cout << p.x << " " << (int)p.y << std::endl;
  //}
  int type = line.type < 5 ? line.type : 5;
  if (type == 2 && points_poly[0].x < image.rows * 0.5)
    continue;
  cv::polylines(image, points_poly, false,
                Scalar(color1[type], color2[type], color3[type]), 3, CV_AA,
                0);
}
```

Display of the model results:

Figure 41: **result image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RoadLine` class:

Table 164: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RoadLine` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`RoadLineResult`	run	const cv::Mat & image

Table 164: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
std::vector< `RoadLineResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `RoadLine`.

Prototype

```
std::unique_ptr< RoadLine > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 165: **create Arguments**

Type	Name	Description
const std::string &	model_name	String of model name
bool	need_preprocess	normalize with mean/scale or not, default value is true.

Returns

An instance of `RoadLine` class.

getInputWidth

Function to get InputWidth of the lanedetect network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the lanedetect network.

getInputHeight

Function to get InputHeight of the lanedetect network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the lanedetect network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function to get running result of the `RoadLine` network.

Prototype

```
RoadLineResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 166: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data , input image (cv::Mat) need to resized as 640x480.

Returns

The struct of `RoadLineResult`

run

Function to get running result of the `RoadLine` network in batch mode.

Prototype

```
std::vector< RoadLineResult > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 167: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `RoadLineResult`

vitis::ai::RoadLinePostProcess

Class of the roadline post-process. It will initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::RoadLinePostProcess` class:

Table 168: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `RoadLinePostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
`RoadLineResult`	road_line_post_process	void

Table 168: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
std::vector< `RoadLineResult` >	road_line_post_process	void

Functions

create

Create an `RoadLinePostProcess` object.

Prototype

```
std::unique_ptr< RoadLinePostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 169: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique printer of `RoadLinePostProcess`.

road_line_post_process

Run roadline post-process.

Prototype

```
            RoadLineResult road_line_post_process(int inWidth, int
inHeight, unsigned int idx)=0;
```

Returns

The struct of `RoadLineResult`.

road_line_post_process

Run roadline post-process in batch mode.

Prototype

```
std::vector< RoadLineResult > road_line_post_process(const std::vector< int
> &inWidth, const std::vector< int > &inHeight)=0;
```

Returns

The vector of struct of `RoadLineResult`.

vitis::ai::Segmentation

Base class for `Segmentation`.

Declaration `Segmentation` Network number of segmentation classes label 0 name: "unlabeled" label 1 name: "ego vehicle" label 2 name: "rectification border" label 3 name: "out of roi" label 4 name: "static" label 5 name: "dynamic" label 6 name: "ground" label 7 name: "road" label 8 name: "sidewalk" label 9 name: "parking" label 10 name: "rail track" label 11 name: "building" label 12 name: "wall" label 13 name: "fence" label 14 name: "guard rail" label 15 name: "bridge" label 16 name: "tunnel" label 17 name: "pole" Input is an image (cv:Mat).

Output is result of running the `Segmentation` network.

Sample code :

```
   auto det =vitis::ai::Segmentation::create("fpn", true);

   auto img= cv::imread("sample_segmentation.jpg");
   int width = det->getInputWidth();
   int height = det->getInputHeight();
   cv::Mat image;
   cv::resize(img, image, cv::Size(width, height), 0, 0,
              cv::INTER_LINEAR);
   auto result = det->run_8UC1(image);
   for (auto y = 0; y < result.segmentation.rows; y++) {
```

```
      for (auto x = 0; x < result.segmentation.cols; x++) {
            result.segmentation.at<uchar>(y,x) *= 10;
        }
    }
    cv::imwrite("segres.jpg",result.segmentation);

    auto resultshow = det->run_8UC3(image);
    resize(resultshow.segmentation, resultshow.segmentation,
cv::Size(resultshow.cols * 2, resultshow.rows * 2));
    cv::imwrite("sample_segmentation_result.jpg",resultshow.segmentation);
```

Figure 42: **segmentation Visualization Result Image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Segmentation` class:

Table 170: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Segmentation` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`SegmentationResult`	run_8UC1	const cv::Mat & image
std::vector< `SegmentationResult` >	run_8UC1	const std::vector< cv::Mat > & images
`SegmentationResult`	run_8UC3	const cv::Mat & image

Table 170: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
std::vector< `SegmentationResult` >	run_8UC3	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Segmentation`.

Prototype

```
std::unique_ptr< Segmentation > create(const std::string &model_name, bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 171: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Segmentation` class.

getInputWidth

Function to get InputWidth of the segmentation network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the segmentation network.

getInputHeight

Function to get InputHeight of the segmentation network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run_8UC1

Function to get running result of the segmentation network.

***Note*:** The type of CV_8UC1 of the segmentation result.

Prototype

```
SegmentationResult run_8UC1(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 172: **run_8UC1 Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

A result that includes segmentation output data.

run_8UC1

Function to get running results of the segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC1 of the segmentation result.

Prototype

```
std::vector< SegmentationResult > run_8UC1(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run_8UC1` function arguments.

Table 173: **run_8UC1 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `SegmentationResult`.

run_8UC3

Function to get running result of the segmentation network.

***Note*:** The type of CV_8UC3 of the segmentation result.

Prototype

```
SegmentationResult run_8UC3(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 174: **run_8UC3 Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

A result that include segmentation image and shape;.

run_8UC3

Function to get running results of the segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC3 of the segmentation result.

Prototype

```
std::vector< SegmentationResult > run_8UC3(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run_8UC3` function arguments.

Table 175: **run_8UC3 Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `SegmentationResult`.

vitis::ai::Segmentation3D

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Segmentation3D` class:

Table 176: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< Segmentation3D >	create	bool need_mean_scale_process
int	getInputWidth	void
int	getInputHeight	void

Table 176: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
size_t	get_input_batch	void
`Segmentation3DResult`	run	std::vector< std::vector< float >> & array
std::vector< `Segmentation3DResult` >	run	std::vector< std::vector< std::vector< float >>> & arrays

Functions

create

Factory function to get an instance of derived classes of class 3Dsegmentation.

Prototype

```
std::unique_ptr< Segmentation3D > create(const std::string &model_name,
bool need_mean_scale_process=false);
```

Parameters

The following table lists the `create` function arguments.

Table 177: **create Arguments**

Type	Name	Description
bool	need_mean_scale_process	Normalize with mean/scale or not, true by default.

Returns

An instance of the 3Dsegmentation class.

getInputWidth

Function to get InputWidth of the 3D segmentation network.

