OpenCV configuration options reference

Prev Tutorial: OpenCV installation overview

Next Tutorial: Installation in Linux

Table of Contents

Introduction

Note

We assume you have read **OpenCV installation overview** tutorial or have experience with CMake.

Configuration options can be set in several different ways:

- Command line: `cmake -Doption=value ...`
- Initial cache files: `cmake -C my_options.txt ...`
- Interactive via GUI

In this reference we will use regular command line.

Most of the options can be found in the root cmake script of OpenCV: `opencv/CMakeLists.txt`. Some options can be defined in specific modules.

It is possible to use CMake tool to print all available options:

```
# initial configuration
cmake ../opencv

# print all options
cmake -L

# print all options with help message
cmake -LH

# print all options including advanced
cmake -LA
```

Most popular and useful are options starting with `WITH_`, `ENABLE_`, `BUILD_`, `OPENCV_`.

Default values vary depending on platform and other options values.

General options

Build with extra modules

`OPENCV_EXTRA_MODULES_PATH` option contains a semicolon-separated list of directories containing extra modules which will be added to the build. Module directory must have compatible layout and CMakeLists.txt, brief description can be found in the Coding Style Guide.

Examples:

```
# build with all modules in opencv_contrib
cmake -DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules ../opencv

# build with one of opencv_contrib modules
cmake -DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules/bgsegm ../opencv

# build with two custom modules (semicolon must be escaped in bash)
cmake -DOPENCV_EXTRA_MODULES_PATH=../my_mod1\;../my_mod2 ../opencv
```

Note

Only 0- and 1-level deep module locations are supported, following command will raise an error:

```
cmake -DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib ../opencv
```

Debug build

`CMAKE_BUILD_TYPE` option can be used to enable debug build; resulting binaries will contain debug symbols and most of compiler optimizations will be turned off. To enable debug symbols in Release build turn the `BUILD_WITH_DEBUG_INFO` option on.

On some platforms (e.g. Linux) build type must be set at configuration stage:

```
cmake -DCMAKE_BUILD_TYPE=Debug ../opencv
cmake --build .
```

On other platforms different types of build can be produced in the same build directory (e.g. Visual Studio, XCode):

```
cmake <options> ../opencv
cmake --build . --config Debug
```

If you use GNU libstdc++ (default for GCC) you can turn on the `ENABLE_GNU_STL_DEBUG` option, then C++ library will be used in Debug mode, e.g. indexes will be bound-checked during vector element access.

Many kinds of optimizations can be disabled with `CV_DISABLE_OPTIMIZATION` option:

- Some third-party libraries (e.g. IPP, Lapack, Eigen)
- Explicit vectorized implementation (universal intrinsics, raw intrinsics, etc.)
- Dispatched optimizations
- Explicit loop unrolling

See also

https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html

https://gcc.gnu.org/onlinedocs/libstdc++/manual/using_macros.html

https://github.com/opencv/opencv/wiki/CPU-optimizations-build-options

Static build

`BUILD_SHARED_LIBS` option control whether to produce dynamic (.dll, .so, .dylib) or static (.a, .lib) libraries. Default value depends on target platform, in most cases it is `ON` .

Example:

```
cmake -DBUILD_SHARED_LIBS=OFF ../opencv
```

See also

https://en.wikipedia.org/wiki/Static_library

`ENABLE_PIC` sets the CMAKE_POSITION_INDEPENDENT_CODE option. It enables or disable generation of "position-independent code". This option must be enabled when building dynamic libraries or static libraries intended to be linked into dynamic libraries. Default value is `ON` .

See also

https://en.wikipedia.org/wiki/Position-independent_code

Generate pkg-config info

`OPENCV_GENERATE_PKGCONFIG` option enables `.pc` file generation along with standard CMake package. This file can be useful for projects which do not use CMake for build.

Example:

```
cmake -DOPENCV_GENERATE_PKGCONFIG=ON ../opencv
```

Note

Due to complexity of configuration process resulting `.pc` file can contain incomplete list of third-party dependencies and may not work in some configurations, especially for static builds. This feature is not officially supported since 4.x version and is disabled by default.

Build tests, samples and applications

There are two kinds of tests: accuracy (`opencv_test_*`) and performance (`opencv_perf_*`). Tests and applications are enabled by default. Examples are not being built by default and should be enabled explicitly.

