Vitis AI Library

Programming Guide

UG1355 (v1.1) March 23, 2020

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
03/23/2020 Version 1.1	
Entire document	Updated the content for Vitis AI v1.1.
12/02/2019 Version 1.0	
Initial release.	N/A

Table of Contents

Overview

Vitis AI Library

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

This document is compatible with version 1.x of the Vitis AI library.

Block Diagram

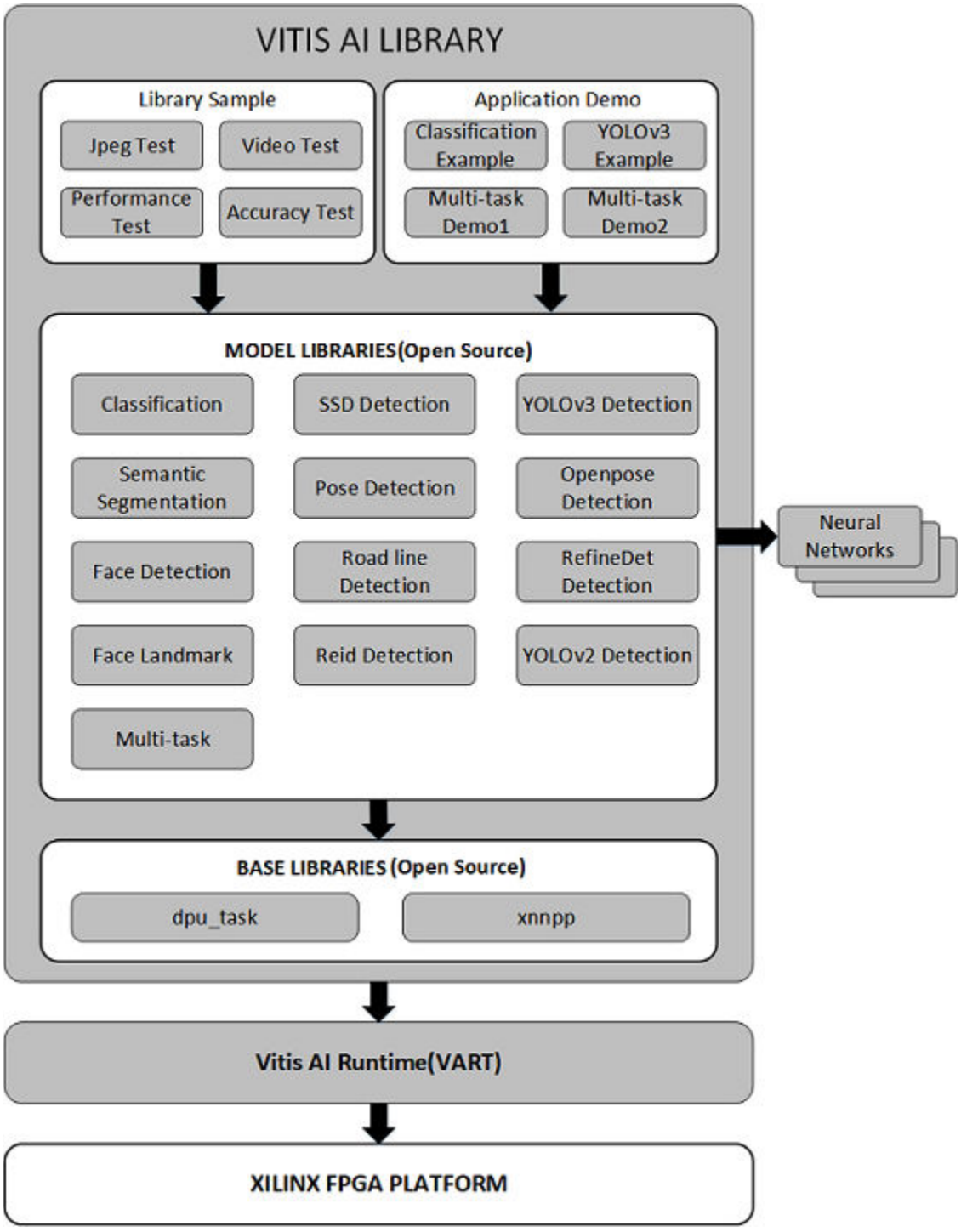
The Vitis AI library contains four parts, the base libraries, the model libraries, the library test samples and the application demos.

The base libraries mainly 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 build-in modules such as optimization and acceleration.

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

The library test samples are used to quickly test and evaluate the model libraries. The application demos show users how to use the library to develop applications. The Vitis AI library block diagram is shown in the following figure.