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the 3D segmentation network.

getInputHeight

Function to get InputHeight of the 3D segmentation network.

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the 3D segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

run

Function of get running result of the 3D segmentation network.

Prototype

```
            Segmentation3DResult run(std::vector< std::vector< float >>
&array)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 178: **run Arguments**

Type	Name	Description
std::vector< std::vector< float >> &	array	Input data of 3D object data in vector<float> mode.

Returns

`Segmentation3DResult`.

run

Function of get running result of the 3D segmentation network in batch mode.

Prototype

```
std::vector< Segmentation3DResult > run(std::vector< std::vector<
std::vector< float >>> &arrays)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 179: **run Arguments**

Type	Name	Description
std::vector< std::vector< std::vector< float >>> &	arrays	A vector of Input data of 3D object data in vector<float> mode.

Returns

A vector of `Segmentation3DResult`.

vitis::ai::Segmentation8UC1

The Class of `Segmentation8UC1`. This class run function `run(const cv::Mat& image)` return a cv::Mat with the type is cv_8UC1. Sample code :

```
   auto det =
vitis::ai::Segmentation8UC1::create(vitis::ai::SEGMENTATION_FPN);
  auto img = cv::imread("sample_segmentation.jpg");
   int width = det->getInputWidth();
   int height = det->getInputHeight();
   cv::Mat image;
   cv::resize(img, image, cv::Size(width, height), 0, 0,
              cv::INTER_LINEAR);
   auto result = det->run(image);
   for (auto y = 0; y < result.segmentation.rows; y++) {
```

```
    for (auto x = 0; x < result.segmentation.cols; x++) {
          result.segmentation.at<uchar>(y,x) *= 10;
      }
  }
  cv::imwrite("segres.jpg",result.segmentation);
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Segmentation8UC1` class:

Table 180: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Segmentation8UC1` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`SegmentationResult`	run	const cv::Mat & image
std::vector< `SegmentationResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Segmentation8UC1`.

Prototype

```
std::unique_ptr< Segmentation8UC1 > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 181: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Segmentation8UC1` class.

getInputWidth

Function to get InputWidth of the segmentation network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the segmentation network.

getInputHeight

Function to get InputHeight of the segmentation network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputHeight of the segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function to get running result of the segmentation network.

***Note*:** The result cv::Mat of the type is CV_8UC1.

Prototype

```
SegmentationResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 182: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of the image (cv::Mat)

Returns

A Struct of `SegmentationResult` ,the result of segmentation network.

run

Function to get running results of the segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC1 of the Result's segmentation.

Prototype

```
std::vector< SegmentationResult > run(const std::vector< cv::Mat > &images);
```

Parameters

The following table lists the `run` function arguments.

Table 183: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `SegmentationResult`.

vitis::ai::Segmentation8UC3

The Class of `Segmentation8UC3`, this class run function `run(const cv::Mat& image)` return a cv::Mat with the type is cv_8UC3 Sample code :

```
   auto det =
vitis::ai::Segmentation8UC3::create(vitis::ai::SEGMENTATION_FPN);
  auto img = cv::imread("sample_segmentation.jpg");

   int width = det->getInputWidth();
   int height = det->getInputHeight();
   cv::Mat image;
   cv::resize(img, image, cv::Size(width, height), 0, 0,
              cv::INTER_LINEAR);
   auto result = det->run(image);
   cv::imwrite("segres.jpg",result.segmentation);
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::Segmentation8UC3` class:

Table 184: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Segmentation8UC3` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`SegmentationResult`	run	const cv::Mat & image
std::vector< `SegmentationResult` >	run	const std::vector< cv::Mat > & images

Functions

create

Factory function to get an instance of derived classes of class `Segmentation8UC3`.

Prototype

```
std::unique_ptr< Segmentation8UC3 > create(const std::string &model_name,
bool need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 185: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `Segmentation8UC3` class.

getInputWidth

Function to get InputWidth of the segmentation network (input image columns).

Prototype

```
int getInputWidth() const ;
```

Returns

InputWidth of the segmentation network.

getInputHeight

Function to get InputWidth of the segmentation network (input image rows).

Prototype

```
int getInputHeight() const ;
```

Returns

InputWidth of the segmentation network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const ;
```

Returns

Batch size.

run

Function to get running result of the segmentation network.

***Note*:** The result cv::Mat of the type is CV_8UC3.

Prototype

```
SegmentationResult run(const cv::Mat &image);
```

Parameters

The following table lists the `run` function arguments.

Table 186: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of the image (cv::Mat)

Returns

`SegmentationResult`, the result of the segmentation network.

run

Function to get running results of the segmentation neuron network in batch mode.

***Note*:** The type of CV_8UC3 of the segmentation result.

Prototype

```
std::vector< SegmentationResult > run(const std::vector< cv::Mat > &images);
```

Parameters

The following table lists the `run` function arguments.

Table 187: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `SegmentationResult`.

vitis::ai::SSD

Base class for detecting position of vehicle, pedestrian, and so on.

Input is an image (cv:Mat).

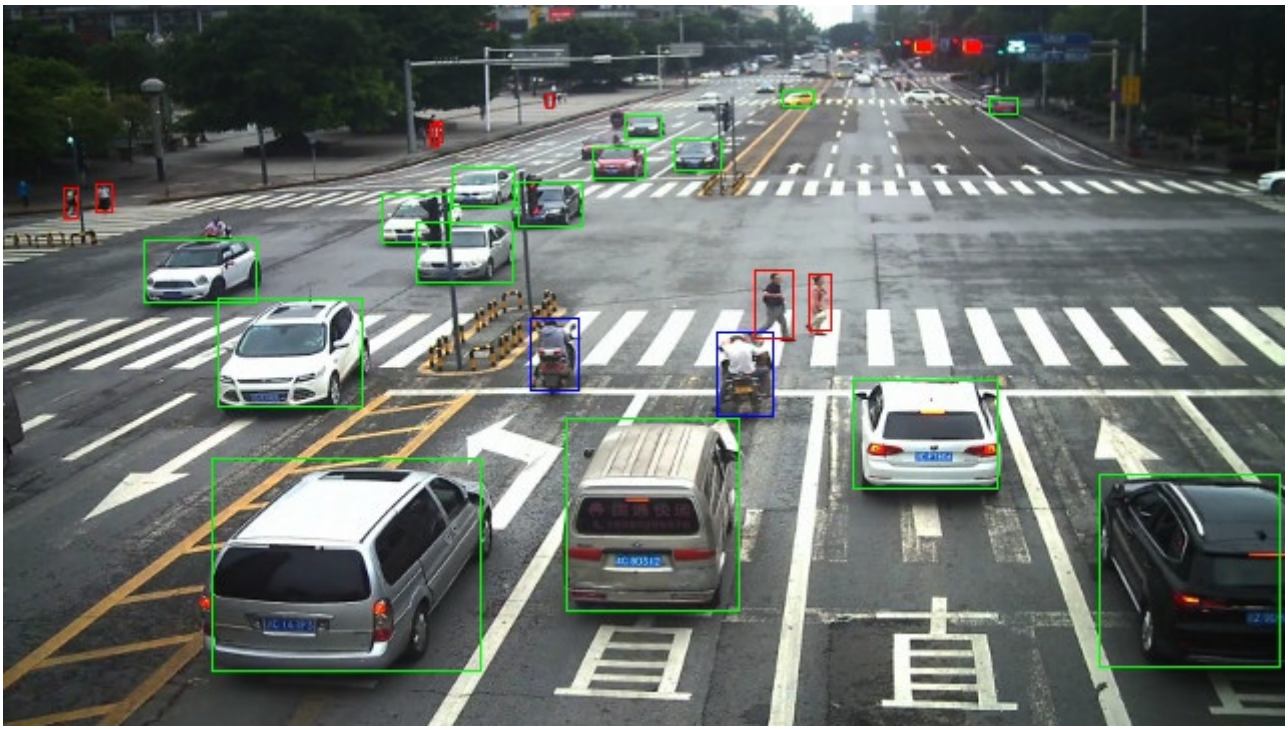
Output is a struct of detection results, named `SSDResult`.