Corresponding *cmake* options:

```
cmake \
  -DBUILD_TESTS=ON \
  -DBUILD_PERF_TESTS=ON \
  -DBUILD_EXAMPLES=ON \
  -DBUILD_opencv_apps=ON \
  ../opencv
```

Build limited set of modules

Each module is a subdirectory of the `modules` directory. It is possible to disable one module:

```
cmake -DBUILD_opencv_calib3d=OFF ../opencv
```

The opposite option is to build only specified modules and all modules they depend on:

```
cmake -DBUILD_LIST=calib3d,videoio,ts ../opencv
```

In this example we requested 3 modules and configuration script has determined all dependencies automatically:

```
--   OpenCV modules:
--     To be built:                 calib3d core features2d flann highgui imgcodecs imgproc ts videoio
```

Downloaded dependencies

Configuration script can try to download additional libraries and files from the internet, if it fails to do it corresponding features will be turned off. In some cases configuration error can occur. By default all files are first downloaded to the `<source>/.cache` directory and then unpacked or copied to the build directory. It is possible to change download cache location by setting environment variable or configuration option:

```
export OPENCV_DOWNLOAD_PATH=/tmp/opencv-cache
cmake ../opencv
# or
cmake -DOPENCV_DOWNLOAD_PATH=/tmp/opencv-cache ../opencv
```

In case of access via proxy, corresponding environment variables should be set before running cmake:

```
export http_proxy=<proxy-host>:<port>
export https_proxy=<proxy-host>:<port>
```

Full log of download process can be found in build directory - `CMakeDownloadLog.txt` . In addition, for each failed download a command will be added to helper scripts in the build directory, e.g. `download_with_wget.sh` . Users can run these scripts as is or modify according to their needs.

CPU optimization level

On x86_64 machines the library will be compiled for SSE3 instruction set level by default. This level can be changed by configuration option:

```
cmake -DCPU_BASELINE=AVX2 ../opencv
```

Note

Other platforms have their own instruction set levels: `VFPV3` and `NEON` on ARM, `VSX` on PowerPC.

Some functions support dispatch mechanism allowing to compile them for several instruction sets and to choose one during runtime. List of enabled instruction sets can be changed during configuration:

```
cmake -DCPU_DISPATCH=AVX,AVX2 ../opencv
```

To disable dispatch mechanism this option should be set to an empty value:

```
cmake -DCPU_DISPATCH= ../opencv
```

It is possible to disable optimized parts of code for troubleshooting and debugging:

```
# disable universal intrinsics
cmake -DCV_ENABLE_INTRINSICS=OFF ../opencv
# disable all possible built-in optimizations
cmake -DCV_DISABLE_OPTIMIZATION=ON ../opencv
```

Note

More details on CPU optimization options can be found in wiki: https://github.com/opencv/opencv/wiki/CPU-optimizations-build-options

Profiling, coverage, sanitize, hardening, size optimization

Following options can be used to produce special builds with instrumentation or improved security. All options are disabled by default.

| Option | Compiler | Description | | `ENABLE_PROFILING` | GCC or Clang | Enable profiling compiler and linker options. | | `ENABLE_COVERAGE` | GCC or Clang | Enable code coverage support. | | `OPENCV_ENABLE_MEMORY_SANITIZER` | N/A | Enable several quirks in code to assist memory sanitizer. | | `ENABLE_BUILD_HARDENING` | GCC, Clang, MSVC | Enable compiler options which reduce possibility of code exploitation. | | `ENABLE_LTO` | GCC, Clang, MSVC | Enable Link Time Optimization (LTO). | | `ENABLE_THIN_LTO` | Clang | Enable thin LTO which incorporates intermediate bitcode to binaries allowing consumers optimize their applications later. |

See also

- GCC instrumentation
- Build hardening
- Interprocedural optimization
- Link time optimization
- ThinLTO

Functional features and dependencies

There are many optional dependencies and features that can be turned on or off. *cmake* has special option allowing to print all available configuration parameters:

```
cmake -LH ../opencv
```

Options naming conventions

There are three kinds of options used to control dependencies of the library, they have different prefixes:

- Options starting with `WITH_` enable or disable a dependency
- Options starting with `BUILD_` enable or disable building and using 3rdparty library bundled with OpenCV

- Options starting with `HAVE_` indicate that dependency have been enabled, can be used to manually enable a dependency if automatic detection can not be used.

