Reading and Writing Images and Video

imdecode

Reads an image from a buffer in memory.

C++: Mat **imdecode**(InputArray buf, int flags)

C++: Mat **imdecode**(InputArray buf, int flags, Mat* dst)

C: IplImage* **cvDecodeImage**(const CvMat* buf, int iscolor=CV_LOAD_IMAGE_COLOR)

C: CvMat* **cvDecodeImageM**(const CvMat* buf, int iscolor=CV_LOAD_IMAGE_COLOR)

Python: `cv2.imdecode`(buf, flags) → retval

Parameters:
- buf – Input array or vector of bytes.
- flags – The same flags as in **imread()** .
- dst – The optional output placeholder for the decoded matrix. It can save the image reallocations when the function is called repeatedly for images of the same size.

The function reads an image from the specified buffer in the memory. If the buffer is too short or contains invalid data, the empty matrix/image is returned.

See **imread()** for the list of supported formats and flags description.

Note: In the case of color images, the decoded images will have the channels stored in `B G R` order.

imencode

Encodes an image into a memory buffer.

C++: bool **imencode**(const string& ext, InputArray img, vector<uchar>& buf, const vector<int>& params=vector<int>())

C: CvMat* **cvEncodeImage**(const char* ext, const CvArr* image, const int* params=0)

Python: `cv2.imencode`(ext, img[, params]) → retval, buf

Parameters:
- ext – File extension that defines the output format.
- img – Image to be written.
- buf – Output buffer resized to fit the compressed image.
- params – Format-specific parameters. See **imwrite()** .

The function compresses the image and stores it in the memory buffer that is resized to fit the result. See **imwrite()** for the list of supported formats and flags description.

Note: cvEncodeImage returns single-row matrix of type CV_8UC1 that contains encoded image as array of bytes.

imread

Loads an image from a file.

C++: Mat **imread**(const string& filename, int flags=1)

Python: cv2.**imread**(filename[, flags]) → retval

C: IplImage* **cvLoadImage**(const char* filename, int iscolor=CV_LOAD_IMAGE_COLOR)

C: CvMat* **cvLoadImageM**(const char* filename, int iscolor=CV_LOAD_IMAGE_COLOR)

Python: cv.**LoadImage**(filename, iscolor=CV_LOAD_IMAGE_COLOR) → None

Python: cv.**LoadImageM**(filename, iscolor=CV_LOAD_IMAGE_COLOR) → None

Parameters:
- filename – Name of file to be loaded.
- flags –

  Flags specifying the color type of a loaded image:

  - CV_LOAD_IMAGE_ANYDEPTH - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.
  - CV_LOAD_IMAGE_COLOR - If set, always convert image to the color one
  - CV_LOAD_IMAGE_GRAYSCALE - If set, always convert image to the grayscale one
  - >0 Return a 3-channel color image.

    Note: In the current implementation the alpha channel, if any, is stripped from the output image. Use negative

value if you need the alpha channel.

- =0 Return a grayscale image.
- <0 Return the loaded image as is (with alpha channel).

The function `imread` loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (`Mat::data==NULL`). Currently, the following file formats are supported:

- Windows bitmaps - `*.bmp, *.dib` (always supported)
- JPEG files - `*.jpeg, *.jpg, *.jpe` (see the *Notes* section)
- JPEG 2000 files - `*.jp2` (see the *Notes* section)
- Portable Network Graphics - `*.png` (see the *Notes* section)
- Portable image format - `*.pbm, *.pgm, *.ppm` (always supported)
- Sun rasters - `*.sr, *.ras` (always supported)
- TIFF files - `*.tiff, *.tif` (see the *Notes* section)

Note:

- The function determines the type of an image by the content, not by the file extension.
- On Microsoft Windows* OS and MacOSX*, the codecs shipped with an OpenCV image (libjpeg, libpng, libtiff, and libjasper) are used by default. So, OpenCV can always read JPEGs, PNGs, and TIFFs. On MacOSX, there is also an option to use native MacOSX image readers. But beware that currently these native image loaders give images with different pixel values because of the color management embedded into MacOSX.
- On Linux*, BSD flavors and other Unix-like open-source operating systems, OpenCV looks for codecs supplied with an OS image. Install the relevant packages (do not forget the development files, for example, "libjpeg-dev" , in Debian* and Ubuntu*) to get the codec support or turn on the `OPENCV_BUILD_3RDPARTY_LIBS` flag in CMake.