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



Features

The Vitis™ AI library features include.

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

Working with the Vitis AI Library

To be able to use the Vitis™ AI library, you need to prepare the development board and cross-compilation environment. During the development, you need to 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, and the Xilinx® Alveo™ U50 Data Center accelerator card.

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 minus 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.

Class Documentation

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(
               "resnet_50",
               true);
auto result = network->run(image);
for (const auto &r : result.scores) {
   auto score = r.score;
   auto index = network->lookup(r.index);
}
```

Quick Function Reference

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

Table 1: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `Classification` >	create	const std::string & model_name bool need_preprocess
const char *	lookup	int index
`vitis::ai::ClassificationResult`	run	const cv::Mat & image
std::vector< `vitis::ai::ClassificationResult` >	run	const std::vector< cv::Mat > & images

Table 1: **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 `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 2: **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.

lookup

Get the classification corresponding by index.

Prototype

```
const char * lookup(int index);
```

Parameters

The following table lists the `lookup` function arguments.

Table 3: **lookup Arguments**

Type	Name	Description
int	index	The network result

Returns

`Classification` description, if index < 0, return empty string

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 4: **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 5: **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 cols).

Prototype

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

Returns

InputWidth of the classification network

getInputHeight

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

Prototype

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

Returns

InputHeight of the classification 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 =0;
```

Returns

Batch size.

vitis::ai::DpuTask

Base class for run a DPU task.

Quick Function Reference

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

Table 6: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `DpuTask` >	create	kernel_name
void	run	void
void	setMeanScaleBGR	const std::vector< float > & mean const std::vector< float > & scale
void	setImageBGR	const cv::Mat & img
void	setImageRGB	const cv::Mat & img
std::vector< float >	getMean	void
std::vector< float >	getScale	void
std::vector< vitis::ai::library::InputTensor >	getInputTensor	void
std::vector< vitis::ai::library::OutputTensor >	getOutputTensor	void
size_t	get_num_of_kernels	void
const vitis::ai::DpuMeta &	get_dpu_meta_info	void

Functions

create

A static method to create a DPU task.

Prototype

```
std::unique_ptr< DpuTask > create(const std::string &kernal_name);
```

Parameters

The following table lists the `create` function arguments.

Table 7: **create Arguments**

Type	Name	Description
Commented parameter kernel_name does not exist in function create.	kernel_name	The dpu kernel name. for example, if kernel_name is "resnet_50", the following dpu model files are searched. ./ libdpumodelrestnet_50.so /usr/lib/libdpumodelrestnet_50.so

Returns

A `DpuTask` instance.

run

Run the dpu task.

***Note*:** Before invoking this function. An input data should be properly copied to input tensors, via `setImageBGR` or `setImageRGB`.

Prototype

```
void run(size_t idx)=0;
```

Returns

setMeanScaleBGR

Set the mean/scale values.

***Note*:** By default, no mean-scale processing, after invoking this function, mean-scale processing is enabled. You cannot turn it off after enabling.

Prototype

```
void setMeanScaleBGR(const std::vector< float > &mean, const std::vector<
float > &scale)=0;
```

Parameters

The following table lists the `setMeanScaleBGR` function arguments.

Table 8: **setMeanScaleBGR Arguments**

Type	Name	Description
const std::vector< float > &	mean	Mean, Normalization is used.
const std::vector< float > &	scale	Scale, Normalization is used.

Returns

setImageBGR

Copy a input image in BGR format to the input tensor.

Prototype

```
void setImageBGR(const cv::Mat &img)=0;
```

Parameters

The following table lists the `setImageBGR` function arguments.

Table 9: **setImageBGR Arguments**

Type	Name	Description
const cv::Mat &	img	The input image (cv::Mat).

setImageRGB

Copy a input image in RGB format to the input tensor.

Prototype

```
void setImageRGB(const cv::Mat &img)=0;
```

Parameters

The following table lists the `setImageRGB` function arguments.

Table 10: **setImageRGB Arguments**

Type	Name	Description
const cv::Mat &	img	The input image(cv::Mat).

getMean

Get the mean values.

Prototype

```
std::vector< float > getMean()=0;
```

Returns

Mean values

getScale

Get the scale values.

Prototype

```
std::vector< float > getScale()=0;
```

Returns

Scale values

getInputTensor

Get the input tensors.

Prototype

```
std::vector< vitis::ai::library::InputTensor > getInputTensor(size_t idx)=0;
```

Returns

The input tensors

getOutputTensor

Get the output tensors.