When `WITH_` option is enabled:

- If `BUILD_` option is enabled, 3rdparty library will be built and enabled => `HAVE_` set to `ON`
- If `BUILD_` option is disabled, 3rdparty library will be detected and enabled if found => `HAVE_` set to `ON` if dependency is found

Heterogeneous computation

CUDA support

`WITH_CUDA` (default: *OFF*)

Many algorithms have been implemented using CUDA acceleration, these functions are located in separate modules. CUDA toolkit must be installed from the official NVIDIA site as a prerequisite. For cmake versions older than 3.9 OpenCV uses own `cmake/FindCUDA.cmake` script, for newer versions - the one packaged with CMake. Additional options can be used to control build process, e.g. `CUDA_GENERATION` or `CUDA_ARCH_BIN` . These parameters are not documented yet, please consult with the `cmake/OpenCVDetectCUDA.cmake` script for details.

Note

Since OpenCV version 4.0 all CUDA-accelerated algorithm implementations have been moved to the *opencv_contrib* repository. To build *opencv* and *opencv_contrib* together check **Build with extra modules**.

Some tutorials can be found in the corresponding section: **GPU-Accelerated Computer Vision (cuda module)**

See also

CUDA-accelerated Computer Vision

https://en.wikipedia.org/wiki/CUDA

TODO: other options: `WITH_CUFFT` , `WITH_CUBLAS` , `WITH_NVCUVID` ?

OpenCL support

`WITH_OPENCL` (default: *ON*)

Multiple OpenCL-accelerated algorithms are available via so-called "Transparent API (T-API)". This integration uses same functions at the user level as regular CPU implementations. Switch to the OpenCL execution branch happens if input and output image arguments are passed as opaque **cv::UMat** objects. More information can be found in the brief introduction and **OpenCL support**

At the build time this feature does not have any prerequisites. During runtime a working OpenCL runtime is required, to check it run `clinfo` and/or `opencv_version --opencl` command. Some parameters of OpenCL integration can be modified using environment variables, e.g. `OPENCV_OPENCL_DEVICE` . However there is no thorough documentation for this feature yet, so please check the source code in `modules/core/src/ocl.cpp` file for details.

See also

https://en.wikipedia.org/wiki/OpenCL

TODO: other options: `WITH_OPENCL_SVM` , `WITH_OPENCLAMDFFT` , `WITH_OPENCLAMDBLAS` , `WITH_OPENCL_D3D11_NV` , `WITH_VA_INTEL`

Image reading and writing (imgcodecs module)

Built-in formats

Following formats can be read by OpenCV without help of any third-party library:

- BMP
- HDR (`WITH_IMGCODEC_HDR`)
- Sun Raster (`WITH_IMGCODEC_SUNRASTER`)
- PPM, PGM, PBM, PFM (`WITH_IMGCODEC_PXM` , `WITH_IMGCODEC_PFM`)

PNG, JPEG, TIFF, WEBP support

Formats	Option	Default	Force build own
PNG	`WITH_PNG`	*ON*	`BUILD_PNG`
JPEG	`WITH_JPEG`	*ON*	`BUILD_JPEG`
TIFF	`WITH_TIFF`	*ON*	`BUILD_TIFF`
WEBP	`WITH_WEBP`	*ON*	`BUILD_WEBP`
JPEG2000 with OpenJPEG	`WITH_OPENJPEG`	*ON*	`BUILD_OPENJPEG`
JPEG2000 with JasPer	`WITH_JASPER`	*ON* (see note)	`BUILD_JASPER`
EXR	`WITH_OPENEXR`	*ON*	`BUILD_OPENEXR`

All libraries required to read images in these formats are included into OpenCV and will be built automatically if not found at the configuration stage. Corresponding `BUILD_*` options will force building and using own libraries, they are enabled by default on some platforms, e.g. Windows.

Note

OpenJPEG have higher priority than JasPer which is deprecated. In order to use JasPer, OpenJPEG must be disabled.

GDAL integration

`WITH_GDAL` (default: *OFF*)

GDAL is a higher level library which supports reading multiple file formats including PNG, JPEG and TIFF. It will have higher priority when opening files and can override other backends. This library will be searched using cmake package mechanism, make sure it is installed correctly or manually set `GDAL_DIR` environment or cmake variable.

GDCM integration

`WITH_GDCM` (default: *OFF*)

Enables DICOM medical image format support through GDCM library. This library will be searched using cmake package mechanism, make sure it is installed correctly or manually set `GDCM_DIR` environment or cmake variable.