Sample code :

```
  Mat img = cv::imread("sample_ssd.jpg");
  auto ssd = vitis::ai::SSD::create("ssd_traffic_pruned_0_9",true);
  auto results = ssd->run(img);
  for(const auto &r : results.bboxes){
     auto label = r.label;
     auto x = r.x * img.cols;
     auto y = r.y * img.rows;
     auto width = r.width * img.cols;
     auto heigth = r.height * img.rows;
     auto score = r.score;
     std::cout << "RESULT: " << label << "\t" << x << "\t" << y << "\t" <<
width
         << "\t" << height << "\t" << score << std::endl;
  }
```

Display of the model results:

Figure 43: **detection result**



Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::SSD` class:

Table 188: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `SSD` >	create	const std::string & model_name bool need_preprocess
std::unique_ptr< `SSD` >	create	void
`vitis::ai::SSDResult`	run	const cv::Mat & image
std::vector< `vitis::ai::SSDResult` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `SSD`.

Prototype

```
std::unique_ptr< SSD > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 189: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `SSD` class.

create

Prototype

```
std::unique_ptr< SSD > create(const std::string &model_name, xir::Attrs
*attrs, bool need_preprocess=true);
```

run

Function to get running results of the `SSD` neuron network.

Prototype

```
vitis::ai::SSDResult run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 190: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`SSDResult`.

run

Function to get running results of the `SSD` neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::SSDResult > run(const std::vector< cv::Mat >
&images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 191: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `SSDResult`.

getInputWidth

Function to get InputWidth of the `SSD` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `SSD` network.

getInputHeight

Function to get InputHeight of the `SSD` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `SSD` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::SSDPostProcess

Class of the `SSD` post-process. It initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::SSDPostProcess` class:

Table 192: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `SSDPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
std::vector< `SSDResult` >	ssd_post_process	void

Functions

create

Create an `SSDPostProcess` object.

Prototype

```
std::unique_ptr< SSDPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 193: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

An unique printer of `SSDPostProcess`.

ssd_post_process

The batch mode post-processing function of the `SSD` network.

Prototype

```
std::vector< SSDResult > ssd_post_process(size_t batch_size)=0;
```

Returns

The vector of struct of `SSDResult`.

vitis::ai::TFRefineDetPostProcess

Class of the tfrefinedet post-process. It initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::TFRefineDetPostProcess` class:

Table 194: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `TFRefineDetPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
`RefineDetResult`	tfrefinedet_post_process	void
std::vector< `RefineDetResult` >	tfrefinedet_post_process	void

Functions

create

Create an `TFRefineDetPostProcess` object.

Prototype

```
std::unique_ptr< TFRefineDetPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config);
```

Parameters

The following table lists the `create` function arguments.

Table 195: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique pointer of `TFRefineDetPostProcess`.

tfrefinedet_post_process

Run tfrefinedet post-process.

Prototype

```
RefineDetResult tfrefinedet_post_process(unsigned int idx)=0;
```

Returns

The struct of `RefineDetResult`.

tfrefinedet_post_process

Run batch mode of tfrefinedet post-process.

Prototype

```
std::vector< RefineDetResult > tfrefinedet_post_process()=0;
```

Returns

The vector of struct of `RefineDetResult`.

vitis::ai::TFSSD

Base class for detecting 90 objects of the COCO dataset.

Input is an image (cv:Mat).

Output is a struct of detection results, named `TFSSDResult`.

Sample code :

```
  Mat img = cv::imread("sample_tfssd.jpg");
  auto tfssd = vitis::ai::TFSSD::create("ssd_resnet_50_fpn_coco_tf",true);
  auto results = tfssd->run(img);
  for(const auto &r : results.bboxes){
     auto label = r.label;
     auto x = r.x * img.cols;
     auto y = r.y * img.rows;
     auto width = r.width * img.cols;
     auto height = r.height * img.rows;
     auto score = r.score;
     std::cout << "RESULT: " << label << "\t" << x << "\t" << y << "\t" <<
width
        << "\t" << height << "\t" << score << std::endl;
  }
```

Display of the model results:

Figure 44: **detection result**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::TFSSD` class:

Table 196: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `TFSSD` >	create	const std::string & model_name bool need_preprocess
`vitis::ai::TFSSDResult`	run	const cv::Mat & img
std::vector< `vitis::ai::TFSSDResult` >	run	const std::vector< cv::Mat > & imgs
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `SSD`.

Prototype

```
std::unique_ptr< TFSSD > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 197: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `TFSSD` class.

run

Function of get result of the ssd neuron network.

Prototype

```
vitis::ai::TFSSDResult run(const cv::Mat &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 198: **run Arguments**

Type	Name	Description
const cv::Mat &	img	Input data of input image (cv::Mat).

Returns

`TFSSDResult`.

run

Function to get running results of the `SSD` neuron network in batch mode.

Prototype

```
std::vector< vitis::ai::TFSSDResult > run(const std::vector< cv::Mat >
&imgs)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 199: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	imgs	Input data of input images (vector<cv::Mat>).The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `TFSSDResult`.

getInputWidth

Function to get InputWidth of the `SSD` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `TFSSD` network.

getInputHeight

Function to get InputHeight of the `SSD` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `TFSSD` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::TFSSDPostProcess

Class of the `TFSSD` post-process. It initializes the parameters once instead of computing them each time the program executes.

