Ingenic® Neural Network Inference Framework Venus

Programming Manual

Date: Jan. 2022

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

Ingenic®

Neural Network Inference Framework Programming Manual

Release history

Date	Author	Revision	Change
Jan.2022	Nancy	1.0.1	First release
Feb.2022	Monday	1.0.2	1、Add image preprocessing class sample

Disclaimer

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

CONTENTS

1 Overview of Venus

Venus is a lightweight, efficient, easy-to-expand and high-performance netural network inference framework. It supports 8bit/4bit/2bit quantitative inference, 8bit post-quantitative inference and mixed precision inference. It supports several convolutions and operators, such as common convolution, Depthwise convolution, pooling, activation, full connection, lstm, concat and other operators, it also supports multi-input and branching networks, implements high-performance NPU accelerator and SIMD512 acceleration engine based on AIE platform. The bottom operator is deeply optimized to make full use of the computing performance of AIE platform. Through efficient and compact memory management structure design, memory reuse between nodes without computing dependence is achieved, memory resource consumption is greatly reduced, and cross-model memory reuse is supported to further reduce memory consumption and improve memory utilization. Venus supports high-speed image processing operations such as hardware-based image matting, scaling, rotation, affine transformation, perspective transformation and color space conversion, which are currently mainly used in T40, T02 and X2500 chip platforms.

2 API Introduction

2.1 Data Structure

- **Tensor**
  - Definition

    Represents multidimensional tensor data, the main data emission format is NHWC format
  - Interface

shape	get tensor shape information [batch, height, width, channel]
reshape	change tensor shape information based on input shape
mudata	gets a readable and writable pointer to the specified type of tensor data.
data	gets a read-only pointer to the specified type of tensor data

- **shape_t**
  - Definition

    Represents the shape information of tensor
  - Attribute

batch	batch of four-dimensional tensor
height	tensor's height
width	tensor's width
channel	tensor's channel

- **Bbox_t**
  - Definition

    A coordinate frame representing the target
  - Attribute

x0	target frame’s top left coordinate
y0	target frame’s top left coordinate
x1	target frame’s lower right coordinate
y1	target frame’s lower right coordinate

- **ObjBbox_t**
  - Definition

    Coordinate frame, corresponding category and confidence level representing the output of target detection network.

- Attribute

box	coordinate frame of the target
score	the confidence level for the target box
class_id	category of target

- **PaddingType**
  - Definition

    Type of image boundary filling
  - Attribute

NONE	no filling
BOTTOM_RIGHT	bottom right filling
SYMMETRY	symmetrical filling around

- **AddressLocate**
  - Definition

    Enter the source address of the pointer
  - Attribute

NMEM_VIRTUAL	pointer is a virtual address from nmem
RMEM_PHYSICAL	the pointer is the physical address from rmem

- **ChannelLayout**
  - Definition

    Image format
  - Attribute

NONE	defaults to null
NV12	NV12 image format
BGRA	BGRA image format
RGBA	RGBA image format
ARGB	ARGB image format
RGB	RGB image format

- **BsPadType**
  - Definition

    Type of image boundary fill
  - Attribute

NONE	no filling
BOTTOM_RIGHT	bottom right filling
SYMMETRY	symmetrical filling around

- **BsBaseParam**
  - Definition

    Image Processing Basic Parameters
  - Attribute

in_layout	input image format
out_layout	output image format
coef_off_enable	whether to use default conversion parameters
coef	nv12 to BGRA
offset	nv12 to BGRA pixel offset

- **BsExtendParam**
  - Definition

    Image processing parameters
  - Attribute

pad_type	the type of image boundary filling (BsPadType)

pad_val boundary pixel fill value (BGR)

- **BsCommonParam**
  - Definition
    Image processing calls common interface parameters
  - Attribute

input_height	height of the input image
input_width	width of the input image
input_line_stride	line spacing of the input image

2.2 Forward Inference API

- **venus_init**
  - Function description
    Initialize the running environment, return status code
  - Interface definition
    int venus_ init(void);
  - Parameter list
    None
  - Return value
    0 represents success, otherwise failure.
- **venus_deinit**
  - Function description
    Release the running environment and return the status code
  - Interface definition
    int venus_deinit();
  - Parameter list
    None
  - Return value
    0 represents success, otherwise failure.
- **BaseNet**
  - Definition
    Inference Engine Base Class for Venus Framework
  - load_model
    - Function description
      Loading model parameters from memory or from a file
    - Interface definition
      *int load_model(*const *char* model_path, bool memory_model=false, int start_off=0);*
    - Parameter list

model_path	When memory_model is True, model_path represents the pointer to the memory area of the model parameter.When memory_model is False, model_path represents the file name of the model file.
memory_model	Indicates whether to load the model from the memory. When it is set to True, it means to load the model from the memory. When it is set to False, it means to load the model from the file. It is set to False by default. Currently, only false is supported.
start_off	From model_path, it indicates the number of bytes skipped by

the first address of file or memory specified by path. Used for multi model packaging and loading.

- Return value

  0 represents success, otherwise failure.

- **get_forward_memory_size**
  - Function description

    Get the size of memory required for the forward inference phase of the model
  - Interface definition

    int get_forward_memory_*size(size_t &memory_size);*
  - Parameter list

    memory_size variable reference holding memory size
  - Return value

    0 represents success, otherwise failure.
- **int set_forward_memory**
  - Function description

    When the memory sharing mode is external, set the first address of the forward reasoning memory through this interface
  - Interface definition

    *int set_forward_memory(*void **memory);*
  - Parameter list

    memory the first address of memory
  - Return value

    0 represents success, otherwise failure.
- **get_input**
  - Function description

    The Tensor of the network input node is obtained by the sequence number of the network input node. If the network has more than one input, the interface can be called multiple times to get it.
  - Interface definition

    std::unique_ptr<Tensor> *get_input(int index);*
  - Parameter list

    index the sequence number of the network input node
  - Return value

    Returns the smart pointer to the network input node Tensor
- **get_input_by_name**
  - Function description

    The Tensor of the network input node is obtained by the name of the network input node. If the network has more than one input, the interface can be called multiple times to get it.
  - Interface definition

    std::unique_ptr<Tensor> get*_input_by_name(std::string &name);*
  - Parameter list

    name the name of the network input node
  - Return value

    Returns the smart pointer to the network input node Tensor
- **get_output**
  - Function description

    The Tensor of the network output node is obtained by the sequence number of the network output node, which can be called multiple times if the network has multiple outputs.
  - Interface definition

std::unique_ptr<const Tensor> *get_output(int index);*

- Parameter list

  index the sequence number of the network output node

- Return value

  Returns the smart pointer to the network output node tensor

- **get_output_by_name**
  - Function description

    Obtain the tensor of the network output node through the name of the network output node. If the network has multiple outputs, you can call the interface for multiple times.
  - Interface definition

    std::unique_ptr<const Tensor> get_output_by_name(std::string &name);
  - Parameter list

    name the name of network output node
  - Return value

    Returns the smart pointer to the network output node tensor
- **run**
  - Function description

    Perform model forward reasoning and return status codes.
  - Interface definition

    int run();
  - Parameter list

    None
  - Return value

    0 represents success, otherwise failure.