Prototype

```
std::vector< vitis::ai::library::OutputTensor > getOutputTensor(size_t
idx)=0;
```

Returns

The output tensors.

get_num_of_kernels

get the number of tasks

Prototype

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

get_dpu_meta_info

get meta info a model

Prototype

```
const vitis::ai::DpuMeta & get_dpu_meta_info() const =0;
```

vitis::ai::FaceDetect

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

Input is an image (cv::Mat).

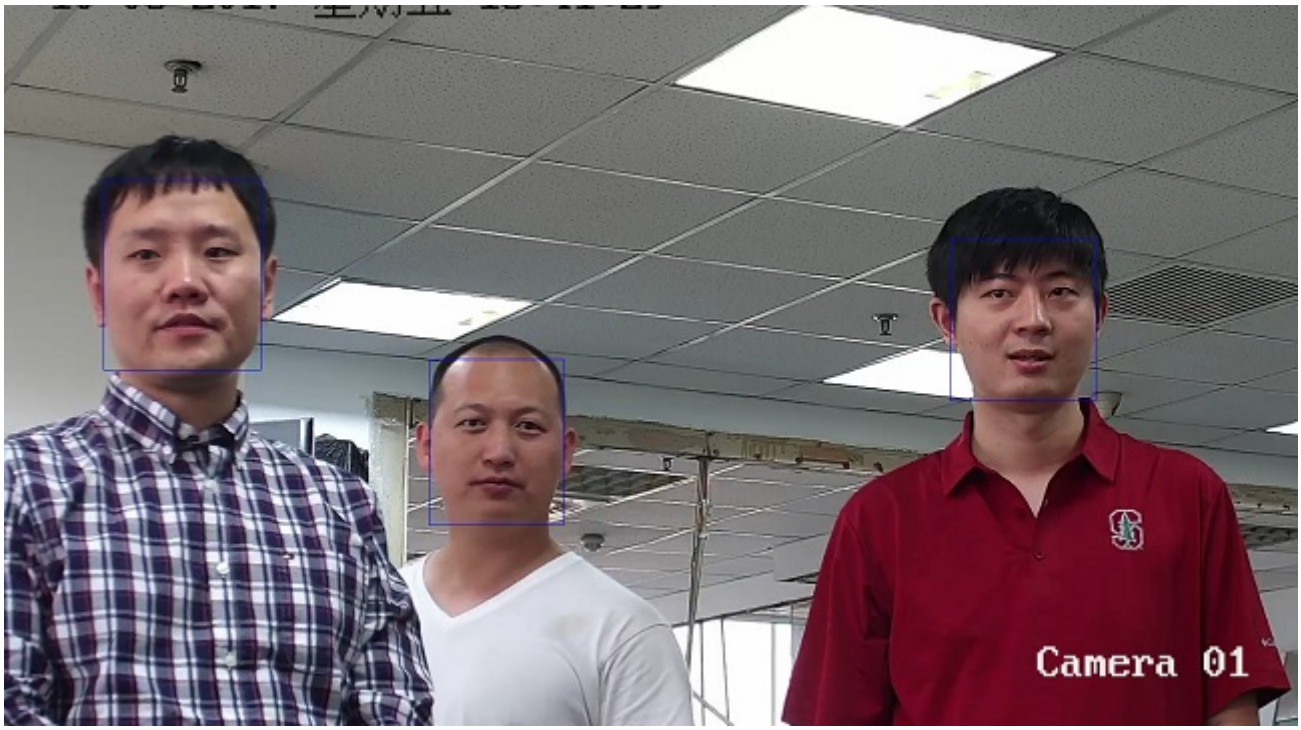
Output is a vector of position and score for faces in the input image.