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::TFSSDPostProcess` class:

Table 200: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `TFSSDPostProcess` >	create	const std::vector< vitis::ai::library::InputTensor > & input_tensors const std::vector< vitis::ai::library::OutputTensor > & output_tensors const vitis::ai::proto::DpuModelParam & config
`TFSSDResult`	ssd_post_process	void
std::vector< `TFSSDResult` >	ssd_post_process	void

Functions

create

Create an `TFSSDPostProcess` object.

Prototype

```
std::unique_ptr< TFSSDPostProcess > create(const std::vector<
vitis::ai::library::InputTensor > &input_tensors, const std::vector<
vitis::ai::library::OutputTensor > &output_tensors, const
vitis::ai::proto::DpuModelParam &config, const std::string &dirname);
```

Parameters

The following table lists the `create` function arguments.

Table 201: **create Arguments**

Type	Name	Description
const std::vector< vitis::ai::library::InputTensor > &	input_tensors	A vector of all input-tensors in the network. Usage: input_tensors[input_tensor_index].
const std::vector< vitis::ai::library::OutputTensor > &	output_tensors	A vector of all output-tensors in the network. Usage: output_tensors[output_index].
const vitis::ai::proto::DpuModelParam &	config	The DPU model configuration information.

Returns

A unique printer of `TFSSDPostProcess`.

ssd_post_process

The post-processing function of the `TFSSD` network.

Prototype

```
TFSSDResult ssd_post_process(unsigned int idx)=0;
```

Returns

The struct of `TFSSDResult`.

ssd_post_process

The batch mode post-processing function of the `TFSSD` network.

Prototype

```
std::vector< TFSSDResult > ssd_post_process()=0;
```

Returns

The vector of struct of `TFSSDResult`.

vitis::ai::YOLOv2

Base class for detecting objects in the input image(cv::Mat). Input is an image(cv::Mat). Output is the position of the objects in the input image. Sample code:

```
auto img = cv::imread("sample_yolov2.jpg");
auto model = vitis::ai::YOLOv2::create("yolov2_voc");
auto result = model->run(img);
for (const auto &bbox : result.bboxes) {
  int label = bbox.label;
  float xmin = bbox.x * img.cols + 1;
  float ymin = bbox.y * img.rows + 1;
  float xmax = xmin + bbox.width * img.cols;
  float ymax = ymin + bbox.height * img.rows;
  if (xmax > img.cols)
    xmax = img.cols;
  if (ymax > img.rows)
    ymax = img.rows;
  float confidence = bbox.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);
}
```

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::YOLOv2` class:

Table 202: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `YOLOv2` >	create	const std::string & model_name bool need_preprocess
`YOLOv2Result`	run	const cv::Mat & image
std::vector< `YOLOv2Result` >	run	const std::vector< cv::Mat > & images
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `YOLOv2`.

Prototype

```
std::unique_ptr< YOLOv2 > create(const std::string &model_name, bool
need_preprocess=true);
```

Parameters

The following table lists the `create` function arguments.

Table 203: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `YOLOv2` class.

run

Function to get running result of the `YOLOv2` neuron network.

Prototype

```
YOLOv2Result run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 204: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

A Struct of `YOLOv2Result`.

run

Function to get running result of the `YOLOv2` neuron network in batch mode.

Prototype

```
std::vector< YOLOv2Result > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 205: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `YOLOv2Result`.

getInputWidth

Function to get InputWidth of the `YOLOv2` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `YOLOv2` network

getInputHeight

Function to get InputHeight of the `YOLOv2` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `YOLOv2` network.

get_input_batch

Function to get the number of images processed by the DPU at one time.

***Note*:** Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

vitis::ai::YOLOv3

Base class for detecting objects in the input image (cv::Mat).

Input is an image (cv::Mat).

Output is the position of the pedestrians in the input image.

Sample code:

```
   auto yolo =
vitis::ai::YOLOv3::create("yolov3_adas_pruned_0_9", true);
   Mat img = cv::imread("sample_yolov3.jpg");

   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";
     if (label == 0) {
       rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255,
0),
                 1, 1, 0);
     } else if (label == 1) {
       rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(255, 0,
0),
                 1, 1, 0);
     } else if (label == 2) {
       rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 0,
255),
                 1, 1, 0);
     } else if (label == 3) {
       rectangle(img, Point(xmin, ymin), Point(xmax, ymax),
                 Scalar(0, 255, 255), 1, 1, 0);
     }

   }
   imwrite("sample_yolov3_result.jpg", img);
```

Display of the model results:

Figure 45: **out image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::YOLOv3` class:

Table 206: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `YOLOv3` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
`YOLOv3Result`	run	const cv::Mat & image
std::vector< `YOLOv3Result` >	run	const std::vector< cv::Mat > & images
size_t	get_input_batch	void

Functions

create

Factory function to get an instance of derived classes of class `YOLOv3`.

Prototype

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

Parameters

The following table lists the `create` function arguments.

Table 207: **create Arguments**

Type	Name	Description
const std::string &	model_name	Model name
bool	need_preprocess	Normalize with mean/scale or not, default value is true.

Returns

An instance of `YOLOv3` class.

getInputWidth

Function to get InputWidth of the `YOLOv3` network (input image columns).

Prototype

```
int getInputWidth() const =0;
```

Returns

InputWidth of the `YOLOv3` network

getInputHeight

Function to get InputHeight of the `YOLOv3` network (input image rows).

Prototype

```
int getInputHeight() const =0;
```

Returns

InputHeight of the `YOLOv3` network.

run

Function to get running result of the `YOLOv3` neuron network.

Prototype

```
YOLOv3Result run(const cv::Mat &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 208: **run Arguments**

Type	Name	Description
const cv::Mat &	image	Input data of input image (cv::Mat).

Returns

`YOLOv3Result`.

run

Function to get running result of the `YOLOv3` neuron network in batch mode.

Prototype

```
std::vector< YOLOv3Result > run(const std::vector< cv::Mat > &images)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 209: **run Arguments**

Type	Name	Description
const std::vector< cv::Mat > &	images	Input data of input images (std:vector<cv::Mat>). The size of input images equals batch size obtained by get_input_batch.

Returns

The vector of `YOLOv3Result`.

get_input_batch

Function to get the number of images processed by the DPU at one time.

Note: Different DPU core the batch size may be different. This depends on the IP used.

