[Draft]

- Author: Vadim Pisarevsky with many inputs from OpenCV team
- Link: https://github.com/opencv/opencv/issues/18995
- Status: Open
- Platforms: All
- Complexity: A few man-years

Introduction

OpenCV 5.0 is the 20-th year anniversary release, initially scheduled for 2020, although the gold release will likely happen in 2021 (as of the moment of writing this document in 2020 December). First of all, as anniversary release, it should be great and should be a strong statement that OpenCV is here, it's modern, up-to-date library and it is going to retain its title of one of the most popular CV API in the world. Also, since it's a major release, it's another good chance to fix some things in a bit more radical way than we can in minor updates. But on the other hand, it should continue the tradition started with OpenCV 2.x – i.e. be a modular C++ library with a rather stable API, so that it should be more or less easy to migrate to it from the previous versions of OpenCV.

The document outlines the list of features and changes planned for OpenCV 5.x, not necessarily 5.0. And some of the features might be postponed till OpenCV 6 or even later versions. As they say, plan is nothing, but planning is everything.

The major directions for 5.0 release are:

- New license
- Clean up API and refresh it to make make the library more future-proof. Make better use of the latest C++ standards and much improved C++ compiler that fully support those standards.
- Better efficiency on various architectures, CPUs and GPUs; Better ARM support and added RISC-V support are two of the major topics.
- Emphasis on deep learning. More architectures will be supported, including the ones from GSoC 2020 for text detection & recognition, object tracking, color calibration etc. Also, deep learning engine itself will be made more modular and will be further optimized.
- Improved and extended 3D vision functionality
- Better documentation, improved samples
- Even better Python support

Let's now discuss the major new/changed things in 5.x.

Highlights

New License

It's done already. OpenCV has migrated to Apache 2 license:

- Evolution proposal with rationale and the details: https://github.com/opencv/opencv/wiki/OE-32.--Change-OpenCV-License-to-Apache-2
- OpenCV 4.5.0 announcement: https://opencv.org/opencv-4-5-0/

C++14 as the minimum required version of C++. Drop C API.

It's time to upgrade requirements to the commonly available versions of the main languages that we use in OpenCV. By the way, modern 4.x versions of OpenCV already support C++ 14, C++ 17 and even C++ 20. At the same time, we

drop support of earlier versions of C++ for leaner, more efficient code and to lighten the support burden.

Also, after 20 years of service it's finally time for C API to completely disappear from OpenCV. Some bits of C API can be retained internally, but will not be exposed anymore.

(?) Modules

Likely not in OpenCV 5.0, but in some further OpenCV 5.x releases, when OpenCV is compiled as C++ 20 library, we are going to compile OpenCV modules as C++ modules. Unfortunately, different compilers offer slightly different implementations of this very useful concept, and it's not quite mature yet, but the advantages are clear, and we will be working on it.

Named parameters in complex algorithms

The goal is to provide cleaner, easier to use API for complex algorithms. See https://github.com/opencv/opencv/wiki/OE-34.-Named-Parameters. This change is compatible with modern versions of clang, GCC and MSVC (C++ 20 needs to be enabled in the case of MSVC), and it also perfectly maps to Python and Swift languages, even though the binding generators need to be updated.

More convenient error reporting and logging. Easier debugging.

One thing where we can definitely make life for OpenCV developers and contributors easier is logging and better error reporting. Probably, we can adopt GoogleTest-like macros, e.g. CV_ASSERT_LT, CV_ASSERT_TRUE etc. maybe even something more advanced if modern C++ standards allow that. Those macros should let, just like in GTest, optional "<< "message" parts. The old error codes, now used in OpenCV, are mostly obsolete. We should keep this API for compatibility and still store error code in cv::Exception. But new macros should just do checks and provide a convenient way to supply extra error messages.

There should be convenient and unified logging capabilities across the whole library. See https://github.com/opencv/opencv/wiki/OE-11.-Logging

Improved matrix calculus/linear algebra support in OpenCV

Lapack (a subset of it) is now always available in OpenCV; see https://github.com/opencv/opencv/wiki/OE-12.-Lapack. In order to reduce fragmentation and the number of branches to tests, `Eigen` calls will be gradually eliminated from the library and replaced with either Lapack or our own matrix operations. For Eigen+OpenCV users the conversions functions from OpenCV matrix types to/from Eigen matrices will still be available.