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 = rz.height * image.rows;
}
```

Display of the model results:

Figure 2: **result image**



Quick Function Reference

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

Table 11: **Quick Function Reference**

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

Table 11: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
std::vector< `FaceDetectResult` >	run	images

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 12: **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.

getInputWidth

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

Prototype

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

Returns

InputWidth of the facedetect network

getInputHeight

Function to get InputHeigth 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 differnt. 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 range 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 13: **setThreshold Arguments**

Type	Name	Description
float	threshold	The detect threshold,the value range 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 14: **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 , filter 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 > &img)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 15: **run Arguments**

Type	Name	Description
Commented parameter images does not exist in function run.	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 `FaceDetectResult`.

vitis::ai::FaceLandmark

Base class for detecting five key points,and 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 contains multiple faces will return 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: image" width=100px

Figure 3: **result image**

Quick Function Reference

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

Table 16: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `FaceLandmark` >	create	const std::string & model_name bool need_preprocess
int	getInputWidth	void
int	getInputHeight	void
size_t	get_input_batch	void
`FaceLandmarkResult`	run	const cv::Mat & input_image
std::vector< `FaceLandmarkResult` >	run	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 17: **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.

getInputWidth

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

Prototype

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

Returns

InputWidth of the face landmark network.

getInputHeight

Function to get InputHeigth 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 differnt. 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 18: **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_image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 19: **run Arguments**

Type	Name	Description
Commented parameter images does not exist in function run.	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::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 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 4: **result image**

Quick Function Reference

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

Table 20: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `MultiTask` >	create	const std::string & model_name bool need_preprocess
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 21: **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 cols).

Prototype

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

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHight 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 differnt. 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 22: **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 23: **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 24: **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 25: **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 26: **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

Table 26: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
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 27: **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 cols).

Prototype

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

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHight 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 28: **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 29: **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 towards; 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 30: **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 31: **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 cols).

Prototype