Video reading and writing (videoio module)

TODO: how videoio works, registry, priorities

Video4Linux

`WITH_V4L` (Linux; default: *ON*)

Capture images from camera using Video4Linux API. Linux kernel headers must be installed.

FFmpeg

`WITH_FFMPEG` (default: *ON*)

Integration with FFmpeg library for decoding and encoding video files and network streams. This library can read and write many popular video formats. It consists of several components which must be installed as prerequisites for the build:

- *avcodec*
- *avformat*
- *avutil*
- *swscale*
- *avresample* (optional)

Exception is Windows platform where a prebuilt plugin library containing FFmpeg will be downloaded during a configuration stage and copied to the `bin` folder with all produced libraries.

Note

Libav library can be used instead of FFmpeg, but this combination is not actively supported.

GStreamer

`WITH_GSTREAMER` (default: *ON*)

Enable integration with GStreamer library for decoding and encoding video files, capturing frames from cameras and network streams. Numerous plugins can be installed to extend supported formats list. OpenCV allows running arbitrary GStreamer pipelines passed as strings to **cv::VideoCapture** and **cv::VideoWriter** objects.

Various GStreamer plugins offer HW-accelerated video processing on different platforms.

Microsoft Media Foundation

`WITH_MSMF` (Windows; default: *ON*)

Enables MSMF backend which uses Windows' built-in Media Foundation framework. Can be used to capture frames from camera, decode and encode video files. This backend have HW-accelerated processing support (`WITH_MSMF_DXVA` option, default is *ON*).

Note

Older versions of Windows (prior to 10) can have incompatible versions of Media Foundation and are known to have problems when used from OpenCV.

DirectShow

`WITH_DSHOW` (Windows; default: *ON*)

This backend uses older DirectShow framework. It can be used only to capture frames from camera. It is now deprecated in favor of MSMF backend, although both can be enabled in the same build.

AVFoundation

`WITH_AVFOUNDATION` (Apple; default: *ON*)

AVFoundation framework is part of Apple platforms and can be used to capture frames from camera, encode and decode video files.

Other backends

There are multiple less popular frameworks which can be used to read and write videos. Each requires corresponding library or SDK installed.

Option	Default	Description
`WITH_1394`	*ON*	IIDC IEEE1394 support using DC1394 library
`WITH_OPENNI`	*OFF*	OpenNI can be used to capture data from depth-sensing cameras. Deprecated.
`WITH_OPENNI2`	*OFF*	OpenNI2 can be used to capture data from depth-sensing cameras.
`WITH_PVAPI`	*OFF*	PVAPI is legacy SDK for Prosilica GigE cameras. Deprecated.
`WITH_ARAVIS`	*OFF*	Aravis library is used for video acquisition using Genicam cameras.
`WITH_XIMEA`	*OFF*	XIMEA cameras support.
`WITH_XINE`	*OFF*	XINE library support.
`WITH_LIBREALSENSE`	*OFF*	RealSense cameras support.
`WITH_MFX`	*OFF*	MediaSDK library can be used for HW-accelerated decoding and encoding of raw video streams.
`WITH_GPHOTO2`	*OFF*	GPhoto library can be used to capure frames from cameras.
`WITH_ANDROID_MEDIANDK`	*ON*	MediaNDK library is available on Android since API level 21.

videoio plugins

Since version 4.1.0 some *videoio* backends can be built as plugins thus breaking strict dependency on third-party libraries and making them optional at runtime. Following options can be used to control this mechanism:

Option	Default	Description
`VIDEOIO_ENABLE_PLUGINS`	*ON*	Enable or disable plugins completely.
`VIDEOIO_PLUGIN_LIST`	*empty*	Comma- or semicolon-separated list of backend names to be compiled as plugins. Supported names are *ffmpeg*, *gstreamer*, *msmf*, *mfx* and *all*.

Check **OpenCV installation overview** for standalone plugins build instructions.

Parallel processing

Some of OpenCV algorithms can use multithreading to accelerate processing. OpenCV can be built with one of threading backends.