Prototype

```
size_t get_input_batch() const =0;
```

Returns

Batch size.

Data Structure Index

The following is a list of data structures:

- vitis::ai::ANNORET
- vitis::ai::ClassificationResult
- vitis::ai::ClassificationResult::Score
- vitis::ai::Covid19SegmentationResult
- vitis::ai::DISPLAY_PARAM
- vitis::ai::FaceDetectResult
- vitis::ai::FaceDetectResult::BoundingBox
- vitis::ai::FaceFeatureFixedResult
- vitis::ai::FaceFeatureFloatResult
- vitis::ai::FaceLandmarkResult
- vitis::ai::FaceQuality5ptResult
- vitis::ai::HourglassResult
- vitis::ai::HourglassResult::PosePoint
- vitis::ai::MedicalDetectionResult
- vitis::ai::MedicalDetectionResult::BoundingBox
- vitis::ai::MedicalSegcellResult
- vitis::ai::MedicalSegmentationResult
- vitis::ai::MultiTaskResult
- vitis::ai::OpenPoseResult
- vitis::ai::OpenPoseResult::PosePoint
- vitis::ai::PPResult
- vitis::ai::PlateDetectResult

- vitis::ai::PlateDetectResult::BoundingBox
- vitis::ai::PlateDetectResult::Point
- vitis::ai::PlateNumResult
- vitis::ai::PointPillarsResult
- vitis::ai::PoseDetectResult
- vitis::ai::PoseDetectResult::Pose14Pt
- vitis::ai::RefineDetResult
- vitis::ai::RefineDetResult::BoundingBox
- vitis::ai::ReidResult
- vitis::ai::RetinaFaceResult
- vitis::ai::RetinaFaceResult::BoundingBox
- vitis::ai::RoadLineResult
- vitis::ai::RoadLineResult::Line
- vitis::ai::SSDResult
- vitis::ai::SSDResult::BoundingBox
- vitis::ai::Segmentation3DResult
- vitis::ai::SegmentationResult
- vitis::ai::TFSSDResult
- vitis::ai::TFSSDResult::BoundingBox
- vitis::ai::VehicleResult
- vitis::ai::YOLOv2Result
- vitis::ai::YOLOv2Result::BoundingBox
- vitis::ai::YOLOv3Result
- vitis::ai::YOLOv3Result::BoundingBox
- xilinx::ai::FaceQualityResult

vitis::ai::ANNORET

Struct of the result returned by the pointpillars neuron network in the annotation mode. It is mainly used for accuracy test or bev image drawing.

Declaration

```
typedef struct
{
  std::vector< std::string > name,
  V1I label,
  V1F truncated,
  V1I occluded,
  V1F alpha,
  V2I bbox,
  V2F dimensions,
  V2F location,
  V1F rotation_y,
  V1F score,
  V2F box3d_camera,
  V2F box3d_lidar  void clear(),

} vitis::ai::ANNORET;
```

Table 210: **Structure vitis::ai::ANNORET member description**

Member	Description
name	Name of detected result in vector: such as Car Cylist Pedestrian.
label	Label of detected result.
truncated	Truncated information.
occluded	Occluded information.
alpha	Alpha information.
bbox	bbox information.
dimensions	Dimensions information.
location	Location information.
rotation_y	rotation_y information.
score	Score information.
box3d_camera	box3d_camera information.
box3d_lidar	box3d_lidar information.
clear	Inner function to clear all fields.

vitis::ai::ClassificationResult

Struct of the result with the classification network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< Score > scores,
  int type  const char * lookup(int index),

} vitis::ai::ClassificationResult;
```

Table 211: **Structure vitis::ai::ClassificationResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
scores	A vector of objects width confidence in the first k, k defaults to 5 and can be modified through the model configuration file.
type	`Classification` label type.
lookup	The classification corresponding by index.

vitis::ai::ClassificationResult::Score

The struct of index and confidence for an object.

Declaration

```
typedef struct
{
  int index,
  float score
} vitis::ai::ClassificationResult::Score;
```

Table 212: **Structure vitis::ai::ClassificationResult::Score member description**

Member	Description
index	The index of the result in the ImageNet.
score	Confidence of this category.

vitis::ai::Covid19SegmentationResult

Declaration

```
typedef struct
{
  int width,
  int height,
  cv::Mat positive_classification,
  cv::Mat infected_area_classification
} vitis::ai::Covid19SegmentationResult;
```

Table 213: **Structure vitis::ai::Covid19SegmentationResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
positive_classification	Positive detection result. The cv::Mat type is CV_8UC1 or CV_8UC3.
infected_area_classification	Infected area detection result. The cv::Mat type is CV_8UC1 or CV_8UC3.

vitis::ai::DISPLAY_PARAM

Four data structure get from the calibration information. It is mainly used for accuracy test or bev image drawing. See detail in readme in the overview for more information.

Declaration

```
typedef struct
{
  V2F P2,
  V2F rect,
  V2F Trv2c,
  V2F p2rect
} vitis::ai::DISPLAY_PARAM;
```

Table 214: **Structure vitis::ai::DISPLAY_PARAM member description**

Member	Description
P2	P2 information:
rect	rect information
Trv2c	Trv2c information.
p2rect	p2rect information

vitis::ai::FaceDetectResult

Struct of the result with the facedetect network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > rects
} vitis::ai::FaceDetectResult;
```

Table 215: **Structure vitis::ai::FaceDetectResult member description**

Member	Description
width	Width of an input image.
height	Height of an input image.
rects	All faces, filtered by confidence >= detect threshold.

vitis::ai::FaceDetectResult::BoundingBox

The coordinate and confidence of a face.

Declaration

```
typedef struct
{
  float x,
  float y,
  float width,
  float height,
  float score
} vitis::ai::FaceDetectResult::BoundingBox;
```

Table 216: **Structure vitis::ai::FaceDetectResult::BoundingBox member description**

Member	Description
x	x-coordinate , x is normalized relative to the input image columns ,the value range from 0 to 1.
y	y-coordinate , y is normalized relative to the input image rows ,the value range from 0 to 1.
width	face width , width is normalized relative to the input image columns , the value range from 0 to 1.
height	face height , heigth is normalized relative to the input image rows ,the value range from 0 to 1.
score	face confidence, the value range from 0 to 1.

vitis::ai::FaceFeatureFixedResult

The result of `FaceFeature`. It is a 512 dimensions vector, fix point values.