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

Returns

InputWidth of the multitask network.

getInputHeight

Function to get InputHight 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_8UC3 of the `MultiTaskResult.segmentation`.

Prototype

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

Parameters

The following table lists the `run` function arguments.

Table 32: **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 33: **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 34: **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 35: **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].

Table 35: **create Arguments** *(cont'd)*

Type	Name	Description
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).

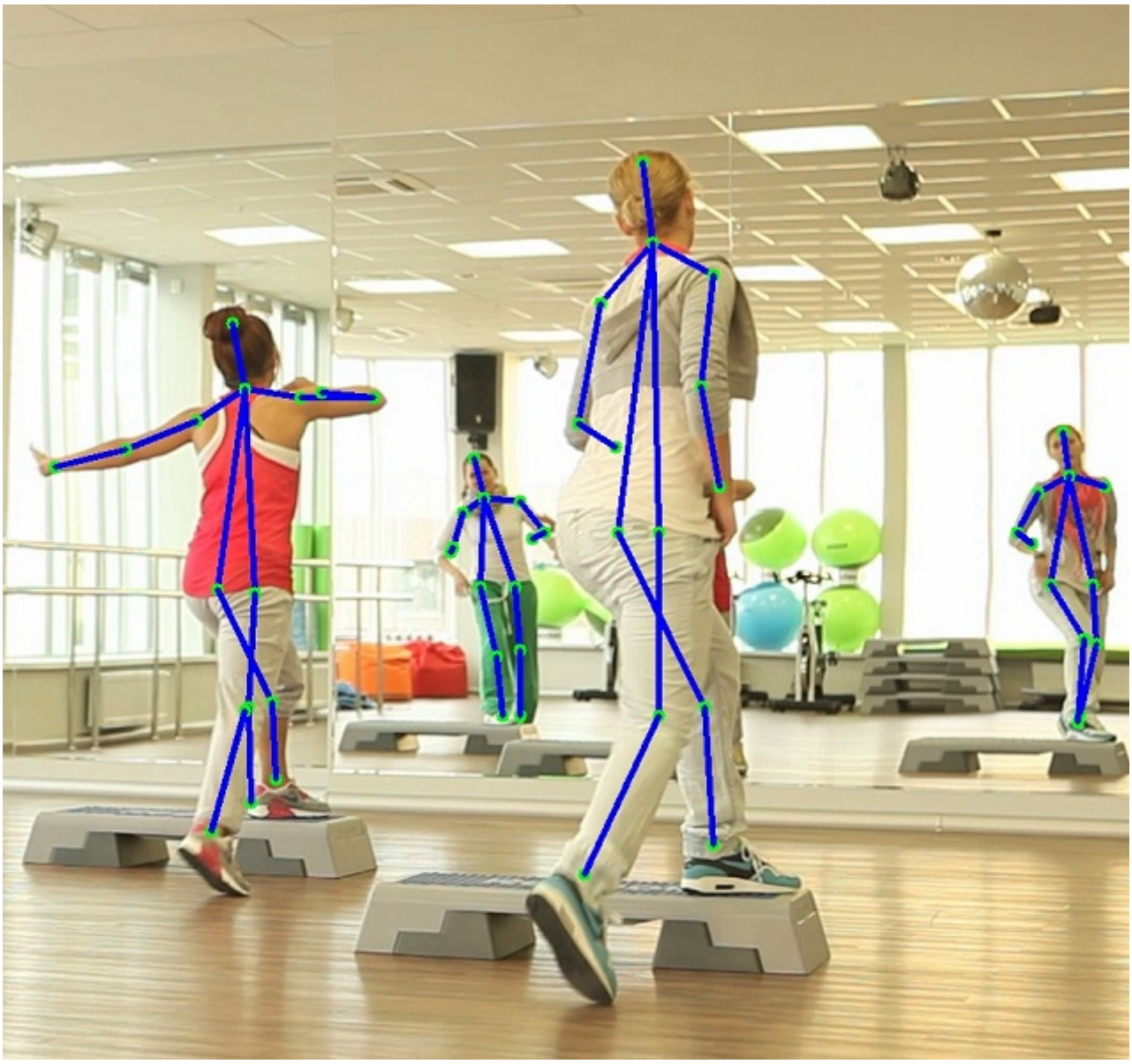
Output is a `OpenPoseResult`.

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 5: **openpose result image**



Quick Function Reference

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

Table 36: **Quick Function Reference**

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

Table 36: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
`OpenPoseResult`	run	const cv::Mat & image
std::vector< `OpenPoseResult` >	run	void
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 37: **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 38: **run Arguments**

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

Returns

`OpenPoseResult`.

run

Prototype

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

getInputWidth

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

Prototype

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

Returns

InputWidth of the openpose network

getInputHeight

Function to get InputHeigth of the openpose network (input image rows).

Prototype

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

Returns

InputHeight of the openpose network.

get_input_batch

Prototype

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

vitis::ai::PoseDetect

Base class for detecting a pose from a input image (cv::Mat).

Input an image (cv::Mat).

***Note*:** Support detect a signle 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 6: **pose detect image**

Quick Function Reference

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

Table 39: **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 40: **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 cols).

Prototype

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

Returns

InputWidth of the `PoseDetect` network.

getInputHeight

Function to get InputHeigth 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 differnt. 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 41: **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 42: **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 pedestrian in the input image (cv::Mat).

Input is an image (cv::Mat).

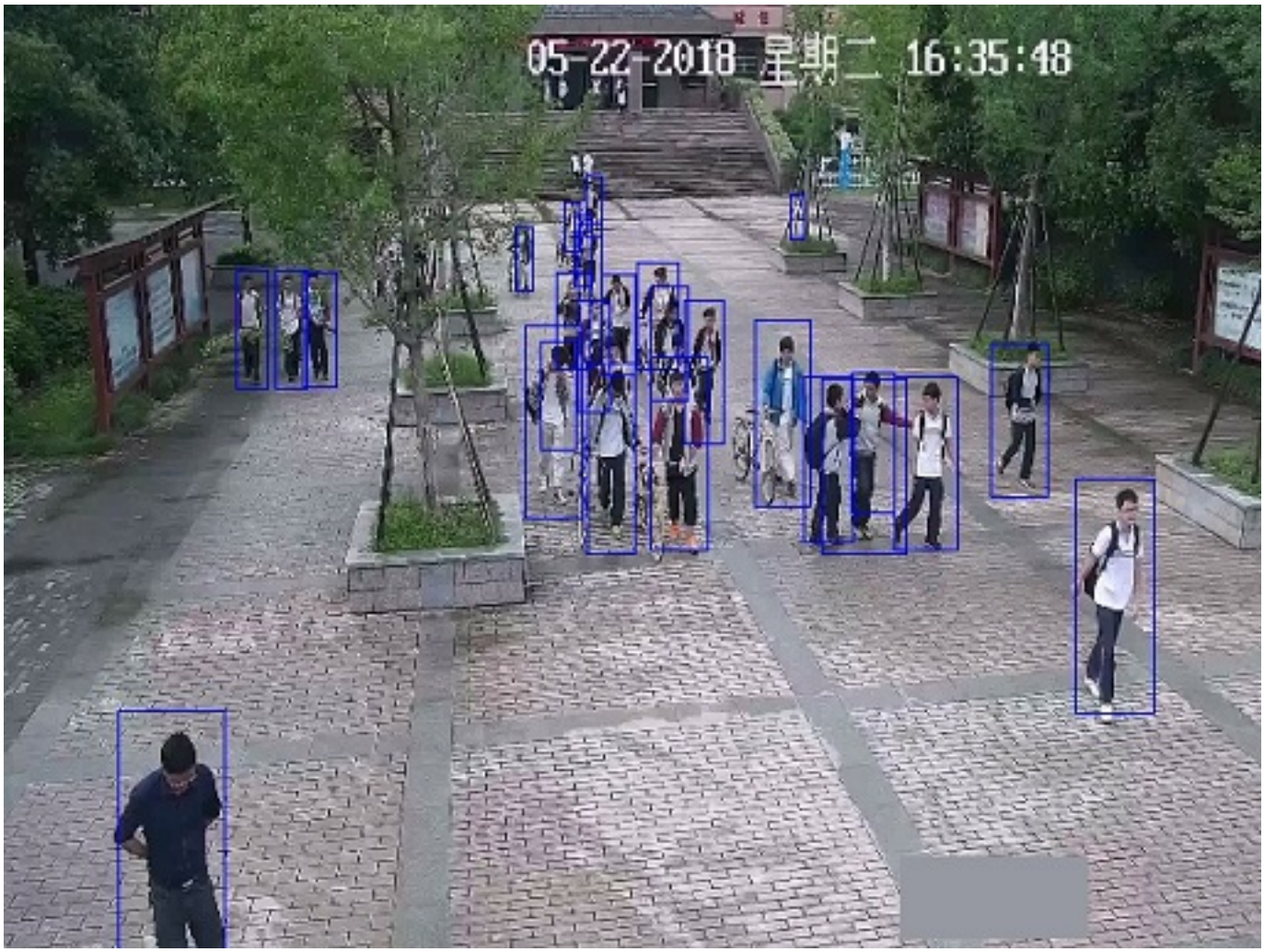
Output is position and score of pedestrian in the input image.

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 7: **result image**



Quick Function Reference

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

Table 43: **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	void

Table 43: **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 44: **create Arguments**

Type	Name	Description
const std::string &	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 45: **run Arguments**

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

Returns

A Struct of `RefineDetResult`.

run

Prototype

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

getInputWidth

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

Prototype

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

Returns

InputWidth of the refinedet network

getInputHeight

Function to get InputHeigth of the refinedet network (input image rows).

Prototype

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

Returns

InputHeight of the refinedet network.

get_input_batch

Prototype

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

vitis::ai::RefineDetPostProcess

Class of the refinedet post-process, it will initialize the parameters once instead of compute them every time when the program execute.

Quick Function Reference

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

Table 46: **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
`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 47: **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 `RefineDetPostProcess`.

refine_det_post_process

Run refinedet post-process.

Prototype