Note: In the case of color images, the decoded images will have the channels stored in `B G R` order.

imwrite

Saves an image to a specified file.

C++: bool **`imwrite`**(const string& filename, InputArray img, const vector<int>& params=vector<int>())

Python: `cv2.imwrite(filename, img[, params]) → retval`

C: int **cvSaveImage**(const char* filename, const CvArr* image, const int* params=0)

Python: `cv.SaveImage(filename, image) → None`

Parameters:
- filename – Name of the file.
- image – Image to be saved.
- params –

  Format-specific save parameters encoded as pairs `paramId_1, paramValue_1, paramId_2, paramValue_2, ...` . The following parameters are currently supported:

  - For JPEG, it can be a quality (`CV_IMWRITE_JPEG_QUALITY`) from 0 to 100 (the higher is the better). Default value is 95.
  - For PNG, it can be the compression level (`CV_IMWRITE_PNG_COMPRESSION`) from 0 to 9. A higher value means a smaller size and longer compression time. Default value is 3.
  - For PPM, PGM, or PBM, it can be a binary format flag (`CV_IMWRITE_PXM_BINARY`), 0 or 1. Default value is 1.

The function `imwrite` saves the image to the specified file. The image format is chosen based on the `filename` extension (see **imread()** for the list of extensions). Only 8-bit (or 16-bit unsigned (`CV_16U`) in case of PNG, JPEG 2000, and TIFF) single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function. If the format, depth or channel order is different, use **Mat::convertTo()** , and **cvtColor()** to convert it before saving. Or, use the universal XML I/O functions to save the image to XML or YAML format.

It is possible to store PNG images with an alpha channel using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535. The sample below shows how to create such a BGRA image and store to PNG file. It also demonstrates how to set custom compression parameters

```
#include <vector>
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

void createAlphaMat(Mat &mat)
{
```

```
    for (int i = 0; i < mat.rows; ++i) {
        for (int j = 0; j < mat.cols; ++j) {
            Vec4b& rgba = mat.at<Vec4b>(i, j);
            rgba[0] = UCHAR_MAX;
            rgba[1] = saturate_cast<uchar>((float (mat.cols - j)) / ((float)mat.cols
            rgba[2] = saturate_cast<uchar>((float (mat.rows - i)) / ((float)mat.rows
            rgba[3] = saturate_cast<uchar>(0.5 * (rgba[1] + rgba[2]));
        }
    }
}

int main(int argv, char **argc)
{
    // Create mat with alpha channel
    Mat mat(480, 640, CV_8UC4);
    createAlphaMat(mat);

    vector<int> compression_params;
    compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9);

    try {
        imwrite("alpha.png", mat, compression_params);
    }
    catch (runtime_error& ex) {
        fprintf(stderr, "Exception converting image to PNG format: %s\n", ex.what()
        return 1;
    }

    fprintf(stdout, "Saved PNG file with alpha data.\n");
    return 0;
}
```

VideoCapture

class **VideoCapture**

Class for video capturing from video files or cameras. The class provides C++ API for capturing video from cameras or for reading video files. Here is how the class can be used:

```
#include "opencv2/opencv.hpp"

using namespace cv;

int main(int, char**)
{
    VideoCapture cap(0); // open the default camera
    if(!cap.isOpened())  // check if we succeeded
        return -1;

    Mat edges;
    namedWindow("edges",1);
    for(;;)
    {
        Mat frame;
        cap >> frame; // get a new frame from camera
```

```
        cvtColor(frame, edges, CV_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
        Canny(edges, edges, 0, 30, 3);
        imshow("edges", edges);
        if(waitKey(30) >= 0) break;
    }
    // the camera will be deinitialized automatically in VideoCapture destructor
    return 0;
}
```

Note: In C API the black-box structure `CvCapture` is used instead of `VideoCapture`.