Backend	Option	Default	Platform	Description
pthreads	`WITH_PTHREADS_PF`	*ON*	Unix-like	Default backend based on pthreads library is available on Linux, Android and other Unix-like platforms. Thread pool is implemented in OpenCV and can be controlled with environment variables `OPENCV_THREAD_POOL_*` . Please check sources in *modules/core/src/parallel_impl.cpp* file for details.
Concurrency	N/A	*ON*	Windows	Concurrency runtime is available on Windows and will be turned *ON* on supported platforms unless other backend is enabled.
GCD	N/A	*ON*	Apple	Grand Central Dispatch is available on Apple platforms and will be turned *ON* automatically unless other backend is enabled. Uses global system thread pool.
TBB	`WITH_TBB`	*OFF*	Multiple	Threading Building Blocks is a cross-platform library for parallel programming.
OpenMP	`WITH_OPENMP`	*OFF*	Multiple	OpenMP API relies on compiler support.
HPX	`WITH_HPX`	*OFF*	Multiple	High Performance ParallelX is an experimental backend which is more suitable for multiprocessor environments.

Note

OpenCV can download and build TBB library from GitHub, this functionality can be enabled with the `BUILD_TBB` option.

Threading plugins

Since version 4.5.2 OpenCV supports dynamically loaded threading backends. At this moment only separate compilation process is supported: first you have to build OpenCV with some *default* parallel backend (e.g. pthreads), then build each plugin and copy resulting binaries to the *lib* or *bin* folder.

Option	Default	Description

PARALLEL_ENABLE_PLUGINS	ON	Enable plugin support, if this option is disabled OpenCV will not try to load anything

Check **OpenCV installation overview** for standalone plugins build instructions.

GUI backends (highgui module)

OpenCV relies on various GUI libraries for window drawing.

Option	Default	Platform	Description
`WITH_GTK`	*ON*	Linux	GTK is a common toolkit in Linux and Unix-like OS-es. By default version 3 will be used if found, version 2 can be forced with the `WITH_GTK_2_X` option.
`WITH_WIN32UI`	*ON*	Windows	WinAPI is a standard GUI API in Windows.
N/A	*ON*	macOS	Cocoa is a framework used in macOS.
`WITH_QT`	*OFF*	Cross-platform	Qt is a cross-platform GUI framework.

Note

OpenCV compiled with Qt support enables advanced *highgui* interface, see **Qt New Functions** for details.

OpenGL

`WITH_OPENGL` (default: *OFF*)

OpenGL integration can be used to draw HW-accelerated windows with following backends: GTK, WIN32 and Qt. And enables basic interoperability with OpenGL, see **OpenGL interoperability** and **OpenGL support** for details.

highgui plugins

Since OpenCV 4.5.3 GTK backend can be build as a dynamically loaded plugin. Following options can be used to control this mechanism:

Option	Default	Description
`HIGHGUI_ENABLE_PLUGINS`	*ON*	Enable or disable plugins completely.
`HIGHGUI_PLUGIN_LIST`	*empty*	Comma- or semicolon-separated list of backend names to be compiled as plugins. Supported names are *gtk*, *gtk2*, *gtk3*, and *all*.

Check **OpenCV installation overview** for standalone plugins build instructions.

Deep learning neural networks inference backends and options (dnn module)

OpenCV have own DNN inference module which have own build-in engine, but can also use other libraries for optimized processing. Multiple backends can be enabled in single build. Selection happens at runtime automatically or manually.

Option	Default	Description
`WITH_PROTOBUF`	*ON*	Enables protobuf library search. OpenCV can either build own copy of the library or use external one. This dependency is required by the *dnn* module, if it can't be found module will be disabled.
`BUILD_PROTOBUF`	*ON*	Build own copy of *protobuf*. Must be disabled if you want to use external library.
`PROTOBUF_UPDATE_FILES`	*OFF*	Re-generate all .proto files. *protoc* compiler compatible with used version of *protobuf* must be installed.
`OPENCV_DNN_OPENCL`	*ON*	Enable built-in OpenCL inference backend.
`WITH_INF_ENGINE`	*OFF*	**Deprecated since OpenVINO 2022.1** Enables Intel Inference Engine (IE) backend. Allows to execute networks in IE format (.xml + .bin). Inference Engine must be installed either as part of OpenVINO toolkit, either as a standalone library built from sources.
`INF_ENGINE_RELEASE`	*2020040000*	**Deprecated since OpenVINO 2022.1** Defines version of Inference Engine library which is tied to OpenVINO toolkit version. Must be a 10-digit string, e.g. *2020040000* for OpenVINO 2020.4.
`WITH_NGRAPH`	*OFF*	**Deprecated since OpenVINO 2022.1** Enables Intel NGraph library support. This library is part of Inference Engine backend which allows executing arbitrary networks read from files in multiple formats supported by OpenCV: Caffe, TensorFlow, PyTorch, Darknet, etc.. NGraph library must be installed, it is included into Inference Engine.
`WITH_OPENVINO`	*OFF*	Enable Intel OpenVINO Toolkit support. Should be used for OpenVINO>=2022.1 instead of `WITH_INF_ENGINE` and `WITH_NGRAPH` .
`OPENCV_DNN_CUDA`	*OFF*	Enable CUDA backend. CUDA, CUBLAS and CUDNN must be installed.
`WITH_HALIDE`	*OFF*	Use experimental Halide backend which can generate optimized code for dnn-layers at runtime. Halide must be installed.
`WITH_VULKAN`	*OFF*	Enable experimental Vulkan backend. Does not require additional dependencies, but can use external Vulkan headers (`VULKAN_INCLUDE_DIRS`).