Declaration

```
typedef struct
{
  std::array< int8_t, 512 > vector_t  int width,
  int height,
  float scale  std::unique_ptr< vector_t > feature,

} vitis::ai::FaceFeatureFixedResult;
```

Table 217: **Structure vitis::ai::FaceFeatureFixedResult member description**

Member	Description
vector_t	The 512 dimensions vector, in fix point format.
width	Width of an input image.
height	Height of an input image.
scale	The fix point.
feature	

vitis::ai::FaceFeatureFloatResult

The result of `FaceFeature`. It is a 512 dimensions vector, float value.

Declaration

```
typedef struct
{
  std::array< float, 512 > vector_t  int width,
  int height  std::unique_ptr< vector_t > feature,

} vitis::ai::FaceFeatureFloatResult;
```

Table 218: **Structure vitis::ai::FaceFeatureFloatResult member description**

Member	Description
vector_t	The 512 dimensions vector.
width	Width of an input image.
height	Height of an input image.
feature	

vitis::ai::FaceLandmarkResult

Struct of the result returned by the facelandmark network.

Declaration

```
typedef struct
{
  std::array< std::pair< float, float >, 5 > points
} vitis::ai::FaceLandmarkResult;
```

Table 219: **Structure vitis::ai::FaceLandmarkResult member description**

Member	Description
points	Five key points coordinate, this array of <x,y> has five elements ,x / y is normalized relative to width / height, the value range from 0 to 1.

vitis::ai::FaceQuality5ptResult

The struct of result returned by the facequality5pt network.

Declaration

```
typedef struct
{
  int width,
  int height,
  float score,
  std::array< std::pair< float, float >, 5 > points
} vitis::ai::FaceQuality5ptResult;
```

Table 220: **Structure vitis::ai::FaceQuality5ptResult member description**

Member	Description
width	Width of a input image.
height	Height of a input image.
score	The quality of face. The value range is from 0 to 1 if the option "orginal_quality" in the model prototxt is false, it is a normal mode. If the option "original_quality" is true, the quality score can be larger than 1, this is a special mode only for accuracy test.
points	Five key points coordinate. An array of <x,y> has five elements where x and y are normalized relative to input image columns and rows. The value range is from 0 to 1.

vitis::ai::HourglassResult

Result with the openpose network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< PosePoint > poses
} vitis::ai::HourglassResult;
```

Table 221: **Structure vitis::ai::HourglassResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
poses	A vector of pose, pose is represented by a vector of `PosePoint`. Joint points are arranged in order 0: head, 1: neck, 2: L_shoulder, 3:L_elbow, 4: L_wrist, 5: R_shoulder, 6: R_elbow, 7: R_wrist, 8: L_hip, 9:L_knee, 10: L_ankle, 11: R_hip, 12: R_knee, 13: R_ankle

vitis::ai::HourglassResult::PosePoint

Struct of a coordinate point and the point type.

Declaration

```
typedef struct
{
  int type,
  cv::Point2f point
} vitis::ai::HourglassResult::PosePoint;
```

Table 222: **Structure vitis::ai::HourglassResult::PosePoint member description**

Member	Description
type	Point type. • `1` : "valid" • `3` : "invalid"
point	Coordinate point.

vitis::ai::MedicalDetectionResult

Struct of the result returned by the medical refinedet network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::MedicalDetectionResult;
```

Table 223: **Structure vitis::ai::MedicalDetectionResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
bboxes	All objects, a vector of `BoundingBox`.

vitis::ai::MedicalDetectionResult::BoundingBox

Struct of an object coordinate ,confidence and classification.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::MedicalDetectionResult::BoundingBox;
```

Table 224: **Structure vitis::ai::MedicalDetectionResult::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence.
x	x-coordinate, x is normalized relative to the input image columns ,the value range from 0 to 1.
y	y-coordinate ,y is normalized relative to the input image rows ,the value range from 0 to 1.
width	width, width is normalized relative to the input image columns ,the value range from 0 to 1.
height	height, height is normalized relative to the input image rows ,the value range from 0 to 1.

vitis::ai::MedicalSegcellResult

Struct of the result returned by the segmentation neuron network.

Declaration

```
typedef struct
{
  int width,
  int height,
  cv::Mat segmentation
} vitis::ai::MedicalSegcellResult;
```

Table 225: **Structure vitis::ai::MedicalSegcellResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
segmentation	segmentation result in cv::Mat mode

vitis::ai::MedicalSegmentationResult

Struct of the result returned by the `MedicalSegmentation` neuron network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< cv::Mat > segmentation
} vitis::ai::MedicalSegmentationResult;
```

Table 226: **Structure vitis::ai::MedicalSegmentationResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
segmentation	A vector of cv::Mat (segmentation result).

vitis::ai::MultiTaskResult

Struct of the result returned by the `MultiTask` network, when you need to visualize.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< VehicleResult > vehicle,
  cv::Mat segmentation
} vitis::ai::MultiTaskResult;
```

Table 227: **Structure vitis::ai::MultiTaskResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
vehicle	Detection result of `SSD` task.
segmentation	`Segmentation` result to visualize, cv::Mat type is CV_8UC1 or CV_8UC3.

vitis::ai::OpenPoseResult

Result with the openpose network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< std::vector< PosePoint > > poses
} vitis::ai::OpenPoseResult;
```

Table 228: **Structure vitis::ai::OpenPoseResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
poses	A vector of pose. Pose is represented by a vector of `PosePoint`. Joint points are arranged in order 0: head, 1: neck, 2: L_shoulder, 3:L_elbow, 4: L_wrist, 5: R_shoulder, 6: R_elbow, 7: R_wrist, 8: L_hip, 9:L_knee, 10: L_ankle, 11: R_hip, 12: R_knee, 13: R_ankle

vitis::ai::OpenPoseResult::PosePoint

Struct of a coordinate point and the point type.

Declaration

```
typedef struct
{
  int type,
  cv::Point2f point
} vitis::ai::OpenPoseResult::PosePoint;
```

Table 229: **Structure vitis::ai::OpenPoseResult::PosePoint member description**

Member	Description
type	Point type. • `1` : "valid" • `3` : "invalid"
point	Coordinate point.

vitis::ai::PlateDetectResult

Struct of the result returned by the platedetect network.