Note:

- A basic sample on using the VideoCapture interface can be found at opencv_source_code/samples/cpp/starter_video.cpp
- Another basic video processing sample can be found at opencv_source_code/samples/cpp/video_dmtx.cpp
- (Python) A basic sample on using the VideoCapture interface can be found at opencv_source_code/samples/python2/video.py
- (Python) Another basic video processing sample can be found at opencv_source_code/samples/python2/video_dmtx.py
- (Python) A multi threaded video processing sample can be found at opencv_source_code/samples/python2/video_threaded.py

VideoCapture::VideoCapture

VideoCapture constructors.

C++: `VideoCapture::VideoCapture()`

C++: `VideoCapture::VideoCapture(const string& filename)`

C++: `VideoCapture::VideoCapture(int device)`

Python: `cv2.VideoCapture()` → <VideoCapture object>

Python: `cv2.VideoCapture(filename)` → <VideoCapture object>

Python: `cv2.VideoCapture(device)` → <VideoCapture object>

C: CvCapture* **cvCaptureFromCAM**(int device)

Python: `cv.CaptureFromCAM(index)` → CvCapture

C: CvCapture* **cvCaptureFromFile**(const char* filename)

Python: `cv.CaptureFromFile`(filename) → CvCapture

Parameters:
- filename – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)
- device – id of the opened video capturing device (i.e. a camera index). If there is a single camera connected, just pass 0.

Note: In C API, when you finished working with video, release `CvCapture` structure with `cvReleaseCapture()`, or use `Ptr<CvCapture>` that calls `cvReleaseCapture()` automatically in the destructor.

VideoCapture::open

Open video file or a capturing device for video capturing

C++: bool `VideoCapture::open`(const string& filename)

C++: bool `VideoCapture::open`(int device)

Python: `cv2.VideoCapture.open`(filename) → retval

Python: `cv2.VideoCapture.open`(device) → retval

Parameters:
- filename – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)
- device – id of the opened video capturing device (i.e. a camera index).

The methods first call `VideoCapture::release()` to close the already opened file or camera.

VideoCapture::isOpened

Returns true if video capturing has been initialized already.

C++: bool `VideoCapture::isOpened`()

Python: `cv2.VideoCapture.isOpened`() → retval

If the previous call to `VideoCapture` constructor or `VideoCapture::open` succeeded, the method returns true.

VideoCapture::release

Closes video file or capturing device.

C++: void `VideoCapture::`**`release`**()

Python: `cv2.VideoCapture.`**`release`**() → None

C: void **`cvReleaseCapture`**(CvCapture** capture)

The methods are automatically called by subsequent **`VideoCapture::open()`** and by `VideoCapture` destructor.

The C function also deallocates memory and clears `*capture` pointer.

VideoCapture::grab

Grabs the next frame from video file or capturing device.

C++: bool `VideoCapture::`**`grab`**()

Python: `cv2.VideoCapture.`**`grab`**() → retval

C: int **`cvGrabFrame`**(CvCapture* capture)

Python: `cv.`**`GrabFrame`**(capture) → int

The methods/functions grab the next frame from video file or camera and return true (non-zero) in the case of success.

The primary use of the function is in multi-camera environments, especially when the cameras do not have hardware synchronization. That is, you call `VideoCapture::grab()` for each camera and after that call the slower method `VideoCapture::retrieve()` to decode and get frame from each camera. This way the overhead on demosaicing or motion jpeg decompression etc. is eliminated and the retrieved frames from different cameras will be closer in time.

Also, when a connected camera is multi-head (for example, a stereo camera or a Kinect device), the correct way of retrieving data from it is to call *VideoCapture::grab* first and then call **`VideoCapture::retrieve()`** one or more times with different values of the `channel` parameter. See https://github.com/Itseez/opencv/tree/master/samples/cpp/openni_capture.cpp

VideoCapture::retrieve

Decodes and returns the grabbed video frame.

C++: bool `VideoCapture::` **`retrieve`**(Mat& image, int channel=0)

Python: `cv2.VideoCapture.` **`retrieve`**([image[, channel]]) → retval, image

C: IplImage* **`cvRetrieveFrame`**(CvCapture* capture, int streamIdx=0)

Python: `cv.` **`RetrieveFrame`**(capture) → image

The methods/functions decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

> Note: OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using **`cvCloneImage()`** and then do whatever you want with the copy.