WITH_TENGINE	*OFF*	Enable experimental Tengine backend for ARM CPUs. Tengine library must be installed.

Installation layout

Installation root

To install produced binaries root location should be configured. Default value depends on distribution, in Ubuntu it is usually set to `/usr/local`. It can be changed during configuration:

```
cmake -DCMAKE_INSTALL_PREFIX=/opt/opencv ../opencv
```

This path can be relative to current working directory, in the following example it will be set to `<absolute-path-to-build>/install`:

```
cmake -DCMAKE_INSTALL_PREFIX=install ../opencv
```

After building the library, all files can be copied to the configured install location using the following command:

```
cmake --build . --target install
```

To install binaries to the system location (e.g. `/usr/local`) as a regular user it is necessary to run the previous command with elevated privileges:

```
sudo cmake --build . --target install
```

Note

On some platforms (Linux) it is possible to remove symbol information during install. Binaries will become 10-15% smaller but debugging will be limited:

```
cmake --build . --target install/strip
```

Components and locations

Options cane be used to control whether or not a part of the library will be installed:

Option	Default	Description
INSTALL_C_EXAMPLES	*OFF*	Install C++ sample sources from the *samples/cpp* directory.
INSTALL_PYTHON_EXAMPLES	*OFF*	Install Python sample sources from the *samples/python* directory.
INSTALL_ANDROID_EXAMPLES	*OFF*	Install Android sample sources from the *samples/android* directory.
INSTALL_BIN_EXAMPLES	*OFF*	Install prebuilt sample applications (`BUILD_EXAMPLES` must be enabled).
INSTALL_TESTS	*OFF*	Install tests (`BUILD_TESTS` must be enabled).
OPENCV_INSTALL_APPS_LIST	*all*	Comma- or semicolon-separated list of prebuilt applications to install (from *apps* directory)

Following options allow to modify components' installation locations relatively to install prefix. Default values of these options depend on platform and other options, please check the *cmake/OpenCVInstallLayout.cmake* file for details.

Option	Components
OPENCV_BIN_INSTALL_PATH	applications, dynamic libraries (*win*)
OPENCV_TEST_INSTALL_PATH	test applications
OPENCV_SAMPLES_BIN_INSTALL_PATH	sample applications
OPENCV_LIB_INSTALL_PATH	dynamic libraries, import libraries (*win*)
OPENCV_LIB_ARCHIVE_INSTALL_PATH	static libraries
OPENCV_3P_LIB_INSTALL_PATH	3rdparty libraries
OPENCV_CONFIG_INSTALL_PATH	cmake config package
OPENCV_INCLUDE_INSTALL_PATH	header files
OPENCV_OTHER_INSTALL_PATH	extra data files
OPENCV_SAMPLES_SRC_INSTALL_PATH	sample sources
OPENCV_LICENSES_INSTALL_PATH	licenses for included 3rdparty components
OPENCV_TEST_DATA_INSTALL_PATH	test data
OPENCV_DOC_INSTALL_PATH	documentation
OPENCV_JAR_INSTALL_PATH	JAR file with Java bindings
OPENCV_JNI_INSTALL_PATH	JNI part of Java bindings
OPENCV_JNI_BIN_INSTALL_PATH	Dynamic libraries from the JNI part of Java bindings

Following options can be used to change installation layout for common scenarios:

Option	Default	Description
INSTALL_CREATE_DISTRIB	*OFF*	Tune multiple things to produce Windows and Android distributions.

INSTALL_TO_MANGLED_PATHS	*OFF*	Adds one level to several installation locations to allow side-by-side installations. For example, headers will be installed to _/usr/include/opencv-4.4.0_ instead of _/usr/include/opencv4_ with this option enabled.

Miscellaneous features

Option	Default	Description
OPENCV_ENABLE_NONFREE	*OFF*	Some algorithms included in the library are known to be protected by patents and are disabled by default.
OPENCV_FORCE_3RDPARTY_BUILD	*OFF*	Enable all BUILD_ options at once.
ENABLE_CCACHE	*ON* (on Unix-like platforms)	Enable ccache auto-detection. This tool wraps compiler calls and caches results, can significantly improve re-compilation time.
ENABLE_PRECOMPILED_HEADERS	*ON* (for MSVC)	Enable precompiled headers support. Improves build time.
BUILD_DOCS	*OFF*	Enable documentation build (*doxygen*, *doxygen_cpp*, *doxygen_python*, *doxygen_javadoc* targets). Doxygen must be installed for C++ documentation build. Python and BeautifulSoup4 must be installed for Python documentation build. Javadoc and Ant must be installed for Java documentation build (part of Java SDK).
ENABLE_PYLINT	*ON* (when docs or examples are enabled)	Enable python scripts check with Pylint (*check_pylint* target). Pylint must be installed.
ENABLE_FLAKE8	*ON* (when docs or examples are enabled)	Enable python scripts check with Flake8 (*check_flake8* target). Flake8 must be installed.
BUILD_JAVA	*ON*	Enable Java wrappers build. Java SDK and Ant must be installed.
BUILD_FAT_JAVA_LIB	*ON* (for static Android builds)	Build single *opencv_java* dynamic library containing all library functionality bundled with Java bindings.
BUILD_opencv_python2	*ON*	Build python2 bindings (deprecated). Python with development files and numpy must be installed.
BUILD_opencv_python3	*ON*	Build python3 bindings. Python with development files and numpy must be installed.

TODO: need separate tutorials covering bindings builds

Automated builds

Some features have been added specifically for automated build environments, like continuous integration and packaging systems.

Option	Default	Description
ENABLE_NOISY_WARNINGS	*OFF*	Enables several compiler warnings considered *noisy*, i.e. having less importance than others. These warnings are usually ignored but in some cases can be worth being checked for.
OPENCV_WARNINGS_ARE_ERRORS	*OFF*	Treat compiler warnings as errors. Build will be halted.
ENABLE_CONFIG_VERIFICATION	*OFF*	For each enabled dependency (WITH_ option) verify that it has been found and enabled (HAVE_ variable). By default feature will be silently turned off if dependency was not found, but with this option enabled cmake configuration will fail. Convenient for packaging systems which require stable library configuration not depending on environment fluctuations.
OPENCV_CMAKE_HOOKS_DIR	*empty*	OpenCV allows to customize configuration process by adding custom hook scripts at each stage and substage. cmake scripts with predefined names located in the directory set by this variable will be included before and after various configuration stages. Examples of file names: *CMAKE_INIT.cmake*, *PRE_CMAKE_BOOTSTRAP.cmake*, *POST_CMAKE_BOOTSTRAP.cmake*, etc.. Other names are not documented and can be found in the project cmake files by searching for the *ocv_cmake_hook* macro calls.
OPENCV_DUMP_HOOKS_FLOW	*OFF*	Enables a debug message print on each cmake hook script call.

Contrib Modules

Following build options are utilized in opencv_contrib modules, as stated previously, these extra modules can be added to your final build by setting DOPENCV_EXTRA_MODULES_PATH option.

Option	Default	Description
WITH_CLP	*OFF*	Will add coinor linear programming library build support which is required in videostab module. Make sure to install the development libraries of coinor-clp.

Other non-documented options

BUILD_ANDROID_PROJECTS BUILD_ANDROID_EXAMPLES ANDROID_HOME ANDROID_SDK ANDROID_NDK ANDROID_SDK_ROOT

CMAKE_TOOLCHAIN_FILE

WITH_CAROTENE WITH_CPUFEATURES WITH_EIGEN WITH_OPENVX WITH_DIRECTX WITH_VA WITH_LAPACK WITH_QUIRC BUILD_ZLIB BUILD_ITT WITH_IPP BUILD_IPP_IW