Declaration

```
typedef struct
{
  int width,
  int height,
  BoundingBox box,
  Point top_left,
  Point top_right,
  Point bottom_left,
  Point bottom_right
} vitis::ai::PlateDetectResult;
```

Table 230: **Structure vitis::ai::PlateDetectResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
box	The position of plate.
top_left	The top_left point.
top_right	The top_right point.
bottom_left	The bottom_left point.
bottom_right	The bottom_right point.

vitis::ai::PlateDetectResult::BoundingBox

Declaration

```
typedef struct
{
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::PlateDetectResult::BoundingBox;
```

Table 231: **Structure vitis::ai::PlateDetectResult::BoundingBox member description**

Member	Description
score	Plate confidence, the value ranges from 0 to 1.
x	x-coordinate of the plate, x is normalized relative to input image columns, the value ranges from 0 to 1.
y	y-coordinate of the plate, y is normalized relative to input image rows, the value ranges from 0 to 1.
width	Plate width, width is normalized relative to input image columns, the value ranges from 0 to 1.
height	Plate height, height is normalized relative to input image rows, the value ranges from 0 to 1.

vitis::ai::PlateDetectResult::Point

Plate coordinate point.

Declaration

```
typedef struct
{
  float x,
  float y
} vitis::ai::PlateDetectResult::Point;
```

Table 232: **Structure vitis::ai::PlateDetectResult::Point member description**

Member	Description
x	x-coordinate, the value ranges from 0 to 1.
y	y-coordinate, the value ranges from 0 to 1.

vitis::ai::PlateNumResult

Struct of the result of the platenum network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::string plate_number,
  std::string plate_color
} vitis::ai::PlateNumResult;
```

Table 233: **Structure vitis::ai::PlateNumResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
plate_number	The plate number.
plate_color	The plate color, Blue / Yellow.

vitis::ai::PointPillarsResult

Struct of the final result returned by the pointpillars neuron network encapsulated with width/height information.

Declaration

```
typedef struct
{
  int width,
  int height,
  PPResult ppresult
} vitis::ai::PointPillarsResult;
```

Table 234: **Structure vitis::ai::PointPillarsResult member description**

Member	Description
width	Width of network input.
height	Height of network input.
ppresult	Final result returned by the pointpillars neuron network.

vitis::ai::PoseDetectResult

Struct of the result returned by the posedetect network.

Declaration

```
typedef struct
{
  cv::Point2f Point  int width,
  int height  Pose14Pt pose14pt,

} vitis::ai::PoseDetectResult;
```

Table 235: **Structure vitis::ai::PoseDetectResult member description**

Member	Description
Point	A coordinate points.
width	Width of input image.
height	Height of input image.
pose14pt	The pose of input image.

vitis::ai::PoseDetectResult::Pose14Pt

A pose, represented by 14 coordinate points.

Declaration

```
typedef struct
{
  Point right_shoulder,
  Point right_elbow,
  Point right_wrist,
  Point left_shoulder,
  Point left_elbow,
```

```
  Point left_wrist,
  Point right_hip,
  Point right_knee,
  Point right_ankle,
  Point left_hip,
  Point left_knee,
  Point left_ankle,
  Point head,
  Point neck
} vitis::ai::PoseDetectResult::Pose14Pt;
```

Table 236: **Structure vitis::ai::PoseDetectResult::Pose14Pt member description**

Member	Description
right_shoulder	R_shoulder coordinate.
right_elbow	R_elbow coordinate.
right_wrist	R_wrist coordinate.
left_shoulder	L_shoulder coordinate.
left_elbow	L_elbow coordinate.
left_wrist	L_wrist coordinate.
right_hip	R_hip coordinate.
right_knee	R_knee coordinate.
right_ankle	R_ankle coordinate.
left_hip	L_hip coordinate.
left_knee	L_knee coordinate.
left_ankle	L_ankle coordinate.
head	Head coordinate.
neck	Neck coordinate.

vitis::ai::PPResult

Struct of the final result returned by the pointpillars neuron network.

Declaration

```
typedef struct
{
  V2F final_box_preds,
  V1F final_scores,
  V1I label_preds
} vitis::ai::PPResult;
```

Table 237: **Structure vitis::ai::PPResult member description**

Member	Description
final_box_preds	Final box predicted.
final_scores	Final scores predicted.
label_preds	Final label predicted.

vitis::ai::RefineDetResult

Struct of the result with the refinedet network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::RefineDetResult;
```

Table 238: **Structure vitis::ai::RefineDetResult member description**

Member	Description
width	Width of the input image.
height	Height of the input image.
bboxes	The vector of `BoundingBox`.

vitis::ai::RefineDetResult::BoundingBox

Struct of an object coordinates and confidence.

Declaration

```
typedef struct
{
  float x,
  float y,
  float width,
  float height,
  int label,
  float score
} vitis::ai::RefineDetResult::BoundingBox;
```

Table 239: **Structure vitis::ai::RefineDetResult::BoundingBox member description**

Member	Description
x	x-coordinate , x is normalized relative to the input image columns ,the value ranges from 0 to 1.
y	y-coordinate , y is normalized relative to the input image rows ,the value ranges from 0 to 1.
width	Body width , width is normalized relative to the input image columns , the value ranges from 0 to 1.
height	Body height , height is normalized relative to the input image rows , the value ranges from 0 to 1.
label	Body detection label, the value ranges from 0 to 21.
score	Body detection confidence, the value ranges from 0 to 1.

vitis::ai::ReidResult

Result with the ReID network.

Declaration

```
typedef struct
{
  int width,
  int height,
  cv::Mat feat
} vitis::ai::ReidResult;
```

Table 240: **Structure vitis::ai::ReidResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
feat	The feature of input image.

vitis::ai::RetinaFaceResult

Struct of the result with the retinaface network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes,
  std::vector< std::array< std::pair< float, float >, 5 > > landmarks
} vitis::ai::RetinaFaceResult;
```

Table 241: **Structure vitis::ai::RetinaFaceResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
bboxes	All faces, filtered by confidence >= detect threshold.
landmarks	Landmarks.

vitis::ai::RetinaFaceResult::BoundingBox

The coordinate and confidence of a face.

Declaration

```
typedef struct
{
  float x,
  float y,
  float width,
  float height,
  float score
} vitis::ai::RetinaFaceResult::BoundingBox;
```

Table 242: **Structure vitis::ai::RetinaFaceResult::BoundingBox member description**

Member	Description
x	x-coordinate, x is normalized relative to the input image columns ,the value range from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows ,the value range from 0 to 1.
width	Face width, width is normalized relative to the input image columns , the value range from 0 to 1.
height	Face height, height is normalized relative to the input image rows ,the value range from 0 to 1.
score	Face confidence, the value ranges from 0 to 1.

vitis::ai::RoadLineResult

Struct of the result returned by the roadline network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< Line > lines
} vitis::ai::RoadLineResult;
```

Table 243: **Structure vitis::ai::RoadLineResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
lines	The vector of line.

vitis::ai::RoadLineResult::Line

Struct of the result returned by the roadline network.