Speaking of our own matrix operations, a decent quaternion support has been recently added (https://github.com/opencv/opencv/pull/18335) and is being improved.

Extended set of basic types; built-in color space information

In OpenCV 4.x the following data types are supported:

- `CV_8S`, `CV_8U`: 8-bit signed and unsigned integers
- `CV_16S`, `CV_16U`: 16-bit signed and unsigned integers
- `CV_32S`: 32-bit signed integers
- `CV_32F`, `CV_64F`: 32-bit and 64-bit floating-point numbers
- (new in OpenCV 4.x) `CV_16F`: 16-bit floating-point number (IEEE754 "half" format)

and also the "uniform" tuples of such numbers, such as CV_8UC3 (3-tuple of bytes), CV_32FC2 (a pair of floating-point numbers) etc.

Until recently, this value set was sufficient. But now, when people are interested in efficient deep learning and, in general, low-power computing, some new interesting formats appeared, such as `bfloat16` and various flavors of `posit`. Of course, it can be very impractical to implement the whole set of OpenCV operations for each data type. Even the current data types are not universally supported in OpenCV. But still it would be useful to be able to represent various data types and let users implement the missing algorithms.

Since the beginning of the library and till now we allocate just 3 bits to specify the "depth", i.e. the type of each data channel, and all 8 options are already taken, as you may see. The solution is to extend this number to, e.g., 8 bits. Then we have a plenty of space for new types.

Another proposed feature attempts to answer one of the old questions from OpenCV users - is a particular image BGR or RGB? In deep learning era it is more relevant when ever, since some of the networks are trained on RGB images, whereas other on BGR ones, and so incorrect order of the channel may slightly decrease accuracy, which will be annoying and hard-to-catch.

The proposed solution: add `colorspace` field to `cv::Mat`. By default, each new mat is initialized with `colorspace=COLOR_GENERIC` and so it behaves as usual. But we modify some popular functions to copy the source image color space into the destination image. `cv::resize()`, `cv::warpAffine()`, `Mat::copyTo()`, `cv::GaussianBlur()`, `cv::flip()` are examples of such functions. Modify `cv::cvtColor()`, `cv::imread()` and `VideCapture::retrieve()` to set the proper colorspace; modify `cv::imwrite()`, `cv::dnn::Net::forward()` to handle the colorspace properly. That does not seem to be many changes, but it will probably help to eliminate some bugs. Note that currently it's planned to introduce only RGB-related colorspace values (`COLOR_RGB`, `COLOR_BGR`), not YUV, HSV, Lab or such — they all will map to `COLOR_GENERIC`.

Improvements in DNN

[TODO] this needs to be put into a separate evolution proposal

Modular architecture with external DNN backends

This feature would be useful at least for 2 reasons:

1. It will let people create other DNN backends without touching OpenCV DNN code itself. We want to keep OpenCV DNN a popular framework for inference, and so we may still want people to contribute their backends to OpenCV and we will integrate them. That will help to avoid custom forks and fragmentation (at the cost of extra pressure on the core team). But a dedicated backend API with more or less isolated implementations of backends will let us/users configure OpenCV more easily and disable/enable certain backends. Some unstable backends can be moved to opencv_contrib and polished there.
2. Create opportunities for the further growth. We will clearly separate the engine (including memory manager, graph manager, fusion etc.), the importers and the layers. We will see more clearly how much freedom, and what level of customization different backends need. This will let us further improve DNN without making it a complete mess.

Perhaps, the backend engine API should be specified in C++, not in C, but it should be a few purely abstract C++ classes (cv::Algorithm style API). As long as some backend implementation provides smart pointers to those interfaces, we could use it.

Automatic model loading with caching for samples (at least in Python samples, but preferably in C++ as well).