```
RefineDetResult refine_det_post_process(unsigned int idx)=0;
```

Returns

The 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 48: **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	void
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 49: **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 `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 50: **run Arguments**

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

Returns

`ReidResult`.

run

Prototype

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

getInputWidth

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

Prototype

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

Returns

InputWidth of the reid network

getInputHeight

Function to get InputHeigth of the reid network (input image rows).

Prototype

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

Returns

InputHeight of the reid network.

get_input_batch

Prototype

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

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 maked 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);
}
```

Quick Function Reference

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

Table 51: **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
std::vector< `RoadLineResult` >	run	void

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 52: **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 cols).

Prototype

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

Returns

InputWidth of the lanedetect network.

getInputHeight

Function to get InputHight of the lanedetect network (input image rows).

Prototype

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

Returns

InputHeight of the lanedetect network.

get_input_batch

Prototype

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

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 53: **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

Prototype

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

vitis::ai::RoadLinePostProcess

Class of the roadline post-process, it will initialize the parameters once instead of compute them every time when the program execute.

Quick Function Reference

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

Table 54: **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

Table 54: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
`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 55: **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 `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`.

vitis::ai::Segmentation

Base class for `Segmentation`.

Declaration `Segmentation` Network num 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 8: **segmentation visulization result image**

Quick Function Reference

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

Table 56: **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
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 57: **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 cols).

Prototype

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

Returns

InputWidth of the segmentation network.

getInputHeight

Function to get InputHight 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 differnt. 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 Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run_8UC1` function arguments.

Table 58: **run_8UC1 Arguments**

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

Returns

a result include 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 Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run_8UC1` function arguments.

Table 59: **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 Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run_8UC3` function arguments.

Table 60: **run_8UC3 Arguments**

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

Returns

a result 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 Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run_8UC3` function arguments.

Table 61: **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::Segmentation8UC1

The Class of `Segmentation8UC1`, this class run function 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 62: **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

Table 62: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
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 63: **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 cols).

Prototype

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

Returns

InputWidth of the segmentation network.

getInputHeight

Function to get InputHight of the segmentation network (input image cols).

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 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 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 64: **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 Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run` function arguments.

Table 65: **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 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 66: **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 67: **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 cols).