- **net_create**
  - Function description

    Create inference engine handle, return engine handle
  - Interface definition

    std::unique_ptr<BaseNet> net_create(TensorFormat input_data_fmt=TensorFormat::NHWC, ShareMemoryMode smem_mode=ShareMemoryMode::DEFAULT);
  - Parameter list

    input_data_fmt representing the type of input image, has two options: NHWC and NV12.

    smem_mode which indicates how memory is shared. DEFAULT, SHARE_ ONE_ THREAD, SET_ FROM_ The EXTERNAL options represent three memory sharing modes, one for the model, one for multiple models in the same thread, and one for users exploiting inferential memory from outside.
  - Return value

    Returns the inference engine handle. NULL represents failure, otherwise success.

2.3 Tools API

- **Image Preprocessing API**
  - **warp_resize**
    - Function description

      Scale the input image to return the scaled image
    - Interface definition

      *int warp_resize(const Tensor &input, Tensor &output, BsExtendParam *param)*

◆ Parameter list

input input image
output output image
param image scaling parameters

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by param->padtype.

➢ When the input is in NV12 format, the width and height of the input must be multiple of 2. When the input is in BGRA, RGBA, ARGB formats, the input width is required to be a multiple of 2, and the output format is consistent with the input format.

➢ When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.

■ **crop_resize**

◆ Function description

The input image is scaled by the boxes coordinate set, and the scaled image set is returned.

◆ Interface definition

*int crop_resize(const Tensor &input, std::vector<Tensor> &output, std::vector<Bbox_t>&boxes,BsExtendParam *param)*

◆ Parameter list

Input input image
output output image collection
boxes the coordinates of the boxes that need to be scaled and the set of widths and heights.
param image scaling parameters

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ When the aspect ratio of the picked ROI image and the output image is not consistent, you can select the output direct zoom image, the equal-scale zoom image

filled around, and the equal-scale zoom image filled at the bottom right by param->padtype.

- When the input is in NV12 format, that is, Bbox_t. The x0, y0, x1, Y1 in the data structure must all be even numbers; When the input is in BGRA, RGBA, ARGB formats, the input width (x1-x0) is required to be a multiple of 2, and the output format is consistent with the input format.
- When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment restriction for the width and height of the output.

■ **warp_affine**

◆ Function description

Perform affine transformation on input image based on affine transformation matrix.

◆ Interface definition

int warp_affine(const Tensor *&input, Tensor &output,Tensor &matrix, BsExtendParam *param)*

◆ Parameter list

input	input image
Output	output image
matrix	set of affine transformation matrices
param	parameters of the image affine transformation

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

- When the input is in NV12 format, the width and height of the input must be multiple of 2. When the input is in BGRA, RGBA, ARGB formats, the input width is required to be a multiple of 2, and the output format is consistent with the input format.
- When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.

■ **crop_affine**

◆ Function description

Carry out a matting operation on the original map according to the input boxes, and then perform an affine transformation on the matted graph according to its affine transformation matrix.

◆ Interface definition

int crop_affine(const Tensor &input,std::vector<Tensor> &output, std::vector<Tensor> &matrix, std*::vector<Bbox_t>&boxes,BsExtendParam *param)*

◆ Parameter list

input	input image
output	output image collection
matrix	set of affine transformation matrices

boxes the coordinates of the box requiring an affine transformation and a set of widths and heights

param parameters of the image affine transformation

◆ Parameter specification

Parameter name	Parameter name	Parameter name	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ When the input is in NV12 format, that is, Bbox_t The x0, y0, x1, Y1 in the data structure must all be even numbers; When the input is in BGRA, RGBA, ARGB formats, the input width (x1-x0) is required to be a multiple of 2, and the output format is consistent with the input format.

➢ When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.

■ **warp_perspective**

◆ Function description

Perform perspective transformation on the input image (NV12) based on the perspective transformation matrix

◆ Interface definition

*int warp_perspective(const Tensor &input, Tensor &output,Tensor &matrix, BsExtendParam *param)*

◆ Parameter list

input input image

output output image

matrix set of affine transformation matrices

param parameters of the image perspective transformation

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ When the input is in NV12 format, that is, Bbox_t The x0, y0, x1, Y1 in the data structure must all be even numbers; When the input is in BGRA, RGBA, ARGB formats, the input width (x1-x0) is required to be a multiple of 2, and the output format is consistent with the input format.

➢ When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.

■ **crop_perspective**

◆ Function description

Perform a cutout operation on the original drawing based on the boxes input, and then a perspective transformation on the finished drawing based on their respective perspective transformation matrices

◆ Interface definition

int crop_perspective(const Tensor &input,std::vector<Tensor> &output, std::vector<Tensor> &matrix*, std::vector<Bbox_t>&boxes,BsExtendParam *param)*

◆ Parameter list

input	input image
output	output image set
matrix	set of affine transformation matrices
boxes	the coordinates of the box requiring an affine transformation and a set of widths and heights
param	parameters of the image perspective transformation

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ When the input is in NV12 format, that is, Bbox_t The x0, y0, x1, Y1 in the data structure must all be even numbers; When the input is in BGRA, RGBA, ARGB formats, the input width (x1-x0) is required to be a multiple of 2, and the output format is consistent with the input format.

➢ When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.

■ **common_resize**

◆ Function description

The input image (NV12) is scaled to return the scaled image data (BGRA/RGBA/ARGB/NV12).

◆ Interface definition

common_resize(const void *input*, Tensor &output, AddressLocate input_locate,BsCommonParam *param)*

◆ Parameter list

input	input data pointer
output	output image
input_locate	address properties of the input image
param	image scaling parameters

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Width height	2x2 ~ 1024x2048

		even alignment	
output	BGRA、RGBA、ARGB、NV12	See Notes	2x2 ~ 1024x2048

- Return value
  0 represents success, otherwise failure.
- Notes
  - When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.
  - When the output is in NV12 format, the width and height of the output should be multiple of 2. When the output is in BGRA, RGBA, ARGB formats, there is no alignment requirement for the width and height of the output.
  - The memory of the input image must come from memory space (rmem/nmem) with a continuous physical address. When "input_locate" is set to NMEM_VIRTUAL, so the input must be a virtual address on nmem (a pointer obtained from Tensor's data/mudata interface, or a memory pointer requested through nmem_memalign); When "input_locate" is set to RMEM_PHYSICAL, the input must be a physical address on rmem (the physical address of the video stream obtained through the relevant interface of the isvp platform).

■ **similar_transform**

- Function description
  The similarity transformation matrix is calculated based on the input and output coordinates.
- Interface definition
  similar_transform(Tensor &input_src, Tensor &input_dst,Tesnor &output)
- Parameter list
  input_src enter matrix coordinates
  Input_dst output matrix coordinates
  output similarity transformation matrix parameters
- Parameter specification

Parameter name	data format
input_src	1*3*2*1
input_dst	1*3*2*1
output	1*3*3*1

- Return value
  0 represents success, otherwise failure.
- Notes
  - The call function generates a 3*3 matrix parameter for subsequent similarity transformation calculations

■ **get_affine_transform**

- Function description
  The affine transformation matrix is calculated based on the input and output coordinates.
- Interface definition
  get_affine_transform(Tensor &input_src, Tensor &input_dst,Tesnor &output)
- Parameter list
  input_src coordinates of the input matrix
  Input_dst coordinates of the output matrix

output parameters of the similarity transformation matrix

- Parameter specification

Parameter name	data format
input_src	1*3*2*1
input_dst	1*3*2*1
output	1*3*2*1

- Return value

  0 represents success, otherwise failure.

- Notes

  - Call function to generate a 3*2 matrix parameter for subsequent affine transformation calculations

（The following is the old version API, which will not be updated later. It is recommended to use the API described above）

■ **warp_covert_nv2bgr**

- Function description

  Perform color space conversion on the input image (NV12), returning the converted image (BGRA)

- Interface definition

  int warp_covert_nv2bgr(const Tensor *&input, Tensor &output)*

- Parameter list

  input input image
  output output image

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.

- Notes

  - The formula of YUV to BGR (bt601) is as follows:

    B = 1.164(Y - 16) + 2.018(U - 128)
    G = 1.164(Y - 16) - 0.391(U - 128) - 0.813(V - 128)
    R = 1.164(Y - 16) + 1.596(V - 128)

  - The conversion parameters of other formats (bt709 / bt656, etc.) can be configured.

■ **warp_resize_bgra**

- Function description

  Scale the input image (BGRA) to return the scaled image (BGRA)

- Interface definition

  int warp_resize_bgra(const Tensor &input， Tensor &output, PaddingType padtype,uint8_t padval*)*

- Parameter list

  input input image
  output output image
  padtype boundary pixel filling

padval boundary pixel fill value (BGR)

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.

- Notes

  ➢ When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.

■ **warp_resize_nv12**

- Function description

  Scale the input image (NV12) to return the scaled image (BGRA/NV12)

- Interface definition

  int warp_resize_nv12(const Tensor &input, Tensor &output, bool cvtbgra,PaddingType padtype*, uin8_t padval)*

- Parameter list

  input input image
  output output image
  cvtbgra convert output image to bgra
  padtype boundary pixel filling method
  padval pixel fill (BGR)

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	See Notes	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.

- Notes

  ➢ When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.

  ➢ When the output is in NV12 format, the width and height of the output should be multiple of 2. When output is in BGRA format, there is no alignment requirement for output width and height.

■ **crop_resize_bgra**

- Function description

  The input image (BGRA) is scaled based on the input boxes coordinate set, and the scaled image (BGRA) set is returned.

- Interface definition

  int crop_resize_bgra(const Tensor *&input, std::vector<Tensor> &output,*

std::vector<Bbox_t>&boxes, PaddingType padtype, uint8_t padval)

- Parameter list

input	input image
output	output image set
boxes	the coordinates of the boxes that need to be scaled and the set of widths and heights.
padtype	boundary pixel filling
padval	boundary pixel fill value (BGR)

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

- Return value

0 represents success, otherwise failure.

- Notes

➢ When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.

➢ The width of each box in boxes must be a multiple of 2, or Bbox_the (x1-x0) of T is a multiple of 2.

■ **crop_resize_nv12**

- Function description

The input image (NV12) is scaled by the boxes coordinate set input, and the scaled image (NV12) set is returned.

- Interface definition

int crop_resize_nv12(const Tensor *&input, std::vector<Tensor> &output, std::vector<Bboxt>&boxes*, *PaddingType padtype, uint8_t padval)*

- Parameter list

input	input image
output	output image collection.
boxes	the coordinates of the boxes that need to be scaled and the set of widths and heights.
padtype	boundary pixel filling.
padval	boundary pixel fill value (BGR)

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

- Return value

0 represents success, otherwise failure.

- Notes
  - When the stripped ROI image and the output image do not have the same aspect ratio, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.
  - The upper left and lower right coordinates of each box in boxes must be even (divisible by 2), that is, Bbox_t The x0, y0, x1, Y1 in the data structure must all be even numbers.

■ **common_resize_nv12**

- Function description

  The input image (NV12) is scaled to return the scaled image data (BGRA/NV12).
- Interface definition

  common_resize_nv12(const void **input, Tensor &output, int img_h, int img_w, int line_stride, bool cvtbgra, PaddingType padtype, uint8_t padval, AddressLocate input_*locate*)*
- Parameter list

input	input data pointer
output	output image
img_h	input image height
img_w	input image width
line_stride	enter image line spacing
cvtbgra	whether to convert the output image to BGRA
padtype	boundary pixel filling.
input_locate	enter image address properties
padval	boundary pixel fill value (BGR)

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.
- Notes
  - When the aspect ratio of the input and output images is inconsistent, you can select the output direct zoom image, the equal-scale zoom image filled around, and the equal-scale zoom image filled at the bottom right by padtype.
  - When the output is in NV12 format, the width and height of the output should be multiple of 2. When output is in BGRA format, there is no alignment requirement for output width and height.
  - The memory of the input image must come from memory space (rmem/nmem) with a continuous physical address. When “input_locate” is set to NMEM_VIRTUAL, so the input must be a virtual address on nmem (a pointer obtained from Tensor's data/mudata interface, or a memory pointer requested through nmem_memalign); When “input_locate” is set to RMEM_PHYSICAL, the input must be a physical address on rmem (the physical address of the video stream obtained through the relevant interface of the isvp platform).

■ **wrap_affine_bgra**

◆ Function description

Perform affine transformation on input image (BGRA) based on affine transformation matrix

◆ Interface definition

int wrap_affine_bgra(const Tensor *&input, Tensor &output, Tensor &matrix)*

◆ Parameter list

input	input image
output	output image
matrix	affine transformation matrix

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ Default fill beyond bounds 0.

■ **wrap_affine_nv12**

◆ Function description

Perform affine transformation on input image (NV12) based on affine transformation matrix.

◆ Interface definition

int wrap_affine_nv12(const Tensor &input, Tensor &output, Tensor &matrix, bool cvtbgra*)*

◆ Parameter list

input	input image
output	output image
matrix	affine transformation matrix
cvtbgra	output bgra image

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ Default Fill Out of Boundary to Full Black.

■ **crop_affine_bgra**

◆ Function description

Perform a cutout on the original (BGRA) according to the input boxes, and then perform an affine transformation on the finished graph according to its affine transformation matrix.

- Interface definition

  int crop_affine_bgra(const Tensor *&input, std::vector<Tensor> &output, std::vector<Tensor>* &matrixes*, std::vector<Bbox_t>&boxes)*

- Parameter list

input	input image
output	a collection of output images
matrixes	a collection of affine transformation matrices.
Boxes	Boxes, the coordinates of the box that need an affine transformation and a set of width and height.

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.

- Notes

  ➢ The width of each box in boxes must be a multiple of 2, that is, bbox_ (x1-x0) of T is a multiple of 2.

■ **crop_affine_nv12**

- Function description

  Match the original graph (NV12) according to the input boxes, then affine transform the finished graph according to their affine transformation matrix.

- Interface definition

  int crop_affine_nv12(const Tensor *&input,std::vector<Tensor> &output, std::vector<Tensor>* &matrixes*, bool cvtbgra, std::vector<Bbox_t>&boxes)*

- Parameter list

input	input image
output	output image collection
matrixes	a collection of affine transformation matrices.
boxes	the coordinates of the box that need an affine transformation and a set of width and height.
cvtbgra	whether to output images in BGRA format.

- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.

- Notes
  - The upper left and lower right coordinates of each box in boxes must be even (divisible by 2), that is, x0, y0, x1, Y1 in the Bbox_t data structure must be even.

■ **wrap_perspective_bgra**

- Function description

  Perform perspective transformation on the input image (BGRA) based on the perspective transformation matrix.
- Interface definition

  int wrap_perspective_bgra(const Tensor &input, Tensor &output, Tensor &matrix)
- Parameter list

  input input image
  output output image
  matrix perspective transformation matrix
- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.
- Notes
  - Default fill beyond bounds 0

■ **wrap_perspective_nv12**

- Function description

  Performs a perspective transformation operation on the input image (NV12) according to the perspective transformation matrix.
- Interface definition

  int wrap_perspective_*bgra(const Tensor &input, Tensor &output, Tensor &matrix, bool cvtbgra)*
- Parameter list

  input input image
  output output image
  matrix perspective transformation matrix
  cvtbgra whether to output images in BGRA format.
- Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048

- Return value

  0 represents success, otherwise failure.
- Notes
  - The default fill beyond the boundary is all black.

■ **crop_perspective_bgra**

◆ Function description

Based on the input boxes, a cutout operation is performed on the original graph (BGRA), and then a perspective transformation is performed on the finished graph according to its respective perspective transformation matrix.

◆ Interface definition

*int crop_perspective_bgra(const Tensor &input, std::vector<Tensor> &output, std::*vector<*Tensor> &matrixes, std::vector<Bbox_t>&boxes)*

◆ Parameter list

input	input image
output	output image collection.
matrixes	a collection of affine transformation matrices.
boxes	the coordinates of the box that need an affine transformation and a set of width and height.

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	BGRA	Multiple of width 2	2x2 ~ 1024x2048
output	BGRA	-	2x2 ~ 1024x2048
boxes	-	See Notes	2x2 ~ 1024x2048

◆ Return value

0 represents success, otherwise failure.

◆ Notes

➢ The width of each box in boxes must be a multiple of 2, that is, the Bbox_t's (x1-x0) is a multiple of 2.

■ **crop_perspective_nv12**

◆ Function description

The input boxes are used to perform a cutout operation on the original (NV12), and then the cutouts are transformed into perspective according to their respective perspective transformation matrices.

◆ Interface definition

*int crop_*perspective*_nv12(const Tensor &input, std::vector<Tensor> &output, std::vector<*Tensor> *&matrixes, bool cvtbgra, std::vector<Bbox_t>&boxes)*

◆ Parameter list

input	input image
output	output image collection.
matrixes	a collection of affine transformation matrices.
boxes	the coordinates of the box that need an affine transformation and a set of width and height.
cvtbgra	whether to output images in BGRA format.

◆ Parameter specification

Parameter name	Supported image types	Alignment Requirement	resolving power
input	NV12	Multiple of width and height of 2	2x2 ~ 1024x2048
output	NV12、BGRA	Multiple of width and height of 2	2x2 ~ 1024x2048

boxes	-	See Notes	2x2 ~ 1024x2048

- Return value
  0 represents success, otherwise failure.
- NotesNotesNotesNotesNotes
  - The upper left and lower right coordinates of each box in boxes must be even (divisible by 2), that is, x0, y0, x1, Y1 in the Bbox_t data structure must be even.

Network post-processing API

nms

- Function description
  Non-maximum suppression of Bbox output from target detection network.
- Interface definition
  void nms(std::vector<ObjBbox_t> &input, std::vector<ObjBbox_t> &output, float nms_threshold = 0.3, NmsType type = NmsType::HARD_NMS)
- Parameter list

input	input coordinate box array
output	output coordinate box array
nms_threshold	threshold of NMS
type	type of NMS（HARD_NMS、SOFT_NMS）

- Return value
  None

generate_box

- Function description
  Generate candidate boxes according to anchor.
- Interface definition
  void *generate_box(std::vector<Tensor> &features, std::vector<float> &strides, std::vector<float> &anchor, std::vector<ObjBbox_t> &candidate_boxes, int* img_w, int img_h, int classes, int box_num,float box_score_threshold,* DetectorType *detector_type = DetectorType::YOLOV3)*
- Parameter list

features	input features map
strides	the downsampling multiple corresponding to the input features map
anchor	preset anchor
candidate_boxes	generated candidate box
img_w	width of network input image
img_h	the height of the network input image
classes	number of categories for target detection.
box_num	number of boxes per anchor point.
box_score_threshold	score threshold for filtering redundant boxes
detector_type	type of detector (yoov3 series or yoov5 Series)

- Return value
  None

Memory management API

nmem_malloc

- Function description
  Request a block of memory space on nmem based on the number of bytes entered, and return the corresponding first address pointer.
- Interface definition
  *void *nmem_malloc(unsigned int size)*

- Parameter list

  size the number of bytes of memory that need to be requested.
- Return value

  NULL indicates that the request failed, otherwise returns the first address pointer corresponding to the block of memory requested

- **nmem_memalign**
  - Function description

    Requests a block of memory space on nmem based on the number of bytes entered, aligns the first address to the specified number of bytes, and returns the corresponding first address pointer.
  - Interface definition

    *void *nmem_memalign(unsigned int align,unsigned int size)*
  - Parameter list

    align the first address of the memory block is aligned by alignment bytes.

    size the number of bytes of memory that need to be requested.
  - Return value

    NULL indicates that the request failed, otherwise the first address pointer corresponding to the block of memory requested is returned.
- **nmem_free**
  - Function description

    Freeing nmem memory pointed to by input pointer
  - Interface definition

    void nmem*_free(void *ptr)*
  - Parameter list

    ptr pointer to nmem memory block
  - Return value

    None
- **memcopy**
  - Function description

    Copy a block of data from the source address to the destination address.
  - Interface definition

    *void memcopy(void *dst, void *src, int n)*
  - Parameter list

    dst destination address

    src source address

    n number of bytes to copy
  - Return value

    None

Complete example：

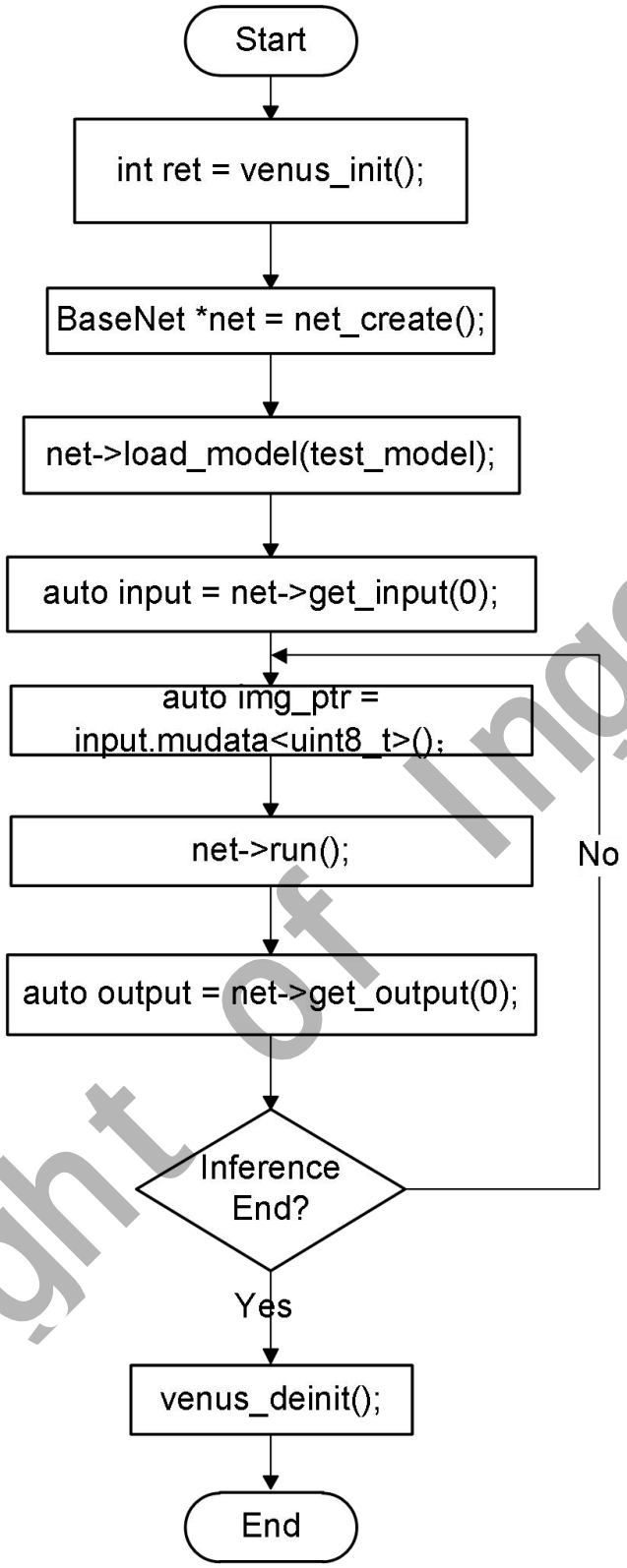
After obtaining the magik software package, please refer to the directory for details：your_root/InferenceKit/nna1/mips720-glibc229/sample/imgproc_run/

2.4 Inference process

For your convenience, Venus has a good API design that hides a lot of details that you don't need to invest time in. You only need seven simple steps to complete model forward inference using Venus on T40 and T02 chip platforms.

1. System initialization. Call venus_ Init, complete the initialization of the system environment.
2. Create a BaseNet handle. By calling net_create, this interface creates a handle to the inference engine.
3. Load the model. Load the required models for inference by calling “load_model” from the BaseNet handle.
4. Prepare the input data. Takes the input node Tensor of the model and copies the image data into a tensor by calling “get_input” from the BaseNet handle.
5. Perform reasoning. Model inference is performed by calling the BaseNet handle method, run.
6. Get results. Gets the output node Tensor of the model by calling get_output from the BaseNet handle.
7. Call venus_deinit to release the system resources.

The specific inference flow is shown in Figure 2-1.



The inference examples configuration is as follows:

1、test_model represents the file of model name.

2、The model have only one input node and one output node.

Figure 2-1 Venus Inference workflow

2.4.1 Operator set

Operator type	Reasoning bit width	remarks
Convolution	8bit/4bit/2bit	
DepthWiseConvolution	8bit/4bit/2bit	
BatchNorm	8bit/4bit/2bit	When reasoning, it is integrated into convolution
BiasAdd	8bit/4bit/2bit	When reasoning, it is integrated into convolution
FullConnected	8bit/4bit/2bit	
Pooling	12bit/10bit/8bit/4bit/2bit	MaxPool/AvgPool/GlobalAvgPool
UpSample	Float/12bit/10bit/8bit/4bit/2bit	
Relu	8bit/4bit/2bit	When reasoning, it is integrated into convolution
Relu6	8bit/4bit/2bit	When reasoning, it is integrated into convolution
PRelu	8bit/4bit/2bit	When reasoning, it is integrated into convolution
LeakyRelu	8bit/4bit/2bit	When reasoning, it is integrated into convolution
Silu	8bit/4bit/2bit	When reasoning, it is integrated into convolution
Tanh	8bit/4bit/2bit	When reasoning, it is integrated into convolution
Sigmoid	8bit/4bit/2bit	When reasoning, it is integrated into convolution
Swish	8bit	When reasoning, it is integrated into convolution
Lstm	8bit/4bit	
Concat	8bit/10bit/12bit	
Softmax	Float	
Mul	Float/12bit/10bit/8bit/4bit/2bit	
Add	12bit/10bit/8bit/4bit/2bit	
Eltwise	12bit/10bit/8bit	Min/Max/Sub
Sum	Float	
Permute	8bit	
Slice	8bit/4bit/2bit	
Padding	8bit	
Normalize	8bit/10bit/12bit	

Table 2-1 Venus Operator list

2.4.1.1 Convolution Operator Description

Input bit width / Parameter support	2bit	4bit	8bit
Weight bit width	2bit/4bit	2bit/4bit/6bit/8bit	2bit/4bit/6bit/8bit

Output bit width	2bit/4bit/8bit/ float	2bit/4bit/8bit/float	4bit/8bit/float
Kernel Size	arbitrarily	arbitrarily	arbitrarily
Stride	<=2	<=2	<=2
Dilation	support	support	support
Batch	Not Supported	support	support
Act Tpye	relu/relu6	relu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish	relu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish

Table 2-2 Convolution Operator Description

2.4.1.2 DepthWiseConvolution Operator Description

Input bit width / Parameter support	2bit	4bit	8bit
Weight bit width	2bit/4bit	2bit/4bit	2bit/4bit/6bit/8bit
Output bit width	2bit/4bit	2bit/4bit	8bit
Kernel Size	arbitrarily	arbitrarily	arbitrarily
Stride	arbitrarily	arbitrarily	<=2
Batch	support	support	support
Act Tpye	relu/relu6	relu/relu6	relu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish

Table 2-3 DepthWiseConvolution Operator Description

2.4.1.3 FullConnected Operator Description

Input bit width / Parameter support	2bit	4bit	8bit

Weight bit width	2bit/4bit/8bit	2bit/4bit/8bit	2bit/4bit/8bit
Output bit width	2bit/4bit/float	2bit/4bit/8bit/float	8bit/float
Batch	Not Supported	Not Supported	support
Act Tpye	When output bit width is float supportrelu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish, For other bit widths supportrelu/relu6	When output bit width is float supportrelu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish, For other bit widths supportrelu/relu6	relu/relu6/prelu/leaky_relu/tanh/silu/sigmoid/swish

Table 2-4 FullConnected Operator Description

2.4.1.4 Add Operator Description

Parameter support \ Input bit width	2bit	4bit	8bit	10bit/12bit
Batch	support	support	support	support
Act Tpye	None	RELU/NONE/LINEAR/RELU6/HARD_SIGMOID TANH/SWISH/SIGMOID /SILU/LEAKY_RELU	NONE/RELU/RELU6/ SWISH/TANH/SIGMOID	NONE/RELU/RELU6/ SWISH/TANH/SIGMOID

Table 2-5 Add Operator Description

2.4.1.5 Mul Operator Description

Parameter support \ Input bit width	2bit	4bit	8bit/10bit/12bit	Fp32
Batch	support	support	support	support
Broadcast type	SINGLE/CHANNEL/ELEMET/SPATIAL	SINGLE/CHANNEL /ELEMET/SPATIAL	SINGLE/CHANNEL /ELEMET/SPATIAL	CHANNEL

Table 2-6 Mul Operator Description

2.4.1.6 Pooling Operator Description

Input bit width / Parameter support	2bit	4bit	8bit	10bit/12bit
Batch	support	support	support	support
PoolType	MAX_POOL/AVG_POOL/GLOBAL_AVG_POOL	MAX_POOL/AVG_POOL/GLOBAL_AVG_POOL	MAX_POOL/AVG_POOL/GLOBAL_AVG_POOL	MAX_POOL/AVG_POOL/GLOBAL_AVG_POOL

Table 2-7 Pooling Operator Description

2.4.1.7 Upsample Operator Description

Input bit width / Parameter support	2bit	4bit	8bit	10bit/12bit	Fp32
Batch	support	support	support	support	support
UppType	FILL_ZERO/ NEARST/ PIXEEL_SHUFFLE/ BILINEAR	FILL_ZERO/ NEARST/ PIXEEL_SHUFFLE	FILL_ZERO/ NEARST/ PIXEEL_SHUFFLE	NEARST	FILL_ZERO/ NEARST

Table 2-8 Upsample Operator Description

2.4.1.8 Concat Operator Description

Input bit width / Parameter support	8bit	10bit	12bit
Batch	support	support	support

Table 2-9 Concat Operator Description

2.4.1.9 Normalize Operator Description

Input bit width / Parameter support	8bit	10bit	12bit
Batch	Not Supported	Not Supported	Not Supported

Table 2-10 Normalize Operator Description

2.4.1.10 LSTM Operator Description

Input bit width / Parameter support	4bit	8bit
Weight bit width	4bit	2bit/4bit/6bit/8bit
Output bit width	4bit	8bit
Batch	Not Supported	Not Supported

Table 2-11 LSTM Operator Description

2.4.1.11 Eltwise Operator Description

Input bit width / Parameter support	8bit	10bit	12bit
BATCH	Not Supported	Not Supported	Not Supported
EltwiseType	Sub/Max/Min	Sub/Max/Min	Sub/Max/Min

Table 2-12 Eltwise Operator Description

2.4.1.12 Sum Operator Description

Input bit width / Parameter support	float
Batch	support
AXIS	2 (in width direction)

Table 2-13 Sum Operator Description

2.4.1.13 Softmax Operator Description

Input bit width / Parameter support	float
Batch	Not Supported
AXIS	2 (in width direction)/ 3 (in channel direction)

Table 2-14 Softmax Operator Description

2.4.1.14 Embedding Operator Description

Parameter support / Input bit width	int32
Feature bit width	8bit
Output bit width	8bit
Batch	Not Supported

Table 2-15 Embedding Operator Description

2.4.1.15 Padding Operator Description

Parameter support / Input bit width	8bit
Batch	support

Table 2-16 Padding Operator Description

2.4.1.16 Slice Operator Description

Parameter support / Input bit width	2bit	4bit	8bit
Batch	support	support	support

Table 2-17 Slice Operator Description

2.4.1.17 Permute Operator Description

Parameter / Input bit width	8bit
Batch	support

Table 2-18 Permute Operator Description

2.4.2 Profiler Tool

Profiler is used to analyze the detailed running time of each layer of the network, the running time and the number of calls for each type of operator, and other related information, so that users can better optimize the network model structure. The statistical results of each layer of the network mainly include layer name, operator type, input feature size, output feature size, weight size, Stride information, Pad information, convolution expansion factor, current layer running time, current layer running time proportion, current layer GOPs and current layer bandwidth, etc. Profiler's

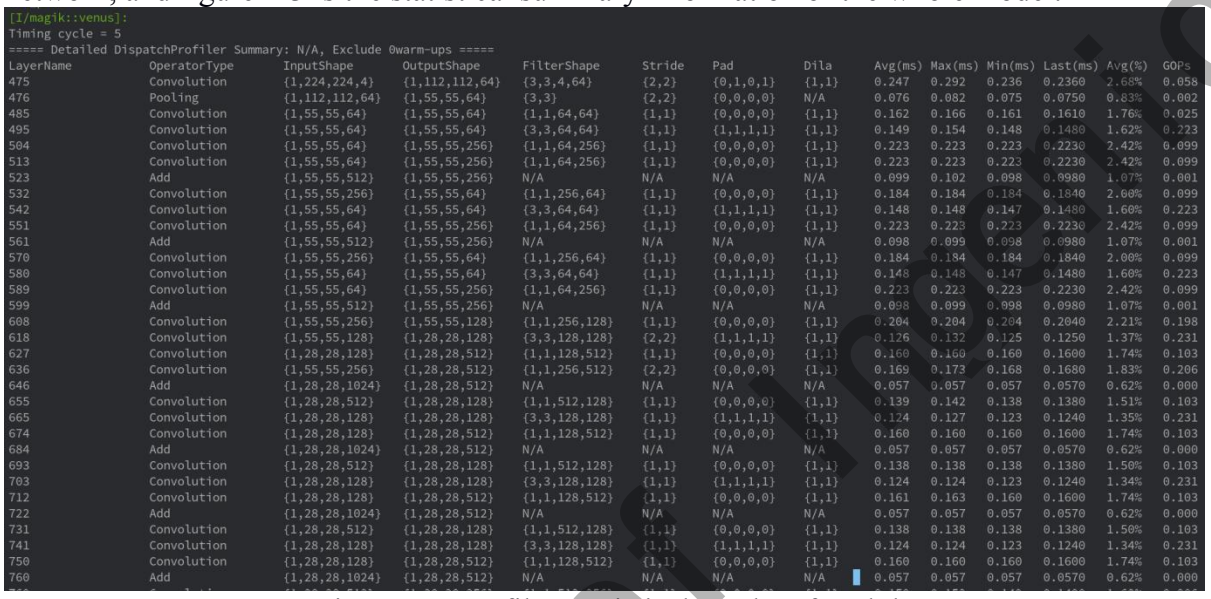
overall summary results include information such as operator type, operation time of each operator, time percentage, GOPs, bandwidth, and number of calls.

2.4.2.1 Opening method

When compiling a Venus-based executable with links, linking libvenus.p.so opens the Profiler tool to perform network performance analysis.

2.4.2.2 Use example

Instantiate the model handle to be analyzed in the main function, and then call the run interface repeatedly. After the model is executed, the profiler log information similar to the following figure will be automatically printed. Figure 2-2 is the detailed statistical information of each layer of the network, and figure 2-3 is the statistical summary information of the whole model.



```
[I/magik::venus]:
Timing cycle = 5
===== Detailed DispatchProfiler Summary: N/A, Exclude 0warm-ups =====
LayerName          OperatorType       InputShape       OutputShape      FilterShape      Stride    Pad         Dila     Avg(ms) Max(ms) Min(ms) Last(ms) Avg(%)  GOPs
475                Convolution        {1,224,224,4}    {1,112,112,64}   {3,3,4,64}       {2,2}     {0,1,0,1}   {1,1}    0.247   0.292   0.236   0.2360   2.68%   0.058
476                Pooling            {1,112,112,64}   {1,55,55,64}     {3,3}            {2,2}     {0,0,0,0}   N/A      0.076   0.082   0.075   0.0750   0.83%   0.002
485                Convolution        {1,55,55,64}     {1,55,55,64}     {1,1,64,64}      {1,1}     {0,0,0,0}   {1,1}    0.162   0.166   0.161   0.1610   1.76%   0.025
495                Convolution        {1,55,55,64}     {1,55,55,64}     {3,3,64,64}      {1,1}     {1,1,1,1}   {1,1}    0.149   0.154   0.148   0.1480   1.62%   0.223
504                Convolution        {1,55,55,64}     {1,55,55,256}    {1,1,64,256}     {1,1}     {0,0,0,0}   {1,1}    0.223   0.223   0.223   0.2230   2.42%   0.099
513                Convolution        {1,55,55,64}     {1,55,55,256}    {1,1,64,256}     {1,1}     {0,0,0,0}   {1,1}    0.223   0.223   0.223   0.2230   2.42%   0.099
523                Add                {1,55,55,512}    {1,55,55,256}    N/A              N/A       N/A         N/A      0.099   0.102   0.098   0.0980   1.07%   0.001
532                Convolution        {1,55,55,256}    {1,55,55,64}     {1,1,256,64}     {1,1}     {0,0,0,0}   {1,1}    0.184   0.184   0.184   0.1840   2.00%   0.099
542                Convolution        {1,55,55,64}     {1,55,55,64}     {3,3,64,64}      {1,1}     {1,1,1,1}   {1,1}    0.148   0.148   0.147   0.1480   1.60%   0.223
551                Convolution        {1,55,55,64}     {1,55,55,256}    {1,1,64,256}     {1,1}     {0,0,0,0}   {1,1}    0.223   0.223   0.223   0.2230   2.42%   0.099
561                Add                {1,55,55,512}    {1,55,55,256}    N/A              N/A       N/A         N/A      0.098   0.099   0.098   0.0980   1.07%   0.001
570                Convolution        {1,55,55,256}    {1,55,55,64}     {1,1,256,64}     {1,1}     {0,0,0,0}   {1,1}    0.184   0.184   0.184   0.1840   2.00%   0.099
580                Convolution        {1,55,55,64}     {1,55,55,64}     {3,3,64,64}      {1,1}     {1,1,1,1}   {1,1}    0.148   0.148   0.147   0.1480   1.60%   0.223
589                Convolution        {1,55,55,64}     {1,55,55,256}    {1,1,64,256}     {1,1}     {0,0,0,0}   {1,1}    0.223   0.223   0.223   0.2230   2.42%   0.099
599                Add                {1,55,55,512}    {1,55,55,256}    N/A              N/A       N/A         N/A      0.098   0.099   0.098   0.0980   1.07%   0.001
608                Convolution        {1,55,55,256}    {1,55,55,128}    {1,1,256,128}    {1,1}     {0,0,0,0}   {1,1}    0.204   0.204   0.204   0.2040   2.21%   0.198
618                Convolution        {1,55,55,128}    {1,28,28,128}    {3,3,128,128}    {2,2}     {1,1,1,1}   {1,1}    0.126   0.132   0.125   0.1250   1.37%   0.231
627                Convolution        {1,28,28,128}    {1,28,28,512}    {1,1,128,512}    {1,1}     {0,0,0,0}   {1,1}    0.160   0.160   0.160   0.1600   1.74%   0.103
636                Convolution        {1,55,55,256}    {1,28,28,512}    {1,1,256,512}    {2,2}     {0,0,0,0}   {1,1}    0.169   0.173   0.168   0.1680   1.83%   0.206
646                Add                {1,28,28,1024}   {1,28,28,512}    N/A              N/A       N/A         N/A      0.057   0.057   0.057   0.0570   0.62%   0.000
655                Convolution        {1,28,28,512}    {1,28,28,128}    {1,1,512,128}    {1,1}     {0,0,0,0}   {1,1}    0.139   0.142   0.138   0.1380   1.51%   0.103
665                Convolution        {1,28,28,128}    {1,28,28,128}    {3,3,128,128}    {1,1}     {1,1,1,1}   {1,1}    0.124   0.127   0.123   0.1240   1.35%   0.231
674                Convolution        {1,28,28,128}    {1,28,28,512}    {1,1,128,512}    {1,1}     {0,0,0,0}   {1,1}    0.160   0.160   0.160   0.1600   1.74%   0.103
684                Add                {1,28,28,1024}   {1,28,28,512}    N/A              N/A       N/A         N/A      0.057   0.057   0.057   0.0570   0.62%   0.000
693                Convolution        {1,28,28,512}    {1,28,28,128}    {1,1,512,128}    {1,1}     {0,0,0,0}   {1,1}    0.138   0.138   0.138   0.1380   1.50%   0.103
703                Convolution        {1,28,28,128}    {1,28,28,128}    {3,3,128,128}    {1,1}     {1,1,1,1}   {1,1}    0.124   0.124   0.123   0.1240   1.34%   0.231
712                Convolution        {1,28,28,128}    {1,28,28,512}    {1,1,128,512}    {1,1}     {0,0,0,0}   {1,1}    0.161   0.163   0.160   0.1600   1.74%   0.103
722                Add                {1,28,28,1024}   {1,28,28,512}    N/A              N/A       N/A         N/A      0.057   0.057   0.057   0.0570   0.62%   0.000
731                Convolution        {1,28,28,512}    {1,28,28,128}    {1,1,512,128}    {1,1}     {0,0,0,0}   {1,1}    0.138   0.138   0.138   0.1380   1.50%   0.103
741                Convolution        {1,28,28,128}    {1,28,28,128}    {3,3,128,128}    {1,1}     {1,1,1,1}   {1,1}    0.124   0.124   0.123   0.1240   1.34%   0.231
750                Convolution        {1,28,28,128}    {1,28,28,512}    {1,1,128,512}    {1,1}     {0,0,0,0}   {1,1}    0.160   0.160   0.160   0.1600   1.74%   0.103
760                Add                {1,28,28,1024}   {1,28,28,512}    N/A              N/A       N/A         N/A      0.057   0.057   0.057   0.0570   0.62%   0.000
```

Figure 2-2 Profiler Statistical results of each layer

```
[I/magik::venus]:
Timing cycle = 5
===== Concise Dispatch Profiler Summary: N/A, Exclude 0 warm-ups =====
OperatorType          Avg(ms)  Max(ms)  Min(ms)  Avg(%)   GOPs     CalledTimes
Add                   0.733    0.738    0.732    7.95%    0.005    16
Convolution           8.392    8.493    8.358    91.03%    7.941    53
Pooling               0.094    0.103    0.092    1.02%    0.002    2
```

Figure 2-3 Profiler Overall statistical results

2.4.3 Complete example

After obtaining the magik package, please refer to the directory of your_root/magik-toolkit/Models/post/soc_sample/T40/venus_sample_ptq_yolov5s/ for details.