VideoCapture::read

Grabs, decodes and returns the next video frame.

C++: VideoCapture& `VideoCapture::` **`operator>>`**(Mat& image)

C++: bool `VideoCapture::` **`read`**(Mat& image)

Python: `cv2.VideoCapture.` **`read`**([image]) → retval, image

C: IplImage* **`cvQueryFrame`**(CvCapture* capture)

Python: `cv.` **`QueryFrame`**(capture) → image

The methods/functions combine **`VideoCapture::grab()`** and **`VideoCapture::retrieve()`** in one call. This is the most convenient method for reading video files or capturing data from decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

> Note: OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using **`cvCloneImage()`** and then do whatever you want with the copy.

VideoCapture::get

Returns the specified `VideoCapture` property

C++: double `VideoCapture::`**`get`**(int propId)

Python: `cv2.VideoCapture.`**`get`**(propId) → retval

C: double **`cvGetCaptureProperty`**(CvCapture* capture, int property_id)

Python: `cv.`**`GetCaptureProperty`**(capture, property_id) → float

Parameters: propId –

Property identifier. It can be one of the following:

- CV_CAP_PROP_POS_MSEC Current position of the video file in milliseconds or video capture timestamp.
- CV_CAP_PROP_POS_FRAMES 0-based index of the frame to be decoded/captured next.
- CV_CAP_PROP_POS_AVI_RATIO Relative position of the video file: 0 - start of the film, 1 - end of the film.
- CV_CAP_PROP_FRAME_WIDTH Width of the frames in the video stream.
- CV_CAP_PROP_FRAME_HEIGHT Height of the frames in the video stream.
- CV_CAP_PROP_FPS Frame rate.
- CV_CAP_PROP_FOURCC 4-character code of codec.
- CV_CAP_PROP_FRAME_COUNT Number of frames in the video file.
- CV_CAP_PROP_FORMAT Format of the Mat objects returned by `retrieve()` .
- CV_CAP_PROP_MODE Backend-specific value indicating the current capture mode.
- CV_CAP_PROP_BRIGHTNESS Brightness of the image (only for cameras).
- CV_CAP_PROP_CONTRAST Contrast of the image (only for cameras).
- CV_CAP_PROP_SATURATION Saturation of the image (only for cameras).
- CV_CAP_PROP_HUE Hue of the image (only for cameras).
- CV_CAP_PROP_GAIN Gain of the image (only for cameras).
- CV_CAP_PROP_EXPOSURE Exposure (only for cameras).
- CV_CAP_PROP_CONVERT_RGB Boolean flags indicating whether images should be converted to RGB.
- CV_CAP_PROP_WHITE_BALANCE Currently not supported

- CV_CAP_PROP_RECTIFICATION Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

Note: When querying a property that is not supported by the backend used by the `VideoCapture` class, value 0 is returned.

VideoCapture::set

Sets a property in the `VideoCapture`.

C++: bool `VideoCapture::` **`set`**(int propId, double value)

Python: `cv2.VideoCapture.` **`set`**(propId, value) → retval

C: int **`cvSetCaptureProperty`**(CvCapture* capture, int property_id, double value)

Python: `cv.` **`SetCaptureProperty`**(capture, property_id, value) → retval

Parameters:

- propId –

  Property identifier. It can be one of the following:

  - CV_CAP_PROP_POS_MSEC Current position of the video file in milliseconds.
  - CV_CAP_PROP_POS_FRAMES 0-based index of the frame to be decoded/captured next.
  - CV_CAP_PROP_POS_AVI_RATIO Relative position of the video file: 0 - start of the film, 1 - end of the film.
  - CV_CAP_PROP_FRAME_WIDTH Width of the frames in the video stream.
  - CV_CAP_PROP_FRAME_HEIGHT Height of the frames in the video stream.
  - CV_CAP_PROP_FPS Frame rate.
  - CV_CAP_PROP_FOURCC 4-character code of codec.
  - CV_CAP_PROP_FRAME_COUNT Number of frames in the video file.
  - CV_CAP_PROP_FORMAT Format of the Mat objects returned by `retrieve()` .
  - CV_CAP_PROP_MODE Backend-specific value indicating the current capture mode.
  - CV_CAP_PROP_BRIGHTNESS Brightness of the image (only for cameras).
  - CV_CAP_PROP_CONTRAST Contrast of the image (only for

cameras).
    - CV_CAP_PROP_SATURATION Saturation of the image (only for cameras).
    - CV_CAP_PROP_HUE Hue of the image (only for cameras).
    - CV_CAP_PROP_GAIN Gain of the image (only for cameras).
    - CV_CAP_PROP_EXPOSURE Exposure (only for cameras).
    - CV_CAP_PROP_CONVERT_RGB Boolean flags indicating whether images should be converted to RGB.
    - CV_CAP_PROP_WHITE_BALANCE Currently unsupported
    - CV_CAP_PROP_RECTIFICATION Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)
- value – Value of the property.

VideoWriter

class **VideoWriter**

Video writer class.

VideoWriter::VideoWriter

VideoWriter constructors

C++: `VideoWriter::`**VideoWriter**()

C++: `VideoWriter::`**VideoWriter**(const string& filename, int fourcc, double fps, Size frameSize, bool isColor=true)

Python: `cv2.`**VideoWriter**([filename, fourcc, fps, frameSize[, isColor]]) → <VideoWriter object>

C: CvVideoWriter* **cvCreateVideoWriter**(const char* filename, int fourcc, double fps, CvSize frame_size, int is_color=1)

Python: `cv.`**CreateVideoWriter**(filename, fourcc, fps, frame_size, is_color=true) → CvVideoWriter

Python: `cv2.VideoWriter.`**isOpened**() → retval

Python: `cv2.VideoWriter.`**open**(filename, fourcc, fps, frameSize[, isColor]) → retval

Python: `cv2.VideoWriter.`**write**(image) → None

Parameters:
- filename – Name of the output video file.

- fourcc – 4-character code of codec used to compress the frames. For example, `CV_FOURCC('P','I','M','1')` is a MPEG-1 codec, `CV_FOURCC('M','J','P','G')` is a motion-jpeg codec etc. List of codes can be obtained at Video Codecs by FOURCC page.
- fps – Framerate of the created video stream.
- frameSize – Size of the video frames.
- isColor – If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

The constructors/functions initialize video writers. On Linux FFMPEG is used to write videos; on Windows FFMPEG or VFW is used; on MacOSX QTKit is used.

ReleaseVideoWriter

Releases the AVI writer.

C: void **cvReleaseVideoWriter**(CvVideoWriter** writer)

The function should be called after you finished using `CvVideoWriter` opened with **`CreateVideoWriter()`**.

VideoWriter::open

Initializes or reinitializes video writer.

C++: bool `VideoWriter::` **open**(const string& filename, int fourcc, double fps, Size frameSize, bool isColor=true)

Python: `cv2.VideoWriter.` **open**(filename, fourcc, fps, frameSize[, isColor]) → retval

The method opens video writer. Parameters are the same as in the constructor **`VideoWriter::VideoWriter()`**.

VideoWriter::isOpened

Returns true if video writer has been successfully initialized.

C++: bool `VideoWriter::` **isOpened**()

Python: `cv2.VideoWriter.` **isOpened**() → retval

VideoWriter::write

Writes the next video frame

C++: VideoWriter& `VideoWriter::`**`operator<<`**(const Mat& image)

C++: void `VideoWriter::`**`write`**(const Mat& image)

Python: `cv2.VideoWriter.`**`write`**(image) → None

C: int **`cvWriteFrame`**(CvVideoWriter* writer, const IplImage* image)

Python: `cv.`**`WriteFrame`**(writer, image) → int

Parameters:
- writer – Video writer structure (OpenCV 1.x API)
- image – The written frame

The functions/methods write the specified image to video file. It must have the same size as has been specified when opening the video writer.

Help and Feedback

You did not find what you were looking for?

Ask a question on the Q&A forum.
If you think something is missing or wrong in the documentation, please file a bug report.