Prototype

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

Returns

InputWidth of the segmentation network.

getInputHeight

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

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 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 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 68: **run Arguments**

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

Returns

`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_8UC3 of the Reuslt's segmentation.

Prototype

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

Parameters

The following table lists the `run` function arguments.

Table 69: **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).

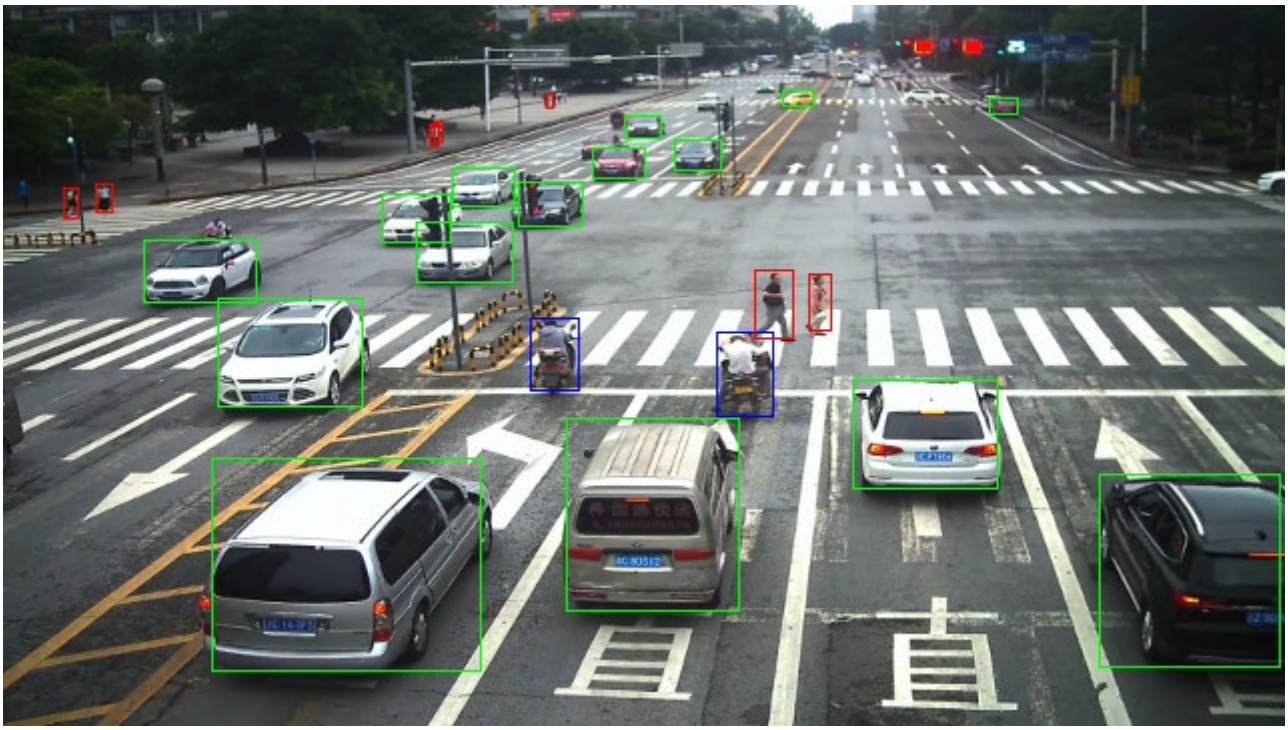
Output is a struct of detection results, named `SSDResult`.

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 9: **detection result**



Quick Function Reference

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

Table 70: **Quick Function Reference**

Type	Name	Arguments
std::unique_ptr< `SSD` >	create	const std::string & model_name bool need_preprocess
`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 71: **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.

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 72: **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 73: **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 cols).

Prototype

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

Returns

InputWidth of the `SSD` network.

getInputHeight

Function to get InputHeigth 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 differnt. 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 will initialize the parameters once instead of compute them every time when the program execute.

Quick Function Reference

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

Table 74: **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
`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 75: **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 post-processing of function of the ssd network.

Prototype

```
            SSDResult ssd_post_process(unsigned int idx)=0;
```

Returns

The struct of `SSDResult`.

---

vitis::ai::YOLOv2

Base class for detecting objects in the input image(cv::Mat). Input is an image(cv::Mat). Output is 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 76: **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	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 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 `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 78: **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 > &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 79: **run Arguments**

Type	Name	Description
Commented parameter images does not exist in function run.	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 cols).

