Reading and Writing Images

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=IMREAD_COLOR)

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)

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)
- WebP - `*.webp` (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)

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 WEBP, it can be a quality (CV_IMWRITE_WEBP_QUALITY) from 1 to 100 (the higher is the better). By default (without any parameter) and for quality above 100 the lossless compression is used.
  - 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 FileStorage 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) * UCHAR_MAX);
```

```
            rgba[2] = saturate_cast<uchar>((float (mat.rows - i)) / ((float)mat.rows) * UCHAR_MAX);
            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;
}
```

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.