Declaration

```
typedef struct
{
  int type,
  std::vector< cv::Point > points_cluster
} vitis::ai::RoadLineResult::Line;
```

Table 244: **Structure vitis::ai::RoadLineResult::Line member description**

Member	Description
type	Road line type, the value range from 0 to 3. • `0` : background • `1` : white dotted line • `2` : white solid line • `3` : yollow line
points_cluster	Point clusters, make line from these.

vitis::ai::Segmentation3DResult

Base class for segmentation 3D object data in the vector<float> mode.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< int > array
} vitis::ai::Segmentation3DResult;
```

Table 245: **Structure vitis::ai::Segmentation3DResult member description**

Member	Description
width	Width of the network model.
height	Height of the network model.
array	Input 3D object data.

vitis::ai::SegmentationResult

Struct of the result returned by the segmentation network.

FPN Num of segmentation classes

- `0` : "unlabeled"

- `1` : "ego vehicle"
- `2` : "rectification border"
- `3` : "out of roi"
- `4` : "static"
- `5` : "dynamic"
- `6` : "ground"
- `7` : "road"
- `8` : "sidewalk"
- `9` : "parking"
- `10` : "rail track"
- `11` : "building"
- `12` : "wall"
- `13` : "fence"
- `14` : "guard rail"
- `15` : "bridge"
- `16` : "tunnel"
- `17` : "pole"
- `18` : "polegroup"

Declaration

```
typedef struct
{
  int width,
  int height,
  cv::Mat segmentation
} vitis::ai::SegmentationResult;
```

Table 246: **Structure vitis::ai::SegmentationResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
segmentation	`Segmentation` result. The cv::Mat type is CV_8UC1 or CV_8UC3.

vitis::ai::SSDResult

Struct of the result returned by the `SSD` neuron network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::SSDResult;
```

Table 247: **Structure vitis::ai::SSDResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
bboxes	All objects, a vector of `BoundingBox`.

vitis::ai::SSDResult::BoundingBox

Struct of an object coordinate, confidence and classification.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::SSDResult::BoundingBox;
```

Table 248: **Structure vitis::ai::SSDResult::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence.
x	x-coordinate, x is normalized relative to the input image columns, the value ranges from 0 to 1.
y	y-coordinate ,y is normalized relative to the input image rows, the value ranges from 0 to 1.
width	Width, width is normalized relative to the input image columns, the value ranges from 0 to 1.
height	Height, height is normalized relative to the input image rows, the value ranges from 0 to 1.

vitis::ai::TFSSDResult

Struct of the result returned by the `TFSSD` neuron network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::TFSSDResult;
```

Table 249: **Structure vitis::ai::TFSSDResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
bboxes	All objects, a vector of `BoundingBox`.

vitis::ai::TFSSDResult::BoundingBox

Struct of an object coordinate, confidence and classification.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::TFSSDResult::BoundingBox;
```

Table 250: **Structure vitis::ai::TFSSDResult::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence.
x	x-coordinate, x is normalized relative to the input image columns, the value ranges from 0 to 1.
y	y-coordinate ,y is normalized relative to the input image rows, the value ranges from 0 to 1.
width	Width, width is normalized relative to the input image columns, the value ranges from 0 to 1.
height	Height, height is normalized relative to the input image rows, the value ranges from 0 to 1.

vitis::ai::VehicleResult

A struct to define detection result of `MultiTask`.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height,
  float angle
} vitis::ai::VehicleResult;
```

Table 251: **Structure vitis::ai::VehicleResult member description**

Member	Description
label	number of classes • 0 : "background" • 1 : "person" • 2 : "car" • 3 : "truck" • 4 : "bus" • 5 : "bike" • 6 : "sign" • 7 : "light"
score	Confidence of this target.
x	x-coordinate, x is normalized relative to the input image columns, the value ranges from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows, the value ranges from 0 to 1.
width	Width, width is normalized relative to the input image columns, the value ranges from 0 to 1.
height	Height, height is normalized relative to the input image rows, the value ranges from 0 to 1.
angle	The angle between the target vehicle and ourself.

vitis::ai::YOLOv2Result

Struct of the result returned by the `YOLOv2` network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::YOLOv2Result;
```

Table 252: **Structure vitis::ai::YOLOv2Result member description**

Member	Description
width	Width of input image.
height	Height of input image.
bboxes	All objects.

vitis::ai::YOLOv2Result::BoundingBox

Struct of an object coordinate, confidence, and classification.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::YOLOv2Result::BoundingBox;
```

Table 253: **Structure vitis::ai::YOLOv2Result::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence, the value ranges from 0 to 1.
x	x-coordinate, x is normalized relative to the input image columns, the value ranges from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows, the value ranges from 0 to 1.
width	Width, width is normalized relative to the input image columns, the value ranges from 0 to 1.
height	Height, height is normalized relative to the input image rows, the value ranges from 0 to 1.

vitis::ai::YOLOv3Result

Struct of the result returned by the `YOLOv3` neuron network.

Note: VOC dataset category:string label[20] = {"aeroplane", "bicycle", "bird", "boat", "bottle", "bus","car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike","person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"};

Note: ADAS dataset category : string label[3] = {"car", "person", "cycle"};

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::YOLOv3Result;
```

Table 254: **Structure vitis::ai::YOLOv3Result member description**

Member	Description
width	Width of input image.
height	Height of output image.
bboxes	All objects, The vector of `BoundingBox`.

vitis::ai::YOLOv3Result::BoundingBox

Struct of detection result with an object.

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::YOLOv3Result::BoundingBox;
```

Table 255: **Structure vitis::ai::YOLOv3Result::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence, the value ranges from 0 to 1.
x	x-coordinate, x is normalized relative to the input image columns, the value ranges from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows, the value ranges from 0 to 1.
width	Width, width is normalized relative to the input image columns, the value ranges from 0 to 1.
height	Height, height is normalized relative to the input image rows, the value ranges from 0 to 1.

xilinx::ai::FaceQualityResult

The result of the facequality network. It is a single float value.

Declaration

```
typedef struct
{
  int width,
  int height,
  float value
} xilinx::ai::FaceQualityResult;
```

Table 256: **Structure xilinx::ai::FaceQualityResult member description**

Member	Description
width	Width of input image.
height	Height of input image.
value	Quality value ranges from 0.0 to 1.0.

XILINX

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.

References

These documents provide supplemental material useful with this guide:

1. *Vitis AI User Guide* (UG1414)
2. *Vitis AI Optimizer User Guide* (UG1333)

Please Read: Important Legal Notices

Copyright