Prototype

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

Returns

InputWidth of the `YOLOv2` network

getInputHeight

Function to get InputHeigth 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 differnt. 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 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 10: **out image**

Quick Function Reference

The following table lists all the functions defined in the `vitis::ai::YOLOv3` class:

Table 80: **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	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 81: **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 cols).

Prototype

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

Returns

InputWidth of the `YOLOv3` network

getInputHeight

Function to get InputHeigth 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 82: **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 > &image)=0;
```

Parameters

The following table lists the `run` function arguments.

Table 83: **run Arguments**

Type	Name	Description
Commented parameter images does not exist in function run.	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 differnt. 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::YOLOv3Result::BoundingBox

vitis::ai::ClassificationResult

Struct of the result with the classification network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< Score > scores
} vitis::ai::ClassificationResult;
```

Table 84: **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 configutation file.

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 85: **Structure vitis::ai::ClassificationResult::Score member description**

Member	Description
index	Result's index in ImageNet.
score	Confidence of this category.

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 86: **Structure vitis::ai::FaceDetectResult member description**

Member	Description
width	Width of a input image.
height	Height of a 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 87: **Structure vitis::ai::FaceDetectResult::BoundingBox member description**

Member	Description
x	x-coordinate , x is normalized relative to the input image cols ,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 cols , 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::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 88: **Structure vitis::ai::FaceLandmarkResult member description**

Member	Description
points	Five key points coordinate, this array of <x,y> has 5 elements ,x / y is normalized relative to width / height, the value range from 0 to 1.

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 89: **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 90: **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 91: **Structure vitis::ai::OpenPoseResult::PosePoint member description**

Member	Description
type	Point type. • `1` : "valid" • `3` : "invalid"
point	Coordinate point.

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 92: **Structure vitis::ai::PoseDetectResult member description**

Member	Description
Point	A coordinate point.

Table 92: **Structure vitis::ai::PoseDetectResult member description** *(cont'd)*

Member	Description
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 93: **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::RefineDetResult

Struct of the result with the refinedet network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::RefineDetResult;
```

Table 94: **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 a object coordinate and confidence.

Declaration

```
typedef struct
{
  float x,
  float y,
  float width,
  float height,
  float score
} vitis::ai::RefineDetResult::BoundingBox;
```

Table 95: **Structure vitis::ai::RefineDetResult::BoundingBox member description**

Member	Description
x	x-coordinate , x is normalized relative to the input image cols ,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	body width , width is normalized relative to the input image cols , the value range from 0 to 1.
height	body height , heigth is normalized relative to the input image rows , the value range from 0 to 1.
score	body detection confidence, the value range 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 96: **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::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 97: **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 98: **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::SegmentationResult

Struct of the result returned by the segementation 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 99: **Structure vitis::ai::SegmentationResult member description**

Member	Description
width	
height	
segmentation	

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 100: **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 101: **Structure vitis::ai::SSDResult::BoundingBox member description**

Member	Description
label	`Classification`.
score	Confidence.
x	x-coordinate, x is normalized relative to the input image cols ,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 cols ,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::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 102: **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 cols ,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 cols ,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.
angle	the angle between the target vehicle and ourself.

vitis::ai::YOLOv2Result

@brie Struct of the result returned by the yolov2 network.

Declaration

```
typedef struct
{
  int width,
  int height,
  std::vector< BoundingBox > bboxes
} vitis::ai::YOLOv2Result;
```

Table 103: **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 104: **Structure vitis::ai::YOLOv2Result::BoundingBox member description**

Member	Description
label	classification.
score	confidence, the range from 0 to 1.
x	x-coordinate, x is normalized relative to the input image cols, its value range from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows, its value range from 0 to 1.
width	width, width is normalized relative to the input image cols, its value from 0 to 1.
height	height, height is normalized relative to the input image rows, its value range 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 105: **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

@Brief Struct of detection result with a object

Declaration

```
typedef struct
{
  int label,
  float score,
  float x,
  float y,
  float width,
  float height
} vitis::ai::YOLOv3Result::BoundingBox;
```

Table 106: **Structure vitis::ai::YOLOv3Result::BoundingBox member description**

Member	Description
label	classification.
score	confidence, the range from 0 to 1.
x	x-coordinate, x is normalized relative to the input image cols, its value range from 0 to 1.
y	y-coordinate, y is normalized relative to the input image rows, its value range from 0 to 1.
width	width, width is normalized relative to the input image cols, its value from 0 to 1.
height	height, height is normalized relative to the input image rows, its value range from 0 to 1.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

Please Read: Important Legal Notices

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright