BMCV Technical Reference Manual

Release master

SOPHGO

Dec 25, 2023

Contents

CHAPTER 1

Declaration

1.1 Disclaimer

Legal Disclaimer

Notice

The purchased products, services and features are stipulated by the contract made between SOPHGO and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise

specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Technical Support

Address
Floor 6, Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Website
https://www.sophgo.com/

Email
sales@sophgo.com

Phone
+86-10-57590723 +86-10-57590724

SDK Release Notes

Versions	Release Date	Description
V2.0.0	2019.09.20	First release.
V2.0.1	2019.11.16	Version V2.0.1 released.
V2.0.3	2020.05.07	Version V2.0.3 released.
V2.2.0	2020.10.12	Version V2.2.0 released.
V2.3.0	2021.01.11	Version V2.3.0 released.
V2.3.1	2021.03.09	Version V2.3.1 released.
V2.3.2	2021.04.01	Version V2.3.2 released.
V2.4.0	2021.05.23	Version V2.4.0 released.
V2.5.0	2021.09.02	Version V2.5.0 released.
V2.6.0	2021.01.30	Version V2.6.0 released.
V2.7.0	2022.03.16	Version V2.7.0 released.

CHAPTER 2

Introduction to BMCV

2.1 Introduction to BMCV

BMCV provides a set of optimized machine vision libraries based on SOPHON Deep learning processors. Through the Tensor Computing Processor and VPP module of the processor, users can complete the operations of color space conversion, scale transformation, affine transformation, perspective transformation, linear transformation, picture frame, JPEG encoding and decoding, BASE64 encoding and decoding, NMS, sequencing, feature matching and so on.

CHAPTER 3

Introduction to bm_image

3.1 bm_image structure

Bmcv api is operated around bm_image, with a bm_image object corresponding to a picture. Users can build bm_image objects through bm_image_create for the applications of each function of bmcv. Users need to destroy it through bm_image_destroy after usage.

3.1.1 bm_image

bm_image structure is defined as follows:

```
struct bm_image {
    int width;
    int height;
    bm_image_format_ext image_format;
    bm_data_format_ext data_type;
    bm_image_private* image_private;
};
```

bm_image structure covers the width and height of the image, the image format (image_format), image data format (data_type), and the private data of the structure.

3.1.2 bm_image_format_ext image_format

image_format has the following enumeration types:

```
typedef enum bm_image_format_ext_{
    FORMAT_YUV420P,
    FORMAT_YUV422P,
    FORMAT_YUV444P,
    FORMAT_NV12,
    FORMAT_NV21,
    FORMAT_NV16,
    FORMAT_NV61,
    FORMAT_NV24,
    FORMAT_RGB_PLANAR,
    FORMAT_BGR_PLANAR,
    FORMAT_RGB_PACKED,
    FORMAT_BGR_PACKED,
    FORMAT_RGBP_SEPARATE,
    FORMAT_BGRP_SEPARATE,
    FORMAT_GRAY,
    FORMAT_COMPRESSED,
    FORMAT_HSV_PLANAR,
    FORMAT_ARGB_PACKED,
    FORMAT_ABGR_PACKED,
    FORMAT_YUV444_PACKED,
    FORMAT_YVU444_PACKED,
    FORMAT_YUV422_YUYV,
    FORMAT_YUV422_YVYU,
    FORMAT_YUV422_UYVY,
    FORMAT_YUV422_VYUY,
    FORMAT_RGBYP_PLANAR,
    FORMAT_HSV180_PACKED,
    FORMAT_HSV256_PACKED,
    FORMAT_BAYER
} bm_image_format_ext;
```

Description of each format:

- FORMAT_YUV420P

  It means to pre-create a picture in YUV420 format with three planes

- FORMAT_YUV422P

  It means to pre-create a picture in YUV422 format with three planes

- FORMAT_YUV444P

  It means to pre-create a picture in YUV444 format with three planes

- FORMAT_NV12

  It means to pre-create a picture in NV12 format with two planes

- FORMAT_NV21

It means to pre-create a picture in NV21 format with two planes

- FORMAT_NV16

  It means to pre-create a picture in NV16 format with two planes

- FORMAT_NV61

  It means to pre-create a picture in NV61 format with two planes

- FORMAT_RGB_PLANAR

  It means to pre-create a picture in RGB format with one plane and separately sequenced RGB

- FORMAT_BGR_PLANAR

  It means to pre-create a picture in BGR format with one plane and separately sequenced BGR

- FORMAT_RGB_PACKED

  It means to pre-create a picture in RGB format with one plane and staggered sequenced RGB

- FORMAT_BGR_PACKED

  It means to pre-create a picture in BGR format with one plane and staggered sequenced BGR

- FORMAT_RGBP_SEPARATE

  It means to pre-create a picture in RGB planar format. RGB is arranged separately and occupies one plane respectively. There are three planes in total.

- FORMAT_BGRP_SEPARATE

  It means to pre-create a picture in BGR planar format. BGR is arranged separately and occupies one plane respectively. There are three planes in total.

- FORMAT_GRAY

  It means to pre-create a gray image format picture with a plane

- FORMAT_COMPRESSED

  It means to pre-create a picture in VPU internally compressed format. There are four planes in total, and the contents are as follows:

  plane0: Y compressed table

  plane1: Y compressed data

  plane2: CbCr compressed table

  plane3: CbCr compressed data

- FORMAT_HSV_PLANAR

It means to pre-create a picture in HSV planar format with three planes. The range of H is 0 to 180

- FORMAT_ARGB_PACKED

  It means to pre-create a picture in ARGB format with one plane and staggered sequenced ARGB

- FORMAT_ABGR_PACKED

  It means to pre-create a picture in ABGR format with one plane and staggered sequenced ABGR

- FORMAT_YUV444_PACKED

  It means to pre-create a picture in YUV444 format with one plane and staggered sequenced YUV

- FORMAT_YVU444_PACKED

  It means to pre-create a picture in YVU444 format with one plane and staggered sequenced YVU

- FORMAT_YUV422_YUYV

  It means to pre-create a picture in YUV422 format with one plane and staggered sequenced YUYV

- FORMAT_YUV422_YVYU

  It means to pre-create a picture in YUV422 format with one plane and staggered sequenced YVYU

- FORMAT_YUV422_UYVY

  It means to pre-create a picture in YUV422 format with one plane and staggered sequenced UYVY

- FORMAT_YUV422_VYUY

  It means to pre-create a picture in YUV422 format with one plane and staggered sequenced VYUY

- FORMAT_RGBYP_PLANAR

  It means to pre-create a picture in RGBY format with four planes and separately sequenced RGBY

- FORMAT_HSV180_PACKED

  It means to pre-create a picture in HSV planar format with one plane and staggered sequenced HSV. The range of H is 0 to 180.

- FORMAT_HSV256_PACKED

  It means to pre-create a picture in HSV planar format with one plane and staggered sequenced HSV. The range of H is 0 to 255.

- FORMAT_BAYER

  It means to pre-create a bayer image format picture with a plane. The pixel arrangement is BGGR, RGGB, GRBG or GBRG, and the width and height need to be even.

3.1.3 bm_data_format_ext data_type

data_type has the following enumeration types:

```
typedef enum bm_image_data_format_ext_{
   DATA_TYPE_EXT_FLOAT32,
   DATA_TYPE_EXT_1N_BYTE,
   DATA_TYPE_EXT_4N_BYTE,
   DATA_TYPE_EXT_1N_BYTE_SIGNED,
   DATA_TYPE_EXT_4N_BYTE_SIGNED,
   DATA_TYPE_EXT_FP16,
   DATA_TYPE_EXT_BF16,
}bm_image_data_format_ext;
```

Description of incoming parameters:

- DATA_TYPE_EXT_FLOAT32

  Indicating that the created image data format is single-precision floating-point number

- DATA_TYPE_EXT_1N_BYTE

  Indicating that the created image data format is ordinary unsigned 1N UINT8

- DATA_TYPE_EXT_4N_BYTE

  Indicating that the created image data format is 4N UINT8, that is, four unsigned INT8 image data are staggered. One bm_image object actually contains four pictures with the same attributes

- DATA_TYPE_EXT_1N_BYTE_SIGNED

  Indicating that the created image data format is ordinary signed 1N INT8

- DATA_TYPE_EXT_4N_BYTE_SIGNED

  Indicating that the created image data format is 4N INT8, that is, the four signed INT8 image data are staggered

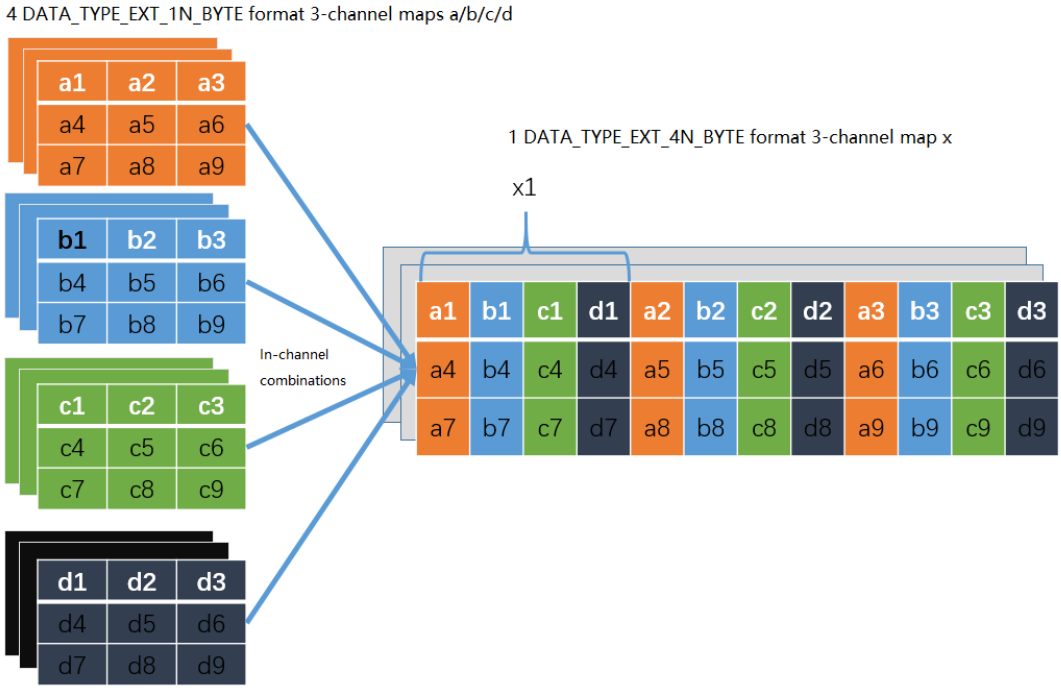
- DATA_TYPE_EXT_FP16

  Indicating that the created image data format is a half-precision floating-point number. Use 5bit to represent the exponent and 10bit to represent the decimal

- DATA_TYPE_EXT_BF16

  Indicating that the created image data format is 16bit floating point number, which is actually truncated data for FLOAT32 single precision floating point number. Use 8bit to represent the exponent and 7bit to represent the decimal

· For 4N arrangement, please refer to the following figure:



As shown above, the 4Byte of the i-th position in the corresponding channel of the 4 1N format images are put together as a 32-bit DWORD as the value of the i-th position in the corresponding channel of the 4N format image.For example, a1/b1/c1/d1 in channel 1 synthesis x1; for cases with less than 4 maps, the placeholder in map x still needs to be preserved.

4N only supports RGB-related formats, not YUV-related formats and FORMAT_COMPRESSED.

3.2 bm_image_create

We do not recommend that users directly fill bm_image structure, but create/destroy bm_image structure through the following API:

Interface form:

```
bm_status_t bm_image_create(
    bm_handle_t handle,
    int img_h,
    int img_w,
    bmcv_image_format_ext image_format,
    bmcv_data_format_ext data_type,
    bm_image *image,
    int* stride);
```

Description of incoming parameters:

- bm_handle_t handle

  input parameter. HDC (handle of device' s capacity) to be obtained through bm_dev_request

- int img_h

  Input parameter. Image height

- int img_w

  Input parameter. Image width

- bmcv_image_format_ext image_format

  Input parameter. Create bm_image format as required. The supported image formats are introduced in bm_image_format_ext.

- bm_image_format_ext data_type

  Input parameter. Create bm_image data format as required. The supported data formats are introduced in bm_image_data_format_ext

- bm_image *image

  Output parameter. Output filled bm_image structure pointer

- int* stride

  Input parameter. Stride describes the device memory layout associated with the created bm-image. The width stride value of each plane is counted in bytes.Defaults to the same width as a line of data (in BYTE count) when not filled.

Description of returning parameters:

The successful call of bmcv_image_create will return to BM_SUCCESS, and fill in the output image pointer structure. This structure records the size and related format of the image. But at this time, it is not associated with any device memory, and there is no device memory corresponding to the application data.

Notice:

1) The width and height of the following picture formats can be odd, and the interface will be adjusted to even numbers before completing the corresponding functions. However, it is recommended to use width and height with even numbers so as to maximize efficiency.
   - FORMAT_YUV420P
   - FORMAT_NV12
   - FORMAT_NV21
   - FORMAT_NV16
   - FORMAT_NV61
2) The width or stride of images with the format of FORMAT_COMPRESSED must be 64 aligned, otherwise it will return failure.

3) The default value of the stride parameter is NULL. At this time, the data of each plane is packed in compact, not in stride by default.

4) If the stride is not NULL, it will check whether the width stride value in the stride is legal. The so-called legality means the stride of all planes corresponding to image_format is greater than the default stride. The default stride value is calculated as follows:

```
int data_size = 1;
switch (data_type) {
    case DATA_TYPE_EXT_FLOAT32:
        data_size = 4;
        break;
    case DATA_TYPE_EXT_4N_BYTE:
    case DATA_TYPE_EXT_4N_BYTE_SIGNED:
        data_size = 4;
        break;
    default:
        data_size = 1;
        break;
}
int default_stride[3] = {0};
switch (image_format) {
    case FORMAT_YUV420P: {
        image_private->plane_num = 3;
        default_stride[0] = width * data_size;
        default_stride[1] = (ALIGN(width, 2) >> 1) * data_size;
        default_stride[2] = default_stride[1];
        break;
    }
    case FORMAT_YUV422P: {
        default_stride[0] = res->width * data_size;
        default_stride[1] = (ALIGN(res->width, 2) >> 1) * data_size;
        default_stride[2] = default_stride[1];
        break;
    }
    case FORMAT_YUV444P: {
        default_stride[0] = res->width * data_size;
        default_stride[1] = res->width * data_size;
        default_stride[2] = default_stride[1];
        break;
    }
    case FORMAT_NV12:
    case FORMAT_NV21: {
        image_private->plane_num = 2;
        default_stride[0] = width * data_size;
        default_stride[1] = ALIGN(res->width, 2) * data_size;
        break;
    }
    case FORMAT_NV16:
    case FORMAT_NV61: {
        image_private->plane_num = 2;
        default_stride[0] = res->width * data_size;
```

(continues on next page)

(continued from previous page)

```
        default_stride[1] = ALIGN(res->width, 2) * data_size;
        break;
    }
    case FORMAT_GRAY: {
        image_private->plane_num = 1;
        default_stride[0] = res->width * data_size;
        break;
    }
    case FORMAT_COMPRESSED: {
        image_private->plane_num = 4;
        break;
    }
    case FORMAT_BGR_PACKED:
    case FORMAT_RGB_PACKED: {
        image_private->plane_num = 1;
        default_stride[0] = res->width * 3 * data_size;
        break;
    }
    case FORMAT_BGR_PLANAR:
    case FORMAT_RGB_PLANAR: {
        image_private->plane_num = 1;
        default_stride[0] = res->width * data_size;
        break;
    }
    case FORMAT_BGRP_SEPARATE:
    case FORMAT_RGBP_SEPARATE: {
        image_private->plane_num = 3;
        default_stride[0] = res->width * data_size;
        default_stride[1] = res->width * data_size;
        default_stride[2] = res->width * data_size;
        break;
    }
}
```

3.3 bm_image_destroy

Destroy bm_image object and bm_image_create are used in pairs. It is recommended to destroy the object where it is created to avoid unnecessary memory leakage.

Interface form:

```
bm_status_t bm_image_destroy(
        bm_image image
);
```

Description of incoming parameters:

- bm_image image

  Input parameter. The object of bm_image to be destroyed.

Description of returning parameters:

Successful return will destroy the bm_image object. If the device memory of this object is applied by bm_image_alloc_dev_mem, the space will be released, otherwise the device memory of the object will not be released and managed by the user.

Note:

The bm_image_destroy(bm_image image) interface is designed with a structure as a formal reference, which internally frees the memory pointed to by image.image_private, but changes to the pointer image.image_private cannot be passed outside the function, resulting in a wild pointer problem on the second call.

In order to achieve the best compatibility of customer code for sdk, no changes are made to the interface at this time.

It is recommended to use bm_image_destroy (image) followed by image.image_private = NULL to avoid wild pointer problems when multithreading.

3.4 bm_image_copy_host_to_device

Interface form:

```
bm_status_t bm_image_copy_host_to_device(
      bm_image image,
      void* buffers[]
);
```

This API copies the host-side data to the corresponding device memory of bm_image structure.

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object of the device memory data to be filled.

- void* buffers[]

  Input parameter. Host side pointer, buffers are pointers to different plane data, and the number should be decided by the number of planes corresponding to image_format when creating bm_image. The amount of data per plane is decided by the width, height, stride, image_format, data_type when creating bm-image. The specific calculation method is as follows:

```
switch (res->image_format) {
   case FORMAT_YUV420P: {
      width[0]  = res->width;
      width[1]  = ALIGN(res->width, 2) / 2;
      width[2]  = width[1];
      height[0] = res->height;
```

(continues on next page)

(continued from previous page)

```
        height[1] = ALIGN(res->height, 2) / 2;
        height[2] = height[1];
        break;
    }
    case FORMAT_YUV422P: {
        width[0]  = res->width;
        width[1]  = ALIGN(res->width, 2) / 2;
        width[2]  = width[1];
        height[0] = res->height;
        height[1] = height[0];
        height[2] = height[1];
        break;
    }
    case FORMAT_YUV444P: {
        width[0]  = res->width;
        width[1]  = width[0];
        width[2]  = width[1];
        height[0] = res->height;
        height[1] = height[0];
        height[2] = height[1];
        break;
    }
    case FORMAT_NV12:
    case FORMAT_NV21: {
        width[0]  = res->width;
        width[1]  = ALIGN(res->width, 2);
        height[0] = res->height;
        height[1] = ALIGN(res->height, 2) / 2;
        break;
    }
    case FORMAT_NV16:
    case FORMAT_NV61: {
        width[0]  = res->width;
        width[1]  = ALIGN(res->width, 2);
        height[0] = res->height;
        height[1] = res->height;
        break;
    }
    case FORMAT_GRAY: {
        width[0]  = res->width;
        height[0] = res->height;
        break;
    }
    case FORMAT_COMPRESSED: {
        width[0]  = res->width;
        height[0] = res->height;
        break;
    }
    case FORMAT_BGR_PACKED:
    case FORMAT_RGB_PACKED: {
        width[0]  = res->width * 3;
```

(continues on next page)

(continued from previous page)

```
        height[0] = res->height;
        break;
    }
    case FORMAT_BGR_PLANAR:
    case FORMAT_RGB_PLANAR: {
        width[0]  = res->width;
        height[0] = res->height * 3;
        break;
    }
    case FORMAT_RGBP_SEPARATE:
    case FORMAT_BGRP_SEPARATE: {
        width[0]  = res->width;
        width[1]  = width[0];
        width[2]  = width[1];
        height[0] = res->height;
        height[1] = height[0];
        height[2] = height[1];
        break;
    }
}
```

Therefore, the amount of data corresponding to the buffers of each plane pointed to by the host pointer should be the above calculated plane_byte_size value. For example, FORMAT_BGR_PLANAR only needs the first address of one buffer, while FORMAT_RGBP_SEPARATE needs three buffers.

Description of returning value:

BM_SUCCESS when the function returns successfully.

Note:

1. If bm_image is not created by bm_image_create, a failure will be returned.
2. If the incoming bm_image object is not associated with device memory, it will automatically apply for the device memory corresponding to each plane_private->plane_byte_size, and copy the host data to the requested device memory. If the application for device memory fails, the API call will fail.
3. If the format of the incoming bm_image object is FORMAT_COMPRESSED, it will directly return failure. FORMAT_COMPRESSED does not support copying input by host pointer.
4. If the copy fails, the API call fails.

3.5 bm_image_copy_device_to_host

Interface form:

```
bm_status_t bm_image_copy_device_to_host(
      bm_image image,
      void* buffers[]
);
```

Description of incoming parameters::

- bm_image image

  Input parameter. The bm_image object whose data is to be transmitted.

- void* buffers[]

  Output parameter. Host-side pointer, buffers are pointers to data of different planes. The amount of data to be transmitted by each plane can be obtained through bm_image_get_byte_size API.

Note:

1. If bm_image is not created by bm_image_create, a failure will be returned.
2. If bm_image is not associated with device memory, a failure will be returned.
3. If the data transmission fails, the API call will fail.
4. If the function returns successfully, the data in the associated device memory will be copied to the host-side buffers.

3.6 bm_image_attach

If users want to manage device memory by themselves, or if device memory is generated by external components (VPU/VPP, etc.), they can call the following API to connect this device memory with bm_image.

Interface form:

```
bm_status_t bm_image_attach(
      bm_image image,
      bm_device_mem_t* device_memory
);
```

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object to be associated.

· bm_device_mem_t* device_memory

Input parameter. Fill the device_memory required by bm_image. The number should be decided by the plane number of image_format when creating bm_image.

Note:

1. bm_image_create will return fail if bm_image is not created.
2. bm_image object will be associated with the device_memory object when the function is called successfully.
3. bm_image will not manage device_memory associated in this way. That is to say, the device_memory will not be released when the image is destroyed. Users need to manage this device_memory by themselves.

3.7 bm_image_detach

Interface form:

```
bm_status_t bm_image_detach(
      bm_image
);
```

Description of incoming parameters:

· bm_image image

Input parameter. The bm_image object to be disassociated.

Note:

1. It will return fail if the incoming bm_image object is not created.
2. The bm_device will dissociate with the bm_image object when the function returns successfully. The bm_image object will no longer be associated with device_memory.
3. If the disassociated device_memory is automatically applied internally, this device memory will be released.
4. If the object is not associated with any device memory, a direct success will be returned.

3.8 bm_image_alloc_dev_mem

Interface form:

```
bm_status_t bm_image_alloc_dev_mem(
      bm_image      image
);
```

The API applies for internal memory management of the bm_image object. The size of the applied device memory is the sum of the size of the device memory required by each plane. The calculation method of plane_byte_size is introduced in bm_image_copy_host_to_device. It can also be confirmed by calling the bm_image_get_byte_size API.

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object to apply for device memory.

Note:

1. It will return fail if the bm_image object is not created.
2. It will return fail if the image format is FORMAT_COMPRESSED.
3. If the bm_image object is associated with device memory, a direct success will be returned.
4. The requested device memory is managed internally which will not be released again when destroyed or no longer in use.

3.9 bm_image_alloc_dev_mem_heap_mask

Interface form:

```
bm_status_t bm_image_alloc_dev_mem_heap_mask(
      bm_image      image,
      int           heap_mask
);
```

The API applies for internal management memory for the bm_image object. The applied device memory size is the sum of the device memory sizes required by each plane. The calculation method of plane_byte_size is introduced in bm_image_copy_host_to_device. It can also be confirmed by calling bm_image_get_byte_size API.

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object to apply for device memory.

- int heap_mask

  Input parameter. Select the mask of one or more heap IDs. Each bit indicates whether a heap ID is valid, 1 indicates that it can be allocated on the heap, 0 indicates that it cannot be allocated on the heap, the lowest bit indicates heap0, and so on. For example, heap_mask=2 indicates specific allocation of space on heap1, heap_mask=5 indicates allocation of space on heap0 or heap2.

Note:

1. If the bm_image object is not created, a failure will be returned.
2. If the image format is FORMAT_COMPRESSES, a failure will be returned.
3. If the bm_image object is associated with device memory, a direct success will be returned.
4. The requested device memory is managed internally which will not be released again when destroyed or no longer in use.

Heap Note:

heap id	bm1684 VPP	bm1684x VPP	Correspondence
heap0	W	R/W	TPU
heap1	R/W	R/W	JPU/VPP
heap2	R/W	R/W	VPU

3.10 bm_image_get_byte_size

获取 bm_image 对象各个 plane 字节大小。

Interface form:

```
bm_status_t bm_image_get_byte_size(
        bm_image image,
        int* size
);
```

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object to apply for device memory.

- int* size

  Output parameter. The number of bytes of each plane returned.

Note:

1. It will return fail if the bm_image object is not created.

2. It will return fail if the image format is FORMAT_COMPRESSED and is not associated with external device memory.
3. When the function is called successfully, the device memory byte size required for each plane will be filled in the size pointer. The calculation method of size is introduced in bm_image_copy_host_to_device.
4. If the bm_image object is not associated with external device memory, in addition to FORMAT_COMPRESSED, other formats can still successfully return and fill in size.

3.11 bm_image_get_device_mem

Interface form:

```
bm_status_t bm_image_get_device_mem(
      bm_image image,
      bm_device_mem_t* mem
);
```

Description of incoming parameters:

- bm_image image

  Input parameter. The bm_image object to apply for device memory.

- bm_device_mem_t* mem

  Output parameter. The bm_device_mem_t structure of each returned plane.

Note:

1. When the function is returned successfully, the device will fill in the bm_device_mem_t structure associated with each plane of the bm_image object in the mem pointer.
2. It will return fail if the bm_image object is not associated with device memory.
3. It will return fail if the bm_image object is not created.
4. If the device memory structure is applied internally, please do not release it in case of double free.

3.12 bm_image_alloc_contiguous_mem

Allocate contiguous memory for multiple images.

Interface form:

```
bm_status_t bm_image_alloc_contiguous_mem(
      int          image_num,
      bm_image     *images
);
```

Description of incoming parameters:

- int image_num

  Input parameter. The number of images to be allocated.

- bm_image *images

  Input parameter. The pointer of the image whose memory is to be allocated.

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、If the incoming image has been allocated or attached the memory, the existing memory should be detached first. Otherwise, a failure will be returned.

3、All images to be allocated should have the same size, otherwise, an error will be returned.

4、If the memory of the image to be destroyed is allocated by calling this API, users should first call bm_image_free_contiguous_mem to release the allocated memory, and then implement bm_image_destroy to destroy image.

5、bm_image_alloc_contiguous_mem and bm_image_free_contiguous_mem should be used in pairs.

3.13 bm_image_alloc_contiguous_mem_heap_mask

Allocate continuous memory for multiple images on the specified heap.

Interface form:

```
bm_status_t bm_image_alloc_contiguous_mem_heap_mask(
      int          image_num,
      bm_image     *images,
      int          heap_mask
);
```

Description of incoming parameters:

- int image_num

  Input parameter. The number of images to be allocated.

- bm_image *images

  Input parameter. Pointer to the image of the memory to be allocated.

- int heap_mask

  Input parameter. Select the mask of one or more heap IDs. Each bit indicates whether a heap ID is valid, 1 indicates that it can be allocated on the heap, 0 indicates that it cannot be allocated on the heap, the lowest bit indicates heap0, and so on. For example, heap_mask=2 indicates specific allocation of space on heap1, heap_mask=5 indicates allocation of space on heap0 or heap2.

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、If the incoming image has been allocated or the memory has been attached, the existing memory should be detached first. Otherwise, a failure will be returned.

3、All images to be allocated should have the same size, otherwise, an error will be returned.

4、If the memory of the image to be destroyed is allocated by calling this API, users should first call bm_image_free_contiguous_mem to release the allocated memory, and then implement bm_image_destroy to destroy image.

5、bm_image_alloc_contiguous_mem and bm_image_free_contiguous_mem should be used in pairs.

Heap Note:

heap id	bm1684 VPP	bm1684x VPP	Correspondence
heap0	W	R/W	TPU
heap1	R/W	R/W	JPU/VPP
heap2	R/W	R/W	VPU

3.14 bm_image_free_contiguous_mem

Release the contiguous memory of multiple images allocated by bm_image_alloc_contiguous_mem.

Interface form:

```
bm_status_t bm_image_free_contiguous_mem(
      int image_num,
      bm_image *images
);
```

Description of incoming parameters:

- int image_num

  Input parameter. Number of images to be released

- bm_image *images

  Input parameters. Pointer to the image of the memory to be released

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、All images to be released should be of the same size.

3、bm_image_alloc_contiguous_mem and bm_image_free_contiguous_mem should be used in pairs. The released memory of bm_image_free_contiguous_mem must be allocated by bm_image_alloc_contiguous_mem.

4、Users should first call bm_image_free_contiguous_mem, release the memory in the image, and then destroy image by calling bm_image_destroy.

3.15 bm_image_attach_contiguous_mem

Attach a piece of contiguous memory to multiple images.

Interface form:

```
bm_status_t bm_image_attach_contiguous_mem(
      int image_num,
      bm_image * images,
      bm_device_mem_t dmem
);
```

Description of incoming parameters:

- int image_num

  Input parameter. The number of images in the memory to be attached.

- bm_image *images

  Input parameter. Pointer to the image of the memory to be attached.

- bm_device_mem_t dmem

  Input parameter. Allocated device memory information.

Description of returning parameters:

- BM_SUCCESS: success

- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、All images to be attached should have the same size, otherwise an error will be returned.

3.16 bm_image_dettach_contiguous_mem

Detach a piece of contiguous memory from multiple images.

Interface form:

```
bm_status_t bm_image_dettach_contiguous_mem(
        int image_num,
        bm_image * images
);
```

Description of incoming parameters:

- int image_num

  Input parameter. The number of images in the memory to be detached.

- bm_image *images

  Input parameter. Pointer to the image of the memory to be detached.

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、All images to be detached should have the same size, otherwise an error will be returned.

3、bm_image_attach_contiguous_mem and bm_image_detach_contiguous_mem should be used in pairs. The detached device memory of bm_image_detach_contiguous must be attached to the image through bm_image_attach_contiguous_mem.

3.17 bm_image_get_contiguous_device_mem

Get the device memory information of contiguous memory from multiple images with contiguous memory.

Interface form:

```
bm_status_t bm_image_get_contiguous_device_mem(
      int image_num,
      bm_image *images,
      bm_device_mem_t * mem
);
```

Description of incoming parameters:

- int image_num

  Input parameter. The number of images to be obtained.

- bm_image *images

  Input parameter. Image pointer to get information.

- bm_device_mem_t * mem

  Output parameter. The obtained device memory information of contiguous memory.

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Note:

1、image_num should be greater than 0, otherwise an error will be returned.

2、The filled image should be the same size, otherwise, an error will be returned.

3、The memory of the filled image must be contiguous, otherwise, an error will be returned.

4、The memory of the filled image must be obtained through bm_image_alloc_contiguous_mem or bm_image_attach_contiguous_mem.

3.18 bm_image_get_format_info

This interface is used to get some information about the bm_image.

Interface form:

```
bm_status_t bm_image_get_format_info(
      bm_image *            src,
      bm_image_format_info_t *info
);
```

Input parameters description:

- bm_image* src

  Input parameter. The target bm_image to obtain information.

- bm_image_foramt_info_t *info

  Output parameter. Save the data structure of the required information and return it to the user. See the data structure description below for detailst.

Return parameters description:

- BM_SUCCESS: success
- Other: failed

Data structure description:

```
typedef struct bm_image_format_info {
      int                 plane_nb;
      bm_device_mem_t         plane_data[8];
      int                 stride[8];
      int                 width;
      int                 height;
      bm_image_format_ext     image_format;
      bm_image_data_format_ext data_type;
      bool                default_stride;
} bm_image_format_info_t;
```

- int plane_nb

  Number of planes for this image

- bm_device_mem_t plane_data[8]

  Device memory of each plane

- int stride[8];

  Stride value of each plane

- int width;

  Width of the image

- int height;

  Height of the image

- bm_image_format_ext image_format;

  Image format

- bm_image_data_format_ext data_type;

  Storage data type of the image

- bool default_stride;

  Whether the defaulted stride value is used.

3.19 bm_image_get_stride

This interface is used to get the stride information of the bm_image object.

Interface form:

```
bm_status_t bm_image_get_stride(
      bm_image image,
      int *stride
);
```

Input parameter description:

- bm_image image

  Input parameter. The target bm_image to obtain stride information.

- int *stride

  Output parameter. Pointer to store the stride of each plane.

Return parameter description:

- BM_SUCCESS: success
- Other: failed

3.20 bm_image_get_plane_num

This interface is used to get the number of plane of the target bm_image object.

Interface form:

```
int bm_image_get_plane_num(bm_image image);
```

Input parameter description:

- bm_image image

  Input parameter. The target bm_image to obtain the number of planes.

Return parameter description:

The return value is the number of planes of the target bm_image.

3.21 bm_image_is_attached

This interface is used to judge whether the target has attached storage space.

Interface form:

```
bool bm_image_is_attached(bm_image image);
```

Input parameter description:

- bm_image image

  Input parameters. To determine whether attach to the target bm_image of the storage space.

Return parameter description:

If the target bm_image has attached the storage space, it will return true; otherwise, it will return false.

3.22 bm_image_get_handle

This interface is used to obtain the handle through bm_image.

Interface form:

```
bm_handle_t bm_image_get_handle(bm_image* image);
```

Input parameter description

- bm_image image

  Input parameter. The target bm_image to obtain handle.

Return parameter description:

The return value is the handle of the target bm_image.

3.23 bm_image_write_to_bmp

This interface is used to output bm_image objects as bitmaps (.bmp).

Interface form:

```
bm_status_t bm_image_write_to_bmp(
      bm_image    input,
      const char* filename);
```

Parameter description:

- bm_image input

Input parameter. Input bm_image.

- const char* filename

Input parameter. The path and file name of the saved bitmap file.

Return value description:

- BM_SUCCESS: success
- Others: failed

Note:

1. Before calling bm_image_write_to_bmp(), you must ensure that the input image has been created correctly and is_attached, otherwise the function will return failure.

CHAPTER 4

bm_image device memory management

4.1 bm_image device memory management

The bm_image structure needs to be associated with relevant device memory. Only when there is the data you need in the device memory can the subsequent bmcv API be called. Whether calling bm_image_alloc_dev_mem internally or calling bm_image_attach associated with external memory, users can connect the bm_image object with the device memory.

Users can call the following API to judge whether the bm_image object has been connected to the device memory:

```
bool bm_image_is_attached(
      bm_image image
);
```

Incoming parameters description:

- bm_image image

  Input parameter. The bm_image object to be judged.

Return parameters description:

1. If the bm_image object is not created, a false will be returned;
2. Whether the returning bm_image object of the function is associated with a piece of device memory. If it is associated, it will return true; otherwise, it will return false.

Note:

1. In general, calling the bmcv API requires the incoming bm_image object to be associated with device memory. Otherwise, it will return failure. If the output

bm_image object is not associated with device memory, it will internally call the bm_image_alloc_dev_mem function and apply internal memory.

2. The applied memory when bm_image calls bm_image_alloc_dev_mem is automatically managed internally. The memory will be automatically released without the management of the caller when calling bm_image_destroy, bm_image_detach, bm_image_attach and other device memory. Conversely, if the bm_image_attach is connected with a device memory, it means that the memory will be managed by the caller. The memory will not be automatically released when calling bm_image_destroy, bm_image_detach, bm_image_attach or other device memory. It needs to be manually released by the caller.

3. At present, device memory is divided into three memory spaces: heap0, heap1 and heap2. The difference among the three is whether the hardware VPP module of the processor has reading permission. Therefore, if an API needs to be implemented by specifying the hardware VPP module, the input bm_image of the API must be guaranteed to be saved in heap1 or heap2.

heap id	bm1684 VPP	bm1684x VPP
heap0	Unreadable	readable
heap1	readable	readable
heap2	readable	readable

CHAPTER 5

BMCV API

5.1 BMCV API

Briefly explain which part of the hardware implements the BMCV API

The following interfaces BM1684X have not yet implemented:

- bmcv_image_canny
- bmcv_image_dct
- bmcv_image_draw_lines
- bmcv_fft
- bmcv_image_lkpyramid
- bmcv_image_morph
- bmcv_image_sobel

num	API	BM1684	BM1684X
1	bmcv_as_strided	NOT SUPPORT	TPU
2	bmcv_image_absdiff	TPU	TPU
3	bmcv_image_add_weighted	TPU	TPU
4	bmcv_base64	SPACC	SPACC
5	bmcv_image_bayer2rgb	NOT SUPPORT	TPU
6	bmcv_image_bitwise_and	TPU	TPU
7	bmcv_image_bitwise_or	TPU	TPU
8	bmcv_image_bitwise_xor	TPU	TPU

continues on next page

Table 5.1 – continued from previous page

num	API	BM1684	BM1684X
9	bmcv_calc_hist	TPU	TPU
10	bmcv_image_canny	TPU	TPU
11	bmcv_image_convert_to	TPU	VPP+TPU
12	bmcv_image_copy_to	TPU	VPP+TPU
13	bmcv_image_dct	TPU	TPU
14	bmcv_distance	TPU	TPU
15	bmcv_image_draw_lines	CPU	VPP
16	bmcv_image_draw_rectangle	TPU	VPP
17	bmcv_feature_match	TPU	TPU
18	bmcv_fft	TPU	TPU
19	bmcv_image_fill_rectangle	TPU	VPP
20	bmcv_image_gaussian_blur	TPU	TPU
21	bmcv_gemm	TPU	TPU
22	bmcv_image_jpeg_enc	JPU	JPU
23	bmcv_image_jpeg_dec	JPU	JPU
24	bmcv_image_laplacian	TPU	TPU
25	bmcv_matmul	TPU	TPU
26	bmcv_min_max	TPU	TPU
27	bmcv_nms_ext	TPU	TPU
28	bmcv_nms	TPU	TPU
29	bmcv_image_resize	VPP+TPU	VPP
30	bmcv_image_sobel	TPU	TPU
31	bmcv_sort	TPU	TPU
32	bmcv_image_storage_convert	VPP+TPU	VPP
33	bmcv_image_threshold	TPU	TPU
34	bmcv_image_transpose	TPU	TPU
35	bmcv_image_vpp_basic	VPP	VPP
36	bmcv_image_vpp_convert_padding	VPP	VPP
37	bmcv_image_vpp_convert	VPP	VPP
38	bmcv_image_vpp_csc_matrix_convert	VPP	VPP
39	bmcv_image_vpp_stitch	VPP	VPP
40	bmcv_image_warp_affine	TPU	TPU
41	bmcv_image_warp_perspective	TPU	TPU
42	bmcv_image_watermark_superpose	NOT SUPPORT	TPU
43	bmcv_nms_yolo	TPU	TPU
44	bmcv_cmulp	TPU	TPU
45	bmcv_faiss_indexflatIP	NOT SUPPORT	TPU
46	bmcv_faiss_indexflatL2	NOT SUPPORT	TPU
47	bmcv_image_yuv2bgr_ext	TPU	VPP
48	bmcv_image_yuv2hsv	TPU	VPP+TPU
49	bmcv_batch_topk	TPU	TPU
50	bmcv_image_put_text	CPU	CPU
51	bmcv_hm_distance	NOT SUPPORT	TPU

continues on next page

Table 5.1 – continued from previous page

num	API	BM1684	BM1684X
52	bmcv_axpy	TPU	TPU
53	bmcv_image_pyramid_down	TPU	TPU

Note:

For BM1684 and BM1684X, the implementation of the following two operators requires a combination of BMCPU and Tensor Computing Processor

num	API
1	bmcv_image_lkpyramid
2	bmcv_image_morph

5.2 bmcv_image_yuv2bgr_ext

This interface convert YUV format to RGB format.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_yuv2bgr_ext(
      bm_handle_t handle,
      int image_num,
      bm_image* input,
      bm_image* output
);
```

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int image_num

  Input parameter. The number of input/output images.

- bm_image* input

  Input parameter. The input bm_image object pointer.

- bm_image* output

  Output parameter. The output bm_image object pointer.

Description of returning parameters:

- BM_SUCCESS: success
- Other: failed

Code example

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>

int main(int argc, char *argv[]) {
    bm_handle_t handle;
    bm_dev_request(&handle, 0);

    int image_n = 1;
    int image_h = 1080;
    int image_w = 1920;
    bm_image src, dst;
    bm_image_create(handle, image_h, image_w, FORMAT_NV12,
          DATA_TYPE_EXT_1N_BYTE, &src);
    bm_image_create(handle, image_h, image_w, FORMAT_BGR_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &dst);
    std::shared_ptr<u8*> y_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w]);
    std::shared_ptr<u8*> uv_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w / 2]);
    memset((void *)(*y_ptr.get()), 148, image_h * image_w);
    memset((void *)(*uv_ptr.get()), 158, image_h * image_w / 2);
    u8 *host_ptr[] = {*y_ptr.get(), *uv_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);
    bmcv_image_yuv2bgr_ext(handle, image_n, &src, &dst);
    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_dev_free(handle);
    return 0;
}
```

Note:

1. This API inputs image objects in NV12/NV21/NV16/NV61/YUV420P formats, and fills the converted RGB data results into the device memory associated with the output image object
2. The API only supports :

- The API supports the following image formats of input bm_image:

num	input image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_NV16
4	FORMAT_NV61
5	FORMAT_YUV420P
6	FORMAT_YUV422P

- The API supports the following image formats of output bm_image:

num	output image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_BGR_PLANAR

- bm1684 supports the following data formats:

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_4N_BYTE

- bm1684x supports the following data formats

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE

It will return fail if the required input/output formats are not met.

3. It will return fail if all input and output bm_image structures not created in advance.
4. The image_format, data_type, width and height of all input bm_image objects must be the same. The image_format, data_type, width and height of all output bm_image objects must be the same. The width and height of all input/output bm_image objects must be the same. Otherwise, a failure will be returned.
5. image_num indicates the number of input objects. If the data format of output bm_image is DATA_TYPE_EXT_4N_BYTE, only output one bm_image 4N object. On the contrary, the number of output objects is image_num.
6. image_num must be greater than or equal to 1 and less than or equal to 4, otherwise, a failure will be returned.
7. All input objects must attach device memory, otherwise, a failure will be returned.

8. If the output object does not attach device memory, it will internally call bm_image_alloc_dev_mem to apply for internally managed device memory and fills the converted RGB data into device memory.

5.3 bmcv_image_warp_affine

The interface implements the affine transformation of the image, and the operations of rotation, translation and scaling. Affine transformation is a linear transformation from two-dimensional coordinates (x, y) to two-dimensional coordinates (x0, y0). The implementation of this interface is to find the corresponding coordinates in the input image for each pixel of the output image, so as to form a new image. Its mathematical expression is as follows:

$$\begin{cases} x_0 = a_1 x + b_1 y + c_1 \\ y_0 = a_2 x + b_2 y + c_2 \end{cases}$$

The corresponding homogeneous coordinate matrix is expressed as:

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The coordinate transformation matrix is a 6-point matrix, which is a coefficient matrix for deriving the input image coordinates from the output image coordinates, which can be obtained by the corresponding 3-point coordinates on the input and output image. In facial detection, the face transformation matrix can be obtained through facial detection points.

bmcv_affine_matrix defines a coordinate transformation matrix in the order of float m[6] = {a1, b1, c1, a2, b2, c2}. bmcv_affine_image_matrix defines that there are several transformation matrices in an image. Generally speaking, when an image has multiple faces, it will correspond to multiple transformation matrices.

```
typedef struct bmcv_affine_matrix_s{
        float m[6];
} bmcv_warp_matrix;

typedef struct bmcv_affine_image_matrix_s{
        bmcv_affine_matrix *matrix;
        int matrix_num;
} bmcv_affine_image_matrix;
```

Processor model support

This interface supports BM1684/BM1684X.

Interface form 1:

```
bm_status_t bmcv_image_warp_affine(
        bm_handle_t handle,
        int image_num,
```

(continues on next page)

(continued from previous page)

```
    bmcv_affine_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0
);
```

Interface form 2:

```
bm_status_t bmcv_image_warp_affine_similar_to_opencv(
    bm_handle_t handle,
    int image_num,
    bmcv_affine_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0
);
```

This interface is an interface to align opencv affine transformations.

Input parameter description

- bm_handle_t handle

  Input parameter. The input bm_handle handle.

- int image_num

  Input parameter. The number of input images, up to 4.

- bmcv_affine_image_matrix matrix[4]

  Input parameter. The transformation matrix data structure corresponding to each image. Support up to 4 images.

- bm_image* input

  Input parameter. Input bm_image. For 1N mode, up to 4 bm_image; for 4N mode, up to one bm_image.

- bm_image* output

  Output parameter. Output bm_image. It requires calling bmcv_image_create externally. Users are recommended to call bmcv_image_attach to allocate the device memory. If users do not call attach, the device memory will be allocated internally. For output bm_image, its data type is consistent with the input, that is, if the input is 4N mode, the output is also 4N mode; if the input is 1N mode, the output is also 1N mode. The size of the required bm_image is the sum of the transformation matrix of all images. For example, input a 4N mode bm_image, and the transformation matrix of four pictures is [3,0,13,5]. The total transformation matrix is 3 + 0 + 13 + 5 = 21. Since the output is in 4N mode, it needs (21 + 4-1) / 4 = 6 bm_image output.

- int use_bilinear

Input parameter. Whether to use bilinear interpolation. If it is 0, use nearest interpolation. If it is 1, use bilinear interpolation. The default is nearest interpolation. The performance of nearest interpolation is better than bilinear interpolation. Therefore, it is recommended to choose nearest interpolation first. Users can select bilinear interpolation unless there are requirements for accuracy.

Return parameters description:

- BM_SUCCESS: success
- Other: failed

Note

1. The API supports the following image_format:

num	image_format
1	FORMAT_BGR_PLANAR
2	FORMAT_RGB_PLANAR

2. The API supports the following data_type in bm1684 :

num	data_type
1	DATA_TYPE_EXT_1N_BYTE
2	DATA_TYPE_EXT_4N_BYTE

3. The API supports the following data_type in bm1684x :

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

4. The API' s input and output of bm_image both support stride.
5. It is required that the width, height, image_format and data_type of the input bm_image must be consistent.
6. It is required that the width, height, image_format and data_type of the output bm_image must be consistent.

Code example

```
#inculde "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>
#include <iostream>
```

(continues on next page)

(continued from previous page)

```
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"

int main(int argc, char *argv[]) {
    bm_handle_t handle;

    int image_h = 1080;
    int image_w = 1920;

    int dst_h = 256;
    int dst_w = 256;
    int use_bilinear = 0;
    bm_dev_request(&handle, 0);
    bmcv_affine_image_matrix matrix_image;
    matrix_image.matrix_num = 1;
    std::shared_ptr<bmcv_affine_matrix> matrix_data
        = std::make_shared<bmcv_affine_matrix>();
    matrix_image.matrix = matrix_data.get();

    matrix_image.matrix->m[0] = 3.848430;
    matrix_image.matrix->m[1] = -0.02484;
    matrix_image.matrix->m[2] = 916.7;
    matrix_image.matrix->m[3] = 0.02;
    matrix_image.matrix->m[4] = 3.8484;
    matrix_image.matrix->m[5] = 56.4748;

    bm_image src, dst;
    bm_image_create(handle, image_h, image_w, FORMAT_BGR_PLANAR,
        DATA_TYPE_EXT_1N_BYTE, &src);
    bm_image_create(handle, dst_h, dst_w, FORMAT_BGR_PLANAR,
        DATA_TYPE_EXT_1N_BYTE, &dst);

    std::shared_ptr<u8*> src_ptr = std::make_shared<u8*>(
        new u8[image_h * image_w * 3]);
    memset((void *)(*src_ptr.get()), 148, image_h * image_w * 3);
    u8 *host_ptr[] = {*src_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);

    bmcv_image_warp_affine(handle, 1, &matrix_image, &src, &dst, use_bilinear);

    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_dev_free(handle);

    return 0;
}
```

5.4 bmcv_image_warp_perspective

The interface implments the transmission transformation of the image, also known as projection transformation or perspective transformation. The transmission transformation projects the picture to a new visual plane, which is a nonlinear transformation from two-dimensional coordinates (x0, y0) to two-dimensional coordinates (x, y). The implementation of the API is to obtain the coordinates of the corresponding input image for each pixel coordinate of the output image, and then form a new image. Its mathematical expression is as follows:

$$\begin{cases} x' = a_1x + b_1y + c_1 \\ y' = a_2x + b_2y + c_2 \\ w' = a_3x + b_3y + c_3 \\ x_0 = x'/w' \\ y_0 = y'/w' \end{cases}$$

The corresponding homogeneous coordinate matrix is expressed as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{cases} x_0 = x'/w' \\ y_0 = y'/w' \end{cases}$$

The coordinate transformation matrix is a 9-point matrix (usually c3=1). Through the transformation matrix, the corresponding coordinates of the original input image can be derived from the coordinates of the output image. The transformation matrix can be obtained by inputting the coordinates of 4 points corresponding to the output image.

In order to complete the transmission transformation more conveniently, the library provides two forms of interfaces for users: one is that the user provides the transformation matrix to the interface as input; the other interface is to provide the coordinates of four points in the input image as input, which is suitable for transmitting an irregular quadrilateral into a rectangle with the same size as the output. As shown in the figure below, the input image A' B' C' D' can be mapped into the output image ABCD. The user only needs to provide the coordinates of four points A' B' C' D' in the input image, the interface will automatically calculate the transformation matrix according to the coordinates of these four vertices and the coordinates of the four vertices of the output image, so as to complete the function.

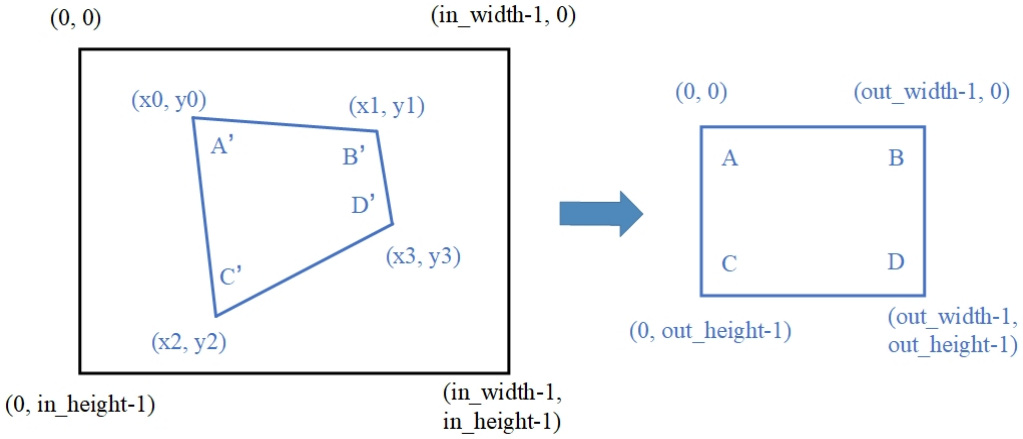
Processor model support

This interface supports BM1684/BM1684X.

Interface form 1:

```
bm_status_t bmcv_image_warp_perspective(
      bm_handle_t handle,
      int image_num,
      bmcv_perspective_image_matrix matrix[4],
```

(continues on next page)



(continued from previous page)

```
        bm_image* input,
        bm_image* output,
        int use_bilinear = 0
);
```

Among them, bmcv_perspective_matrix defines a coordinate transformation matrix in the order of float m[9] = {a1, b1, c1, a2, b2, c2, a3, b3, c3}. bmcv_perspective_image_Matrix defines several transformation matrices in an image, which can implements the transmission transformation of multiple small images in an image.

```
typedef struct bmcv_perspective_matrix_s{
        float m[9];
} bmcv_perspective_matrix;

typedef struct bmcv_perspective_image_matrix_s{
        bmcv_perspective_matrix *matrix;
        int matrix_num;
} bmcv_perspective_image_matrix;
```

Interface form 2:

```
bm_status_t bmcv_image_warp_perspective_with_coordinate(
        bm_handle_t handle,
        int image_num,
        bmcv_perspective_image_coordinate coord[4],
        bm_image* input,
        bm_image* output,
        int use_bilinear = 0
);
```

Among them, bmcv_perspective_coordinate defines the coordinates of the four vertices of the quadrilateral, which are stored in the order of top left, top right, bottom left and bottom right. bmcv_perspective_image_coordinate defines the

coordinates of several groups of quadrangles in an image, which can complete the transmission transformation of multiple small images in an image.

```
typedef struct bmcv_perspective_coordinate_s{
    int x[4];
    int y[4];
} bmcv_perspective_coordinate;

typedef struct bmcv_perspective_image_coordinate_s{
    bmcv_perspective_coordinate *coordinate;
    int coordinate_num;
} bmcv_perspective_image_coordinate;
```

Interface form 3:

```
bm_status_t bmcv_image_warp_perspective_similar_to_opencv(
    bm_handle_t handle,
    int image_num,
    bmcv_perspective_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0
);
```

The transformation matrix defined by bmcv_perspective_image_matrix in this interface is the same as the transformation matrix required to be input by the warpPerspective interface of opencv, and is the inverse of the matrix defined by the structure of the same name in interface 1, and the other parameters are the same as interface 1.

```
typedef struct bmcv_perspective_matrix_s{
    float m[9];
} bmcv_perspective_matrix;

typedef struct bmcv_perspective_image_matrix_s{
    bmcv_perspective_matrix *matrix;
    int matrix_num;
} bmcv_perspective_image_matrix;
```

输入参数说明

- bm_handle_t handle

  Input parameter. The input bm_handle handle.

- int image_num

  Input parameter. The number of input images, up to 4.

- bmcv_perspective_image_matrix matrix[4]

  Input parameter. The transformation matrix data structure corresponding to each image. Support up to 4 images.

· bmcv_perspective_image_coordinate coord[4]

  Input parameter. The quadrilateral coordinate information corresponding to each image. Support up to 4 images.

· bm_image* input

  Input parameter. Input bm_image. For 1N mode, up to 4 bm_image; for 4N mode, up to 1 bm_image.

· bm_image* output

  Output parameter. Output bm_image. It requires calling bmcv_image_create externally. Users are recommended to call bmcv_image_attach to allocate the device memory. If users do not call attach, the device memory will be allocated internally. For output bm_image, its data type is consistent with the input, that is, if the input is 4N mode, the output is also 4N mode; if the input is 1N mode, the output is also 1N mode. The size of the required bm_image is the sum of the transformation matrix of all images. For example, input a 4N mode bm_image, and the transformation matrix of four pictures is [3,0,13,5]. The total transformation matrix is 3 + 0 + 13 + 5 = 21. Since the output is in 4N mode, it needs (21 + 4-1) / 4 = 6 bm_image output.

· int use_bilinear

  Input parameter. Whether to use bilinear interpolation. If it is 0, use nearest interpolation. If it is 1, use bilinear interpolation. The default is nearest interpolation. The performance of nearest interpolation is better than bilinear interpolation. Therefore, it is recommended to choose nearest interpolation first. Users can select bilinear interpolation unless there are requirements for accuracy.

Return parameter description:

· BM_SUCCESS: success

· Other: failed

注意事项

1. The interface requires that all coordinate points of the output image can find the corresponding coordinates in the original input image, which cannot exceed the size of the original image. It is recommended to give priority to interface 2, which can automatically meet this requirement.

2. The API supports the following image_format:

num	image_format
1	FORMAT_BGR_PLANAR
2	FORMAT_RGB_PLANAR

3. The API supports the following data_type in bm1684:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE
2	DATA_TYPE_EXT_4N_BYTE

4. The API supports the following data_type in bm1684x:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

5. The API' s input and output of bm_image both support stride.
6. It is required that the width, height, image_format and data_type of the input bm_image must be consistent.
7. It is required that the width, height, image_format and data_type of the output bm_image must be consistent.

Code example

```
#inculde "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>
#include <iostream>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"

int main(int argc, char *argv[]) {
    bm_handle_t handle;

    int image_h = 1080;
    int image_w = 1920;

    int dst_h = 1080;
    int dst_w = 1920;
    int use_bilinear = 0;
    bm_dev_request(&handle, 0);
    bmcv_perspective_image_matrix matrix_image;
    matrix_image.matrix_num = 1;
    std::shared_ptr<bmcv_perspective_matrix> matrix_data
            = std::make_shared<bmcv_perspective_matrix>();
    matrix_image.matrix = matrix_data.get();

    matrix_image.matrix->m[0] = 0.529813;
    matrix_image.matrix->m[1] = -0.806194;
    matrix_image.matrix->m[2] = 1000.000;
    matrix_image.matrix->m[3] = 0.193966;
    matrix_image.matrix->m[4] = -0.019157;
```

(continues on next page)

(continued from previous page)

```
    matrix_image.matrix->m[5] = 300.000;
    matrix_image.matrix->m[6] = 0.000180;
    matrix_image.matrix->m[7] = -0.000686;
    matrix_image.matrix->m[8] = 1.000000;

    bm_image src, dst;
    bm_image_create(handle, image_h, image_w, FORMAT_BGR_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &src);
    bm_image_create(handle, dst_h, dst_w, FORMAT_BGR_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &dst);

    std::shared_ptr<u8*> src_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w * 3]);
    memset((void *)(*src_ptr.get()), 148, image_h * image_w * 3);
    u8 *host_ptr[] = {*src_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);

    bmcv_image_warp_perspective(handle, 1, &matrix_image, &src, &dst, use_
↪bilinear);

    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_dev_free(handle);

    return 0;
}
```

5.5 bmcv_image_watermark_superpose

This interface is used to overlay one or more watermarks on the image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form 1:

```
bm_status_t bmcv_image_watermark_superpose(
                bm_handle_t handle,
                bm_image * image,
                bm_device_mem_t * bitmap_mem,
                int bitmap_num,
                int bitmap_type,
                int pitch,
                bmcv_rect_t * rects,
                bmcv_color_t color)
```

This interface can realize the specified positions of different input maps and overlay different watermarks.

Interface form 2:

```
bm_status_t bmcv_image_watermark_repeat_superpose(
                bm_handle_t handle,
                bm_image image,
                bm_device_mem_t bitmap_mem,
                int bitmap_num,
                int pitch,
                bmcv_rect_t * rects,
                bmcv_color_t color)
```

This interface is a simplified version of Interface 1, and a watermark can be superimposed repeatedly at different positions in a picture.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- bm_image* image

  Input parameter. The bm_image on which users need to add watermarks.

- bm_device_mem_t* bitmap_mem

  Input parameters. Pointer to bm_device_mem_t Object of watermarks.

- int bitmap_num

  Input parameters. Number of watermarks, It refers to the number of bmcv_rect_t objects contained in the rects pointer, also the number of bm_image objects contained in the image pointer, and the number of bm_device_mem_t objects contained in the bitmap_mem pointer.

- int bitmap_type

  Input parameters. Watermark type: value 0 indicates that the watermark is an 8bit data type (with transparency information), and value 1 indicates that the watermark is a 1bit data type (without transparency information).

- int pitch

  Input parameters. The number of byte per line of the watermark file can be interpreted as the width of the watermark.

- bmcv_rect_t* rects

  Input parameters. Watermark position pointer, including the starting point, width and height of each watermark. Please refer to the following data type description for details.

- bmcv_color_t color

  Input parameters. The color of the watermark. Please refer to the following data type description for details.

Return value description:

- BM_SUCCESS: success
- Other: failed

Data type description:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;

typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} bmcv_color_t;
```

- start_x describes the starting horizontal coordinate of where the watermask is located in the original image. It starts at 0 from left to right and takes values in the range [0, width).
- start_y describes the starting vertical coordinate of where the watermask is located in the original image. It starts at 0 from top to bottom and takes values in the range [0, height).
- crop_w describes the width of the crop image.
- crop_h describes the height of the crop image.
- r R component of color
- g G component of color
- b B component of color

Note:

1. bm1684x:

- bm1684x supports the following data_type of bm_image:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

- bm1684x supports the following image_format of bm_image:

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

Returns a failure if the input and output format requirements are not met.

2. All input and output bm_image structures must be created in advance, or a failure will be returned.
3. The maximum number of watermarks can be 512.
4. If the watermark area exceeds the width and height of the original image, a failure will be returned.

5.6 bmcv_image_crop

The interface can crop out several small images from an original image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_crop(
      bm_handle_t       handle,
      int               crop_num,
      bmcv_rect_t*      rects,
      bm_image          input,
      bm_image*         output
);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. bm_handle handle.

- int crop_num

Input parameter. The number of crop small images is required, which is the length of the content pointed to by the pointer rects and the number of output bm_image.

- bmcv_rect_t* rects

  Input parameter. It refers to the information related to the crop, including the starting coordinates, crop width and height. For details, please refer to the data type description below. The pointer points to the information of several crop boxes, and the number of boxes is determined by crop_num.

- bm_image input

  Input parameter. The creation of input bm_image requires the external calling of bmcv_image _create. Image memory can use bm_image_alloc_dev_mem or use bm_image_copy_host_to_device to applicate memory, or use bmcv_image_attach to attach existing memory.

- bm_image* output

  Output parameter. The pointer of output bm_image whose number is crop_num. The creation of bm_image requires the external calling of bm_image_create. New image memory can be opened through bm_image_alloc_dev_mem. Users can also use bmcv_image_attach to attach the existing memory. If users do not actively allocate, the memory will be allocated automatically within the API.

Return parameter description:

- BM_SUCCESS: success
- Other: failed

Data type description:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;
```

- start_x describes the starting horizontal coordinate of where the crop image is located in the original image. It starts at 0 from left to right and takes values in the range [0, width).
- start_y describes the starting vertical coordinate of where the crop image is located in the original image. It starts at 0 from top to bottom and takes values in the range [0, height).
- crop_w describes the width of the crop image, that is, the width of the corresponding output image.
- crop_h describes the height of the crop image, that is, the height corresponding to the output image.

Supported format

Crop currently supports the following image_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_GRAY

bm1684 crop currently supports the following data_type

num	data_type
1	DATA_TYPE_EXT_FLOAT32
2	DATA_TYPE_EXT_1N_BYTE
3	DATA_TYPE_EXT_1N_BYTE_SIGNED

bm1684x crop currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling bmcv_image_crop(), you must ensure that the input image memory has been applied.
2. Data_type and image_format of input must be the same.
3. To avoid memory overruns, start_x + crop_w must be less than or equal to the width of the input image; start_y + crop_h must be less than or equal to the height of the input image.

Code example:

```
int channel   = 3;
int in_w      = 400;
int in_h      = 400;
int out_w     = 800;
int out_h     = 800;
int    dev_id = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
        new unsigned char[channel * in_w * in_h],
```

(continues on next page)

(continued from previous page)

```
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * out_w * out_h],
        std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * in_w * in_h; i++) {
    src_data[i] = rand() % 255;
}
// calculate res
bmcv_rect_t crop_attr;
crop_attr.start_x   = 0;
crop_attr.start_y   = 0;
crop_attr.crop_w    = 50;
crop_attr.crop_h    = 50;
bm_image input, output;
bm_image_create(handle,
        in_h,
        in_w,
        FORMAT_RGB_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
        out_h,
        out_w,
        FORMAT_RGB_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_crop(handle, 1, &crop_attr, input, &output)) {
    std::cout << "bmcv_copy_to error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.7 bmcv_image_resize

The interface is used to change image size, such as zoom in, zoom out, matting and other functions.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_resize(
        bm_handle_t handle,
        int input_num,
        bmcv_resize_image resize_attr[4],
        bm_image* input,
        bm_image* output
);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. bm_handle handle.

- int input_num

  Input parameter. Input the number of images, up to 4. If input_num > 1, then multiple input images must be stored continuously (you can use bm_image_alloc_contiguous_mem to apply for continuous space for multiple images).

- bmcv_resize_image resize_attr [4]

  Input parameter. The resize parameter corresponding to each image. Support up to 4 images.

- bm_image* input

  Input parameter. Input bm_image. Each bm_image requires an external call of bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to applicate memory, or use bmcv_image_attach to attach existing memory.

- bm_image* output

  Output parameter. Output bm_image. Each bm_image requires an external call to bmcv_image_create and create the image memory through bm_image_alloc_dev_mem to open up new memory, or use bmcv_image_attach to attach the existing memory. If it is not actively allocated, it will be allocated within the API itself.

Return parameter description:

- BM_SUCCESS: success

- Other: failed

Data type description:

```
typedef struct bmcv_resize_s{
      int start_x;
      int start_y;
      int in_width;
      int in_height;
      int out_width;
      int out_height;
}bmcv_resize_t;

typedef struct bmcv_resize_image_s{
      bmcv_resize_t *resize_img_attr;
      int roi_num;
      unsigned char stretch_fit;
      unsigned char padding_b;
      unsigned char padding_g;
      unsigned char padding_r;
      unsigned int interpolation;
}bmcv_resize_image;
```

- bmcv_resize_image describes the resize configuration information in an image.
- roi_num describes the total number of subimages in an image that need to be resized.
- stretch_fit indicates whether the image is scaled according to the original scale. 1 indicates that it is not necessary to scale according to the original scale, and 0 indicates that it is scaled according to the original scale. When this method is adopted, the places in the resulting image that are scaled will be filled with specific values.
- padding_b means when stretch_fit is set to 0, the filled value on channel b.
- padding_r means when stretch_fit is set to 0, the filled value on channel r.
- padding_g means when stretch_fit is set to 0, the filled value on channel g.
- interpolation represents the algorithm used in the thumbnail. BMCV_INTER_NEAREST represents the nearest neighbor algorithm, BMCV_INTER_LINEAR represents the linear interpolation algorithm,BMCV_INTER_BICUBIC represents the bi-triple interpolation algorithm.

  bm1684 supports BMCV_INTER_NEAREST,BMCV_INTER_LINEAR,BMCV_INTER_BICU

  bm1684x supports BMCV_INTER_NEAREST,BMCV_INTER_LINEAR.
- start_x describes the start abscissa of resize (relative to the original image), which is commonly used for matting function.
- start_y describes the start ordinate of resize (relative to the original image), which is commonly used for matting function.
- in_width describes the width of the crop image.
- in_height describes the height of the crop image.

- out_width describes the width of the output image.
- out_height describes the height of the output image.

Code example:

```
int image_num = 4;
int crop_w = 711, crop_h = 400, resize_w = 711, resize_h = 400;
int image_w = 1920, image_h = 1080;
int img_size_i = image_w * image_h * 3;
int img_size_o = resize_w * resize_h * 3;
std::unique_ptr<unsigned char[]> img_data(
        new unsigned char[img_size_i * image_num]);
std::unique_ptr<unsigned char[]> res_data(
        new unsigned char[img_size_o * image_num]);
memset(img_data.get(), 0x11, img_size_i * image_num);
memset(res_data.get(), 0, img_size_o * image_num);
bmcv_resize_image resize_attr[image_num];
bmcv_resize_t resize_img_attr[image_num];
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  resize_img_attr[img_idx].start_x = 0;
  resize_img_attr[img_idx].start_y = 0;
  resize_img_attr[img_idx].in_width = crop_w;
  resize_img_attr[img_idx].in_height = crop_h;
  resize_img_attr[img_idx].out_width = resize_w;
  resize_img_attr[img_idx].out_height = resize_h;
}
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  resize_attr[img_idx].resize_img_attr = &resize_img_attr[img_idx];
  resize_attr[img_idx].roi_num = 1;
  resize_attr[img_idx].stretch_fit = 1;
  resize_attr[img_idx].interpolation = BMCV_INTER_NEAREST;
}

bm_image input[image_num];
bm_image output[image_num];
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  int input_data_type = DATA_TYPE_EXT_1N_BYTE;
  bm_image_create(handle,
      image_h,
      image_w,
      FORMAT_BGR_PLANAR,
      (bm_image_data_format_ext)input_data_type,
      &input[img_idx]);
}
bm_image_alloc_contiguous_mem(image_num, input, 1);
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  unsigned char * input_img_data = img_data.get() + img_size_i * img_idx;
  bm_image_copy_host_to_device(input[img_idx],
  (void **)&input_img_data);
}
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  int output_data_type = DATA_TYPE_EXT_1N_BYTE;
```

(continues on next page)

(continued from previous page)

```
  bm_image_create(handle,
      resize_h,
      resize_w,
      FORMAT_BGR_PLANAR,
      (bm_image_data_format_ext)output_data_type,
      &output[img_idx]);
}
bm_image_alloc_contiguous_mem(image_num, output, 1);
bmcv_image_resize(handle, image_num, resize_attr, input, output);
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  unsigned char *res_img_data = res_data.get() + img_size_o * img_idx;
  bm_image_copy_device_to_host(output[img_idx],
                               (void **)&res_img_data);
}
bm_image_free_contiguous_mem(image_num, input);
bm_image_free_contiguous_mem(image_num, output);
for(int i = 0; i < image_num; i++) {
  bm_image_destroy(input[i]);
  bm_image_destroy(output[i]);
}
```

Supported format::

1. resize supports the conversion of the following image_format:
2. resize supports the conversion between data types in the following cases:

bm1684 supports the following data_type:

- 1 vs 1 : one image resizes (crop) one image
- 1 vs N : one image resizes (crop) multiple image

1	DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE	1 vs 1
2	DATA_TYPE_EXT_FLOAT32 ——> DATA_TYPE_EXT_FLOAT32	1 vs 1
3	DATA_TYPE_EXT_4N_BYTE ——> DATA_TYPE_EXT_4N_BYTE	1 vs 1
4	DATA_TYPE_EXT_4N_BYTE ——> DATA_TYPE_EXT_1N_BYTE	1 vs 1
5	DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE	1 vs N
6	DATA_TYPE_EXT_FLOAT32 ——> DATA_TYPE_EXT_FLOAT32	1 vs N
7	DATA_TYPE_EXT_4N_BYTE ——> DATA_TYPE_EXT_1N_BYTE	1 vs N

bm1684x supports the following data_type:

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

Note:

1. Before calling bmcv_image_resize(), users must ensure that the input image memory has been applied.
2. bm1684:the maximum size supported is 2048*2048, the minimum size is 16*16, and the maximum zoom ratio is 32.

   bm1684x:the maximum size supported is 8192*8192, the minimum size is 8*8, and the maximum zoom ratio is 128.

5.8 bmcv_image_convert_to

The interface is used to do the linear change of image pixels. The specific data relationship can be expressed by the following formula:

$$y = kx + b$$

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_convert_to (
      bm_handle_t handle,
      int input_num,
      bmcv_convert_to_attr convert_to_attr,
      bm_image* input,
      bm_image* output
);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- int input_num

  Input parameter. The number of input images. If input_num > 1, then multiple input images must be stored continuously (users can use bm_image_alloc_contiguous_mem to apply continuous space for multiple images).

- bmcv_convert_to_attr convert_to_attr

  Input parameter. The configuration parameter corresponding to each image.

- bm_image* input

  Input parameter. The input bm_image. The creation of each bm_image require the calling of bmcv_image_create externally. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image* output

  Output parameter. The output bm_image. The creation of each bm_image require the calling of bmcv_image_create externally. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

Return Value Description:

- BM_SUCCESS: success
- Other: failed

Data Type Description:

```
typedef struct bmcv_convert_to_attr_s{
      float alpha_0;
      float beta_0;
      float alpha_1;
      float beta_1;
      float alpha_2;
      float beta_2;
} bmcv_convert_to_attr;
```

- alpha_0 describes the coefficient of the linear transformation of the 0th channel
- beta_0 describes the offset of the linear transformation of the 0th channel
- alpha_1 describes the coefficient of the linear transformation of the 1st channel
- beta_1 describes the offset of linear transformation of the 1st channel
- alpha_2 describes the coefficient of the linear transformation of the 2nd channel
- beta_2 describes the offset of linear transformation of the 2nd channel

Code Example:

```
int image_num = 4, image_channel = 3;
int image_w = 1920, image_h = 1080;
bm_image input_images[4], output_images[4];
bmcv_convert_to_attr convert_to_attr;
convert_to_attr.alpha_0 = 1;
convert_to_attr.beta_0 = 0;
convert_to_attr.alpha_1 = 1;
convert_to_attr.beta_1 = 0;
convert_to_attr.alpha_2 = 1;
convert_to_attr.beta_2 = 0;
int img_size = image_w * image_h * image_channel;
std::unique_ptr<unsigned char[]> img_data(
      new unsigned char[img_size * image_num]);
std::unique_ptr<unsigned char[]> res_data(
      new unsigned char[img_size * image_num]);
memset(img_data.get(), 0x11, img_size * image_num);
```

(continues on next page)

(continued from previous page)

```
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  bm_image_create(handle,
        image_h,
        image_w,
        FORMAT_BGR_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &input_images[img_idx]);
}
bm_image_alloc_contiguous_mem(image_num, input_images, 0);
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  unsigned char *input_img_data = img_data.get() + img_size * img_idx;
  bm_image_copy_host_to_device(input_images[img_idx],
        (void **)&input_img_data);
}

for (int img_idx = 0; img_idx < image_num; img_idx++) {
  bm_image_create(handle,
        image_h,
        image_w,
        FORMAT_BGR_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &output_images[img_idx]);
}
bm_image_alloc_contiguous_mem(image_num, output_images, 1);
bmcv_image_convert_to(handle, image_num, convert_to_attr, input_images,
        output_images);
for (int img_idx = 0; img_idx < image_num; img_idx++) {
  unsigned char *res_img_data = res_data.get() + img_size * img_idx;
  bm_image_copy_device_to_host(output_images[img_idx],
        (void **)&res_img_data);
}
bm_image_free_contiguous_mem(image_num, input_images);
bm_image_free_contiguous_mem(image_num, output_images);
for(int i = 0; i < image_num; i++) {
  bm_image_destroy(input_images[i]);
  bm_image_destroy(output_images[i]);
}
```

Supported Format:

1. This interface supports the conversion of the following image_format:

· FORMAT_BGR_PLANAR ——> FORMAT_BGR_PLANAR

· FORMAT_RGB_PLANAR ——> FORMAT_RGB_PLANAR

· FORMAT_GRAY ——> FORMAT_GRAY

2. This interface supports the conversion of data type in the following cases:

bm1684 supports the fllowing data_type:

· DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_FLOAT32

- DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE
- DATA_TYPE_EXT_1N_BYTE_SIGNED ——> DATA_TYPE_EXT_1N_BYTE_SIGNED
- DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE_SIGNED
- DATA_TYPE_EXT_FLOAT32 ——> DATA_TYPE_EXT_FLOAT32
- DATA_TYPE_EXT_4N_BYTE ——> DATA_TYPE_EXT_FLOAT32

bm1684x supports the fllowing data_type:

- DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_FLOAT32
- DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE
- DATA_TYPE_EXT_1N_BYTE_SIGNED ——> DATA_TYPE_EXT_1N_BYTE_SIGNED
- DATA_TYPE_EXT_1N_BYTE ——> DATA_TYPE_EXT_1N_BYTE_SIGNED
- DATA_TYPE_EXT_FLOAT32 ——> DATA_TYPE_EXT_FLOAT32

Note:

1. Before calling bmcv_image_convert_to(), users must ensure that the input image memory has been applied.
2. The input width, height, data _type and image_format must be the same.
3. The output width, height, data_type and image_format must be the same.
4. The width and height of the input image must be equal to the width and height of the output image.
5. image_num must be greater than 0.
6. The stride of the output image must be equal to the width.
7. The stride of the input image must be greater than or equal to the width.
8. bm1684 supports the maximum size is 2048*2048 and the minimum size is 16*16. When the image format is DATA_TYPE_EXT_4N_BYTE, w*h should not be greater than 1024*1024.

   bm1684x supports the maximum size is 4096*4096 and the minimum size is 16*16.

5.9 bmcv_image_csc_convert_to

The API can combine crop, color-space-convert, resize, padding, convert_to, and any number of functions for multiple images.

```
bm_status_t bmcv_image_csc_convert_to(
    bm_handle_t         handle,
    int                 in_img_num,
    bm_image*           input,
```

(continues on next page)

(continued from previous page)

```
    bm_image*           output,
    int*                crop_num_vec = NULL,
    bmcv_rect_t*        crop_rect = NULL,
    bmcv_padding_atrr_t* padding_attr = NULL,
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
    csc_type_t          csc_type = CSC_MAX_ENUM,
    csc_matrix_t*       matrix = NULL,
    bmcv_convert_to_attr* convert_to_attr);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int in_img_num

  Input parameter. The number of input bm_image.

- bm_image* input

  Input parameter. Input bm_image object pointer whose length to the space is decided by in_img_num.

- bm_image* output

  Output parameter. Output bm_image image object pointer whose length to the space is jointly decided by in_img_num and crop_num_vec, that is, the sum of the number of crops of all input images.

- int* crop_num_vec = NULL

  Input parameter. The pointer points to the number of crops for each input image, and the length of the pointing space is decided by in_img_num. NULL can be filled in if the crop function is not used.

- bmcv_rect_t * crop_rect = NULL

  Input parameter. The specific format is defined as follows:

  ```
  typedef struct bmcv_rect {
      int start_x;
      int start_y;
      int crop_w;
      int crop_h;
  } bmcv_rect_t;
  ```

  The parameters of the crop on the input image corresponding to each output bm_image object, including the X coordinate of the starting point, the Y coordinate of the starting

point, the width of the crop image and the height of the crop image. The top left vertex of the image is used as the coordinate origin. If you do not use the crop function, you can fill in NULL.

- bmcv_padding_atrr_t* padding_attr = NULL

  Input parameter. The location information of the target thumbnails of all crops in the dst image and the pixel values of each channel to be padding. If you do not use the padding function, you can set the function to NULL.

```
typedef struct bmcv_padding_atrr_s {
    unsigned int  dst_crop_stx;
    unsigned int  dst_crop_sty;
    unsigned int  dst_crop_w;
    unsigned int  dst_crop_h;
    unsigned char padding_r;
    unsigned char padding_g;
    unsigned char padding_b;
    int           if_memset;
} bmcv_padding_atrr_t;
```

  1. Offset information of the top left corner vertex of the target thumbnail relative to the dst image origin (top left corner): dst_crop_stx and dst_crop_sty;
  2. The width and height of the target thumbnail after resize: dst_crop_w and dst_crop_h;
  3. If dst image is in RGB format, the required pixel value information of each channel: padding_r, padding_g, padding_b. When if_memset=1, it is valid. If it is a GRAY image, you can set all three values to the same value;
  4. if_memset indicates whether to memset dst image according to the padding value of each channel within the API. Only images in RGB and GRAY formats are supported. If it is set to 0, users need to call the API in bmlib according to the pixel value information of padding before calling the API to directly perform memset operation on device memory. If users are not concerned about the value of padding, it can be set to 0 to ignore this step.

- bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR

  Input parameter. Resize algorithm selection, including BMCV_INTER_NEAREST, BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC, which is the bilinear difference by default.

  – bm1684 supports : BMCV_INTER_NEAREST,

    BMCV_INTER_LINEAR, BMCV_INTER_BICUBIC.

  – bm1684x supports:

    BMCV_INTER_NEAREST, BMCV_INTER_LINEAR.

- csc_type_t csc_type = CSC_MAX_ENUM

  Input parameters. color space convert Parameter type selection, fill CSC_MAX_ENUM then use the default value. The default is CSC_YCbCr2RGB_BT601 or CSC_RGB2YCbCr_BT601. The supported types include:

CSC_YCbCr2RGB_BT601
CSC_YPbPr2RGB_BT601
CSC_RGB2YCbCr_BT601
CSC_YCbCr2RGB_BT709
CSC_RGB2YCbCr_BT709
CSC_RGB2YPbPr_BT601
CSC_YPbPr2RGB_BT709
CSC_RGB2YPbPr_BT709
CSC_USER_DEFINED_MATRIX
CSC_MAX_ENUM

- csc_matrix_t* matrix = NULL

Input parameter for the selection of color space convert parameter type. Fill in CSC_MAX_ENUM to use the default value, which is by default CSC_YCbCr2RGB_BT601 or CSC_RGB2YCbCr_BT601. The supported types include:

```
typedef struct {
    int csc_coe00;
    int csc_coe01;
    int csc_coe02;
    int csc_add0;
    int csc_coe10;
    int csc_coe11;
    int csc_coe12;
    int csc_add1;
    int csc_coe20;
    int csc_coe21;
    int csc_coe22;
    int csc_add2;
} __attribute__((packed)) csc_matrix_t;
```

- bmcv_convert_to_attr* convert_to_attr

Input parameter for linear transformation coefficient.

```
typedef struct bmcv_convert_to_attr_s{
        float alpha_0;
        float beta_0;
        float alpha_1;
        float beta_1;
        float alpha_2;
```

(continues on next page)

(continued from previous page)

```
    float beta_2;
} bmcv_convert_to_attr;
```

- alpha_0 describes the coefficient of the linear transformation of the 0th channel
- beta_0 describes the offset of the linear transformation of the 0th channel
- alpha_1 describes the coefficient of the linear transformation of the 1st channel
- beta_1 describes the offset of linear transformation of the 1st channel
- alpha_2 describes the coefficient of the linear transformation of the 2nd channel
- beta_2 describes the offset of linear transformation of the 2nd channel

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

- bm1684x supports the following:

1. bm1684x supports the following data_type:

num	input data_type	output data_type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

2. bm1684x supports the following color formats of input bm_image:

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_COMPRESSED
16	FORMAT_YUV444_PACKED
17	FORMAT_YVU444_PACKED
18	FORMAT_YUV422_YUYV
19	FORMAT_YUV422_YVYU
20	FORMAT_YUV422_UYVY
21	FORMAT_YUV422_VYUY

3. bm1684x supports the following color formats of output bm_image:

num	output image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY
12	FORMAT_RGBYP_PLANAR
13	FORMAT_BGRP_SEPARATE
14	FORMAT_HSV180_PACKED
15	FORMAT_HSV256_PACKED

4. bm1684x vpp does not support FORMAT_COMPRESSED to FORMAT_HSV180_PACKED or FORMAT_HSV256_PACKED

5. The zoom ratio of the image ((crop.width / output.width) and (crop.height / output.height)) is limited to 1 / 128 ~ 128.
6. The width and height (src.width, src.height, dst.width, dst.height) of input and output are limited to 8 ~ 8192.
7. The input must be associated with device memory, otherwise, a failure will be returned.
8. The usage of FORMAT_COMPRESSED format is described in the bm1684 section.

· bm1684 supports the following:

1. The format and some requirements that the API needs to meet are shown in the following table:

src format	dst format	Other Limitation
RGB_PACKED	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
BGR_PACKED	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
RGB_PLANAR	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
BGR_PLANAR	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
RGBP_SEPARATE	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
BGRP_SEPARATE	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
GRAY	GRAY	Condition 1
YUV420P	RGB_PLANAR	Condition 4
	BGR_PLANAR	Condition 4
NV12	RGB_PLANAR	Condition 4
	BGR_PLANAR	Condition 4
COMPRESSED	RGB_PLANAR	Condition 4
	BGR_PLANAR	Condition 4

of which:

· Condition 1: src.width >= crop.x + crop.width, src.height >= crop.y + crop.height

· Condition 2: src.width, src.height, dst.width, dst.height must be an integral multiple of 2, src.width >= crop.x + crop.width, src.height >= crop.y + crop.heigh

· Condition 3: dst.width, dst.height must be an integral multiple of 2, src.width == dst.width, src.height == dst.height, crop.x == 0, crop.y == 0, src.width >= crop.x + crop.width, src.height >= crop.y + crop.height

· Condition 4: src.width, src.height must be an integral multiple of 2, src.width >= crop.x + crop.width, src.height >= crop.y + crop.height

2. The device mem of input bm_image cannot be on heap0.

3. The stride of all input and output images must be 64 aligned.
4. The addresses of all input and output images must be aligned with 32 byte.
5. The zoom ratio of the image ((crop.width / output.width) and (crop.height / output.height)) is limited to 1 / 32 ~ 32.
6. The width and height (src.width, src.height, dst.width, dst.height) of input and output are limited to 16 ~ 4096.
7. The input must be associated with device memory, otherwise, a failure will be returned.
8. FORMAT_COMPRESSED is a built-in compression format after VPU decoding. It includes four parts: Y compressed table, Y compressed data, CbCr compressed table and CbCr compressed data. Please note the order of the four parts in bm_image is slightly different from that of the AVFrame in FFMPEG. If you need to attach the device memory data in AVFrame to bm_image, the corresponding relationship is as follows. For details of AVFrame, please refer to "VPU User Manual" .

```
bm_device_mem_t src_plane_device[4];
src_plane_device[0] = bm_mem_from_device((u64)avframe->data[6],
      avframe->linesize[6]);
src_plane_device[1] = bm_mem_from_device((u64)avframe->data[4],
      avframe->linesize[4] * avframe->h);
src_plane_device[2] = bm_mem_from_device((u64)avframe->data[7],
      avframe->linesize[7]);
src_plane_device[3] = bm_mem_from_device((u64)avframe->data[5],
      avframe->linesize[4] * avframe->h / 2);

bm_image_attach(*compressed_image, src_plane_device);
```

5.10 bmcv_image_storage_convert

The interface converts the data corresponding to the source image format into the format data of the target image and fills it in the device memory associated with the target image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_storage_convert(
      bm_handle_t handle,
      int image_num,
      bm_image* input_image,
      bm_image* output_image
);
```

Incoming parameters description:

· bm_handle_t handle

Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int image_num

  Input parameter. The number of input/output images.

- bm_image* input

  Input parameter. Input bm_image object pointer.

- bm_image* output

  Output parameter. Output bm_image object pointer.

Return parameters description:

- BM_SUCCESS: success
- Other: failed

Note

1. The API supports the following formats: - bm1684 supports two-two conversion of all the following formats:

num	image_format	data type
1	FORMAT_RGB_PLANAR	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_4N_BYTE
4	FORMAT_BGR_PLANAR	DATA_TYPE_EXT_FLOAT32
5		DATA_TYPE_EXT_1N_BYTE
6		DATA_TYPE_EXT_4N_BYTE
7	FORMAT_RGB_PACKED	DATA_TYPE_EXT_FLOAT32
8		DATA_TYPE_EXT_1N_BYTE
9		DATA_TYPE_EXT_4N_BYTE
10	FORMAT_BGR_PACKED	DATA_TYPE_EXT_FLOAT32
11		DATA_TYPE_EXT_1N_BYTE
12		DATA_TYPE_EXT_4N_BYTE
13	FORMAT_RGBP_SEPARATE	DATA_TYPE_EXT_FLOAT32
14		DATA_TYPE_EXT_1N_BYTE
15		DATA_TYPE_EXT_4N_BYTE
16	FORMAT_BGRP_SEPARATE	DATA_TYPE_EXT_FLOAT32
17		DATA_TYPE_EXT_1N_BYTE
18		DATA_TYPE_EXT_4N_BYTE
19	FORMAT_NV12	DATA_TYPE_EXT_1N_BYTE
20	FORMAT_NV21	DATA_TYPE_EXT_1N_BYTE
21	FORMAT_NV16	DATA_TYPE_EXT_1N_BYTE
22	FORMAT_NV61	DATA_TYPE_EXT_1N_BYTE
23	FORMAT_YUV420P	DATA_TYPE_EXT_1N_BYTE
24	FORMAT_YUV444P	DATA_TYPE_EXT_1N_BYTE
25	FORMAT_GRAY	DATA_TYPE_EXT_1N_BYTE

If the input and output image objects are not in the above format, return failure.

- bm1684x supports the following data_type:

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

- bm1684x supports the following color formats of input bm_image:

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_COMPRESSED
16	FORMAT_YUV444_PACKED
17	FORMAT_YVU444_PACKED
18	FORMAT_YUV422_YUYV
19	FORMAT_YUV422_YVYU
20	FORMAT_YUV422_UYVY
21	FORMAT_YUV422_VYUY

· bm1684x supports the following color formats of output bm_image:

num	output image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY
12	FORMAT_RGBYP_PLANAR
13	FORMAT_BGRP_SEPARATE
14	FORMAT_HSV180_PACKED
15	FORMAT_HSV256_PACKED

If the input/output image object is not in the above format, a failure will be returned.

2. All input and output bm_image structures must be created in advance, or a failure will

be returned.

3. All the image_format, data_type, width and height of all input bm_image objects must be the same. All the image_format, data_type, width and height of all output bm_image objects must be the same. The width and height of the input and output bm_image object must be the same, or a failure will be returned.

4. image_num indicates the number of input images. If the input image data format is DATA_TYPE_EXT_4N_BYTE, the number of input bm_image object is one, and the number of valid images in 4N is image_num. If the input image data format is not DATA_TYPE_EXT_4N_BYTE, the number of input bm_image is image_num. If the output image data format is DATA_TYPE_EXT_4N_BYTE, the number of output bm_image object is one, and the number of valid images in 4N is image_num. If the output image data format is not DATA_TYPE_EXT_4N_BYTE, the number of output bm_image is image_num.

5. image_num must be greater than or equal to 1 and less than or equal to 4, otherwise, a failure will be returned.

6. All input objects must attach device memory, otherwise, a failure will be returned.

7. If the output object does not attach device memory, the device will externally call bm_image_alloc_dev_mem to apply for internally managed device memory and fills the converted data into device memory.

8. If the input image and output image have the same format a direct success will be returned, and the original data will not be copied to the output image.

9. Currently do not support the image format conversion when image_w > 8192. A failure will be returned when image_w > 8192.

Code example:

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>

int main(int argc, char *argv[]) {
  bm_handle_t handle;
  bm_dev_request(&handle, 0);

  int image_n = 1;
  int image_h = 1080;
  int image_w = 1920;
  bm_image src, dst;
  bm_image_create(handle, image_h, image_w, FORMAT_NV12,
      DATA_TYPE_EXT_1N_BYTE, &src);
```

(continues on next page)

(continued from previous page)

```
    bm_image_create(handle, image_h, image_w, FORMAT_BGR_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &dst);
    std::shared_ptr<u8*> y_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w]);
    std::shared_ptr<u8*> uv_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w / 2]);
    memset((void *)(*y_ptr.get()), 148, image_h * image_w);
    memset((void *)(*uv_ptr.get()), 158, image_h * image_w / 2);
    u8 *host_ptr[] = {*y_ptr.get(), *uv_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);
    bmcv_image_storage_convert(handle, image_n, &src, &dst);
    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_dev_free(handle);
    return 0;
}
```

5.11 bmcv_image_vpp_basic

There is a special video post-processing module VPP on BM1684 and BM1684X. Under certain conditions, it can do the functions of clip, color-space-convert, resize and padding at one time, faster than Tensor Computing Processor. The API can combine crop, color-space-convert, resize, padding and any number of functions for multiple images.

```
bm_status_t bmcv_image_vpp_basic(
    bm_handle_t         handle,
    int                 in_img_num,
    bm_image*           input,
    bm_image*           output,
    int*                crop_num_vec = NULL,
    bmcv_rect_t*        crop_rect = NULL,
    bmcv_padding_atrr_t*  padding_attr = NULL,
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
    csc_type_t          csc_type = CSC_MAX_ENUM,
    csc_matrix_t*       matrix = NULL);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int in_img_num

Input parameter. The number of input bm_image.

- bm_image* input

  Input parameter. Input bm_image object pointer whose length to the space is decided by in_img_num.

- bm_image* output

  Output parameter. Output bm_image image object pointer whose length to the space is jointly decided by in_img_num and crop_num_vec, that is, the sum of the number of crops of all input images.

- int* crop_num_vec = NULL

  Input parameter. The pointer points to the number of crops for each input image, and the length of the pointing space is decided by in_img_num. NULL can be filled in if the crop function is not used.

- bmcv_rect_t * crop_rect = NULL

  Input parameter. The specific format is defined as follows:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;
```

  The parameters of the crop on the input image corresponding to each output bm_image object, including the X coordinate of the starting point, the Y coordinate of the starting point, the width of the crop image and the height of the crop image. The top left vertex of the image is used as the coordinate origin. If you do not use the crop function, you can fill in NULL.

- bmcv_padding_atrr_t* padding_attr = NULL

  Input parameter. The location information of the target thumbnails of all crops in the dst image and the pixel values of each channel to be padding. If you do not use the padding function, you can set the function to NULL.

```
typedef struct bmcv_padding_atrr_s {
    unsigned int  dst_crop_stx;
    unsigned int  dst_crop_sty;
    unsigned int  dst_crop_w;
    unsigned int  dst_crop_h;
    unsigned char padding_r;
    unsigned char padding_g;
    unsigned char padding_b;
    int           if_memset;
} bmcv_padding_atrr_t;
```

1. Offset information of the top left corner vertex of the target thumbnail relative to the dst image origin (top left corner): dst_crop_stx and dst_crop_sty;
2. The width and height of the target thumbnail after resize: dst_crop_w and dst_crop_h;
3. If dst image is in RGB format, the required pixel value information of each channel: padding_r, padding_g, padding_b. When if_memset=1, it is valid. If it is a GRAY image, you can set all three values to the same value;
4. if_memset indicates whether to memset dst image according to the padding value of each channel within the API. Only images in RGB and GRAY formats are supported. If it is set to 0, users need to call the API in bmlib according to the pixel value information of padding before calling the API to directly perform memset operation on device memory. If users are not concerned about the value of padding, it can be set to 0 to ignore this step.

· bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR

Input parameter. Resize algorithm selection, including BMCV_INTER_NEAREST, BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC, which is the bilinear difference by default.

- bm1684 supports : BMCV_INTER_NEAREST,

  BMCV_INTER_LINEAR, BMCV_INTER_BICUBIC.

- bm1684x supports:

  BMCV_INTER_NEAREST, BMCV_INTER_LINEAR.

· csc_type_t csc_type = CSC_MAX_ENUM

Input parameters. color space convert Parameter type selection, fill CSC_MAX_ENUM then use the default value. The default is CSC_YCbCr2RGB_BT601 or CSC_RGB2YCbCr_BT601. The supported types include:

CSC_YCbCr2RGB_BT601
CSC_YPbPr2RGB_BT601
CSC_RGB2YCbCr_BT601
CSC_YCbCr2RGB_BT709
CSC_RGB2YCbCr_BT709
CSC_RGB2YPbPr_BT601
CSC_YPbPr2RGB_BT709
CSC_RGB2YPbPr_BT709
CSC_USER_DEFINED_MATRIX
CSC_MAX_ENUM

- csc_matrix_t* matrix = NULL

Input parameter for the selection of color space convert parameter type. Fill in CSC_MAX_ENUM to use the default value, which is by default CSC_YCbCr2RGB_BT601 or CSC_RGB2YCbCr_BT601. The supported types include:

```
typedef struct {
    int csc_coe00;
    int csc_coe01;
    int csc_coe02;
    int csc_add0;
    int csc_coe10;
    int csc_coe11;
    int csc_coe12;
    int csc_add1;
    int csc_coe20;
    int csc_coe21;
    int csc_coe22;
    int csc_add2;
} __attribute__((packed)) csc_matrix_t;
```

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

- bm1684x supports the following:

1. bm1684x supports the following data_type:

num	input data_type	output data_type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

2. bm1684x supports the following color formats of input bm_image:

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_COMPRESSED
16	FORMAT_YUV444_PACKED
17	FORMAT_YVU444_PACKED
18	FORMAT_YUV422_YUYV
19	FORMAT_YUV422_YVYU
20	FORMAT_YUV422_UYVY
21	FORMAT_YUV422_VYUY

3. bm1684x supports the following color formats of output bm_image:

num	output image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY
12	FORMAT_RGBYP_PLANAR
13	FORMAT_BGRP_SEPARATE
14	FORMAT_HSV180_PACKED
15	FORMAT_HSV256_PACKED

4. bm1684x vpp does not support FORMAT_COMPRESSED to FORMAT_HSV180_PACKED or FORMAT_HSV256_PACKED

5. The zoom ratio of the image ((crop.width / output.width) and (crop.height / output.height)) is limited to 1 / 128 ~ 128.
6. The width and height (src.width, src.height, dst.width, dst.height) of input and output are limited to 8 ~ 8192.
7. The input must be associated with device memory, otherwise, a failure will be returned.
8. The usage of FORMAT_COMPRESSED format is described in the bm1684 section.

· bm1684 supports the following:

1. The format and some requirements that the API needs to meet are shown in the following table:

src format	dst format	Other Limitation
RGB_PACKED	RGB_PACKED	Condition 1
	RGB_PLANAR	Condition 1
	BGR_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
BGR_PACKED	RGB_PACKED	Condition 1
	RGB_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	BGR_PLANAR	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
RGB_PLANAR	RGB_PACKED	Condition 1
	RGB_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	BGR_PLANAR	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
BGR_PLANAR	RGB_PACKED	Condition 1
	RGB_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	BGR_PLANAR	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
RGBP_SEPARATE	RGB_PACKED	Condition 1
	RGB_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	BGR_PLANAR	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
BGRP_SEPARATE	RGB_PACKED	Condition 1

continues on next page

Table 5.2 – continued from previous page

src format	dst format	Other Limitation
	RGB_PLANAR	Condition 1
	BGR_PACKED	Condition 1
	BGR_PLANAR	Condition 1
	RGBP_SEPARATE	Condition 1
	BGRP_SEPARATE	Condition 1
GRAY	GRAY	Condition 1
YUV420P	YUV420P	Condition 2
COMPRESSED	YUV420P	Condition 2
RGB_PACKED	YUV420P	Condition 3
RGB_PLANAR		Condition 3
BGR_PACKED		Condition 3
BGR_PLANAR		Condition 3
RGBP_SEPARATE		Condition 3
BGRP_SEPARATE		Condition 3
YUV420P	RGB_PACKED	Condition 4
	RGB_PLANAR	Condition 4
	BGR_PACKED	Condition 4
	BGR_PLANAR	Condition 4
	RGBP_SEPARATE	Condition 4
	BGRP_SEPARATE	Condition 4
NV12	RGB_PACKED	Condition 4
	RGB_PLANAR	Condition 4
	BGR_PACKED	Condition 4
	BGR_PLANAR	Condition 4
	RGBP_SEPARATE	Condition 4
	BGRP_SEPARATE	Condition 4
COMPRESSED	RGB_PACKED	Condition 4
	RGB_PLANAR	Condition 4
	BGR_PACKED	Condition 4
	BGR_PLANAR	Condition 4
	RGBP_SEPARATE	Condition 4
	BGRP_SEPARATE	Condition 4

of which:

- Condition 1: src.width >= crop.x + crop.width, src.height >= crop.y + crop.height
- Condition 2: src.width, src.height, dst.width, dst.height must be an integral multiple of 2, src.width >= crop.x + crop.width, src.height >= crop.y + crop.heigh
- Condition 3: dst.width, dst.height must be an integral multiple of 2, src.width == dst.width, src.height == dst.height, crop.x == 0, crop.y == 0, src.width >= crop.x + crop.width, src.height >= crop.y + crop.height
- Condition 4: src.width, src.height must be an integral multiple of 2, src.width >= crop.x + crop.width, src.height >= crop.y + crop.height

2. The device mem of input bm_image cannot be on heap0.
3. The stride of all input and output images must be 64 aligned.
4. The addresses of all input and output images must be aligned with 32 byte.
5. The zoom ratio of the image ((crop.width / output.width) and (crop.height / output.height)) is limited to 1 / 32 ~ 32.
6. The width and height (src.width, src.height, dst.width, dst.height) of input and output are limited to 16 ~ 4096.
7. The input must be associated with device memory, otherwise, a failure will be returned.
8. FORMAT_COMPRESSED is a built-in compression format after VPU decoding. It includes four parts: Y compressed table, Y compressed data, CbCr compressed table and CbCr compressed data. Please note the order of the four parts in bm_image is slightly different from that of the AVFrame in FFMPEG. If you need to attach the device memory data in AVFrame to bm_image, the corresponding relationship is as follows. For details of AVFrame, please refer to "VPU User Manual" .

```
bm_device_mem_t src_plane_device[4];
src_plane_device[0] = bm_mem_from_device((u64)avframe->data[6],
      avframe->linesize[6]);
src_plane_device[1] = bm_mem_from_device((u64)avframe->data[4],
      avframe->linesize[4] * avframe->h);
src_plane_device[2] = bm_mem_from_device((u64)avframe->data[7],
      avframe->linesize[7]);
src_plane_device[3] = bm_mem_from_device((u64)avframe->data[5],
      avframe->linesize[4] * avframe->h / 2);

bm_image_attach(*compressed_image, src_plane_device);
```

5.12 bmcv_image_vpp_convert

The API converts the input image format into the output image format, and supports the function of crop and resize. It supports the output of multiple crops from one input and resizing to the output image size.

```
bm_status_t bmcv_image_vpp_convert(
    bm_handle_t  handle,
    int output_num,
    bm_image input,
    bm_image *output,
    bmcv_rect_t *crop_rect,
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR
);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int output_num

  Output parameter. The number of output bm_image, which is equal to the number of crop of src image. One src crop outputs one dst bm_image.

- bm_image input

  Input parameters. Input bm_image object.

- bm_image* output

  Output parameter. Output bm_image object pointer.

- bmcv_rect_t * crop_rect

  Input parameter. The specific format is defined as follows:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;
```

  The parameters of the crop on the input image corresponding to each output bm_image object, including the X coordinate of the starting point, the Y coordinate of the starting point, the width of the crop image and the height of the crop image.

- bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR

  Input parameters. Resize algorithm selection, including BMCV_INTER_NEAREST 、BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC, which is bilinear difference by default.

  bm1684 supports BMCV_INTER_NEAREST,BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC.

  bm1684x supports BMCV_INTER_NEAREST and BMCV_INTER_LINEAR.

Return parameter description:

- BM_SUCCESS: success
- Other: failed

Note:

1. The format and some requirements that the API needs to meet are the same as the table of bmcv_image_vpp_basic.

2. The width and height (src.width, src.height, dst.width, dst.height) of bm1684 input and output are limited to 16 ~ 4096.

   The width and height (src.width, src.height, dst.width, dst.height) of bm1684x input and output are limited to 8 ~ 8192, and zoom 128 times.

3. The input must be associated with device memory, otherwise, a failure will be returned.

4. FORMAT_COMPRESSED is a built-in compression format after VPU decoding. It includes four parts: Y compressed table, Y compressed data, CbCr compressed table and CbCr compressed data. Please note the order of the four parts in bm_image is slightly different from that of the AVFrame in FFMPEG. If you need to attach the device memory data in AVFrame to bm_image, the corresponding relationship is as follows. For details of AVFrame, please refer to "VPU User Manual".

```
bm_device_mem_t src_plane_device[4];
src_plane_device[0] = bm_mem_from_device((u64)avframe->data[6],
      avframe->linesize[6]);
src_plane_device[1] = bm_mem_from_device((u64)avframe->data[4],
      avframe->linesize[4] * avframe->h);
src_plane_device[2] = bm_mem_from_device((u64)avframe->data[7],
      avframe->linesize[7]);
src_plane_device[3] = bm_mem_from_device((u64)avframe->data[5],
      avframe->linesize[4] * avframe->h / 2);

bm_image_attach(*compressed_image, src_plane_device);
```

Code example:

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include <memory>
#include "stdio.h"
#include "stdlib.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  bm_handle_t handle;
  int          image_h     = 1080;
  int          image_w     = 1920;
  bm_image     src, dst[4];
  bm_dev_request(&handle, 0);
  bm_image_create(handle, image_h, image_w, FORMAT_NV12,
      DATA_TYPE_EXT_1N_BYTE, &src);
  bm_image_alloc_dev_mem(src, 1);
  for (int i = 0; i < 4; i++) {
    bm_image_create(handle,
      image_h / 2,
```

(continues on next page)

(continued from previous page)

```
            image_w / 2,
            FORMAT_BGR_PACKED,
            DATA_TYPE_EXT_1N_BYTE,
            dst + i);
        bm_image_alloc_dev_mem(dst[i]);
    }
    std::unique_ptr<u8 []> y_ptr(new u8[image_h * image_w]);
    std::unique_ptr<u8 []> uv_ptr(new u8[image_h * image_w / 2]);
    memset((void *)(y_ptr.get()), 148, image_h * image_w);
    memset((void *)(uv_ptr.get()), 158, image_h * image_w / 2);
    u8 *host_ptr[] = {y_ptr.get(), uv_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);

    bmcv_rect_t rect[] = {{0, 0, image_w / 2, image_h / 2},
            {0, image_h / 2, image_w / 2, image_h / 2},
            {image_w / 2, 0, image_w / 2, image_h / 2},
            {image_w / 2, image_h / 2, image_w / 2, image_h / 2}};

    bmcv_image_vpp_convert(handle, 4, src, dst, rect);

    for (int i = 0; i < 4; i++) {
        bm_image_destroy(dst[i]);
    }

    bm_image_destroy(src);
    bm_dev_free(handle);
    return 0;
}
```

5.13 bmcv_image_vpp_convert_padding

The effect of image padding is implemented by using VPP hardware resources and memset operation on dst image. This effect is done through the function of dst crop of vpp. Generally speaking, it is to fill a small image into a large one. Users can crop multiple target images from one src image. For each small target image, users can complete the csc and resize operation at one time, and then fill it into the large image according to its offset information in the large image. The number of crop at a time cannot exceed 256.

```
bm_status_t bmcv_image_vpp_convert_padding(
    bm_handle_t          handle,
    int                  output_num,
    bm_image             input,
    bm_image *           output,
    bmcv_padding_atrr_t * padding_attr,
    bmcv_rect_t *        crop_rect = NULL,
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int output_num

  Output parameter. The number of output bm_image, which is equal to the number of crop of SRC image. One src crop outputs one dst bm_image.

- bm_image input

  Input parameter. Input bm_image object.

- bm_image* output

  Output parameter. Output bm_image object pointer.

- bmcv_padding_atrr_t * padding_attr

  Input parameter. The location information of the target thumbnail of src crop in the dst image and the pixel values of each channel to be pdding.

```
typedef struct bmcv_padding_atrr_s {
    unsigned int    dst_crop_stx;
    unsigned int    dst_crop_sty;
    unsigned int    dst_crop_w;
    unsigned int    dst_crop_h;
    unsigned char padding_r;
    unsigned char padding_g;
    unsigned char padding_b;
    int          if_memset;
} bmcv_padding_atrr_t;
```

1. Offset information of the top left corner of the target small image relative to the dst image origin (top left corner): dst_crop_stx and dst_crop_sty;
2. Width and height of the target small image after resizing: dst_crop_w and dst_crop_h;
3. If dst image is in RGB format, each channel needs padding pixel value information: padding_r, padding_g, padding_b, which is valid when if_memset=1. If it is a GRAY image, users can set all three values to the same;
4. if_memset indicates whether to memset dst image according to the padding value of each channel within the API.

- bmcv_rect_t * crop_rect

  Input parameter. Coordinates, width and height of each target small image on src image.

The specific format is defined as follows:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;
```

- bmcv_resize_algorithm algorithm

  Input parameter. Resize algorithm selection, including BMCV_INTER_NEAREST、BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC . By default, it is set as bilinear difference.

  bm1684 supports BMCV_INTER_NEAREST,BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC.

  bm1684x supports BMCV_INTER_NEAREST and BMCV_INTER_LINEAR.

Description of returning value:

- BM_SUCCESS: success
- Other: failed

Note:

1. The format of dst image of this API only supports:

num	dst image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_BGR_PLANAR
3	FORMAT_RGBP_SEPARATE
4	FORMAT_BGRP_SEPARATE
5	FORMAT_RGB_PACKED
6	FORMAT_BGR_PACKED

2. The format and some requirements that the API needs to meet are consistent to bmcv_image_vpp_basic.

5.14 bmcv_image_vpp_stitch

Use the crop function of vpp hardware to complete image stitching. The src crop , csc, resize and dst crop operation can be completed on the input image at one time. The number of small images stitiched in dst image cannot exceed 256.

```
bm_status_t bmcv_image_vpp_stitch(
    bm_handle_t         handle,
```

(continues on next page)

(continued from previous page)

```
int               input_num,
bm_image*         input,
bm_image          output,
bmcv_rect_t*      dst_crop_rect,
bmcv_rect_t*      src_crop_rect = NULL,
bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. Handle of device' s capacity (HDC) obtained by calling bm_dev_request.

- int input_num

  Input parameter. The number of input bm_image.

- bm_imagei* input

  Input parameter. Input bm_image object pointer.

- bm_image output

  Output parameter. Output bm_image object.

- bmcv_rect_t * dst_crop_rect

  Input parameter. The coordinates, width and height of each target small image on dst images.

- bmcv_rect_t * src_crop_rect

  Input parameter. The coordinates, width and height of each target small image on src image.

  The specific format is defined as follows:

  ```
  typedef struct bmcv_rect {
      int start_x;
      int start_y;
      int crop_w;
      int crop_h;
  } bmcv_rect_t;
  ```

- bmcv_resize_algorithm algorithm

  Input parameter. Resize algorithm selection, including BMCV_INTER_NEAREST, BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC. By default, it is set as bilinear difference.

bm1684 supports BMCV_INTER_NEAREST, BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC.

bm1684x supports BMCV_INTER_NEAREST and BMCV_INTER_LINEAR.

Return value description:

- BM_SUCCESS: success
- Other: failed

Notes:

1. The src image of this API does not support data in compressed format.
2. The format and some requirements that the API needs to meet are consistent to bmcv_image_vpp_basic.
3. If the src image is cropped, only one target will be cropped for one src image.

5.15 bmcv_image_vpp_csc_matrix_convert

By default, bmcv_image_vpp_convert uses BT_609 standard for color gamut conversion. In some cases, users need to use other standards or customize csc parameters.

```
bm_status_t bmcv_image_vpp_csc_matrix_convert(
    bm_handle_t  handle,
    int output_num,
    bm_image input,
    bm_image *output,
    csc_type_t csc,
    csc_matrix_t * matrix = nullptr,
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);
```

Processor model support

This interface supports BM1684/BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. Handle of device' s capacity (HDC) obtained by calling bm_dev_request.

- int image_num

  Input parameter. The number of input bm_image

- bm_image input

  Input parameter. Input bm_image object

- bm_image* output

  Output parameter. Output bm_image object pointer

- csc_type_t csc

  Input parameters. Enumeration type of gamut conversion, currently optional:

```
typedef enum csc_type {
    CSC_YCbCr2RGB_BT601 = 0,
    CSC_YPbPr2RGB_BT601,
    CSC_RGB2YCbCr_BT601,
    CSC_YCbCr2RGB_BT709,
    CSC_RGB2YCbCr_BT709,
    CSC_RGB2YPbPr_BT601,
    CSC_YPbPr2RGB_BT709,
    CSC_RGB2YPbPr_BT709,
    CSC_USER_DEFINED_MATRIX = 1000,
    CSC_MAX_ENUM
} csc_type_t;
```

- csc_matrix_t * matrix

  Input parameters. Color gamut conversion custom matrix, valid if and only if csc is CSC_USER_DEFINED_MATRIX.

  The specific format is defined as follows:

```
typedef struct {
    int csc_coe00;
    int csc_coe01;
    int csc_coe02;
    int csc_add0;
    int csc_coe10;
    int csc_coe11;
    int csc_coe12;
    int csc_add1;
    int csc_coe20;
    int csc_coe21;
    int csc_coe22;
    int csc_add2;
} __attribute__((packed)) csc_matrix_t;
```

  bm1684:

$$\begin{cases} dst_0 = (csc\_coe_{00} * src_0 + csc\_coe_{01} * src_1 + csc\_coe_{02} * src_2 + csc\_add_0) >> 10 \\ dst_1 = (csc\_coe_{10} * src_0 + csc\_coe_{11} * src_1 + csc\_coe_{12} * src_2 + csc\_add_1) >> 10 \\ dst_2 = (csc\_coe_{20} * src_0 + csc\_coe_{21} * src_1 + csc\_coe_{22} * src_2 + csc\_add_2) >> 10 \end{cases}$$

  bm1684x:

$$\begin{cases} dst_0 = csc\_coe_{00} * src_0 + csc\_coe_{01} * src_1 + csc\_coe_{02} * src_2 + csc\_add_0 \\ dst_1 = csc\_coe_{10} * src_0 + csc\_coe_{11} * src_1 + csc\_coe_{12} * src_2 + csc\_add_1 \\ dst_2 = csc\_coe_{20} * src_0 + csc\_coe_{21} * src_1 + csc\_coe_{22} * src_2 + csc\_add_2 \end{cases}$$

- bmcv_resize_algorithm algorithm

  Input parameter. Resize algorithm selection, including BMCV_INTER_NEAREST 、 BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC.By default, it is set as bilinear difference.

  bm1684 supports BMCV_INTER_NEAREST, BMCV_INTER_LINEAR and BMCV_INTER_BICUBIC.

  bm1684x supports BMCV_INTER_NEAREST and BMCV_INTER_LINEAR.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. The format and some requirements that the API needs to meet are consistent to vpp_convert.
2. If the color gamut conversion enumeration type does not correspond to the input and output formats. For example, if csc == CSC_YCbCr2RGB_BT601, while input image_format is RGB, a failure will be returned.
3. If csc == CSC_USER_DEFINED_MATRIX while matrix is nullptr, a failure will be returned.

Code example:

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include <memory>
#include "stdio.h"
#include "stdlib.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    bm_handle_t handle;
    int          image_h     = 1080;
    int          image_w     = 1920;
    bm_image       src, dst[4];
    bm_dev_request(&handle, 0);
    bm_image_create(handle, image_h, image_w, FORMAT_NV12,
          DATA_TYPE_EXT_1N_BYTE, &src);
    bm_image_alloc_dev_mem(src, 1);
    for (int i = 0; i < 4; i++) {
        bm_image_create(handle,
            image_h / 2,
```

(continues on next page)

(continued from previous page)

```
        image_w / 2,
        FORMAT_BGR_PACKED,
        DATA_TYPE_EXT_1N_BYTE,
        dst + i);
    bm_image_alloc_dev_mem(dst[i]);
  }
  std::unique_ptr<u8 []> y_ptr(new u8[image_h * image_w]);
  std::unique_ptr<u8 []> uv_ptr(new u8[image_h * image_w / 2]);
  memset((void *)(y_ptr.get()), 148, image_h * image_w);
  memset((void *)(uv_ptr.get()), 158, image_h * image_w / 2);
  u8 *host_ptr[] = {y_ptr.get(), uv_ptr.get()};
  bm_image_copy_host_to_device(src, (void **)host_ptr);

  bmcv_rect_t rect[] = {{0, 0, image_w / 2, image_h / 2},
        {0, image_h / 2, image_w / 2, image_h / 2},
        {image_w / 2, 0, image_w / 2, image_h / 2},
        {image_w / 2, image_h / 2, image_w / 2, image_h / 2}};

  bmcv_image_vpp_csc_matrix_convert(handle, 4, src, dst, CSC_YCbCr2RGB_
↪BT601);

  for (int i = 0; i < 4; i++) {
    bm_image_destroy(dst[i]);
  }

  bm_image_destroy(src);
  bm_dev_free(handle);
  return 0;
}
```

5.16 bmcv_image_jpeg_enc

This API can be used for JPEG encoding of multiple bm_image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_jpeg_enc(
      bm_handle_t handle,
      int         image_num,
      bm_image *  src,
      void *      p_jpeg_data[],
      size_t *    out_size,
      int         quality_factor = 85
);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- int image_num

  Input parameter. The number of input image, up to 4.

- bm_image* src

  Input parameter. Input bm_image pointer. Each bm_image requires an external call to bmcv_image_create to create the image memory. Users can call bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- void * p_jpeg_data,

  Output parameter. The data pointer of the encoded images. Since the interface supports the encoding of multiple images, it is a pointer array, and the size of the array is image_num. Users can choose not to apply for space (that is, each element of the array is NULL). Space will be automatically allocated according to the size of encoded data within the API, but when it is no longer used, users need to release the space manually. Of course, users can also choose to apply for enough space.

- size_t *out_size,

  Output parameter. After encoding, the size of each image (in bytes) is stored in the pointer.

- int quality_factor = 85

  Input parameter. The quality factor of the encoded image. The value is between 0 and 100. The larger the value, the higher the image quality, but the greater the amount of data. On the contrary, the smaller the value, the lower the image quality, and the less the amount of data. This parameter is optional and the default value is 85.

Return value Description:

- BM_SUCCESS: success
- Other: failed

Note:

Currently, the image formats which support encoding include the following:

FORMAT_YUV420P
FORMAT_YUV422P
FORMAT_YUV444P
FORMAT_NV12
FORMAT_NV21
FORMAT_NV16

FORMAT_NV61
FORMAT_GRAY

The interface supports the following data_type:

DATA_TYPE_EXT_1N_BYTE

Sample code

```
int image_h     = 1080;
int image_w     = 1920;
int size        = image_h * image_w;
int format      = FORMAT_YUV420P;
bm_image src;
bm_image_create(handle, image_h, image_w, (bm_image_format_ext)format,
       DATA_TYPE_EXT_1N_BYTE, &src);
std::unique_ptr<unsigned char[]> buf1(new unsigned char[size]);
memset(buf1.get(), 0x11, size);

std::unique_ptr<unsigned char[]> buf2(new unsigned char[size / 4]);
memset(buf2.get(), 0x22, size / 4);

std::unique_ptr<unsigned char[]> buf3(new unsigned char[size / 4]);
memset(buf3.get(), 0x33, size / 4);

unsigned char *buf[] = {buf1.get(), buf2.get(), buf3.get()};
bm_image_copy_host_to_device(src, (void **)buf);

void* jpeg_data = NULL;
size_t out_size = 0;
int ret = bmcv_image_jpeg_enc(handle, 1, &src, &jpeg_data, &out_size);
if (ret == BM_SUCCESS) {
    FILE *fp = fopen("test.jpg", "wb");
    fwrite(jpeg_data, out_size, 1, fp);
    fclose(fp);
}
free(jpeg_data);
bm_image_destroy(src);
```

5.17 bmcv_image_jpeg_dec

The interface can decode multiple JPEG images.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:


```
bm_status_t bmcv_image_jpeg_dec(
        bm_handle_t handle,
        void *      p_jpeg_data[],
        size_t *    in_size,
        int         image_num,
        bm_image *  dst
);
```

Input parameter description:

- bm_handle_t handle

  Input parameters. Handle of bm_handle.

- void * p_jpeg_data[]

  Input parameter. The image data pointer to be decoded. It is a pointer array because the interface supports the decoding of multiple images.

- size_t *in_size

  Input parameter. The size of each image to be decoded (in bytes) is stored in the pointer, that is, the size of the space pointed to by each dimensional pointer of p_jpeg_data.

- int image_num

  Input parameter. The number of input image, up to 4.

- bm_image* dst

  Output parameter. The pointer of output bm_image. Users can choose whether or not to call bm_image_create to create dst bm_image. If users only declare but do not create, it will be automatically created by the interface according to the image information to be decoded. The default format is shown in the following table. When it is no longer needed, users still needs to call bm_image_destroy to destroy it.

Code Stream	Default Output Format
YUV420	FORMAT_YUV420P
YUV422	FORMAT_YUV422P
YUV444	FORMAT_YUV444P
YUV400	FORMAT_GRAY

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. If users do not use bmcv_image_create to create bm_image of dst, they need to set the space pointed by the parameter input pointer to 0.

2. At present, the image formats supported by decoding and their output formats are shown in the following table. If users need to specify one of the following output formats, they can use bmcv_image_create to create their own dst bm_image, so as to decode the picture to one of the following corresponding formats.

Code Stream	Default Output Format
YUV420	FORMAT_YUV420P
	FORMAT_NV12
	FORMAT_NV21
YUV422	FORMAT_YUV422P
	FORMAT_NV16
	FORMAT_NV61
YUV444	FORMAT_YUV444P
YUV400	FORMAT_GRAY

The interface supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Sample code

```
size_t size = 0;
// read input from picture
FILE *fp = fopen(filename, "rb+");
assert(fp != NULL);
fseek(fp, 0, SEEK_END);
*size = ftell(fp);
u8* jpeg_data = (u8*)malloc(*size);
fseek(fp, 0, SEEK_SET);
fread(jpeg_data, *size, 1, fp);
fclose(fp);

// create bm_image used to save output
bm_image dst;
memset((char*)&dst, 0, sizeof(bm_image));
// if you not create dst bm_image it will create automatically inside.
// you can also create dst bm_image here, like this:
// bm_image_create(handle, IMAGE_H, IMAGE_W, FORMAT_YUV420P,
//         DATA_TYPE_EXT_1N_BYTE, &dst);

// decode input
int ret = bmcv_image_jpeg_dec(handle, (void**)&jpeg_data, &size, 1, &dst);
free(jpeg_data);
bm_image_destory(dst);
```

5.18 bmcv_image_copy_to

The interface copies an image to the corresponding memory area of the target image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_copy_to(
      bm_handle_t handle,
      bmcv_copy_to_atrr_t copy_to_attr,
      bm_image          input,
      bm_image          output
);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bmcv_copy_to_atrr_t copy_to_attr

  Input parameter. The attribute configuration corresponding to the API.

- bm_image input

  Input parameter. Input bm_image. The creation of bm_image requires an external call to bmcv_image_create. Users can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Users can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

Return value Description:

- BM_SUCCESS: success
- Other: failed

Data Type Description:

```
typedef struct bmcv_copy_to_atrr_s {
   int          start_x;
   int          start_y;
   unsigned char padding_r;
   unsigned char padding_g;
   unsigned char padding_b;
```

(continues on next page)

(continued from previous page)

```
    int if_padding;
} bmcv_copy_to_atrr_t;
```

- padding_b represents the value filled in the b channel of the extra image when the input image is smaller than the output image.
- padding_r represents the value filled in the r channel of the extra image when the input image is smaller than the output image.
- padding_g represents the value filled in the g channel of the extra image when the input image is smaller than the output image.
- start_x describes the starting abscissa of the output image of copy_to.
- start_y describes the starting ordinate of the output image of copy_to.
- if_padding indicates whether the extra image area needs to be filled with a specific color when the input image is smaller than the output image. 0 indicates no and 1 indicates yes. When the value is filled with 0, the setting of padding_r, padding_g and padding_b will be invalid.

Format support

bm1684 supports the following combination of image_formats and data_type:

num	image_format	data_type
1	FORMAT_BGR_PACKED	DATA_TYPE_EXT_FLOAT32
2	FORMAT_BGR_PLANAR	DATA_TYPE_EXT_FLOAT32
3	FORMAT_BGR_PACKED	DATA_TYPE_EXT_1N_BYTE
4	FORMAT_BGR_PLANAR	DATA_TYPE_EXT_1N_BYTE
5	FORMAT_BGR_PLANAR	DATA_TYPE_EXT_4N_BYTE
6	FORMAT_RGB_PACKED	DATA_TYPE_EXT_FLOAT32
7	FORMAT_RGB_PLANAR	DATA_TYPE_EXT_FLOAT32
8	FORMAT_RGB_PACKED	DATA_TYPE_EXT_1N_BYTE
9	FORMAT_RGB_PLANAR	DATA_TYPE_EXT_1N_BYTE
10	FORMAT_RGB_PLANAR	DATA_TYPE_EXT_4N_BYTE
11	FORMAT_GRAY	DATA_TYPE_EXT_1N_BYTE

bm1684x supports the following data_type of bm_image:

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16
6	DATA_TYPE_EXT_FLOAT32	DATA_TYPE_EXT_FLOAT32

The color formats supported for input and output are:

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

Notes

1. Before calling bmcv_image_copy_to(), users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.
3. To avoid memory overrun, the width + start_x of input image must be less than or equal to the width stride of output image.

Code example:

```
int channel   = 3;
int in_w      = 400;
int in_h      = 400;
int out_w     = 800;
int out_h     = 800;
int    dev_id = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
       new unsigned char[channel * in_w * in_h],
       std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
       new unsigned char[channel * out_w * out_h],
       std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * in_w * in_h; i++) {
   src_data[i] = rand() % 255;
}
// calculate res
bmcv_copy_to_atrr_t copy_to_attr;
copy_to_attr.start_x   = 0;
copy_to_attr.start_y   = 0;
```

(continues on next page)

(continued from previous page)

```
copy_to_attr.padding_r = 0;
copy_to_attr.padding_g = 0;
copy_to_attr.padding_b = 0;
bm_image input, output;
bm_image_create(handle,
        in_h,
        in_w,
        FORMAT_RGB_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
        out_h,
        out_w,
        FORMAT_RGB_PLANAR,
        DATA_TYPE_EXT_1N_BYTE,
        &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_copy_to(handle, copy_to_attr, input, output)) {
    std::cout << "bmcv_copy_to error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);

    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle)
```

5.19 bmcv_image_draw_lines

The function of drawing polygons can be implemented by drawing one or more lines on an image, it also can specify the color and width of lines.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
typedef struct {
    int x;
    int y;
} bmcv_point_t;

typedef struct {
```

(continues on next page)

(continued from previous page)

```
    unsigned char r;
    unsigned char g;
    unsigned char b;
} bmcv_color_t;

bm_status_t bmcv_image_draw_lines(
        bm_handle_t handle,
        bm_image img,
        const bmcv_point_t* start,
        const bmcv_point_t* end,
        int line_num,
        bmcv_color_t color,
        int thickness);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bm_image img

  Input/output parameter. The bm_image of the image to be processed. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- const bmcv_point_t* start

  Input parameter. The coordinate pointer of the starting point of the line segment. The data length pointed to is determined by line_num. The upper left corner of the image is the origin, extending to the right in the x-direction and downward in the y-direction.

- const bmcv_point_t* end

  Input parameter. The coordinate pointer of the end point of the line segment. The data length pointed to is determined by line_num. The upper left corner of the image is the origin, extending to the right in the x-direction and downward in the y-direction.

- int line_num

  Input parameter. The number of lines to be drawn.

- bmcv_color_t color

  Input parameter. The color of the drawn line, which is the value of RGB three channels respectively.

- int thickness

  Input parameter. The width of the drawn line, which is recommended to be set as even numbers for YUV format images.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format Support:

The interface currently supports the following image_format:

num	image_format
1	FORMAT_GRAY
2	FORMAT_YUV420P
3	FORMAT_YUV422P
4	FORMAT_YUV444P
5	FORMAT_NV12
6	FORMAT_NV21
7	FORMAT_NV16
8	FORMAT_NV61

The following data_type is currently supported:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Code example:

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
int thickness = 4
bmcv_point_t start = {0, 0};
bmcv_point_t end = {100, 100};
bmcv_color_t color = {255, 0, 0};
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> data_ptr(
       new unsigned char[channel * width * height],
       std::default_delete<unsigned char[]>());
for (int i = 0; i < channel * width * height; i++) {
   data_ptr.get()[i] = rand() % 255;
}
// calculate res
bm_image img;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
```

(continues on next page)

(continued from previous page)

```
            &img);
bm_image_alloc_dev_mem(img);
bm_image_copy_host_to_device(img, (void **)&(data_ptr.get()));
if (BM_SUCCESS != bmcv_image_draw_lines(handle, img, &start, &end, 1, color,F
↪thickness)) {
    std::cout << "bmcv draw lines error !!!" << std::endl;
    bm_image_destroy(img);
    bm_dev_free(handle);
    return;
}
bm_image_copy_device_to_host(img, (void **)&(data_ptr.get()));
bm_image_destroy(img);
bm_dev_free(handle);
```

5.20 bmcv_image_draw_point

This interface is used to fill one or more points on an image。

Processor model support

This interface only supports BM1684X.

接口形式:

```
bm_status_t bmcv_image_draw_point(
      bm_handle_t   handle,
      bm_image      image,
      int           point_num,
      bmcv_point_t *coord,
      int           length,
      unsigned char r,
      unsigned char g,
      unsigned char b)
```

传入参数说明:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bm_image image

  Input parameter. The bm_image on which users need to draw a point.

- int point_num

  Input parameter. The number of points boxes, which refers to the number of bmcv_point_t objects in the rects pointer.

- bmcv_point_t* rect

Input parameters. Pointer position pointer. Please refer to the data type description below for specific content.

- int length

  Input parameters. The side length of the point, with a value range of [1, 510].

- unsigned char r

  Input parameter. The r component of the color.

- unsigned char g

  Input parameter. The g component of the color.

- unsigned char b

  Input parameter. The b component of the color.

Return value description:

- BM_SUCCESS: success
- Other: failed

Data type description:

```
typedef struct {
    int x;
    int y;
} bmcv_point_t;
```

- x describes the starting abscissa of the point in the original image. Starting from 0 from left to right, the value range is [0, width).
- y describes the starting ordinate of the point in the original image. Starting from 0 from top to bottom, the value range is [0, height).

注意事项:

1. bm1684x supports the following formats of bm_image:

num	input image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_YUV420P
4	RGB_PLANAR
5	RGB_PACKED
6	BGR_PLANAR
7	BGR_PACKED

bm1684x supports the following data_type of bm_image:

num	intput data_type
1	DATA_TYPE_EXT_1N_BYTE

If the input/output format requirements are not met, a failure will be returned.

3. All input and output bm_image structures must be created in advance, or a failure will be returned.
4. All input point object areas must be within the image.
5. When the input is FORMAT_YUV420P, FORMAT_NV12, FORMAT_NV21, length must be an even number.

5.21 bmcv_image_draw_rectangle

This interface is used to draw one or more rectangular boxes on the image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_draw_rectangle(
      bm_handle_t   handle,
      bm_image      image,
      int           rect_num,
      bmcv_rect_t * rects,
      int           line_width,
      unsigned char r,
      unsigned char g,
      unsigned char b)
```

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- bm_image image

  Input parameter. The bm_image on which users need to draw a rectangle.

- int rect_num

  Input parameter. The number of rectangular boxes, which refers to the number of bmcv_rect_t objects in the rects pointer.

- bmcv_rect_t* rect

Input parameter. Pointer to a rectangular box object that contains the starting point, width, and height of the rectangle. Refer to the following data type description for details.

· int line_width

Input parameter. The width of the rectangle line.

· unsigned char r

Input parameter. The r component of the rectangle color.

· unsigned char g

Input parameter. The g component of the rectangle color.

· unsigned char b

Input parameter. The b component of the rectangle color.

Return value description:

· BM_SUCCESS: success

· Other: failed

Data type description:

```
typedef struct bmcv_rect {
    int start_x;
    int start_y;
    int crop_w;
    int crop_h;
} bmcv_rect_t;
```

· start_x describes the starting horizontal coordinate of where the crop image is located in the original image. It starts at 0 from left to right and takes values in the range [0, width).

· start_y describes the starting vertical coordinate of where the crop image is located in the original image. It starts at 0 from top to bottom and takes values in the range [0, height).

· crop_w describes the width of the crop image, that is the width of the corresponding output image.

· crop_h describes the height of the crop image, that is the height of the corresponding output image.

Note:

1. bm1684x:

· bm1684x supports the following data_type of bm_image:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

· bm1684x supports the following image_format of bm_image:

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

Returns a failure if the input and output format requirements are not met.

2. bm1684:

· The API inputs image objects in NV12 / NV21 / NV16 / NV61 / YUV420P / RGB_PLANAR / RGB_PACKED / BGR_PLANAR / BGR_PACKED formats and directly draw a frame on the corresponding device memory without additional memory application and copy.

· At present, the API supports the following image formats of input bm_image:

num	image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_NV16
4	FORMAT_NV61
5	FORMAT_YUV420P
6	FORMAT_RGB_PLANAR
7	FORMAT_BGR_PLANAR
8	FORMAT_RGB_PACKED
9	FORMAT_BGR_PACKED

the API supports the following data format of input bm_image:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

If the input/output format requirements are not met, a failure will be returned.

3. All input and output bm_image structures must be created in advance, or a failure will be returned.
4. If the image is in NV12 / NV21 / NV16 / NV61 / YUV420P format, the line_width will be automatically even aligned.
5. If rect_num is 0, a success will be returned automatically.
6. If line_width is less than zero, a failure will be returned.
7. If all input rectangular objects are outside the image, only the lines within the image will be drawn and a success will be returned.

Code example

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>

int main(int argc, char *argv[]) {
    bm_handle_t handle;
    bm_dev_request(&handle, 0);

    int image_h = 1080;
    int image_w = 1920;
    bm_image src;
    bm_image_create(handle, image_h, image_w, FORMAT_NV12,
          DATA_TYPE_EXT_1N_BYTE, &src);
    std::shared_ptr<u8*> y_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w]);
    memset((void *)(*y_ptr.get()), 148, image_h * image_w);
    memset((void *)(*uv_ptr.get()), 158, image_h * image_w / 2);
    u8 *host_ptr[] = {*y_ptr.get(), *uv_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);
    bmcv_rect_t rect;
    rect.start_x = 100;
    rect.start_y = 100;
    rect.crop_w = 200;
    rect.crop_h = 300;
    bmcv_image_draw_rectangle(handle, src, 1, &rect, 3, 255, 0, 0);
```

(continues on next page)

(continued from previous page)

```
    bm_image_destroy(src);
    bm_dev_free(handle);
    return 0;
}
```

5.22 bmcv_image_put_text

The functions of writing (English) on an image and specifying the color, size and width of words are supported.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
typedef struct {
    int x;
    int y;
} bmcv_point_t;

typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} bmcv_color_t;

bm_status_t bmcv_image_put_text(
        bm_handle_t handle,
        bm_image image,
        const char* text,
        bmcv_point_t org,
        bmcv_color_t color,
        float fontScale,
        int thickness);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image image

  Input / output parameter. The bm_image of image to be processed. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- const char* text

Input parameter. The text content to be written. Currently only supports English.

- bmcv_point_t org

  Input parameter. The coordinate position of the lower left corner of the first character. The upper left corner of the image is the origin, extending to the right in the x-direction and downward in the y-direction.

- bmcv_color_t color

  Input parameter. The color of the drawn line, which is the value of RGB three channels respectively.

- float fontScale

  Input parameter. Font scale.

- int thickness

  Input parameter. The width of the drawn line, which is recommended to be set to an even number for YUV format images.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following images_format:

num	image_format
1	FORMAT_GRAY
2	FORMAT_YUV420P
3	FORMAT_YUV422P
4	FORMAT_YUV444P
5	FORMAT_NV12
6	FORMAT_NV21
7	FORMAT_NV16
8	FORMAT_NV61

The following data_type is currently supported:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Code example:

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
int thickness = 4
float fontScale = 4;
char text[20] = "hello world";
bmcv_point_t org = {100, 100};
bmcv_color_t color = {255, 0, 0};
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> data_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
for (int i = 0; i < channel * width * height; i++) {
    data_ptr.get()[i] = rand() % 255;
}
// calculate res
bm_image img;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &img);
bm_image_alloc_dev_mem(img);
bm_image_copy_host_to_device(img, (void **)&(data_ptr.get()));
if (BM_SUCCESS != bmcv_image_put_text(handle, img, text, org, color, fontScale,
↪ thickness)) {
    std::cout << "bmcv put text error !!!" << std::endl;
    bm_image_destroy(img);
    bm_dev_free(handle);
    return;
}
bm_image_copy_device_to_host(img, (void **)&(data_ptr.get()));
bm_image_destroy(img);
bm_dev_free(handle);
```

5.23 bmcv_image_fill_rectangle

This interface is used to fill one or more rectangles on the image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_fill_rectangle(
        bm_handle_t   handle,
```

(continues on next page)

(continued from previous page)

```
    bm_image     image,
    int          rect_num,
    bmcv_rect_t * rects,
    unsigned char r,
    unsigned char g,
    unsigned char b)
```

Description of incoming parameters::

- bm_handle_t handle

  Input parameter. Handle of device' s capacity (HDC) which is obtained by calling bm_dev_request.

- bm_image image

  Input parameter. The bm_image object on which users want to fill a rectangle.

- int rect_num

  Input parameter. The number of rectangles to be filled, which refers to the number of bmcv_rect_t object contained in the rects pointer.

- bmcv_rect_t* rect

  Input parameter. Pointer to a rectangular object that contains the start point and width height of the rectangle. Refer to the following data type description for details.

- unsigned char r

  Input parameter. The r component of the rectangle fill color.

- unsigned char g

  Input parameter. The g component of the rectangle fill color.

- unsigned char b

  Input parameter. The b component of the rectangle fill color.

Return value Description:

- BM_SUCCESS: success
- Other: failed

Data type description:

```
typedef struct bmcv_rect {
  int start_x;
  int start_y;
  int crop_w;
  int crop_h;
} bmcv_rect_t;
```

- start_x describes the starting horizontal coordinate of where the crop image is located in the original image. It starts at 0 from left to right and takes values in the range [0, width).
- start_y describes the starting vertical coordinate of where the crop image is located in the original image. It starts at 0 from top to bottom and takes values in the range [0, height).
- crop_w describes the width of the crop image, that is, the width of the corresponding output image.
- crop_h describes the height of the crop image, that is, the height of the corresponding output image.

Note:

1. bm1684 supports the following formats of bm_image:

num	input image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_NV16
4	FORMAT_NV61
5	FORMAT_YUV420P
6	RGB_PLANAR
7	RGB_PACKED
8	BGR_PLANAR
9	BGR_PACKED

bm1684x supports the following formats of bm_image:

num	input image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_YUV420P
4	RGB_PLANAR
5	RGB_PACKED
6	BGR_PLANAR
7	BGR_PACKED

bm1684x supports the following data_type of bm_image:

num	intput data_type
1	DATA_TYPE_EXT_1N_BYTE

If the input/output format requirements are not met, a failure will be returned.

2. All input and output bm_image structures must be created in advance, or a failure will be returned.
3. If rect_num is 0, a success will be returned automatically.
4. If the part of all input rectangular objects is outside the image, only the part inside the image will be filled and a success will be returned.

Code example

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>

int main(int argc, char *argv[]) {
    bm_handle_t handle;
    bm_dev_request(&handle, 0);

    int image_h = 1080;
    int image_w = 1920;
    bm_image src;
    bm_image_create(handle, image_h, image_w, FORMAT_NV12,
          DATA_TYPE_EXT_1N_BYTE, &src);
    std::shared_ptr<u8*> y_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w]);
    memset((void *)(*y_ptr.get()), 148, image_h * image_w);
    memset((void *)(*uv_ptr.get()), 158, image_h * image_w / 2);
    u8 *host_ptr[] = {*y_ptr.get(), *uv_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);
    bmcv_rect_t rect;
    rect.start_x = 100;
    rect.start_y = 100;
    rect.crop_w = 200;
    rect.crop_h = 300;
    bmcv_image_fill_rectangle(handle, src, 1, &rect, 255, 0, 0);
    bm_image_destroy(src);
    bm_dev_free(handle);
    return 0;
}
```

5.24 bmcv_image_absdiff

Subtract the pixel values of two images with the same size and take the absolute value.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_absdiff(
      bm_handle_t handle,
      bm_image input1,
      bm_image input2,
      bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bm_image input1

  Input parameter. The bm_image of the first input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image input2

  Input parameter. The bm_image of the second input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following images_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling bmcv_image_absdiff(), users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.

Code example:

```
int channel  = 3;
int width    = 1920;
int height   = 1080;
int dev_id   = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> src2_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
```

(continues on next page)

(continued from previous page)

```
unsigned char * src1_data = src1_ptr.get();
unsigned char * src2_data = src2_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src1_data[i] = rand() % 255;
    src2_data[i] = rand() % 255;
}
// calculate res
bm_image input1, input2, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input1);
bm_image_alloc_dev_mem(input1);
bm_image_copy_host_to_device(input1, (void **)&src1_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input2);
bm_image_alloc_dev_mem(input2);
bm_image_copy_host_to_device(input2, (void **)&src2_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_absdiff(handle, input1, input2, output)) {
    std::cout << "bmcv absdiff error !!!" << std::endl;
    bm_image_destroy(input1);
    bm_image_destroy(input2);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input1);
bm_image_destroy(input2);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.25 bmcv_image_bitwise_and

Bitwise and operate on the corresponding pixel value of two images with the same size.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_bitwise_and(
      bm_handle_t handle,
      bm_image input1,
      bm_image input2,
      bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input1

  Input parameter. The bm_image of the first input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image input2

  Input parameter. The bm_image of the second input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling bmcv_image_bitwise_and(), users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.

Code example:

```
int channel   = 3;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> src2_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
```

(continues on next page)

(continued from previous page)

```
unsigned char * src1_data = src1_ptr.get();
unsigned char * src2_data = src2_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src1_data[i] = rand() % 255;
    src2_data[i] = rand() % 255;
}
// calculate res
bm_image input1, input2, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input1);
bm_image_alloc_dev_mem(input1);
bm_image_copy_host_to_device(input1, (void **)&src1_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input2);
bm_image_alloc_dev_mem(input2);
bm_image_copy_host_to_device(input2, (void **)&src2_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_bitwise_and(handle, input1, input2, output)) {
    std::cout << "bmcv bitwise and error !!!" << std::endl;
    bm_image_destroy(input1);
    bm_image_destroy(input2);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input1);
bm_image_destroy(input2);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.26 bmcv_image_bitwise_or

Bitwise or operate on the corresponding pixel value of two images with the same size.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_bitwise_or(
      bm_handle_t handle,
      bm_image input1,
      bm_image input2,
      bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input1

  Input parameter. The bm_image of the first input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image input2

  Input parameter. The bm_image of the second input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following images_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

The following data are currently supported_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling bmcv_image_bitwise_or(), users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.

Code example:

```
int channel   = 3;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> src2_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
```

(continues on next page)

(continued from previous page)

```
unsigned char * src1_data = src1_ptr.get();
unsigned char * src2_data = src2_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src1_data[i] = rand() % 255;
    src2_data[i] = rand() % 255;
}
// calculate res
bm_image input1, input2, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input1);
bm_image_alloc_dev_mem(input1);
bm_image_copy_host_to_device(input1, (void **)&src1_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input2);
bm_image_alloc_dev_mem(input2);
bm_image_copy_host_to_device(input2, (void **)&src2_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_bitwise_or(handle, input1, input2, output)) {
    std::cout << "bmcv bitwise or error !!!" << std::endl;
    bm_image_destroy(input1);
    bm_image_destroy(input2);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input1);
bm_image_destroy(input2);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.27 bmcv_image_bitwise_xor

Perform bitwise xor operate on the corresponding pixel values of two images with the same size.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_bitwise_xor(
      bm_handle_t handle,
      bm_image input1,
      bm_image input2,
      bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameters. The handle of bm_handle.

- bm_image input1

  Input parameter. The bm_image of the first input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image input2

  Input parameter. The bm_image of the second input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following images_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

The following data are currently supported_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling bmcv_image_bitwise_xor(), users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.

Code example:

```
int channel   = 3;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> src2_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
```

(continues on next page)

(continued from previous page)

```
unsigned char * src1_data = src1_ptr.get();
unsigned char * src2_data = src2_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src1_data[i] = rand() % 255;
    src2_data[i] = rand() % 255;
}
// calculate res
bm_image input1, input2, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input1);
bm_image_alloc_dev_mem(input1);
bm_image_copy_host_to_device(input1, (void **)&src1_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &input2);
bm_image_alloc_dev_mem(input2);
bm_image_copy_host_to_device(input2, (void **)&src2_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_RGB_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_bitwise_xor(handle, input1, input2, output)) {
    std::cout << "bmcv bitwise xor error !!!" << std::endl;
    bm_image_destroy(input1);
    bm_image_destroy(input2);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input1);
bm_image_destroy(input2);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.28 bmcv_image_add_weighted

Fusion of two images of the same size by weighted, as follows:

$$output = alpha * input1 + beta * input2 + gamma$$

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_add_weighted(
        bm_handle_t handle,
        bm_image input1,
        float alpha,
        bm_image input2,
        float beta,
        float gamma,
        bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input1

  Input parameter. The bm_image of the first input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- float alpha

  The weight of the first image.

- bm_image input2

  Input parameter. The bm_image of the second input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- float beta

  The weight of the second image.

- float gamma

  Offset after fusion.

- bm_image output

Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following images_format:

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.

Code example:

```
int channel   = 3;
int width     = 1920;
```

(continues on next page)

(continued from previous page)

```
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> src2_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
unsigned char * src1_data = src1_ptr.get();
unsigned char * src2_data = src2_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
  src1_data[i] = rand() % 255;
  src2_data[i] = rand() % 255;
}
// calculate res
bm_image input1, input2, output;
bm_image_create(handle,
            height,
            width,
            FORMAT_RGB_PLANAR,
            DATA_TYPE_EXT_1N_BYTE,
            &input1);
bm_image_alloc_dev_mem(input1);
bm_image_copy_host_to_device(input1, (void **)&src1_data);
bm_image_create(handle,
            height,
            width,
            FORMAT_RGB_PLANAR,
            DATA_TYPE_EXT_1N_BYTE,
            &input2);
bm_image_alloc_dev_mem(input2);
bm_image_copy_host_to_device(input2, (void **)&src2_data);
bm_image_create(handle,
            height,
            width,
            FORMAT_RGB_PLANAR,
            DATA_TYPE_EXT_1N_BYTE,
            &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_add_weighted(handle, input1, 0.5, input2, 0.5, 0,
↪ output)) {
  std::cout << "bmcv add_weighted error !!!" << std::endl;
  bm_image_destroy(input1);
  bm_image_destroy(input2);
  bm_image_destroy(output);
```

(continues on next page)

(continued from previous page)

```
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input1);
bm_image_destroy(input2);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.29 bmcv_image_threshold

Image thresholding operation.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_threshold(
        bm_handle_t handle,
        bm_image input,
        bm_image output,
        unsigned char thresh,
        unsigned char max_value,
        bm_thresh_type_t type);
```

The thresh types are as follows:

```
typedef enum {
    BM_THRESH_BINARY = 0,
    BM_THRESH_BINARY_INV,
    BM_THRESH_TRUNC,
    BM_THRESH_TOZERO,
    BM_THRESH_TOZERO_INV,
    BM_THRESH_TYPE_MAX
} bm_thresh_type_t;
```

The specific formula of each type is as follows:

Description of parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of the input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use

Enumerator	
THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$

bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

- unsigned char thresh

  Threshold.

- max_value

  Maximum value.

- bm_thresh_type_t type

  Thresholding type.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_BGR_PACKED	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY	FORMAT_GRAY
8	FORMAT_YUV420P	FORMAT_YUV420P
9	FORMAT_YUV422P	FORMAT_YUV422P
10	FORMAT_YUV444P	FORMAT_YUV444P
11	FORMAT_NV12	FORMAT_NV12
12	FORMAT_NV21	FORMAT_NV21
13	FORMAT_NV16	FORMAT_NV16
14	FORMAT_NV61	FORMAT_NV61
15	FORMAT_NV24	FORMAT_NV24

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The image_format and data_type of input and output must be the same.

Code example:

```
int channel  = 1;
int width    = 1920;
int height   = 1080;
int dev_id   = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
```

(continues on next page)

(continued from previous page)

```
    src_data[i] = rand() % 255;
}
// calculate res
bm_image input, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_threshold(handle, input, output, 200, 200, BM_
↪THRESH_BINARY)) {
    std::cout << "bmcv thresh error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.30 bmcv_image_dct

DCT transformation of the image.

The format of the interface is as follows:

```
bm_status_t bmcv_image_dct(
        bm_handle_t handle,
        bm_image input,
        bm_image output,
        bool is_inversed);
```

Processor model support

This interface only supports BM1684.

Description of input parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of the input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

- bool is_inversed

  Input parameter. Whether it is inverse transformation.

Return value description::

- BM_SUCCESS: success
- Other: failed

The coefficients of DCT transformation are only related to the width and height of the image, and the transformation coefficients need to be recalculated every time the above interface is called. Therefore, for images of the same size, in order to avoid repeated calculation of transformation coefficients, the above interface can be divided into two steps:

1. Calculate the transformation coefficient of a specific size;
2. Reuse the reorganized coefficients to perform DCT transformation on images of the same size.

The interface form of calculation coefficient is as follows:

```
bm_status_t bmcv_dct_coeff(
      bm_handle_t handle,
      int H,
      int W,
      bm_device_mem_t hcoeff_output,
      bm_device_mem_t wcoeff_output,
      bool is_inversed);
```

Description of input parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- int H

  Input parameter. The height of the image.

- int W

  Input parameter. The width of the image.

- bm_device_mem_t hcoeff_output

  Output parameter. The device memory space stores the DCT transformation coefficients of the h dimension. For images with the size of H*W, the size of the space is H*H*sizeof (float).

- bm_device_mem_t wcoeff_output

  Output parameter. The device memory space stores DCT transformation coefficients of the w dimension. For images with the size of H*W, the size of the space is W*W*sizeof (float).

- bool is_inversed

  Input parameter. Whether it is inverse transformation.

Return value description:

- BM_SUCCESS: success
- Other: failed

After obtaining the coefficient, transfer it to the following interfaces to start the calculation:

```
bm_status_t bmcv_image_dct_with_coeff(
        bm_handle_t handle,
        bm_image input,
        bm_device_mem_t hcoeff,
        bm_device_mem_t wcoeff,
        bm_image output);
```

Description of input parameters:

- bm_handle_t handle

  Input parameters. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of the input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_device_mem_t hcoeff

  Input parameter. The device memory space stores the DCT transformation coefficients of the h dimension. For the image with the size of H*W, the size of the space is H*H*sizeof (float).

- bm_device_mem_t wcoeff

Input parameter. The device memory space stores the DCT transformation coefficients of the w dimension. For the image with the size of H*W, the size of the space is W*W*sizeof (float).

- bm_image output

  Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_GRAY	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_FLOAT32

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The data_type of input and output must be the same.

Sample code

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<float> src_ptr(
        new float[channel * width * height],
        std::default_delete<float[]>());
std::shared_ptr<float> res_ptr(
        new float[channel * width * height],
        std::default_delete<float[]>());
```

(continues on next page)

(continued from previous page)

```
float * src_data = src_ptr.get();
float * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src_data[i] = rand() % 255;
}
bm_image bm_input, bm_output;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_FLOAT32,
                &bm_input);
bm_image_alloc_dev_mem(bm_input);
bm_image_copy_host_to_device(bm_input, (void **)&src_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_FLOAT32,
                &bm_output);
bm_image_alloc_dev_mem(bm_output);
bm_device_mem_t hcoeff_mem;
bm_device_mem_t wcoeff_mem;
bm_malloc_device_byte(handle, &hcoeff_mem, height*height*sizeof(float));
bm_malloc_device_byte(handle, &wcoeff_mem, width*width*sizeof(float));
bmcv_dct_coeff(handle, bm_input.height, bm_input.width, hcoeff_mem, wcoeff_
↪mem, is_inversed);
bmcv_image_dct_with_coeff(handle, bm_input, hcoeff_mem, wcoeff_mem, bm_
↪output);
bm_image_copy_device_to_host(bm_output, (void **)&res_data);
bm_image_destroy(bm_input);
bm_image_destroy(bm_output);
bm_free_device(handle, hcoeff_mem);
bm_free_device(handle, wcoeff_mem);
bm_dev_free(handle);
```

5.31 bmcv_image_sobel

Sobel operator for edge detection.

Processor model support

This interface only supports BM1684.

Interface form:

```
bm_status_t bmcv_image_sobel(
      bm_handle_t handle,
      bm_image input,
```

(continues on next page)

(continued from previous page)

```
    bm_image output,
    int dx,
    int dy,
    int ksize = 3,
    float scale = 1,
    float delta = 0);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of input image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. The output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to open up new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

- int dx

  The difference order in the x direction.

- int dy

  The difference order in the y direction.

- int ksize = 3

  The size of Sobel core, which must be - 1,1,3,5 or 7. In particular, if it is - 1, use 3×3 Scharr filter; if it is 1, use 3×1 or 1×3 core. The default value is 3.

- float scale = 1

  Multiply the calculated difference result by the coefficient, and the default value is 1.

- float delta = 0

  Add this offset before outputting the final result. The default value is 0.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_BGR_PACKED	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY	FORMAT_GRAY
8	FORMAT_YUV420P	FORMAT_GRAY
9	FORMAT_YUV422P	FORMAT_GRAY
10	FORMAT_YUV444P	FORMAT_GRAY
11	FORMAT_NV12	FORMAT_GRAY
12	FORMAT_NV21	FORMAT_GRAY
13	FORMAT_NV16	FORMAT_GRAY
14	FORMAT_NV61	FORMAT_GRAY
15	FORMAT_NV24	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The data_type of input and output must be the same.
3. The currently supported maximum image width is (2048 - ksize).

Code example:

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
```

(continues on next page)

(continued from previous page)

```
for (int i = 0; i < channel * width * height; i++) {
    src_data[i] = rand() % 255;
}
// calculate res
bm_image input, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_sobel(handle, input, output, 0, 1)) {
    std::cout << "bmcv sobel error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.32 bmcv_image_canny

Canny operator for edge detection.

Processor model support

This interface only supports BM1684.

Interface form:

```
bm_status_t bmcv_image_canny(
        bm_handle_t handle,
        bm_image input,
        bm_image output,
        float threshold1,
        float threshold2,
        int aperture_size = 3,
        bool l2gradient = false);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of input image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to open up new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. The output bm_image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem to open up new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

- float threshold1

  The first threshold in the lag process.

- float threshold2

  The second threshold in the lag process.

- int aperture_size = 3

  The size of Sobel core, which currently supports only 3.

- bool l2gradient = false

  Whether to use L2 norm to calculate image gradient. The default value is false.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_GRAY	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and output must be the same.
3. The currently supported maximum image width is 2048.
4. The stride and width of the input image must be the same.

Code example:

```
int channel  = 1;
int width    = 1920;
int height   = 1080;
int dev_id   = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
   src_data[i] = rand() % 255;
}
// calculate res
bm_image input, output;
bm_image_create(handle,
            height,
            width,
            FORMAT_GRAY,
            DATA_TYPE_EXT_1N_BYTE,
            &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
            height,
            width,
            FORMAT_GRAY,
            DATA_TYPE_EXT_1N_BYTE,
            &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_canny(handle, input, output, 0, 200)) {
   std::cout << "bmcv canny error !!!" << std::endl;
   bm_image_destroy(input);
   bm_image_destroy(output);
   bm_dev_free(handle);
   exit(-1);
```

(continues on next page)

(continued from previous page)

```
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.33 bmcv_image_yuv2hsv

Convert the specified area of YUV image to HSV format.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_yuv2hsv(
      bm_handle_t handle,
      bmcv_rect_t rect,
      bm_image    input,
      bm_image    output);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bmcv_rect_t rect

  Describes the starting coordinates and size of the area to be converted in the original image. Refer to bmcv_image_crop for specific parameters.

- bm_image input

  Input parameter. The bm_image of input image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to open up new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem to open up new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

bm1684:The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_YUV420P	FORMAT_HSV_PLANAR
2	FORMAT_NV12	FORMAT_HSV_PLANAR
3	FORMAT_NV21	FORMAT_HSV_PLANAR

bm1684x: The interface currently

- supports the following input image_format

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_NV12
3	FORMAT_NV21

- supports the following output image_format:

num	output image_format
1	FORMAT_HSV180_PACKED
2	FORMAT_HSV256_PACKED

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.

Code example:

```
int channel   = 2;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
        new unsigned char[channel * width * height],
```

(continues on next page)

(continued from previous page)

```
        std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
for (int i = 0; i < channel * width * height; i++) {
    src_data[i] = rand() % 255;
}
// calculate res
bmcv_rect_t rect;
rect.start_x  = 0;
rect.start_y  = 0;
rect.crop_w   = width;
rect.crop_h   = height;
bm_image input, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_NV12,
                DATA_TYPE_EXT_1N_BYTE,
                &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_HSV_PLANAR,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_yuv2hsv(handle, rect, input, output)) {
    std::cout << "bmcv yuv2hsv error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.34 bmcv_image_gaussian_blur

Gaussian blur of the image.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_gaussian_blur(
        bm_handle_t handle,
        bm_image input,
        bm_image output,
        int kw,
        int kh,
        float sigmaX,
        float sigmaY = 0);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of input image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to open up new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem to open up new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be automatically allocated within the API.

- int kw

  The size of the kernel in the width direction.

- int kh

  The size of the kernel in the height direction.

- float sigmaX

  Gaussian kernel standard deviation in the x-direction.

- float sigmaY = 0

  Gaussian kernel standard deviation in the y-direction. If it is 0, it means that it is the same as the Gaussian kernel standard deviation in the x direction.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_BGR_PACKED	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied for.
2. The data_type and image_format of input and must be the same.
3. The maximum width of the image supported by BM1684 is (2048 - kw), the maximum width supported by BM1684X is 4096, and the maximum height is 8192.

4. The maximum convolution kernel width and height supported by BM1684 is 31, and the maximum convolution kernel width and height supported by BM1684X is 3. **Code example:**

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
std::shared_ptr<unsigned char> res_ptr(
      new unsigned char[channel * width * height],
      std::default_delete<unsigned char[]>());
unsigned char * src_data = src_ptr.get();
unsigned char * res_data = res_ptr.get();
```

(continues on next page)

(continued from previous page)

```
for (int i = 0; i < channel * width * height; i++) {
    src_data[i] = rand() % 255;
}
// calculate res
bm_image input, output;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &input);
bm_image_alloc_dev_mem(input);
bm_image_copy_host_to_device(input, (void **)&src_data);
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &output);
bm_image_alloc_dev_mem(output);
if (BM_SUCCESS != bmcv_image_gaussian_blur(handle, input, output, 3, 3, 0.1)) {
    std::cout << "bmcv gaussian blur error !!!" << std::endl;
    bm_image_destroy(input);
    bm_image_destroy(output);
    bm_dev_free(handle);
    exit(-1);
}
bm_image_copy_device_to_host(output, (void **)&res_data);
bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.35 bmcv_image_transpose

The interface can transpose image width and height.

Processor model support

This interface supports BM1684/BM1684X.

Interface form::

```
bm_status_t bmcv_image_transpose(
        bm_handle_t handle,
        bm_image input,
        bm_image output
);
```

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity), which can be obtained by calling bm_dev_request.

- bm_image input

  Input parameter. The bm_image structure of the input image.

- bm_image output

  Output parameter. The bm_image structure of the output image.

Return value description:

- BM_SUCCESS: success
- Other: failed

Code example

```
#include <iostream>
#include <vector>
#include "bmcv_api_ext.h"
#include "bmlib_utils.h"
#include "common.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <memory>

int main(int argc, char *argv[]) {
    bm_handle_t handle;
    bm_dev_request(&handle, 0);

    int image_n = 1;
    int image_h = 1080;
    int image_w = 1920;
    bm_image src, dst;
    bm_image_create(handle, image_h, image_w, FORMAT_RGB_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &src);
    bm_image_create(handle, image_w, image_h, FORMAT_RGB_PLANAR,
          DATA_TYPE_EXT_1N_BYTE, &dst);
    std::shared_ptr<u8*> src_ptr = std::make_shared<u8*>(
          new u8[image_h * image_w * 3]);
    memset((void *)(*src_ptr.get()), 148, image_h * image_w * 3);
    u8 *host_ptr[] = {*src_ptr.get()};
    bm_image_copy_host_to_device(src, (void **)host_ptr);
    bmcv_image_transpose(handle, src, dst);
    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_dev_free(handle);
    return 0;
}
```

Note:

1. This API requires that the input and output bm_image have the same image format. It supports the following formats:

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_BGR_PLANAR
3	FORMAT_GRAY

2. This API requires that the input and output bm_image have the same data type. It supports the following types:

num	data_type
1	DATA_TYPE_EXT_FLOAT32
2	DATA_TYPE_EXT_1N_BYTE
3	DATA_TYPE_EXT_4N_BYTE
4	DATA_TYPE_EXT_1N_BYTE_SIGNED
5	DATA_TYPE_EXT_4N_BYTE_SIGNED

3. The width of the output image must be equal to the height of the input image, and the height of the output image must be equal to the width of the input image;
4. It supports input images with stride;
5. The Input / output bm_image structure must be created in advance, or a failure will be returned.
6. The input bm_image must attach device memory, otherwise, a failure will be returned.
7. If the output object does not attach device memory, it will internally call bm_image_alloc_dev_mem to apply for internally managed device memory and fill the transposed data into device memory.

5.36 bmcv_image_morph

It can do the basic morphological operation of the image, including dilation and erosion.

Users can use this function in the following two steps:

5.36.1 Get the Device Memory of Kernel

Users can use the following interface to obtain the Device Memory of Kernel during initialization. Of course, users can also customize Kernel and ignore this step directly.

The function passes in the size and shape of the required Kernel and returns the corresponding Device Memory to the subsequent morphological operation interface. In the end, users need to manually free the space.

Processor model support

This interface only supports BM1684.

Interface form:

```
typedef enum {
   BM_MORPH_RECT,
   BM_MORPH_CROSS,
   BM_MORPH_ELLIPSE
} bmcv_morph_shape_t;

bm_device_mem_t bmcv_get_structuring_element(
       bm_handle_t handle,
       bmcv_morph_shape_t shape,
       int kw,
       int kh
       );
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bmcv_morph_shape_t shape

  Input parameter. The shape of Kernel. Currently supporting rectangle, cross and ellipse.

- int kw

  Input parameter. The width of Kernel.

- int kh

  Input parameter. The height of Kernel.

Return value description:

The Device Memory corresponding to the return Kernel.

5.36.2 Morphological operation

Currently supporting dilation and erosion. Users can also implment the following functions through the combination of these two basic operations:

- Opening
- Closing
- Morphological Gradient
- Top Hat
- Black Hat

Interface form:

```
bm_status_t bmcv_image_erode(
        bm_handle_t handle,
        bm_image src,
        bm_image dst,
        int kw,
        int kh,
        bm_device_mem_t kmem
        );

bm_status_t bmcv_image_dilate(
        bm_handle_t handle,
        bm_image src,
        bm_image dst,
        int kw,
        int kh,
        bm_device_mem_t kmem
        );
```

Parameter Description:

- bm_handle_t handle

  Input parameters. The handle of bm_handle.

- bm_image src

  Input parameter. The bm_image to be processed. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to open up new memory, or use bmcv_image_attach to attach existing memory.

- bm_image dst

  Output paramete. The bm_image of the processed image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to open up new memory, or use bmcv_image_attach to attach the existing memory. If users do not apply, the internal memory will be applied automatically.

- int kw

  Input parameter. The width of Kernel.

- int kh

  Input parameter. The height of Kernel.

- bm_device_mem_t kmem

  Input parameter. The Device Memory space that stores Kernel, which can be accessed through the interface bmcv_get_structuring_element. Users can also customize it. The value of 1 means that the pixel is selected, and tthe value of 0 means that the pixel is ignored.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	image_format
1	FORMAT_GRAY
2	FORMAT_RGB_PLANAR
3	FORMAT_BGR_PLANAR
4	FORMAT_RGB_PACKED
5	FORMAT_BGR_PACKED

The following data are currently supported_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Code example:

```
int channel   = 1;
int width     = 1920;
int height    = 1080;
int kw        = 3;
int kh        = 3;
int dev_id    = 0;
bmcv_morph_shape_t shape = BM_MORPH_RECT;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
bm_device_mem_t kmem = bmcv_get_structuring_element(
      handle,
```

(continues on next page)

(continued from previous page)

```
        shape,
        kw,
        kh);
std::shared_ptr<unsigned char> data_ptr(
        new unsigned char[channel * width * height],
        std::default_delete<unsigned char[]>());
for (int i = 0; i < channel * width * height; i++) {
    data_ptr.get()[i] = rand() % 255;
}
// calculate res
bm_image src, dst;
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &src);
bm_image_create(handle,
                height,
                width,
                FORMAT_GRAY,
                DATA_TYPE_EXT_1N_BYTE,
                &dst);
bm_image_alloc_dev_mem(src);
bm_image_alloc_dev_mem(dst);
bm_image_copy_host_to_device(src, (void **)&(data_ptr.get()));
if (BM_SUCCESS != bmcv_image_erode(handle, src, dst, kw, kh, kmem)) {
    std::cout << "bmcv erode error !!!" << std::endl;
    bm_image_destroy(src);
    bm_image_destroy(dst);
    bm_free_device(handle, kmem);
    bm_dev_free(handle);
    return;
}
bm_image_copy_device_to_host(dst, (void **)&(data_ptr.get()));
bm_image_destroy(src);
bm_image_destroy(dst);
bm_free_device(handle, kmem);
bm_dev_free(handle);
```

5.37 bmcv_image_mosaic

This interface is used to print one or more mosaics on the image.

Interface form:

```
bm_status_t bmcv_image_mosaic(
        bm_handle_t handle,
        int mosaic_num,
```

(continues on next page)

(continued from previous page)

```
        bm_image input,
        bmcv_rect_t * mosaic_rect,
        int is_expand)
```

Processor model support

This interface only supports BM1684X.

Description of incoming parameters:

- bm_handle_t handle

  Input parameter. HDC (handle of device' s capacity) obtained by calling bm_dev_request.

- int mosaic_num

  Input parameter. The number of mosaic, which refers to the number of bmcv_rect_t objects in the rects pointer.

- bm_image input

  Input parameter. The bm_image on which users need to add mosaic.

- bmcv_rect_t* mosaic_rect

  Input parameter. Pointer to a mosaic object that contains the starting point, width, and height of the mosaic. Refer to the following data type description for details.

- int is_expand

  Input parameters. Whether to expand columns. A value of 0 means that the column is not expanded, and a value of 1 means that a macro block (8 pixels) is expanded around the original mosaic.

Return value description:

- BM_SUCCESS: success
- Other: failed

Data type description:

```
typedef struct bmcv_rect {
   int start_x;
   int start_y;
   int crop_w;
   int crop_h;
} bmcv_rect_t;
```

- start_x describes the starting horizontal coordinate of where the mosaic is located in the original image. It starts at 0 from left to right and takes values in the range [0, width).

- start_y describes the starting vertical coordinate of where the mosaic is located in the original image. It starts at 0 from top to bottom and takes values in the range [0, height).
- crop_w describes the width of the crop image.
- crop_h describes the height of the crop image.

Note:

1.bm1684x supports the following data_type of bm_image:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

- bm1684x supports the following image_format of bm_image:

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

Returns a failure if the input and output format requirements are not met.

2. All input and output bm_image structures must be created in advance, or a failure will be returned.
3. If the width and height of the mosaic are not aligned with 8, it will automatically align up to 8. If it is in the edge area, the 8 alignment will extend toward the non edge direction.
4. If the mosaic area exceeds the width and height of the original drawing, the exceeding part will be automatically pasted to the edge of the original drawing.
5. Only mosaic sizes above 8x8 are supported.

5.38 bmcv_image_laplacian

Laplacian operator of gradient calculation.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_laplacian(
    bm_handle_t handle,
    bm_image input,
    bm_image output,
    unsigned int ksize);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_image input

  Input parameter. The bm_image of input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. The output bm_image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem to create new memory or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

- int ksize = 3

  The number of Laplacian nucleus. Must be 1 or 3.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format:

num	input image_format	output image_format
1	FORMAT_GRAY	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1. Before calling this interface, users must ensure that the input image memory has been applied.
2. The data_type of input and output must be the same.
3. Currently, the maximum supported image width is 2048.

Code example:

```
int loop =1;
int ih = 1080;
int iw = 1920;
unsigned int ksize = 3;
bm_image_format_ext fmt = FORMAT_GRAY;

fmt = argc > 1 ? (bm_image_format_ext)atoi(argv[1]) : fmt;
ih = argc > 2 ? atoi(argv[2]) : ih;
iw = argc > 3 ? atoi(argv[3]) : iw;
loop = argc > 4 ? atoi(argv[4]) : loop;
ksize = argc >5 ? atoi(argv[5]) : ksize;

bm_status_t ret = BM_SUCCESS;
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS)
    throw("bm_dev_request failed");

bm_image_data_format_ext data_type = DATA_TYPE_EXT_1N_BYTE;
bm_image input;
bm_image output;

bm_image_create(handle, ih, iw, fmt, data_type, &input);
bm_image_alloc_dev_mem(input);

bm_image_create(handle,ih, iw, fmt, data_type, &output);
bm_image_alloc_dev_mem(output);

std::shared_ptr<unsigned char*> ch0_ptr = std::make_shared<unsigned char*>
↪(new unsigned char[ih * iw]);
std::shared_ptr<unsigned char*> tpu_res_ptr = std::make_shared<unsigned char *>
↪(new unsigned char[ih * iw]);
std::shared_ptr<unsigned char*> cpu_res_ptr = std::make_shared<unsigned char *>
↪(new unsigned char[ih*iw]);
```

(continues on next page)

(continued from previous page)

```
for (int i = 0; i < loop; i++) {
    for (int j = 0; j < ih * iw; j++) {
        (*ch0_ptr.get())[j] = j % 256;
    }

    unsigned char *host_ptr[] = {*ch0_ptr.get()};
    bm_image_copy_host_to_device(input, (void **)host_ptr);

    ret = bmcv_image_laplacian(handle, input, output, ksize);
    if (ret) {
        cout << "test laplacian failed" << endl;
        bm_image_destroy(input);
        bm_image_destroy(output);
        bm_dev_free(handle);
        return ret;
    } else {
        host_ptr[0] = *tpu_res_ptr.get();
        bm_image_copy_device_to_host(output, (void **)host_ptr);
    }
}

bm_image_destroy(input);
bm_image_destroy(output);
bm_dev_free(handle);
```

5.39 bmcv_image_lkpyramid

LK pyramid optical flow algorithm. The complete call flow include creation, execution and destruction. The first half of the algorithm uses Tensor Computing Processor, and the second half uses Processor for serial operation. Therefore, for PCIe mode, it is recommended to enable Processor to accelerate. Please refer to Chapter 5 for specific steps.

5.39.1 Create

The internal implementation of the algorithm requires some cache space. Therefore, in order to avoid releasing the space repeatedly, some preparatory work is encapsulated in the creation interface. Users can call the execute interface multiple times by calling it once before starting (create function parameters unchanged). The interface form is as follows:

```
bm_status_t bmcv_image_lkpyramid_create_plan(
        bm_handle_t handle,
        void*& plan,
        int width,
        int height,
        int winW = 21,
        int winH = 21,
        int maxLevel = 3);
```

Processor model support

This interface only supports BM1684.

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- void*& plan

  Output parameter. Handle required by the execution phase.

- int width

  Input parameter. The width of the image to be processed.

- int height

  Input parameter. The height of the image to be processed.

- int winW

  Input parameter. The width of the algorithm processing window, and the default value is 21.

- int winH

  Input parameter. The height of the algorithm processing window, and the default value is 21.

- int maxLevel

  Input parameter. The height of pyramid processing. The default value is 3, and the maximum value currently supported is 5. The larger the parameter value, the longer the execution time of the algorithm. It is recommended to select the acceptable minimum value according to the actual effect.

Return value description:

- BM_SUCCESS: success
- Other: failed

5.39.2 Execute

The plan created with the above interface can start the real execution phase. The interface format is as follows:

```
typedef struct {
    float x;
    float y;
} bmcv_point2f_t;

typedef struct {
```

(continues on next page)

(continued from previous page)

```
    int type;   // 1: maxCount   2: eps   3: both
    int max_count;
    double epsilon;
} bmcv_term_criteria_t;

bm_status_t bmcv_image_lkpyramid_execute(
        bm_handle_t handle,
        void* plan,
        bm_image prevImg,
        bm_image nextImg,
        int ptsNum,
        bmcv_point2f_t* prevPts,
        bmcv_point2f_t* nextPts,
        bool* status,
        bmcv_term_criteria_t criteria = {3, 30, 0.01});
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- const void *plan

  Input parameter. The handle obtained during the creation phase.

- bm_image prevImg

  Input parameter. The bm_image of the previous image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image nextImg

  Input parameter. The bm_image of the next image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- int ptsNum

  Input parameter. The number of points to be tracked.

- bmcv_point2f_t* prevPts

  Input parameter. It is required to track the coordinate pointer of the point in the previous image. Its pointing length is ptsNum.

- bmcv_point2f_t* nextPts

  Output parameter. The coordinate pointer of calculated tracking point in the next image. Its pointing length is ptsNum.

- bool* status

Output parameter. Whether each tracking point in nextPts is valid or not. Its pointing length is ptsNum, which corresponds to the coordinates in nextPts one by one. If it is valid, it is true, otherwise it is false (it means that the corresponding tracking point is not found in the next image, which may exceed the image range).

- bmcv_term_criteria_t criteria

  Input parameter. Iteration end criteria. Type indicates which parameter is used as the judgment condition of end: if it is 1, it is determined by the number of iterations max_count as the end judgment parameter. If it is 2, the error epsilon is the end judgment parameter. If it is 3, both must be met. This parameter will affect the execution time. It is suggested to select the optimal standard according to the actual effect.

Return value description:

- BM_SUCCESS: success
- Other: failed

5.39.3 Destruction

When the execution is completed, the created handle needs to be destroyed. This interface must be the same as the creation interface bmcv_image_lkpyramid_create_plan and used in pairs.

```
void bmcv_image_lkpyramid_destroy_plan(bm_handle_t handle, void *plan);
```

Format support:

The interface currently supports the following image_format:

num	image_format
1	FORMAT_GRAY

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

5.39.4 Sample Code

```
bm_handle_t handle;
bm_status_t ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
    printf("Create bm handle failed. ret = %d\n", ret);
    return -1;
}
ret = bmcv_open_cpu_process(handle);
if (ret != BM_SUCCESS) {
    printf("BMCV enable Processor failed. ret = %d\n", ret);
    bm_dev_free(handle);
    return -1;
}
bm_image_format_ext fmt = FORMAT_GRAY;
bm_image prevImg;
bm_image nextImg;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
↪prevImg);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
↪nextImg);
bm_image_alloc_dev_mem(prevImg);
bm_image_alloc_dev_mem(nextImg);
bm_image_copy_host_to_device(prevImg, (void **)(&prevPtr));
bm_image_copy_host_to_device(nextImg, (void **)(&nextPtr));
void *plan = nullptr;
bmcv_image_lkpyramid_create_plan(
        handle,
        plan,
        width,
        height,
        kw,
        kh,
        maxLevel);
bmcv_image_lkpyramid_execute(
        handle,
        plan,
        prevImg,
        nextImg,
        ptsNum,
        prevPts,
        nextPts,
        status,
        criteria);
bmcv_image_lkpyramid_destroy_plan(handle, plan);
bm_image_destroy(prevImg);
bm_image_destroy(nextImg);
ret = bmcv_close_cpu_process(handle);
if (ret != BM_SUCCESS) {
    printf("BMCV disable Processor failed. ret = %d\n", ret);
    bm_dev_free(handle);
    return -1;
```

(continues on next page)

(continued from previous page)

```
}
bm_dev_free(handle);
```

5.40 bmcv_debug_savedata

This interface is used to input bm_image object to the internally defined binary file for debugging. The binary file format and parsing method are given in the example code.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_debug_savedata(
      bm_image input,
      const char *name
);
```

Parameter Description:

- bm_image input

  Input parameter. Input bm_image.

- const char* name

  Input parameter. The saved binary file path and name.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. Before calling bmcv_debug_savedata(), users must ensure that the input image has been created correctly and guaranteed is_attached, otherwise the function will return a failure.

Code example and binary file parsing method::

```
bm_image input;
bm_image_create(handle,
   1080,
   1920,
   FORMAT_BGR_PLANAR,
   DATA_TYPE_EXT_1N_BYTE,
   &input);
bm_image_alloc_dev_mem(input);
// ... your own function
```

(continues on next page)

(continued from previous page)

```
bmcv_debug_savedata(input, "input.bin");
// now a file named "input.bin" is generated in current folder

// the following code shows how to parse the binary file
FILE *   fp           = fopen("input.bin", "rb");
uint32_t data_offset  = 0;
uint32_t width        = 0;
uint32_t height       = 0;
uint32_t image_format = 0;
uint32_t data_type    = 0;
uint32_t plane_num    = 0;

uint32_t stride[4] = {0};
uint64_t size[4]         = {0};

fread(&data_offset, sizeof(uint32_t), 1, fp);
fread(&width, sizeof(uint32_t), 1, fp);
fread(&height, sizeof(uint32_t), 1, fp);
fread(&image_format, sizeof(uint32_t), 1, fp);
fread(&data_type, sizeof(uint32_t), 1, fp);
fread(&plane_num, sizeof(uint32_t), 1, fp);

fread(size, sizeof(size), 1, fp);
fread(stride, sizeof(stride), 1, fp);

uint32_t channel_stride[4] = {0};
uint32_t batch_stride[4]   = {0};
uint32_t meta_data_size[4] = {0};

uint32_t N[4] = {0};
uint32_t C[4] = {0};
uint32_t H[4] = {0};
uint32_t W[4] = {0};

fread(channel_stride, sizeof(channel_stride), 1, fp);
fread(batch_stride, sizeof(batch_stride), 1, fp);
fread(meta_data_size, sizeof(meta_data_size), 1, fp);

fread(N, sizeof(N), 1, fp);
fread(C, sizeof(C), 1, fp);
fread(H, sizeof(H), 1, fp);
fread(W, sizeof(W), 1, fp);

fseek(fp, data_offset, SEEK_SET);
std::vector<std::unique_ptr<unsigned char[]>> host_ptr;
host_ptr.resize(plane_num);
void* void_ptr[4] = {0};
for (uint32_t i = 0; i < plane_num; i++) {
    host_ptr[i] =
        std::unique_ptr<unsigned char[]>(new unsigned char[size[i]]);
    void_ptr[i] = host_ptr[i].get();
```

(continues on next page)

(continued from previous page)

```
    fread(host_ptr[i].get(), 1, size[i], fp);
}
fclose(fp);
std::cout << "image width " << width << " image height " << height
        << " image format " << image_format << " data type " << data_type
        << " plane num " << plane_num << std::endl;
for (uint32_t i = 0; i < plane_num; i++) {
    std::cout << "plane" << i << " size " << size[i] << " C " << C[i]
            << " H " << H[i] << " W " << W[i] << " stride "
            << stride[i] << std::endl;
}
// The following shows how to recover the image
bm_image recover;
bm_image_create(handle,
                height,
                width,
                (bm_image_format_ext)image_format,
                (bm_image_data_format_ext)data_type,
                &recover,
                (int *)stride);
bm_image_copy_host_to_device(recover, (void **)&void_ptr);
bm_image_write_to_bmp(recover, "recover.bmp");
bm_image_destroy(recover);
```

5.41 bmcv_sort

This interface can sort floating-point data (ascending/descending), and support the index corresponding to the original data after sorting.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_sort(bm_handle_t     handle,
                      bm_device_mem_t src_index_addr,
                      bm_device_mem_t src_data_addr,
                      int             data_cnt,
                      bm_device_mem_t dst_index_addr,
                      bm_device_mem_t dst_data_addr,
                      int             sort_cnt,
                      int             order,
                      bool            index_enable,
                      bool            auto_index);
```

Input parameter description:

- bm_handle_t handle

Input parameter. The handle of input bm_handle.

- bm_device_mem_t src_index_addr

  Input parameter. The address of the index corresponding to each input data. If index_enable rather than auto_index is used, this parameter is valid. bm_device_mem_t is the built-in data type representing the address. The function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the example code.

- bm_device_mem_t src_data_addr

  Input parameter. The address corresponding to the input data to be sorted. bm_device_mem_t is the built-in data type representing the address. The function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the example code.

- int data_cnt

  Input parameter. The number of input data to be sorted.

- bm_device_mem_t dst_index_addr

  Output parameter. The address of the index corresponding to the output data after sorting. If index_enable rather than auto_index is used, this parameter is valid. bm_device_mem_t is the built-in data type representing the address. The function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the example code.

- bm_device_mem_t dst_data_addr

  Output parameter. The address corresponding to the sorted output data. bm_device_mem_t is the built-in data type representing the address. The function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the example code.

- int sort_cnt

  Input parameter. The quantity to be sorted, that is, the number of output results, including the ordered data and the corresponding index. For example, in descending order, if you only need to output the top 3 data, set this parameter to 3.

- int order

  Input parameter. Ascending or descending, 0 means ascending and 1 means descending.

- bool index_enable

  Input parameter. Whether index is enabled. If enabled, the index corresponding to the sorted data can be output; otherwise, src_index_addr and dst_index_addr will be invalid.

- bool auto_index

  Input parameter. Whether to enable the automatic generation of index function. The premise of using this function is index_enable parameter is true. If the parameter is

also true, it means counting from 0 according to the storage order of the input data as index and src_index_addr is invalid, and the index corresponding to the ordered data in the output result is stored in dst_index_addr.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. It is required that sort_cnt <= data_cnt.
2. If the auto index function is required, the precondition is the parameter index_enable is true.
3. The API can support full sorting of 1MB data at most.

Sample code

```
int data_cnt = 100;
int sort_cnt = 50;
float src_data_p[100];
int src_index_p[100];
float dst_data_p[50];
int dst_index_p[50];
for (int i = 0; i < 100; i++) {
   src_data_p[i] = rand() % 1000;
   src_index_p[i] = 100 - i;
}
int order = 0;
bmcv_sort(handle,
        bm_mem_from_system(src_index_p),
        bm_mem_from_system(src_data_p),
        data_cnt,
        bm_mem_from_system(dst_index_p),
        bm_mem_from_system(dst_data_p),
        sort_cnt,
        order,
        true,
        false);
```

5.42 bmcv_base64_enc(dec)

A common encoding method in base64 network transmission, which uses 64 common characters to encode 6-bit binary numbers.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_base64_enc(bm_handle_t handle,
      bm_device_mem_t src,
      bm_device_mem_t dst,
      unsigned long len[2])

bm_status_t bmcv_base64_dec(bm_handle_t handle,
      bm_device_mem_t src,
      bm_device_mem_t dst,
      unsigned long len[2])
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t src

  Input parameter. The address of the input character string. The type is bm_device_mem_t. It is required to call bm_mem_from_system() to convert the data address to the corresponding structure of bm_device_mem_t.

- bm_device_mem_t dst

  Input parameter. The address of the output character string. The type is bm_device_mem_t. It is required to call bm_mem_from_system() to convert the data address to the corresponding structure of bm_device_mem_t.

- unsigned long len[2]

  Input parameter. The length of base64 encoding or decoding, in bytes. Len [0] represents the input length, which needs to be given by the caller. Len [1] is the output length, which is calculated by API.

Return value:

- BM_SUCCESS: success
- Other: failed

Code example:

```
int original_len[2];
int encoded_len[2];
int original_len[0] = (rand() % 134217728) + 1;
int encoded_len[0] = (original_len + 2) / 3 * 4;
char *src = (char *)malloc((original_len + 3) * sizeof(char));
char *dst = (char *)malloc((encoded_len + 3) * sizeof(char));
for (j = 0; j < original_len; j++)
    src[j] = (char)((rand() % 100) + 1);

bm_handle_t handle;
ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
```

(continues on next page)

(continued from previous page)

```
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
bmcv_base64_enc(
    handle,
    bm_mem_from_system(src),
    bm_mem_from_system(dst),
    original_len);

bmcv_base64_dec(
    handle,
    bm_mem_from_system(dst),
    bm_mem_from_system(src),
    original_len);

bm_dev_free(handle);
free(src);
free(dst);
```

Note:

1. The API can encode and decode up to 128MB of data at a time, that is, the parameter len cannot exceed 128MB.
2. The supported incoming address type is system or device at the same time.
3. encoded_len[1] will give the output length, especially when decoding, calculate the number of bits to be removed according to the end of the input.

5.43 bmcv_feature_match

The interface is used to compare the feature points obtained from the network (int8 format) with the feature points in the database (int8 format),and output the best matching top-k.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_feature_match(
    bm_handle_t     handle,
    bm_device_mem_t input_data_global_addr,
    bm_device_mem_t db_data_global_addr,
    bm_device_mem_t output_sorted_similarity_global_addr,
    bm_device_mem_t output_sorted_index_global_addr,
    int             batch_size,
    int             feature_size,
    int             db_size,
```

(continues on next page)

(continued from previous page)

```
    int          sort_cnt = 1,
    int          rshiftbits = 0);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input_data_global_addr

  Input parameter. The address of the feature point data storage to be compared. The data is arranged based on the data format of batch_size * feature_size. The specific meanings of batch_size and feature_size will be introduced below. bm_device_mem_t is the built-in data type representing the address, and the function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the usage in the example code.

- bm_device_mem_t db_data_global_addr

  Input parameter. The address of the feature point data storage of the database. The data is arranged based on the data format of feature_size * db_size. The specific meanings of feature_size and db_size will be introduced below. bm_device_mem_t is the built-in data type representing the address, and the function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the usage in the example code.

- bm_device_mem_t output_sorted_similarity_global_addr

  Output parameter. The storage address of the maximum values (in descending order) of the comparison results obtained by each batch. The specific number of values is determined by sort_cnt. The data is arranged based on the data format of batch_size * sort_cnt. The specific meaning of batch_size will be introduced below. bm_device_mem_t is the built-in data representing the address type, you can use the function bm_mem_from_system(addr) to convert the pointer or address used by ordinary users to For this type, users can refer to the usage in the sample code.

- bm_device_mem_t output_sorted_index_global_addr

  Output parameter. The storage address of the serial number in the database of the comparison result obtained by each batch. For example, for batch 0, if output_sorted_similarity_global_addr is obtained by comparing the input data with the 800th group of feature points in the database, then the data of batch 0 corresponding to the address of output_sorted_index_global_addr is 800. The data in output_sorted_similarity_global_addr is arranged in the data format of batch_size * sort_cnt. The specific meaning of batch_size will be introduced below. bm_device_mem_t is the built-in data type representing the address, and the function bm_mem_from_system (addr) can be used to convert the pointer or address used by ordinary users to this type. Users can refer to the usage in the example code.

- int batch_size

Input parameter. The number of batch whose data is to be input. If the input data has 4 groups of feature points, the batch_size of the data is 4. The maximum batch_size should not exceed 8.

- int feature_size

  Input parameter. The number of feature points of each data group. The maximum feature_size should not exceed 4096.

- int db_size

  Input parameter. The number of groups of data feature points in the database. The maximum db_size should not exceed 500000.

- int sort_cnt

  Input parameter. The number to be sorted in each batch comparison result, that is, the number of output results. If the maximum three comparison results are required, set sort_cnt to 3. The defaulted value is 1. The maximum sort_cnt should not exceed 30.

- int rshiftbits

  Input parameter. The number of digits of shifting the result to the right, which uses round to round the decimal. This parameter defaults to 0.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. The data type of input data and data in the database is char.
2. The output comparison result data type is short, and the output sequence number type is int.
3. The data in the database is arranged in the memory as feature_size * db_size. Therefore, it is necessary to transpose a group of feature points before putting them into the database.
4. The value range of sort_cnt is 1 ~ 30.

Sample code

```
int batch_size = 4;
int feature_size = 512;
int db_size = 1000;
int sort_cnt = 1;
unsigned char src_data_p[4 * 512];
unsigned char db_data_p[512 * 1000];
short output_val[4];
int output_index[4];
for (int i = 0; i < 4 * 512; i++) {
   src_data_p[i] = rand() % 1000;
```

(continues on next page)

(continued from previous page)

```
}
for (int i = 0; i < 512 * 1000; i++) {
    db_data_p[i] = rand() % 1000;
}
bmcv_feature_match(handle,
    bm_mem_from_system(src_data_p),
    bm_mem_from_system(db_data_p),
    bm_mem_from_system(output_val),
    bm_mem_from_system(output_index),
    batch_size,
    feature_size,
    db_size,
    sort_cnt, 8);
```

5.44 bmcv_gemm

This interface is used to implement the general multiplication calculation of float32 type matrix, as shown in the following formula:

$$C = \alpha \times A \times B + \beta \times C$$

Among them, A, B and C are matrices,and α and β are constant coefficients.

The format of the interface is as follows:

```
bm_status_t bmcv_gemm(bm_handle_t     handle,
                      bool            is_A_trans,
                      bool            is_B_trans,
                      int             M,
                      int             N,
                      int             K,
                      float           alpha,
                      bm_device_mem_t A,
                      int             lda,
                      bm_device_mem_t B,
                      int             ldb,
                      float           beta,
                      bm_device_mem_t C,
                      int             ldc);
```

Processor model support

This interface supports BM1684/BM1684X.

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bool is_A_trans

  Input parameter. Set whether matrix A is transposed

- bool is_B_trans

  Input parameter. Set whether matrix B is transposed

- int M

  Input parameter. The number of rows of matrix A and matrix C

- int N

  Input parameter. The number of columns of matrix B and matrix C

- int K

  Input parameter. The number of columns of matrix A and the number of rows of matrix B

- float alpha

  Input parameter. Number multiplication coefficient

- bm_device_mem_t A

  Input parameter. Save the device address or host address of the left matrix A data according to the data storage location. If the data is stored in the host space, it will automatically complete the handling of s2d.

- int lda

  Input parameter. The leading dimension of matrix A, that is, the size of the first dimension, is the number of columns (no transpose) or rows (transpose) of A when there is no stride between rows.

- bm_device_mem_t B

  Input parameter. Save the device address or host address of the right matrix B data according to the data storage location. If the data is stored in the host space, it will automatically complete the handling of s2d.

- int ldb

  Input parameter. The leading dimension of matrix C, that is, the size of the first dimension, is the number of columns (no transpose) or rows (transpose) of B when there is no stride between rows.

- float beta

  Input parameter. Number multiplication factor.

- bm_device_mem_t C

  Output parameter. Save the device address or host address of matrix C data according to the data storage location. If it is the host address, when the beta is not 0, the

transportation of s2d will be completed automatically before calculation, and then the transportation of d2s will be completed automatically after calculation.

- int ldc

  Input parameter. The leading dimension of matrix C, that is, the size of the first dimension, is the number of columns of C when there is no stride between rows.

Return value description:

- BM_SUCCESS: success
- Other: failed

Sample code

```
int M = 3, N = 4, K = 5;
float alpha = 0.4, beta = 0.6;
bool is_A_trans = false;
bool is_B_trans = false;
float *A    = new float[M * K];
float *B    = new float[N * K];
float *C    = new float[M * N];
memset(A, 0x11, M * K * sizeof(float));
memset(B, 0x22, N * K * sizeof(float));
memset(C, 0x33, M * N * sizeof(float));

bmcv_gemm(handle,
        is_A_trans,
        is_B_trans,
        M,
        N,
        K,
        alpha,
        bm_mem_from_system((void *)A),
        is_A_trans ? M : K,
        bm_mem_from_system((void *)B),
        is_B_trans ? K : N,
        beta,
        bm_mem_from_system((void *)C),
        N);
delete A;
delete B;
delete C;
```

5.45 bmcv_gemm_ext

This interface is used to implement the general multiplication calculation of float32 or float16 type matrix, as shown in the following formula:

$$Y = \alpha \times A \times B + \beta \times C$$

Among them, A, B, C and Y are matrices,and α and β are constant coefficients.

The format of the interface is as follows:

```
bm_status_t bmcv_gemm_ext(bm_handle_t     handle,
                  bool           is_A_trans,
                  bool           is_B_trans,
                  int            M,
                  int            N,
                  int            K,
                  float          alpha,
                  bm_device_mem_t A,
                  bm_device_mem_t B,
                  float          beta,
                  bm_device_mem_t C,
                  bm_device_mem_t Y,
                  bm_image_data_format_ext input_dtype,
                  bm_image_data_format_ext output_dtype);
```

Processor model support

This interface only supports BM1684X.

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bool is_A_trans

  Input parameter. Set whether matrix A is transposed

- bool is_B_trans

  Input parameter. Set whether matrix B is transposed

- int M

  Input parameter. The number of rows of matrix A, matrix C and matrix Y

- int N

  Input parameter. The number of columns of matrix B, matrix C and matrix Y

- int K

Input parameter. The number of columns of matrix A and the number of rows of matrix B

- float alpha

  Input parameter. Number multiplication coefficient

- bm_device_mem_t A

  Input parameter. The device address of the left matrix A data is stored according to the data storage location, and the data s2d handling needs to be completed before use.

- bm_device_mem_t B

  Input parameter. The device address of the right matrix B data is stored according to the data storage location, and the data s2d handling needs to be completed before use.

- float beta

  Input parameter. Number multiplication factor.

- bm_device_mem_t C

  Input parameters. The device address of the matrix C data is stored according to the data storage location, and the data s2d handling needs to be done before use.

- bm_device_mem_t Y

  Output parameters. The device address of the matrix Y data, which holds the output result.

- bm_image_data_format_ext input_dtype

  Input parameters. Data type of input matrix A, B, C. Support input FP16-output FP16 or FP32, input FP32-output FP32.

- bm_image_data_format_ext output_dtype

  Input parameters. The data type of the output matrix Y.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note

1. In the case of FP16 input and A matrix transpose, M only supports values less than or equal to 64.
2. This interface does not support FP32 input and FP16 output.

Sample code

```
int M = 3, N = 4, K = 5;
float alpha = 0.4, beta = 0.6;
bool is_A_trans = false;
```

(continues on next page)

(continued from previous page)

```
bool is_B_trans = false;
float *A     = new float[M * K];
float *B     = new float[N * K];
float *C     = new float[M * N];
memset(A, 0x11, M * K * sizeof(float));
memset(B, 0x22, N * K * sizeof(float));
memset(C, 0x33, M * N * sizeof(float));
bm_device_mem_t input_dev_buffer[3];
bm_device_mem_t output_dev_buffer[1];
bm_malloc_device_byte(handle, &input_dev_buffer[0], M * K * sizeof(float));
bm_malloc_device_byte(handle, &input_dev_buffer[1], N * K * sizeof(float));
bm_malloc_device_byte(handle, &input_dev_buffer[2], M * N * sizeof(float));
bm_memcpy_s2d(handle, input_dev_buffer[0], (void *)A);
bm_memcpy_s2d(handle, input_dev_buffer[1], (void *)B);
bm_memcpy_s2d(handle, input_dev_buffer[2], (void *)C);
bm_malloc_device_byte(handle, &output_dev_buffer[0], M * N * sizeof(float));
bm_image_data_format_ext in_dtype = DATA_TYPE_EXT_FLOAT32;
bm_image_data_format_ext out_dtype = DATA_TYPE_EXT_FLOAT32;
bmcv_gemm_ext(handle,
      is_A_trans,
      is_B_trans,
      M,
      N,
      K,
      alpha,
      input_dev_buffer[0],
      input_dev_buffer[1],
      beta,
      input_dev_buffer[2],
      output_dev_buffer[0],
      in_dtype,
      out_dtype);
delete A;
delete B;
delete C;
delete Y;
for (int i = 0; i < 3; i++)
{
  bm_free_device(handle, input_dev_buffer[i]);
}
bm_free_device(handle, output_dev_buffer[0]);
```

5.46 bmcv_matmul

This interface used to implment the multiplication calculation of 8-bit data type matrix, as shown in the following formula:

$$C = (A \times B) >> rshift\_bit \tag{5.1}$$

or

$$C = alpha \times (A \times B) + beta \tag{5.2}$$

Among them,

- A is the input left matrix, and its data type can be unsigned char or 8-bit data of signed char, with the size of (M, K);
- B is the input right matrix, and its data type can be unsigned char or 8-bit data of signed char, with the size of (K, N);
- C is the output result matrix. Its data type length can be int8, int16 or float32, which is determined by user configuration.

  When C is int8 or int16, execute the function of the above formula (5.1) and its symbol depends on A and B. when A and B are both unsigned, C is an unsigned number, otherwise it is signed;

  When C is float32, execute the function of the above formula (5.2) .
- rshift_bit is the right shift number of the matrix product, which is valid only when C is int8 or int16. Since the matrix product may exceed the range of 8-bit or 16-bit, users can configure a certain right shift number to prevent overflow by discarding some accuracy.
- alpha and beta are constant coefficients of float32, which is valid only when C is float32.

The format of the interface is as follows:

```
bm_status_t bmcv_matmul(bm_handle_t     handle,
                        int             M,
                        int             N,
                        int             K,
                        bm_device_mem_t  A,
                        bm_device_mem_t  B,
                        bm_device_mem_t  C,
                        int             A_sign,
                        int             B_sign,
                        int             rshift_bit,
                        int             result_type,
```

(continues on next page)

(continued from previous page)

```
                bool        is_B_trans,
                float       alpha = 1,
                float       beta = 0);
```

Processor model support

This interface supports BM1684/BM1684X.

输入参数说明：

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- int M

  Input parameter. The number of rows of matrix A and matrix C

- int N

  Input parameter. The number of columns of matrix B and matrix C

- int K

  Input parameter. The number of columns of matrix A and the number of rows of matrix B

- bm_device_mem_t A

  Enter parameters. Save its device address or host address according to the data storage location of left matrix A. If the data is stored in the host space, it will automatically complete the handling of S2D.

- bm_device_mem_t B

  Input parameter. Save its device address or host address according to the data storage location of right matrix B. If the data is stored in the host space, it will automatically complete the handling of s2d.

- bm_device_mem_t C

  Output parameter. Save its device address or host address according to the data storage location of matrix C. If it is the host address, when the beta is not 0, the transportation of s2d will be completed automatically before calculation, and then the transportation of d2s will be completed automatically after calculation.

- int A_sign

  Input parameter. The sign of left matrix A, 1 means signed and 0 means unsigned.

- int B_sign

  Input parameter. The sign of right matrix B, 1 means signed and 0 means unsigned.

- int rshift_bit

Input parameter. The right shift number of matrix product is non-negative. Valid only when result_type is equal to 0 or 1.

- int result_type

  Input parameter. The data type of the output result matrix. 0 means int8, 1 means int16, and 2 means float32.

- bool is_B_trans

  Input parameter. Whether the input right matrix B needs to be transposed before calculation.

- float alpha

  Constant coefficient, which is multiplied by input matrices A and B and then multiplied by this coefficient. Only valid when result_type is equal to 2. The default value is 1.

- float beta

  Constant coefficient, add the offset before the output result matrix C. Only valid when result_type is equal to 2. The default value is 0.

Return value description:

- BM_SUCCESS: success
- Other: failed

Sample code

```
int M = 3, N = 4, K = 5;
int result_type = 1;
bool is_B_trans = false;
int rshift_bit = 0;
char *A     = new char[M * K];
char *B     = new char[N * K];
short *C    = new short[M * N];
memset(A, 0x11, M * K * sizeof(char));
memset(B, 0x22, N * K * sizeof(char));

bmcv_matmul(handle,
          M,
          N,
          K,
          bm_mem_from_system((void *)A),
          bm_mem_from_system((void *)B),
          bm_mem_from_system((void *)C),
          1,
          1,
          rshift_bit,
          result_type,
          is_B_trans);

delete A;
```

(continues on next page)

(continued from previous page)

```
delete B;
delete C;
```

5.47 bmcv_distance

Calculate the Euclidean distance between multiple points and a specific point in multidimensional space. The coordinates of the former are stored in continuous device memory, while the coordinates of a specific point are passed in through parameters. The coordinate value is of float type.

The format of the interface is as follows:

```
bm_status_t bmcv_distance(
      bm_handle_t handle,
      bm_device_mem_t input,
      bm_device_mem_t output,
      int dim,
      const float *pnt,
      int len);
```

Processor model support

This interface supports BM1684/BM1684X.

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle

- bm_device_mem_t input

  Input parameter. Device space for storing len point coordinates. Its size is len*dim*sizeof (float).

- bm_device_mem_t output

  Output parameter. Device space for storing len distances. Its size is len*sizeof (float).

- int dim

  Input parameter. Space dimension size.

- const float *pnt

  Input parameter. The coordinate of a specific point, with the length of dim.

- int len

  Input parameter. Number of coordinates to be calculated.

Return value description:

- BM_SUCCESS: success

- Other: failed

Sample code

```
int L = 1024 * 1024;
int dim = 3;
float pnt[8] = {0};
for (int i = 0; i < dim; ++i)
    pnt[i] = (rand() % 2 ? 1.f : -1.f) * (rand() % 100 + (rand() % 100) * 0.01);
float *XHost = new float[L * dim];
for (int i = 0; i < L * dim; ++i)
    XHost[i] = (rand() % 2 ? 1.f : -1.f) * (rand() % 100 + (rand() % 100) * 0.01);
float *YHost = new float[L];
bm_handle_t handle = nullptr;
bm_dev_request(&handle, 0);
bm_device_mem_t XDev, YDev;
bm_malloc_device_byte(handle, &XDev, L * dim * 4);
bm_malloc_device_byte(handle, &YDev, L * 4);
bm_memcpy_s2d(handle, XDev, XHost);
bmcv_distance(handle,
              XDev,
              YDev,
              dim,
              pnt,
              L));
bm_memcpy_d2s(handle, YHost, YDev));
delete [] XHost;
delete [] YHost;
bm_free_device(handle, XDev);
bm_free_device(handle, YDev);
bm_dev_free(handle);
```

5.48 bmcv_min_max

For a group of data stored in continuous space in device memory, the interface can obtain the maximum and minimum values in the data group.

The format of the interface is as follows:

```
bm_status_t bmcv_min_max(bm_handle_t handle,
                         bm_device_mem_t input,
                         float *minVal,
                         float *maxVal,
                         int len);
```

Processor model support

This interface supports BM1684/BM1684X.

Input parameter description:

- bm_handle_t handle

Input parameter. The handle of bm_handle.

- bm_device_mem_t input

  Input parameter. Input the device address of the data.

- float *minVal

  Input parameter. The minimum value obtained after operation. If it is NULL, the minimum value will not be calculated.

- float *maxVal

  Output parameter. The maximum value obtained after operation. If it is NULL, the maximum value will not be calculated.

- int len

  Input parameter. The length of the input data.

Return value description:

- BM_SUCCESS: success
- Other: failed

Sample code

```
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
int len = 1000;
float max = 0;
float min = 0;
float *input = new float[len];
for (int i = 0; i < 1000; i++) {
    input[i] = (float)(rand() % 1000) / 10.0;
}
bm_device_mem_t input_mem;
bm_malloc_device_byte(handle, input_mem, len * sizeof(float))
bm_memcpy_s2d(handle, input_mem, input);
bmcv_min_max(handle,
             input_mem,
             &min,
             &max,
             len,
             2);
bm_free_device(handle, input_mem);
bm_dev_free(handle);
delete [] input;
```

5.49 bmcv_fft

FFT operation. The complete operation includes creation, execution and destruction.

5.49.1 Create

It supports one-dimensional or two-dimensional FFT calculation. The difference is that in the creation process, the later execution and destruction use the same interface.

For one-dimensional FFT, multi-batch operation is supported. The interface form is as follows:

```
bm_status_t bmcv_fft_1d_create_plan(
        bm_handle_t handle,
        int batch,
        int len,
        bool forward,
        void *&plan);
```

Processor model support

This interface only supports BM1684.

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle

- int batch

  Input parameter. Number of batches.

- int int

  Input parameters. The length of each batch.

- bool forward

  Input parameter. Whether it is a forward transformation. False indicates an inverse transformation.

- void *&plan

  Output parameter. The handle required for execution.

Return value description:

- BM_SUCCESS: success
- Other: failed

For two-dimensional M*N FFT, the inerface form is as follows:

```
bm_status_t bmcv_fft_2d_create_plan(
        bm_handle_t handle,
        int M,
        int N,
        bool forward,
        void *&plan);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle

- int M

  Input parameter. The size of the first dimension.

- int N

  Input parameter. The size of the second dimension.

- bool forward

  Input parameter. Whether it is a forward transformation. False indicates an inverse transformation.

- void *&plan

  Output parameter. The handle required for execution.

Return value Description:

- BM_SUCCESS: success
- Other: failed

5.49.2 Execute

Use the plan created above to start the real execution phase. It supports two interfaces: complex input and real input. Their formats are as follows:

```
bm_status_t bmcv_fft_execute(
        bm_handle_t handle,
        bm_device_mem_t inputReal,
        bm_device_mem_t inputImag,
        bm_device_mem_t outputReal,
        bm_device_mem_t outputImag,
        const void *plan);

bm_status_t bmcv_fft_execute_real_input(
        bm_handle_t handle,
        bm_device_mem_t inputReal,
        bm_device_mem_t outputReal,
```

(continues on next page)

(continued from previous page)

```
    bm_device_mem_t outputImag,
    const void *plan);
```

Input parameter description:

- bm_handle_t handle

  Input parameters. The handle of bm_handle

- bm_device_mem_t inputReal

  Input parameter. The device memory space storing the real number of the input data is batch*len*sizeof (float) for one-dimensional FFT and M*N*sizeof (float) for two-dimensional FFT.

- bm_device_mem_t inputImag

  Input parameter. The device memory space storing the imaginary number of the input data. For one-dimensional FFT, its size is batch*len*sizeof (float) and M*N*sizeof (float) for two-dimensional FFT.

- bm_device_mem_t outputReal

  Output parameter. The device memory space storing the real number of the output result is batch*len*sizeof (float) for one-dimensional FFT and M*N*sizeof (float) for two-dimensional FFT.

- bm_device_mem_t outputImag

  Output parameter. The device memory space storing the imaginary number of the output result is batch*len*sizeof (float) for one-dimensional FFT and M*N*sizeof (float) for two-dimensional FFT.

- const void *plan

  Input parameter. The handle obtained during the creation phase.

Return value description:

- BM_SUCCESS: success
- Other: failed

5.49.3 Destruct

When the execution is completed, the created handle needs to be destructed.

```
void bmcv_fft_destroy_plan(bm_handle_t handle, void *plan);
```

5.49.4 Sample code:

```
bool realInput = false;
float *XRHost = new float[M * N];
float *XIHost = new float[M * N];
float *YRHost = new float[M * N];
float *YIHost = new float[M * N];
for (int i = 0; i < M * N; ++i) {
    XRHost[i] = rand() % 5 - 2;
    XIHost[i] = realInput ? 0 : rand() % 5 - 2;
}
bm_handle_t handle = nullptr;
bm_dev_request(&handle, 0);
bm_device_mem_t XRDev, XIDev, YRDev, YIDev;
bm_malloc_device_byte(handle, &XRDev, M * N * 4);
bm_malloc_device_byte(handle, &XIDev, M * N * 4);
bm_malloc_device_byte(handle, &YRDev, M * N * 4);
bm_malloc_device_byte(handle, &YIDev, M * N * 4);
bm_memcpy_s2d(handle, XRDev, XRHost);
bm_memcpy_s2d(handle, XIDev, XIHost);
void *plan = nullptr;
bmcv_fft_2d_create_plan(handle, M, N, forward, plan);
if (realInput)
    bmcv_fft_execute_real_input(handle, XRDev, YRDev, YIDev, plan);
else
    bmcv_fft_execute(handle, XRDev, XIDev, YRDev, YIDev, plan);
bmcv_fft_destroy_plan(handle, plan);
bm_memcpy_d2s(handle, YRHost, YRDev);
bm_memcpy_d2s(handle, YIHost, YIDev);
bm_free_device(handle, XRDev);
bm_free_device(handle, XIDev);
bm_free_device(handle, YRDev);
bm_free_device(handle, YIDev);
bm_dev_free(handle);
```

5.50 bmcv_calc_hist

5.50.1 Histogram

Interface form:

```
bm_status_t bmcv_calc_hist(
        bm_handle_t handle,
        bm_device_mem_t input,
        bm_device_mem_t output,
        int C,
        int H,
        int W,
        const int *channels,
```

(continues on next page)

(continued from previous page)

```
    int dims,
    const int *histSizes,
    const float *ranges,
    int inputDtype);
```

Processor model support

This interface supports BM1684/BM1684X.

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input

  Input parameter. The device memory space stores the input data. The type can be float32 or uint8, which is determined by the parameter inputDtype. Its size is C*H*W*sizeof (Dtype).

- bm_device_mem_t output

  Output parameter. The device memory space stores the output results. The type is float and its size is histSizes[0]*histSizes[1]*···*histSizes[n]*sizeof (float).

- int C

  Input parameter. Number of channels for input data.

- int H

  Input parameter. The height of each channel of the input data.

- int W

  Input parameter. The width of each channel of the input data.

- const int *channels

  Input parameter. The channel list of histogram needs to be calculated. Its length is dims, and the value of each element must be less than C.

- int dims

  Input parameter. The output histogram dimension, which shall not be greater than 3.

- const int *histSizes

  Input parameter. Corresponding to the number of copies of each channel statistical histogram. Its length is dims.

- const float *ranges

  Input parameter. The range of each channel participating in statistics, with a length of 2*dims.

- int inputDtype

  Input parameter. Type of input data: 0 means float, 1 means uint8.

Return value description:

- BM_SUCCESS: success
- Other: failed

Code example:

```
int H = 1024;
int W = 1024;
int C = 3;
int dim = 3;
int channels[3] = {0, 1, 2};
int histSizes[] = {15000, 32, 32};
float ranges[] = {0, 1000000, 0, 256, 0, 256};
int totalHists = 1;
for (int i = 0; i < dim; ++i)
    totalHists *= histSizes[i];
bm_handle_t handle = nullptr;
bm_status_t ret = bm_dev_request(&handle, 0);
float *inputHost = new float[C * H * W];
float *outputHost = new float[totalHists];
for (int i = 0; i < C; ++i)
    for (int j = 0; j < H * W; ++j)
        inputHost[i * H * W + j] = static_cast<float>(rand() % 1000000);
if (ret != BM_SUCCESS) {
    printf("bm_dev_request failed. ret = %d\n", ret);
    exit(-1);
}
bm_device_mem_t input, output;
ret = bm_malloc_device_byte(handle, &input, C * H * W * 4);
if (ret != BM_SUCCESS) {
    printf("bm_malloc_device_byte failed. ret = %d\n", ret);
    exit(-1);
}
ret = bm_memcpy_s2d(handle, input, inputHost);
if (ret != BM_SUCCESS) {
    printf("bm_memcpy_s2d failed. ret = %d\n", ret);
    exit(-1);
}
ret = bm_malloc_device_byte(handle, &output, totalHists * 4);
if (ret != BM_SUCCESS) {
    printf("bm_malloc_device_byte failed. ret = %d\n", ret);
    exit(-1);
}
ret = bmcv_calc_hist(handle,
                     input,
                     output,
                     C,
                     H,
```

(continues on next page)

(continued from previous page)

```
                    W,
                    channels,
                    dim,
                    histSizes,
                    ranges,
                    0);
if (ret != BM_SUCCESS) {
    printf("bmcv_calc_hist failed. ret = %d\n", ret);
    exit(-1);
}
ret = bm_memcpy_d2s(handle, outputHost, output);
if (ret != BM_SUCCESS) {
    printf("bm_memcpy_d2s failed. ret = %d\n", ret);
    exit(-1);
}
bm_free_device(handle, input);
bm_free_device(handle, output);
bm_dev_free(handle);
delete [] inputHost;
delete [] outputHost;
```

5.50.2 Weighted Histogram

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_calc_hist_with_weight(
        bm_handle_t handle,
        bm_device_mem_t input,
        bm_device_mem_t output,
        const float *weight,
        int C,
        int H,
        int W,
        const int *channels,
        int dims,
        const int *histSizes,
        const float *ranges,
        int inputDtype);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input

Input parameter. The device memory space stores the input data, and its size is C*H*W* sizeof (Dtype).

- bm_device_mem_t output

  Output parameter. The device memory space stores the output results. The type is float, and its size is histSizes[0]* histSizes[1]*···*histSizes[n]*sizeof (float).

- const float *weight

  Input parameter. The weight of each element in the channel during histogram statistics. Its size is H*W*sizeof (float). If all values are 1, it has the same function as the ordinary histogram.

- int C

  Input parameter. Number of channels for input data.

- int H

  Input parameter. The height of each channel of the input data

- int W

  Input parameter. The width of each channel of the input data.

- const int *channels

  Input parameter. The channel list of histogram needs to be calculated. Its length is dims, and the value of each element must be less than C.

- int dims

  Input parameter. The output histogram dimension shall not be greater than 3.

- const int *histSizes

  Input parameter. Corresponding to the number of copies of each channel statistical histogram. Its length is dims.

- const float *ranges

  Input parameter. The range of each channel participating in statistics, with a length of 2*dims.

- int inputDtype

  Input parameter. Type of input data: 0 means float, 1 means uint8.

Return value description:

- BM_SUCCESS: success
- Other: failed

5.51 bmcv_nms

The interface is used to eliminate excessive object frames obtained by network calculation and find the best object frame.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_nms(bm_handle_t handle,
                     bm_device_mem_t input_proposal_addr,
                     int proposal_size,
                     float nms_threshold,
                     bm_device_mem_t output_proposal_addr)
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input_proposal_addr

  Input parameter. Input the address where the object box data is located, and input the data structure as face_rect_t. Please refer to the following data structure description for details. Users need to call bm_mem_from_system() to convert the data address to structure corresponding to bm_device_mem_t.

- int proposal_size

  Input parameter. The number of object frames.

- float nms_threshold

  Input parameter. The threshold value of the filtered object frame. The object frame whose score is less than the threshold value will be filtered out.

- bm_device_mem_t output_proposal_addr

  Output parameter. The address where the output object frame data is located, and the output object frame data structure is nms_proposal_t. Please refer to the following data structure description for details. Users need to call bm_mem_from_system() to convert the data address to structure corresponding to bm_device_mem_t.

Return value:

- BM_SUCCESS: success
- Other: failed

Data type description:

face_rect_t describes the coordinate position of an object frame and the corresponding fraction.

```
typedef struct
{
   float x1;
   float y1;
   float x2;
   float y2;
   float score;
}face_rect_t;
```

- x1 describes the abscissa of the left edge of the object frame
- y1 describes the ordinate of the upper edge of the object frame
- x2 describes the abscissa of the right edge of the object frame
- y2 describes the ordinate of the lower edge of the object frame
- score describes the score corresponding to the object frame

nms_proposal_t describes the information of the output object box.

```
 typedef struct
 {
    face_rect_t face_rect[MAX_PROPOSAL_NUM];
    int size;
    int capacity;
    face_rect_t *begin;
    face_rect_t *end;
 } nms_proposal_t;

* face_rect describes the filtered object frame information

* size describes the number of object frames obtained after filtering

* capacity describes the maximum number of object frames after filtering

* begin is not used temporarily

* end is not used temporarily
```

Code example:

```
face_rect_t *proposal_rand = new face_rect_t[MAX_PROPOSAL_NUM];
nms_proposal_t *output_proposal = new nms_proposal_t;
int proposal_size =32;
float nms_threshold = 0.2;
for (int i = 0; i < proposal_size; i++)
{
   proposal_rand[i].x1 = 200;
   proposal_rand[i].x2 = 210 ;
   proposal_rand[i].y1 = 200;
   proposal_rand[i].y2 = 210;
   proposal_rand[i].score = 0.23;
```

(continues on next page)

(continued from previous page)

```
}
bmcv_nms(handle,
        bm_mem_from_system(proposal_rand),
        proposal_size,
        nms_threshold,
        bm_mem_from_system(output_proposal));
delete[] proposal_rand;
delete output_proposal;
```

Note:

The maximum number of proposal that can be entered by this API is 56000.

5.52 bmcv_nms_ext

This interface is the generalized form of bmcv_nms.It supports Hard_NMS/Soft_NMS/Adaptive_NMS/SSD_NMS which is used to eliminate excessive object frames obtained by network calculation and find the best object frame.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_nms_ext(bm_handle_t     handle,
              bm_device_mem_t input_proposal_addr,
              int            proposal_size,
              float          nms_threshold,
              bm_device_mem_t output_proposal_addr,
              int            topk,
              float          score_threshold,
              int            nms_alg,
              float          sigma,
              int            weighting_method,
              float *        densities,
              float          eta)
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input_proposal_addr

  Input parameter. Input the address where the object box data is located, and input the object box data structure as face_rect_t. Please refer to the following data structure description for details. Users need to call bm_mem_from_system() to convert the data address to structure corresponding to bm_device_mem_t.

- int proposal_size

  Input parameter. The number of object frames.

- float nms_threshold

  Input parameter. The threshold value of the filtered object frame. The object frame whose score is less than the threshold value will be filtered out.

- bm_device_mem_t output_proposal_addr

  Output parameter. The address where the output object frame data is located, and the output object frame data structure is nms_proposal_t. Please refer to the following data structure description for details. Users need to call bm_mem_from_system() to convert the data address to structure corresponding to bm_device_mem_t.

- int topk

  Input parameter. It is not currently in use and is reserved for possible subsequent extensions.

- float score_threshold

  Input parameter. The minimum score threshold when using Soft_NMS or Adaptive_NMS. When the score is lower than this value, the box corresponding to the score will be filtered out.

- int nms_alg

  Input parameter. Selection of different NMS algorithms, including Hard_NMS/Soft_NMS/Adaptive_NMS/SSD_NMS.

- float sigma

  Input parameter. When using Soft_NMS or Adaptive_NMS, the parameters of Gaussian re-score function.

- int weighting_method

  Input parameter. When using Soft_NMS or Adaptive_NMS, re-score function options include linear weight and Gaussian weight. Optional parameters:

```
typedef enum {
  LINEAR_WEIGHTING = 0,
  GAUSSIAN_WEIGHTING,
  MAX_WEIGHTING_TYPE
} weighting_method_e;
```

The linear weight expression is as follows:

$$s_i = \begin{cases} s_i, & iou(\mathcal{M}, b_i) < N_t \\ s_i \times (1 - iou(\mathcal{M}, b_i)), & iou(\mathcal{M}, b_i) \geq N_t \end{cases}$$

Gaussian weight expression is as follows:

$$s_i = s_i \times e^{-iou(\mathcal{M},\ b_i)^2/\sigma}$$

In the above two expressions, $\mathcal{M}$ represents the object frame with the largest current score, b_i represents the object frame with score lower than $\mathcal{M}$, s_i represents the score value of the object frame with score lower than $\mathcal{M}$ and N_t represents the NMS threshold, σ corresponds to the parameter float sigma of this interface.

- float* densities

  Input parameters. Adaptive-NMS density value.

- float eta

  Input parameter. SSD-NMS coefficient, used to adjust iou threshold.

Return value:

- BM_SUCCESS: success
- Other: failed

Code example:

```
#include <assert.h>
#include <stdint.h>
#include <stdio.h>
#include <algorithm>
#include <functional>
#include <iostream>
#include <memory>
#include <set>
#include <string>
#include <vector>
#include <math.h>
#include "bmcv_api.h"
#include "bmcv_internal.h"
#include "bmcv_common_bm1684.h"

#define MAX_PROPOSAL_NUM (65535)
typedef float bm_nms_data_type_t;

typedef struct {
    float x1;
    float y1;
    float x2;
    float y2;
    float score;
} face_rect_t;

typedef struct nms_proposal {
    int         size;
    face_rect_t  face_rect[MAX_PROPOSAL_NUM];
    int         capacity;
    face_rect_t *begin;
    face_rect_t *end;
} nms_proposal_t;
```

(continues on next page)

(continued from previous page)

```
typedef enum {
    LINEAR_WEIGHTING = 0,
    GAUSSIAN_WEIGHTING,
    MAX_WEIGHTING_TYPE
} weighting_method_e;

template <typename data_type>
static bool generate_random_buf(std::vector<data_type> &random_buffer,
                                int                    random_min,
                                int                    random_max,
                                int                    scale) {
    for (int i = 0; i < scale; i++) {
        data_type data_val = (data_type)(
            random_min + (((float)((random_max - random_min) * i)) / scale));
        random_buffer.push_back(data_val);
    }
    std::random_shuffle(random_buffer.begin(), random_buffer.end());

    return false;
}

 int main(int argc, char *argv[]) {
     unsigned int seed1 = 100;
    bm_nms_data_type_t  nms_threshold = 0.22;
    bm_nms_data_type_t  nms_score_threshold = 0.22;
    bm_nms_data_type_t  sigma               = 0.4;
    int  proposal_size      = 500;
    int  rand_loop_num      = 10;
    int  weighting_method   = GAUSSIAN_WEIGHTING;
    std::function<float(float, float)> weighting_func;
    int  nms_type = SOFT_NMS;  // ADAPTIVE NMS / HARD NMS / SOFT NMS
    const int soft_nms_total_types = MAX_NMS_TYPE - HARD_NMS - 1;

    for (int rand_loop_idx = 0;rand_loop_idx < (rand_loop_num * soft_nms_total_
↪types);rand_loop_idx++) {
        for (int rand_mode = 0; rand_mode < MAX_RAND_MODE; rand_
↪mode++) {
            std::shared_ptr<Blob<face_rect_t>> proposal_rand =
                std::make_shared<Blob<face_rect_t>>(MAX_PROPOSAL_NUM);
            std::shared_ptr<nms_proposal_t> output_proposal =
                std::make_shared<nms_proposal_t>();

            std::vector<face_rect_t>        proposals_ref;
            std::vector<face_rect_t>        nms_proposal;
            std::vector<bm_nms_data_type_t> score_random_buf;
            std::vector<bm_nms_data_type_t> density_vec;
            std::shared_ptr<Blob<float>>    densities =
                std::make_shared<Blob<float>>(proposal_size);
            generate_random_buf<bm_nms_data_type_t>(
                score_random_buf, 0, 1, 10000);
```

(continues on next page)

(continued from previous page)

```
        face_rect_t *proposal_rand_ptr = proposal_rand.get()->data;
        float eta = ((float)(rand() % 10)) / 10;
        for (int32_t i = 0; i < proposal_size; i++) {
            proposal_rand_ptr[i].x1 =
                ((bm_nms_data_type_t)(rand() % 100)) / 10;
            proposal_rand_ptr[i].x2 = proposal_rand_ptr[i].x1
                + ((bm_nms_data_type_t)(rand() % 100)) / 10;
            proposal_rand_ptr[i].y1 =
                ((bm_nms_data_type_t)(rand() % 100)) / 10;
            proposal_rand_ptr[i].y2 = proposal_rand_ptr[i].y1
                + ((bm_nms_data_type_t)(rand() % 100)) / 10;
            proposal_rand_ptr[i].score = score_random_buf[i];
            proposals_ref.push_back(proposal_rand_ptr[i]);
            densities.get()->data[i] = ((float)(rand() % 100)) / 100;
        }
        assert(proposal_size <= MAX_PROPOSAL_NUM);
        if (weighting_method == LINEAR_WEIGHTING) {
            weighting_func = linear_weighting;
        } else if (weighting_method == GAUSSIAN_WEIGHTING) {
            weighting_func = gaussian_weighting;
        } else {
            std::cout << "weighting_method error: " << weighting_method
                      << std::endl;
        }
        bmcv_nms_ext(handle,
                     bm_mem_from_system(proposal_rand.get()->data),
                     proposal_size,
                     nms_threshold,
                     bm_mem_from_system(output_proposal.get()),
                     1,
                     nms_score_threshold,
                     nms_type,
                     sigma,
                     weighting_method,
                     densities.get()->data,
                     eta);
    }
  }

  return 0;
}
```

Note:

The maximum number of proposal that can be entered by this API is 1024.

5.53 bmcv_nms_yolo

This interface supports yolov3/yolov7, which is used to eliminate too many object boxes obtained by network calculation and find the best object box.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_nms_yolo(
            bm_handle_t handle,
            int     input_num,
            bm_device_mem_t bottom[3],
            int     batch_num,
            int     hw_shape[3][2],
            int     num_classes,
            int     num_boxes,
            int     mask_group_size,
            float nms_threshold,
            float confidence_threshold,
            int     keep_top_k,
            float bias[18],
            float anchor_scale[3],
            float mask[9],
            bm_device_mem_t output,
            int     yolo_flag,
            int     len_per_batch,
            void *ext)
```

Parameter Description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- int input_num

  Input parameter. Input the number of feature maps.

- bm_device_mem_t bottom[3]

  Input parameter.The bottom device address needs to be called bm_mem_from_system() to convert the data address into the structure corresponding to bm_device_mem_t.

- int batch_num

  Input parameter.The number of batches.

- int hw_shape[3][2]

  Input parameter. Input the h, w of the feature map.

- int num_classes

  Input parameter. Number of categories of images.

- int num_boxes

  Input parameter. How many anchor boxes of different scales each grid contains.

- int mask_group_size

  Input parameter. size of mask.

- float nms_threshold

  Input parameter. Threshold for filtering object boxes, object boxes with scores less than this threshold will be filtered out.

- int confidence_threshold

  Input parameter. Confidence.

- int keep_top_k

  Input parameter. Save the first k numbers.

- int bias[18]

  Input parameter. Bias.

- float anchor_scale[3]

  Input parameter. The size of anchor.

- float mask[9]

  Input parameter.Mask.

- bm_device_mem_t output

  Input parameter. For the output device address, you need to call bm_mem_from_system() to convert the data address into the structure corresponding to bm_device_mem_t.

- int yolo_flag

  Input parameter. yolo_flag=0 when yolov3, yolo_flag=2 when yolov7.

- int len_per_batch

  Input parameter. This parameter is not valid and is only intended to maintain interface compatibility.

- int scale

  Input parameter. Target size. This parameter only takes effect in yolov7.

- int *orig_image_shape

  Input parameter. The w/h of the original image is arranged in batches, such as batch4: w1 h1 w2 h2 w3 h3 w4 h4. This parameter only takes effect in yolov7.

- int model_h

  Input parameter.The shape h of the model, this parameter only takes effect in yolov7.

- int model_w

  Input parameter. The shape w of the model, this parameter only takes effect in yolov7.

- void *ext

  Reserved parameters. If you need to add new parameters, you can add them here. Four new parameters have been added to yolov7 as:

```
typedef struct yolov7_info{
    int scale;
    int *orig_image_shape;
    int model_h;
    int model_w;
} yolov7_info_t;
```

In the above structure, int scale: scale_flag. int* orig_image_shape: w/h of the original image, sorted by batch cloth, such as batch4: w1 h1 w2 h2 w3 h3 w4 h4. int model_h: The shape h of the model. int model_w: The shape w of the model. These parameters only take effect in yolov7.

Return value:

- BM_SUCCESS: success
- Other: failed

Code example::

```
#include <time.h>
#include <random>
#include <algorithm>
#include <map>
#include <vector>
#include <iostream>
#include <cmath>
#include <getopt.h>
#include "bmcv_api_ext.h"
#include "bmcv_common_bm1684.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <iostream>
#include <new>
#include <fstream>

#define KEEP_TOP_K    200
#define Dtype float
#define TIME_PROFILE
```

(continues on next page)

(continued from previous page)

```
typedef struct yolov7_info{
    int scale;
    int *orig_image_shape;
    int model_h;
    int model_w;
} yolov7_info_t;

 int main(int argc, char *argv[]) {
    int DEV_ID = 0;
    int H = 16, W = 30;
    int bottom_num = 3;
    int dev_count;
    int f_data_from_file = 0;
    int f_tpu_forward = 1;

    bm_status_t ret = BM_SUCCESS;
    int batch_num = 32;
    int num_classes = 6;
    int num_boxes = 3;
    int yolo_flag = 0; //yolov3: 0, yolov7: 2
    int len_per_batch = 0;
    int keep_top_k = 100;
    float nms_threshold = 0.1;
    float conf_threshold = 0.98f;
    int mask_group_size = 3;
    float bias[18] = {10, 13, 16, 30, 33, 23, 30, 61, 62, 45, 59, 119, 116, 90, 156, 198, 373,
↪ 326};
    float anchor_scale[3] = {32, 16, 8};
    float mask[9] = {6, 7, 8, 3, 4, 5, 0, 1, 2};
    int scale = 0; //for yolov7 post handle
    int model_h = 0;
    int model_w = 0;
    int mode_value_end = 0;
    bm_dev_request(&handle, 0);
    int hw_shape[3][2] = {
        {H*1, W*1},
        {H*2, W*2},
        {H*4, W*4},
    };

    int size_bottom[3];
    float* data_bottom[3];
    int origin_image_shape[batch_num * 2] = {0};
    if (yolo_flag == 1){
        num_boxes = 1;
        len_per_batch = 12096 * 18;
        bottom_num = 1;
    } else if (yolo_flag == 2){
        //yolov7 post handle;
        num_boxes = 1;
```

(continues on next page)

(continued from previous page)

```
    bottom_num = 3;
    mask_group_size = 1;
    scale = 1;
    model_h = 512;
    model_w = 960;
    for (int i = 0 ; i < 3; i++){
    mask[i] = i;
    }

    for (int i = 0; i < 6; i++)
    bias[i] = 1;

    for (int i = 0; i < 3; i++)
    anchor_scale[i] = 1;

    for (int i = 0; i < batch_num; i++){
    origin_image_shape[i*2 + 0] = 1920;
    origin_image_shape[i*2 + 1] = 1080;
    }
}

// alloc input data
for (int i = 0; i < 3; ++i) {
    if (yolo_flag == 1){
    size_bottom[i] = batch_num * len_per_batch;
    } else {
    size_bottom[i] = batch_num * num_boxes *
                (num_classes + 5) * hw_shape[i][0] * hw_shape[i][1];
    }
    try {
    data_bottom[i] = new float[size_bottom[i]];
    }
    catch(std::bad_alloc &memExp)
    {
    std::cerr<<memExp.what()<<std::endl;
    exit(-1);
    }
}

if (f_data_from_file) {
    #if defined(__aarch64__)
    #define DIR     "./imgs/"
    #else
    #define DIR     "test/test_api_bmdnn/bm1684/imgs/"
    #endif
    printf("reading data from: \"" DIR "\"\n");
    char path[256];
    if (yolo_flag == 1) {
    FILE* fp = fopen("./output_ref_data.dat.bmrt", "rb");
    size_t cnt = fread(data_bottom[0],
          sizeof(float), size_bottom[0]*batch_num, fp);
```

(continues on next page)

(continued from previous page)

```
    cnt = cnt;
    fclose(fp);
    } else {
    for (int i = 0; i < batch_num; ++i) {
        sprintf(path, DIR "b%d_13.bin", i);
        FILE* fp = fopen(path, "rb");
        size_t cnt = fread(data_bottom[0] + i * size_bottom[0] / batch_num,
            sizeof(float), size_bottom[0] / batch_num, fp);
        cnt = cnt;
        fclose(fp);

        sprintf(path, DIR "b%d_26.bin", i);
        fp = fopen(path, "rb");
        cnt = fread(data_bottom[1] + i * size_bottom[1] / batch_num,
            sizeof(float), size_bottom[1] / batch_num, fp);
        cnt = cnt;
        fclose(fp);

        sprintf(path, DIR "b%d_52.bin", i);
        fp = fopen(path, "rb");
        cnt = fread(data_bottom[2] + i * size_bottom[2] / batch_num,
            sizeof(float), size_bottom[2] / batch_num, fp);
        cnt = cnt;
        fclose(fp);
    }
    }
} else {
    ofstream file_1("1.txt", std::ios::out);
    ofstream file_2("2.txt", std::ios::out);
    ofstream file_3("3.txt", std::ios::out);

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dist(0, 1);

    // alloc and init input data
    for (int j = 0; j < size_bottom[0]; ++j){
    if (yolo_flag == 2){
        data_bottom[0][j] = dist(gen);
    } else {
        data_bottom[0][j] = (rand() % 1000 - 999.0f) / (124.0f);
    }
    file_1 << data_bottom[0][j] <<endl;
    }

    for (int j = 0; j < size_bottom[1]; ++j){
    if (yolo_flag == 2){
        data_bottom[1][j] = dist(gen);
    } else {
        data_bottom[1][j] = (rand() % 1000 - 999.0f) / (124.0f);
    }
```

(continues on next page)

(continued from previous page)

```
        file_2 << data_bottom[1][j] <<endl;
        }

        for (int j = 0; j < size_bottom[2]; ++j){
        if (yolo_flag == 2){
            data_bottom[2][j] = dist(gen);
        } else {
            data_bottom[2][j] = (rand() % 1000 - 999.0f) / (124.0f);
        }
        file_3 << data_bottom[2][j] <<endl;
        }
    }

    // alloc output data
    float* output_bmdnn;
    float* output_native;
    try {
        output_bmdnn = new float[output_size];
        output_native = new float[output_size];
    }
    catch(std::bad_alloc &memExp)
    {
        std::cerr<<memExp.what()<<std::endl;
        exit(-1);
    }
    memset(output_bmdnn, 0, output_size * sizeof(float));
    memset(output_native, 0, output_size * sizeof(float));

    bm_dev_request(&handle, 0);
    bm_device_mem_t bottom[3] = {
        bm_mem_from_system((void*)data_bottom[0]),
        bm_mem_from_system((void*)data_bottom[1]),
        bm_mem_from_system((void*)data_bottom[2])
    };
    yolov7_info_t *ext = (yolov7_info_t*) malloc (sizeof(yolov7_info_t));
    ext->scale = scale;
    ext->orig_image_shape = origin_image_shape;
    ext->model_h = model_h;
    ext->model_w = model_w;

    ret = bmcv_nms_yolo(
    handle, bottom_num, bottom,
    batch_num, hw_shape, num_classes, num_boxes,
    mask_group_size, nms_threshold, conf_threshold,
    keep_top_k, bias, anchor_scale, mask,
    bm_mem_from_system((void*)output_bmdnn), yolo_flag, len_per_batch,F
↪(void*)ext);

    return 0;
}
```

5.54 bmcv_cmulp

This interface is used to implement the complex number multiplication, as shown in the following formula:

$$\text{outputReal} + \text{outputImag} \times i = (\text{inputReal} + \text{inputImag} \times i) \times (\text{pointReal} + \text{pointImag} \times i)$$

$$\text{outputReal} = \text{inputReal} \times \text{pointReal} - \text{inputImag} \times \text{pointImag}$$

$$\text{outputImag} = \text{inputReal} \times \text{pointImag} + \text{inputImag} \times \text{pointReal}$$

Among that, i is the imaginary unit and satisfying the equation $i^2 = -1$.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_cmulp(
      bm_handle_t     handle,
      bm_device_mem_t inputReal,
      bm_device_mem_t inputImag,
      bm_device_mem_t pointReal,
      bm_device_mem_t pointImag,
      bm_device_mem_t outputReal,
      bm_device_mem_t outputImag,
      int             batch,
      int             len);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t inputReal

  Input parameter. Device addr information of the real part of the input.

- bm_device_mem_t inputImag

  Input parameter. Device addr information of the imaginary part of the input.

- bm_device_mem_t pointReal

  Input parameter. Device addr information of the real part of the point.

- bm_device_mem_t pointImag

  Input parameter. Device addr information of the imaginary part of the point.

- bm_device_mem_t outputReal

  Output parameter. Device addr information of the real part of the output.

- bm_device_mem_t outputImag

  Output parameter. Device addr information of the imaginary part of the output.

- int batch

  Input parameter. The number of batches.

- int len

  Input parameter. The number of the complex numbers in a batch.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. Data type: only support float.

Sample code

```
int L = 5;
int batch = 2;
float *XRHost = new float[L * batch];
float *XIHost = new float[L * batch];
float *PRHost = new float[L];
float *PIHost = new float[L];
for (int i = 0; i < L * batch; ++i) {
    XRHost[i] = rand() % 5 - 2;
    XIHost[i] = rand() % 5 - 2;
}
for (int i = 0; i < L; ++i) {
    PRHost[i] = rand() % 5 - 2;
    PIHost[i] = rand() % 5 - 2;
}
float *YRHost = new float[L * batch];
float *YIHost = new float[L * batch];
bm_handle_t handle = nullptr;
bm_dev_request(&handle, 0);
bm_device_mem_t XRDev, XIDev, PRDev, PIDev, YRDev, YIDev;
bm_malloc_device_byte(handle, &XRDev, L * batch * 4);
bm_malloc_device_byte(handle, &XIDev, L * batch * 4);
bm_malloc_device_byte(handle, &PRDev, L * 4);
bm_malloc_device_byte(handle, &PIDev, L * 4);
bm_malloc_device_byte(handle, &YRDev, L * batch * 4);
bm_malloc_device_byte(handle, &YIDev, L * batch * 4);
bm_memcpy_s2d(handle, XRDev, XRHost);
bm_memcpy_s2d(handle, XIDev, XIHost);
bm_memcpy_s2d(handle, PRDev, PRHost);
bm_memcpy_s2d(handle, PIDev, PIHost);

bmcv_cmulp(handle,
```

(continues on next page)

(continued from previous page)

```
        XRDev,
        XIDev,
        PRDev,
        PIDev,
        YRDev,
        YIDev,
        batch,
        L);
bm_memcpy_d2s(handle, YRHost, YRDev);
bm_memcpy_d2s(handle, YIHost, YIDev);

delete[] XRHost;
delete[] XIHost;
delete[] PRHost;
delete[] PIHost;
delete[] YRHost;
delete[] YIHost;
bm_free_device(handle, XRDev);
bm_free_device(handle, XIDev);
bm_free_device(handle, YRDev);
bm_free_device(handle, YIDev);
bm_free_device(handle, PRDev);
bm_free_device(handle, PIDev);
bm_dev_free(handle);
```

5.55 bmcv_faiss_indexflatIP

This interface is used to calculate inner product distance between query vectors and database vectors, output the top K (sort_cnt) IP-values and the corresponding indices, return BM_SUCCESS if succeed.

Processor model support

This interface only supports BM1684X.

Interface form:

```
bm_status_t bmcv_faiss_indexflatIP(
      bm_handle_t    handle,
      bm_device_mem_t input_data_global_addr,
      bm_device_mem_t db_data_global_addr,
      bm_device_mem_t buffer_global_addr,
      bm_device_mem_t output_sorted_similarity_global_addr,
      bm_device_mem_t output_sorted_index_global_addr,
      int             vec_dims,
      int             query_vecs_num,
      int             database_vecs_num,
      int             sort_cnt,
      int             is_transpose,
```

(continues on next page)

(continued from previous page)

```
    int          input_dtype,
    int          output_dtype);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input_data_global_addr

  Input parameter. Device addr information of the query matrix.

- bm_device_mem_t db_data_global_addr

  Input parameter. Device addr information of the database matrix.

- bm_device_mem_t buffer_global_addr

  Input parameter. Inner product values stored in the buffer.

- bm_device_mem_t output_sorted_similarity_global_addr

  Output parameter. The IP-values matrix.

- bm_device_mem_t output_sorted_index_global_addr

  Output parameter. The result indices matrix.

- int vec_dims

  Input parameter. Vector dimension.

- int query_vecs_num

  Input parameter. The num of query vectors.

- int database_vecs_num

  Input parameter. The num of database vectors.

- int sort_cnt

  Input parameter. Get top sort_cnt values.

- int is_transpose

  Input parameter. db_matrix 0: NO_TRNAS; 1: TRANS.

- int input_dtype

  Input parameter. Support float and char, 5 means float, 1 means char.

- int output_dtype

  Output parameter. Support float and int, 5 means float, 9 means int.

Return value description:

- BM_SUCCESS: success

- Other: failed

Note:

1. The input data type (query vectors) and data in the database (database vectors) are float or char.
2. The data type of the output sorted similarity result is float or int, and that of the corresponding indices is int.
3. Usually, the data in the database is arranged in the memory as database_vecs_num * vec_dims. Therefore, the is_transpose needs to be set to 1.
4. The larger the inner product values of the query vector and the database vector, the higher the similarity of the two vectors. Therefore, the inner product values are sorted in descending order in the process of TopK.
5. The interface is used for Faiss::IndexFlatIP.search() and implemented on BM1684X. According to the continuous memory of Tensor Computing Processor on BM1684X, we can query about 512 inputs of 256 dimensions at a time on a single processor if the database is about 100W.

Sample code

```
int sort_cnt = 100;
int vec_dims = 256;
int query_vecs_num = 1;
int database_vecs_num = 2000000;
int is_transpose = 1;
int input_dtype = 5; // 5: float
int output_dtype = 5;

float *input_data = new float[query_vecs_num * vec_dims];
float *db_data = new float[database_vecs_num * vec_dims];

void matrix_gen_data(float* data, u32 len) {
    for (u32 i = 0; i < len; i++) {
        data[i] = ((float)rand() / (float)RAND_MAX) * 3.3;
    }
}

matrix_gen_data(input_data, query_vecs_num * vec_dims);
matrix_gen_data(db_data, vec_dims * database_vecs_num);

bm_handle_t handle = nullptr;
bm_dev_request(&handle, 0);
bm_device_mem_t query_data_dev_mem;
bm_device_mem_t db_data_dev_mem;
bm_malloc_device_byte(handle, &query_data_dev_mem,
        query_vecs_num * vec_dims * sizeof(float));
bm_malloc_device_byte(handle, &db_data_dev_mem,
        database_vecs_num * vec_dims * sizeof(float));
bm_memcpy_s2d(handle, query_data_dev_mem, input_data);
```

(continues on next page)

(continued from previous page)

```
bm_memcpy_s2d(handle, db_data_dev_mem, db_data);

float *output_dis = new float[query_vecs_num * sort_cnt];
int *output_inx = new int[query_vecs_num * sort_cnt];
bm_device_mem_t buffer_dev_mem;
bm_device_mem_t sorted_similarity_dev_mem;
bm_device_mem_t sorted_index_dev_mem;
bm_malloc_device_byte(handle, &buffer_dev_mem,
      query_vecs_num * database_vecs_num * sizeof(float));
bm_malloc_device_byte(handle, &sorted_similarity_dev_mem,
      query_vecs_num * sort_cnt * sizeof(float));
bm_malloc_device_byte(handle, &sorted_index_dev_mem,
      query_vecs_num * sort_cnt * sizeof(int));

bmcv_faiss_indexflatIP(handle,
                    query_data_dev_mem,
                    db_data_dev_mem,
                    buffer_dev_mem,
                    sorted_similarity_dev_mem,
                    sorted_index_dev_mem,
                    vec_dims,
                    query_vecs_num,
                    database_vecs_num,
                    sort_cnt,
                    is_transpose,
                    input_dtype,
                    output_dtype);
bm_memcpy_d2s(handle, output_dis, sorted_similarity_dev_mem);
bm_memcpy_d2s(handle, output_inx, sorted_index_dev_mem);
delete[] input_data;
delete[] db_data;
delete[] output_similarity;
delete[] output_index;
bm_free_device(handle, query_data_dev_mem);
bm_free_device(handle, db_data_dev_mem);
bm_free_device(handle, buffer_dev_mem);
bm_free_device(handle, sorted_similarity_dev_mem);
bm_free_device(handle, sorted_index_dev_mem);
bm_dev_free(handle);
```

5.56 bmcv_faiss_indexflatL2

This interface is used to calculate squared L2 distance between query vectors and database vectors, output the top K (sort_cnt) L2sqr-values and the corresponding indices, return BM_SUCCESS if succeed.

Processor model support

This interface only supports BM1684X.

Interface form:

```
bm_status_t bmcv_faiss_indexflatL2(
      bm_handle_t     handle,
      bm_device_mem_t input_data_global_addr,
      bm_device_mem_t db_data_global_addr,
      bm_device_mem_t query_L2norm_global_addr,
      bm_device_mem_t db_L2norm_global_addr,
      bm_device_mem_t buffer_global_addr,
      bm_device_mem_t output_sorted_similarity_global_addr,
      bm_device_mem_t output_sorted_index_global_addr,
      int             vec_dims,
      int             query_vecs_num,
      int             database_vecs_num,
      int             sort_cnt,
      int             is_transpose,
      int             input_dtype,
      int             output_dtype);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t input_data_global_addr

  Input parameter. Device addr information of the query matrix.

- bm_device_mem_t db_data_global_addr

  Input parameter. Device addr information of the database matrix.

- bm_device_mem_t query_L2norm_global_addr

  Input parameter. Device addr information of the query norm_L2sqr vector.

- bm_device_mem_t db_L2norm_global_addr

  Input parameter. Device addr information of the database norm_L2sqr vector.

- bm_device_mem_t buffer_global_addr

  Input parameter. Squared L2 values stored in the buffer.

- bm_device_mem_t output_sorted_similarity_global_addr

  Output parameter. The L2sqr-values matrix.

- bm_device_mem_t output_sorted_index_global_addr

  Output parameter. The result indices matrix.

- int vec_dims

  Input parameter. Vector dimension.

- int query_vecs_num

Input parameter. The num of query vectors.

- int database_vecs_num

  Input parameter. The num of database vectors.

- int sort_cnt

  Input parameter. Get top sort_cnt values.

- int is_transpose

  Input parameter. db_matrix 0: NO_TRNAS; 1: TRANS.

- int input_dtype

  Input parameter. Only support float, 5 means float.

- int output_dtype

  Output parameter. Only support float, 5 means float.

Return value description:

- BM_SUCCESS: success
- Other: failed

Note:

1. The input data type (query vectors) and data in the database (database vectors) are float.
2. The data type of the output sorted similarity result is float, and that of the corresponding indices is int.
3. The assumption is that the norm_L2sqr values of the input data and the database data have been computed ahead of time and stored on the processor.
4. Usually, the data in the database is arranged in the memory as database_vecs_num * vec_dims. Therefore, the is_transpose needs to be set to 1.
5. The smaller the squared L2 values of the query vector and the database vector, the higher the similarity of the two vectors. Therefore, the squared L2 values are sorted in ascending order in the process of TopK.
6. The interface is used for Faiss::IndexFlatL2.search() and implemented on BM1684X. According to the continuous memory of Tensor Computing Processor on BM1684X, we can query about 512 inputs of 256 dimensions at a time on a single processor if the database is about 100W.
7. the value of database_vecs_num and sort_cnt needs to meet the condition: database_vecs_num > sort_cnt.

Sample code

```
int sort_cnt = 100;
int vec_dims = 256;
int query_vecs_num = 1;
int database_vecs_num = 2000000;
int is_transpose = 1;
int input_dtype = 5; // 5: float
int output_dtype = 5;

float *input_data = new float[query_vecs_num * vec_dims];
float *db_data = new float[database_vecs_num * vec_dims];
float *vec_query = new float[1 * query_vecs_num];
float *vec_db = new float[1 * database_vecs_num];

void matrix_gen_data(float* data, u32 len) {
    for (u32 i = 0; i < len; i++) {
        data[i] = ((float)rand() / (float)RAND_MAX) * 3.3;
    }
}

void fvec_norm_L2sqr_ref(float* vec, float* matrix, int row_num, int col_num) {
for (int i = 0; i < row_num; i++)
    for (int j = 0; j < col_num; j++)
        vec[i] += matrix[i * col_num + j] * matrix[i * col_num + j];
}

matrix_gen_data(input_data, query_vecs_num * vec_dims);
matrix_gen_data(db_data, vec_dims * database_vecs_num);
fvec_norm_L2sqr_ref(vec_query, input_data, query_vecs_num, vec_dims);
fvec_norm_L2sqr_ref(vec_db, db_data, database_vecs_num, vec_dims);

bm_handle_t handle = nullptr;
bm_dev_request(&handle, 0);
bm_device_mem_t query_data_dev_mem;
bm_device_mem_t db_data_dev_mem;
bm_device_mem_t query_L2norm_dev_mem;
bm_device_mem_t db_L2norm_dev_mem;
bm_malloc_device_byte(handle, &query_data_dev_mem,
        query_vecs_num * vec_dims * sizeof(float));
bm_malloc_device_byte(handle, &db_data_dev_mem,
        database_vecs_num * vec_dims * sizeof(float));
bm_malloc_device_byte(handle, &query_L2norm_dev_mem,
        1 * query_vecs_num * sizeof(float));
bm_malloc_device_byte(handle, &db_L2norm_dev_mem,
        1 * database_vecs_num * sizeof(float));

bm_memcpy_s2d(handle, query_data_dev_mem, input_data);
bm_memcpy_s2d(handle, db_data_dev_mem, db_data);
bm_memcpy_s2d(handle, query_L2norm_dev_mem, vec_query);
bm_memcpy_s2d(handle, db_L2norm_dev_mem, vec_db);

float *output_dis = new float[query_vecs_num * sort_cnt];
```

(continues on next page)

(continued from previous page)

```
int *output_inx = new int[query_vecs_num * sort_cnt];
bm_device_mem_t buffer_dev_mem;
bm_device_mem_t sorted_similarity_dev_mem;
bm_device_mem_t sorted_index_dev_mem;
bm_malloc_device_byte(handle, &buffer_dev_mem,
       query_vecs_num * database_vecs_num * sizeof(float));
bm_malloc_device_byte(handle, &sorted_similarity_dev_mem,
       query_vecs_num * sort_cnt * sizeof(float));
bm_malloc_device_byte(handle, &sorted_index_dev_mem,
       query_vecs_num * sort_cnt * sizeof(int));

bmcv_faiss_indexflatL2(handle,
                       query_data_dev_mem,
                       db_data_dev_mem,
                       query_L2norm_dev_mem,
                       db_L2norm_dev_mem,
                       buffer_dev_mem,
                       sorted_similarity_dev_mem,
                       sorted_index_dev_mem,
                       vec_dims,
                       query_vecs_num,
                       database_vecs_num,
                       sort_cnt,
                       is_transpose,
                       input_dtype,
                       output_dtype);
bm_memcpy_d2s(handle, output_dis, sorted_similarity_dev_mem);
bm_memcpy_d2s(handle, output_inx, sorted_index_dev_mem);
delete[] input_data;
delete[] db_data;
delete[] vec_query;
delete[] vec_db;
delete[] output_similarity;
delete[] output_index;
bm_free_device(handle, query_data_dev_mem);
bm_free_device(handle, db_data_dev_mem);
bm_free_device(handle, query_L2norm_dev_mem);
bm_free_device(handle, db_L2norm_dev_mem);
bm_free_device(handle, buffer_dev_mem);
bm_free_device(handle, sorted_similarity_dev_mem);
bm_free_device(handle, sorted_index_dev_mem);
bm_dev_free(handle);
```

5.57 bmcv_batch_topk

Compute the largest or smallest k number in each db, and return the index.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_batch_topk(
        bm_handle_t     handle,
        bm_device_mem_t src_data_addr,
        bm_device_mem_t src_index_addr,
        bm_device_mem_t dst_data_addr,
        bm_device_mem_t dst_index_addr,
        bm_device_mem_t buffer_addr,
        bool            src_index_valid,
        int             k,
        int             batch,
        int *           per_batch_cnt,
        bool            same_batch_cnt,
        int             src_batch_stride,
        bool            descending);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

- bm_device_mem_t src_data_addr

  Input parameters. The device address information of input_data.

- bm_device_mem_t src_index_addr

  Input parameters. The device address information of input_index, when src_index_valid is true, set this parameter.

- bm_device_mem_t dst_data_addr

  Output parameters. The output_data device address information.

- bm_device_mem_t dst_index_addr

  Output parameters. The output_index device information

- bm_device_mem_t buffer_addr

  Input parameters. The buffer device address information.

- bool src_index_valid

  Input parameters. If true, use src_index, otherwise use auto-generated index.

- int k

  Input parameters. The value of k.

- int batch

  Input parameters. The number of batches.

- int * per_batch_cnt

  Input parameters. The amount of data in each batch.

- bool same_batch_cnt

  Input parameters. Determine whether each batch data is the same.

- int src_batch_stride

  Input parameters. The distance between two batches.

- bool descending

  Input parameters. Ascending or descending order.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

This interface currently only supports float32 type data.

Code example:

```
int batch_num = 100000;
int k = batch_num / 10;
int descending = rand() % 2;
int batch = rand() % 20 + 1;
int batch_stride = batch_num;
bool bottom_index_valid = true;

bm_handle_t handle;
bm_status_t ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
    std::cout << "Create bm handle failed. ret = " << ret << std::endl;
    exit(-1);
}

float* bottom_data = new float[batch * batch_stride * sizeof(float)];
int* bottom_index = new int[batch * batch_stride];
float* top_data = new float[batch * batch_stride * sizeof(float)];
int* top_index = new int[batch * batch_stride];
float* top_data_ref = new float[batch * k * sizeof(float)];
int* top_index_ref = new int[batch * k];
float* buffer = new float[3 * batch_stride * sizeof(float)];
```

(continues on next page)

(continued from previous page)

```
for(int i = 0; i < batch; i++){
    for(int j = 0; j < batch_num; j++){
        bottom_data[i * batch_stride + j] = rand() % 10000 * 1.0f;
        bottom_index[i * batch_stride + j] = i * batch_stride + j;
    }
}

bm_status_t ret = bmcv_batch_topk( handle,
                                   bm_mem_from_system((void*)bottom_data),
                                   bm_mem_from_system((void*)bottom_index),
                                   bm_mem_from_system((void*)top_data),
                                   bm_mem_from_system((void*)top_index),
                                   bm_mem_from_system((void*)buffer),
                                   bottom_index_valid,
                                   k,
                                   batch,
                                   &batch_num,
                                   true,
                                   batch_stride,
                                   descending);

if(ret == BM_SUCCESS){
    int data_cmp = -1;
    int index_cmp = -1;
    data_cmp = array_cmp( (float*)top_data_ref,
                          (float*)top_data,
                          batch * k,
                          "topk data",
                          0);
    index_cmp = array_cmp( (float*)top_index_ref,
                           (float*)top_index,
                           batch * k,
                           "topk index",
                           0);
    if (data_cmp == 0 && index_cmp == 0) {
        printf("Compare success for topk data and index!\n");
    } else {
        printf("Compare failed for topk data and index!\n");
        exit(-1);
    }
} else {
    printf("Compare failed for topk data and index!\n");
    exit(-1);
}
delete [] bottom_data;
delete [] bottom_index;
delete [] top_data;
delete [] top_data_ref;
delete [] top_index;
delete [] top_index_ref;
```

(continues on next page)

(continued from previous page)

```
bm_dev_free(handle);
```

5.58 bmcv_hm_distance

Calculates the Hamming distance of each element in two vectors.

Processor model support

This interface only supports BM1684X.

Interface form:

```
bmcv_hamming_distance(
  bm_handle_t handle,
  bm_device_mem_t input1,
  bm_device_mem_t input2,
  bm_device_mem_t output,
  int bits_len,
  int input1_num,
  int input2_num);
```

Description of parameters:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- bm_image input1

  Input parameters. Device address information for vector 1 data.

- bm_image input2

  Input parameters. Device address information for vector 2 data.

- bm_image output

  Output parameters. Device address information for output vector data.

- int bits_len

  Input parameters. The length of each element in the vector.

- int input1_num

  Input parameters. The num of vector 1 data.

- int input2_num

  Input parameters. The num of vector 2 data.

Return value description:

- BM_SUCCESS: success

- Other: failed

Code example:

```
int bits_len = 8;
int input1_num = 2;
int input2_num = 2562;

int* input1_data = new int[input1_num * bits_len];
int* input2_data = new int[input2_num * bits_len];
int* output_ref  = new int[input1_num * input2_num];
int* output_tpu  = new int[input1_num * input2_num];

memset(input1_data, 0, input1_num * bits_len * sizeof(int));
memset(input2_data, 0, input2_num * bits_len * sizeof(int));
memset(output_ref,  0,  input1_num * input2_num * sizeof(int));
memset(output_tpu,  0,  input1_num * input2_num * sizeof(int));

// fill data
for(int i = 0; i < input1_num * bits_len; i++) input1_data[i] = rand() % 10;
for(int i = 0; i < input2_num * bits_len; i++) input2_data[i] = rand() % 20 + 1;

bm_device_mem_t input1_dev_mem;
bm_device_mem_t input2_dev_mem;
bm_device_mem_t output_dev_mem;

if(BM_SUCCESS != bm_malloc_device_byte(handle, &input1_dev_mem, input1_
↪num * bits_len * sizeof(int))){
    std::cout << "malloc input fail" << std::endl;
    exit(-1);
}

if(BM_SUCCESS != bm_malloc_device_byte(handle, &input2_dev_mem, input2_
↪num * bits_len * sizeof(int))){
    std::cout << "malloc input fail" << std::endl;
    exit(-1);
}

if(BM_SUCCESS != bm_malloc_device_byte(handle, &output_dev_mem, input1_
↪num * input2_num * sizeof(int))){
    std::cout << "malloc input fail" << std::endl;
    exit(-1);
}

if(BM_SUCCESS != bm_memcpy_s2d(handle, input1_dev_mem, input1_data)){
    std::cout << "copy input1 to device fail" << std::endl;
    exit(-1);
}

if(BM_SUCCESS != bm_memcpy_s2d(handle, input2_dev_mem, input2_data)){
    std::cout << "copy input2 to device fail" << std::endl;
    exit(-1);
```

(continues on next page)

(continued from previous page)

```
}

struct timeval t1, t2;
gettimeofday(&t1, NULL);
bm_status_t status = bmcv_hamming_distance(handle,
                                    input1_dev_mem,
                                    input2_dev_mem,
                                    output_dev_mem,
                                    bits_len,
                                    input1_num,
                                    input2_num);
gettimeofday(&t2, NULL);
cout << "--using time = " << ((t2.tv_sec - t1.tv_sec) * 1000000 + t2.tv_usec - t1.
↪tv_usec) << "(us)--" << endl;

if(status != BM_SUCCESS){
    printf("run bmcv_hamming_distance failed status = %d \n", status);
    bm_free_device(handle, input1_dev_mem);
    bm_free_device(handle, input2_dev_mem);
    bm_free_device(handle, output_dev_mem);
    bm_dev_free(handle);
    exit(-1);
}

if(BM_SUCCESS != bm_memcpy_d2s(handle, output_tpu, output_dev_mem)){
        std::cout << "bm_memcpy_d2s fail" << std::endl;
        exit(-1);
}

delete [] input1_data;
delete [] input2_data;
delete [] output_ref;
delete [] output_tpu;
bm_free_device(handle, input1_dev_mem);
bm_free_device(handle, input2_dev_mem);
bm_free_device(handle, output_dev_mem);
```

5.59 bmcv_axpy

This interface implements F = A * X + Y, where A is a constant of size n * c , and F , X , Y are all matrices of size n * c * h * w.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:


```
bm_status_t bmcv_image_axpy(
      bm_handle_t handle,
      bm_device_mem_t tensor_A,
      bm_device_mem_t tensor_X,
      bm_device_mem_t tensor_Y,
      bm_device_mem_t tensor_F,
      int input_n,
      int input_c,
      int input_h,
      int input_w);
```

Description of parameters:

- bm_handle_t handle

  Input parameters. bm_handle handle.

- bm_device_mem_t tensor_A

  Input parameters. The device memory address where the scalar A is stored.

- bm_device_mem_t tensor_X

  Input parameters. The device memory address where matrix X is stored.

- bm_device_mem_t tensor_Y

  Input parameters. The device memory address where matrix Y is stored.

- bm_device_mem_t tensor_F

  Output parameters. The device memory address where the result matrix F is stored.

- int input_n

  Input parameters. The size of n dimension.

- int input_c

  Input parameters. The size of c dimension.

- int input_h

  Input parameters. The size of h dimension.

- int input_w

  Input parameters. The size of w dimension.

Return value description:

- BM_SUCCESS: success
- Other: failed

Code example:


```
#define N (10)
#define C 256 //(64 * 2 + (64 >> 1))
#define H 8
#define W 8
#define TENSOR_SIZE (N * C * H * W)

bm_handle_t handle;
bm_status_t ret = BM_SUCCESS;

bm_dev_request(&handle, 0);
int trials = 0;
if (argc == 1) {
    trials = 5;
}else if(argc == 2){
    trials  = atoi(argv[1]);
}else{
    std::cout << "command input error, please follow this "
            "order:test_cv_axpy loop_num "
         << std::endl;
    return -1;
}

float* tensor_X = new float[TENSOR_SIZE];
float* tensor_A = new float[N*C];
float* tensor_Y = new float[TENSOR_SIZE];
float* tensor_F = new float[TENSOR_SIZE];

for (int idx_trial = 0; idx_trial < trials; idx_trial++) {

    for (int idx = 0; idx < TENSOR_SIZE; idx++) {
        tensor_X[idx] = (float)idx - 5.0f;
        tensor_Y[idx] = (float)idx/3.0f - 8.2f;  //y
    }

    for (int idx = 0; idx < N*C; idx++) {
    tensor_A[idx] = (float)idx * 1.5f + 1.0f;
    }

    struct timeval t1, t2;
    gettimeofday_(&t1);
    ret = bmcv_image_axpy(handle,
                        bm_mem_from_system((void *)tensor_A),
                        bm_mem_from_system((void *)tensor_X),
                        bm_mem_from_system((void *)tensor_Y),
                        bm_mem_from_system((void *)tensor_F),
                        N, C, H, W);
    gettimeofday_(&t2);
    std::cout << "The "<< idx_trial <<" loop "<< " axpy using time: " << ((t2.tv_
↪sec - t1.tv_sec) * 1000000 + t2.tv_usec - t1.tv_usec)  << "us" << std::endl;
}
delete []tensor_A;
```

(continues on next page)

(continued from previous page)

```
delete []tensor_X;
delete []tensor_Y;
delete []tensor_F;
delete []tensor_F_cmp;
bm_dev_free(handle);
```

5.60 bmcv_image_pyramid_down

This interface implements downsampling in image gaussian pyramid operations.

Processor model support

This interface supports BM1684/BM1684X.

Interface form:

```
bm_status_t bmcv_image_pyramid_down(
        bm_handle_t handle,
        bm_image input,
        bm_image output);
```

Description of parameters:

- bm_handle_t handle

  Input parameters. bm_handle handle.

- bm_image input

  Input parameter. The bm_image of the input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. The bm_image of the output image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following image_format and data_type:

num	image_format	data_type
1	FORMAT_GRAY	DATA_TYPE_EXT_1N_BYTE

Code example:

```
int height = 1080;
int width = 1920;
int ow = height / 2;
int oh = width / 2;
int channel = 1;
unsigned char* input_data = new unsigned char [width * height * channel];
unsigned char* output_tpu = new unsigned char [ow * oh * channel];
unsigned char* output_ocv = new unsigned char [ow * oh * channel];

for (int i = 0; i < height * channel; i++) {
    for (int j = 0; j < width; j++) {
        input_data[i * width + j] = rand() % 100;
    }
}

bm_handle_t handle;
bm_status_t ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
    printf("Create bm handle failed. ret = %d\n", ret);
    return -1;
}
bm_image_format_ext fmt = FORMAT_GRAY;
bm_image img_i;
bm_image img_o;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
↪img_i);
bm_image_create(handle, oh, ow, fmt, DATA_TYPE_EXT_1N_BYTE, &img_o);
bm_image_alloc_dev_mem(img_i);
bm_image_alloc_dev_mem(img_o);
bm_image_copy_host_to_device(img_i, (void **)(&input));

struct timeval t1, t2;
gettimeofday_(&t1);
bmcv_image_pyramid_down(handle, img_i, img_o);
gettimeofday_(&t2);
cout << "pyramid down Tensor Computing Processor using time: " << ((t2.tv_sec -F
↪t1.tv_sec) * 1000000 + t2.tv_usec - t1.tv_usec) << "us" << endl;

bm_image_copy_device_to_host(img_o, (void **)(&output));
bm_image_destroy(img_i);
bm_image_destroy(img_o);
bm_dev_free(handle);
```

5.61 bmcv_image_bayer2rgb

Converts bayerBG8 format images to RGB Plannar format.

Processor model support

This interface only supports BM1684X.

Interface form:

```
bm_status_t bmcv_image_bayer2rgb(
      bm_handle_t handle,
      unsigned char* convd_kernel,
      bm_image input
      bm_image output);
```

Parameter Description:

- bm_handle_t handle

  Input parameter. Handle of bm_handle.

- unsigned char* convd_kernel

  Input parameter. Convolutional kernel for convolutional computation.

- bm_image input

  Input parameter. The bm_image of the input image. The creation of bm_image requires an external call to bmcv_image_create. The image memory can use bm_image_alloc_dev_mem or bm_image_copy_host_to_device to create new memory, or use bmcv_image_attach to attach existing memory.

- bm_image output

  Output parameter. Output bm_image. The creation of bm_image requires an external call to bmcv_image_create. Image memory can use bm_image_alloc_dev_mem to create new memory, or use bmcv_image_attach to attach existing memory. If users do not actively allocate, it will be allocated automatically within the API.

Return value description:

- BM_SUCCESS: success
- Other: failed

Format support:

The interface currently supports the following input format:

num	image_format
1	FORMAT_BAYER

The interface currently supports the following output format:

num	image_format
1	FORMAT_RGB_PLANAR

The interface currently supports the following data_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

Note

1、The format of input is bayerBG, the format of output is rgb plannar, and the data_type is uint8.

2、The interface supports the size range of 8*8 ~ 8096*8096, and the width and height of the image need to be even.

Code example:

```
#define KERNEL_SIZE 3 * 3 * 3 * 4 * 64
#define CONVD_MATRIX 12 * 9

const unsigned char convd_kernel[CONVD_MATRIX] = {1, 0, 1, 0, 0, 0, 1, 0, 1,
                                    0, 0, 2, 0, 0, 0, 0, 0, 2,
                                    0, 0, 0, 0, 0, 0, 2, 0, 2,
                                    0, 0, 0, 0, 0, 0, 0, 0, 4, // r R
                                    4, 0, 0, 0, 0, 0, 0, 0, 0, // b B
                                    2, 0, 2, 0, 0, 0, 0, 0, 0,
                                    2, 0, 0, 0, 0, 0, 2, 0, 0,
                                    1, 0, 1, 0, 0, 0, 1, 0, 1,
                                    0, 1, 0, 1, 0, 1, 0, 1, 0,
                                    0, 0, 0, 0, 0, 4, 0, 0, 0, // g1 G1
                                    0, 0, 0, 0, 0, 0, 0, 4, 0, // g2 G2
                                    0, 1, 0, 1, 0, 1, 0, 1, 0};
int width     = 1920;
int height    = 1080;
int dev_id    = 0;
bm_handle_t handle;
bm_status_t dev_ret = bm_dev_request(&handle, dev_id);
std::shared_ptr<unsigned char> src1_ptr(
       new unsigned char[channel * width * height],
       std::default_delete<unsigned char[]>());
bm_image input_img;
bm_image output_img;
bm_image_create(handle, height, width, FORMAT_BAYER, DATA_TYPE_EXT_
↪1N_BYTE, &input_img);
bm_image_create(handle, height, width, FORMAT_RGB_PLANAR, DATA_
↪TYPE_EXT_1N_BYTE, &output_img);
bm_image_alloc_dev_mem(output_img, BMCV_HEAP_ANY);
```

(continues on next page)

(continued from previous page)

```
unsigned char kernel_data[KERNEL_SIZE];
memset(kernel_data, 0, KERNEL_SIZE);
// constructing convd_kernel_data
for (int i = 0;i < 12;i++) {
    for (int j = 0;j < 9;j++) {
        kernel_data[i * 9 * 64 + 64 * j] = convd_kernel[i * 9 + j];
    }
}
unsigned char* input_data[3] = {srcImage.data, srcImage.data + height * width,F
↪srcImage.data + 2 * height * width};
bm_image_copy_host_to_device(input_img, (void **)input_data);
bmcv_image_bayer2rgb(handle, kernel_data, input_img, output_img);
bm_image_copy_device_to_host(output_img, (void **)(&output));
bm_image_destroy(input_img);
bm_image_destroy(output_img);
bm_dev_free(handle);
```

5.62 bmcv_as_strided

This interface can create a view matrix based on the existing matrix and the given step size.

Processor model support

This interface only supports BM1684X.

Interface form:

```
bm_status_t bmcv_as_strided(bm_handle_t     handle,
                    bm_device_mem_t input,
                    bm_device_mem_t output,
                    int             input_row,
                    int             input_col,
                    int             output_row,
                    int             output_col,
                    int             row_stride,
                    int             col_stride);
```

Input parameter description:

- bm_handle_t handle

  Input parameter. bm_handle Handle.

- bm_device_mem_t input

  Input parameter. The device memory address where the input matrix data is stored.

- bm_device_mem_t output

Input parameter. The device memory address where the output matrix data is stored.

- int input_row

  Input parameter. The number of rows of the input matrix.

- int input_col

  Input parameter. The number of columns of the input matrix.

- int output_row

  Input parameter. The number of rows of the output matrix.

- int output_col

  Input parameter. The number of columns of the output matrix.

- int row_stride

  Input parameter. The step size between the rows of the output matrix.

- int col_stride

  Input parameter. The step size between the columns of the output matrix.

Return value description:

- BM_SUCCESS: success
- Other: failed

Example Code

```
#define RAND_MAX 2147483647

int loop = 1;
int input_row = 5;
int input_col = 5;
int output_row = 3;
int output_col = 3;
int row_stride = 1;
int col_stride = 2;

bm_handle_t handle;
bm_status_t ret = BM_SUCCESS;
ret = bm_dev_request(&handle, 0);
if (BM_SUCCESS != ret){
    printf("request dev failed\n");
    return BM_ERR_FAILURE;
}

float* input_data = new float[input_row * input_col];
float* output_data = new float[output_row * output_col];

srand((unsigned int)time(NULL));
```

(continues on next page)

(continued from previous page)

```
for (int i = 0; i < len; i++) {
    input_data[i] = (float)rand() / (float)RAND_MAX * 100;
}

bm_device_mem_t input_dev_mem, output_dev_mem;
bm_malloc_device_byte(handle, &input_dev_mem, input_row * input_col *F
↪sizeof(float));
bm_malloc_device_byte(handle, &output_dev_mem, output_row * output_col *F
↪sizeof(float));

bm_memcpy_s2d(handle, input_dev_mem, input_data);

struct timeval t1, t2;
gettimeofday_(&t1);
ret = bmcv_as_strided(handle,
                      input_dev_mem,
                      output_dev_mem,
                      input_row, input_col,
                      output_row, output_col,
                      row_stride, col_stride);
gettimeofday_(&t2);
std::cout << "as_strided Tensor Computing Processor using time= " << ((t2.tv_
↪sec - t1.tv_sec) * 1000000 + t2.tv_usec - t1.tv_usec) << "(us)" << std::endl;
if (ret != BM_SUCCESS) {
printf("as_strided failed. ret = %d\n", ret);
goto exit;
}

bm_memcpy_d2s(handle, output_data, output_dev_mem);

exit:
    bm_free_device(handle, input_dev_mem);
    bm_free_device(handle, output_dev_mem);
    delete[] output_data;
    delete[] input_data;
    bm_dev_free(handle);
```

CHAPTER 6

PCIe CPU

6.1 PCIe CPU

For operations that are inconvenient to use Tensor Computing Processor acceleration, the cooperation of Processor is required.

If it is SoC mode, the host side is the on-chip ARM A53 processor, which completes the Processor operation.

In case of PCIe mode, the host side is the user' s host, and the Processor operation can be completed at the host side or by using the on-chip ARM A53 processor. The two implementation methods have their own advantages and disadvantages: the former needs to carry input and output data between device and host, but the operation performance may be better than ARM, so users can choose the better method according to their own host processor performance, load and other actual conditions. It is the former by default. If you need to use an on-chip processor, you can turn it on in the following way.

6.1.1 Preparatory Work

If you want to enable the on-chip processor, you need the following two files:

- ramboot_rootfs.itb
- fip.bin

You need to set the path where these two files are located to the environment variable BMCV_CPU_KERNEL_PATH where the program runs, as follows:

$ export BMCV_CPU_KERNEL_PATH=/path/to/kernel_fils/

All implementations of BMCV that require Processor operations are in the library libbmcv_cpu_func.so, you need to add the path of the file to the environment variable BMCV_CPU_KERNEL_PATH where the program runs, as follows:

$ export BMCV_CPU_LIB_PATH=/path/to/lib/

At present, the APIs that require Processor participation are as follows. If the following APIs are not used, this function can be ignored.

num	API
1	bmcv_image_draw_lines
2	bmcv_image_erode
3	bmcv_image_dilate
4	bmcv_image_lkpyramid_execute
5	bmcv_image_morph

6.1.2 Opening and Closing

Users can use the following two interfaces at the beginning and end of the program to turn on and off the function respectively.

```
bm_status_t bmcv_open_cpu_process(bm_handle_t handle);

bm_status_t bmcv_close_cpu_process(bm_handle_t handle);
```

Input parameters description:

- bm_handle_t handle

  Input parameter. The handle of bm_handle.

Return value description:

- BM_SUCCESS: success
- Other: failed