This item may also be relevant for traditional vision samples, but it's critical for deep learning samples. They should be able to download models automatically from various places. See https://github.com/opencv/opencv/pull/18591.

Better efficiency of DNN on ARM, NVidia, RISC-V etc.

The work is already in progress; here we just state that performance will be dramatically improved on various architectures (on NVidia GPUs it's basically the case already). Another big topic is support for specialized deep learning accelerators (so-called VPU's, NPU's, TPU's etc.). But the modular architecture seems to be a strong prerequisite for such activities.

Python improvements

TODO move it to a separate evolution proposal

- drop Python 2.x
- make Python module a directory with binary blob cv2.so and pure python wrappers for some functionality, such as:
  - filestorage I/O
  - code to download DNN models and other files from internet (see above)
  - multi-file reader: https://github.com/opencv/opencv/pull/17753
  - etc.
- extend Python bindings generator to support named arguments: https://github.com/opencv/opencv/wiki/OE-34.-Named-Parameters
- official PIP module with proper build script

RISC-V support

OpenCV provides universal intrinsics for a few years already, and so it's convenient to port it to the new hardware this way. RISC-V is emerging platform for edge computing and we are glad to support it in OpenCV. This is now work in progress, thanks to T-head (https://github.com/opencv/opencv/pull/18394) and Institute of Software, Chinese Academy of Science (https://github.com/opencv/opencv/pull/18228).

State-of-art 3D Module

See https://github.com/opencv/opencv/wiki/OE-33.-3D-Module.

Tracking module

(More detailed description) Some of the best OpenCV trackers are moved from opencv_contrib/tracking and more deep learning trackers are added. Already in master: https://github.com/opencv/opencv/pull/18838

Improvements in UI

- See https://github.com/opencv/opencv/wiki/OE-24.-Module-HighGUI
- One of the big topics is done already; OpenCV 5.0-pre (next branch) supports truetype fonts: https://github.com/opencv/opencv/pull/18760.

More serious testing of complex algorithms

[TODO] make a separate evolution proposal about it.

The goal for OpenCV 5.x is to establish the solid testing methodology for complex algorithms, i.e. depart from simple "smoke" or "toy" tests that just check that functions do not crash to more serious tests that will evaluate performance and probably speed of the complex algorithms on real datasets or their substantial subsets. Here are some examples of such complex algorithms, where more serious testing methodology will be very useful:

- Face detection
- Face analysis (age/gender/emotions/facial features etc.)
- Chessboard/circles/aruco/april tag pattern detection
- QR code detection
- Circle, line detection
- Text detection and recognition
- Tracking
- Stereo
- People detection
- Visual odometry
- Image registration (homography, fundamental matrix, essential matrix ...)

Use some public benchmarks/datasets to test complex OpenCV algorithms, instead of testing the algorithms on just a few samples, which does not show real quality and does not help to catch real regressions. It should probably be some light/express testing mode when only a small part of the benchmark/dataset is used (just a few dozens of samples perhaps); and then it should be more or less complete nightly or weekly testing where the whole dataset or its substantial part is used. Some integral metric is computed and compared to the reference one. There can also be checks for individual samples to catch abnormal behaviour (e.g. some bugs in the code are introduced that affect just a few test cases without affecting integral characteristics). The datasets should be downloadable using the same technique as we currently use to download deep learning models.

In the next phase we can also consider benchmarks that are not assumed to be run locally, but rather require a part of OpenCV with the tested algorithm to be compiled and sent to a server for remote testing. This can hardly be automated and run reliably, but later we can consider it as well. Here the primary goal is not to participate in various competitions, but to test complex OpenCV algorithms much more extensively than we do now.

Improved Documentation

Improving documentation, especially for such a big and complex project as OpenCV, is an endless story, but we have to do it constantly. With migration from Sphinx to Doxygen the documentation has become more complete for sure, but at the same time the old documentation perhaps looks better and has a very convenient navigation pane on the left. CUDA documentation, highly praised by developers, also uses a similar approach.

The documentation should have more “hand-written” style with less auto-generated almost empty descriptions with little source code snippets etc. There should be a navigation pane on the left, which is just convenient and leaves more space for the actual contents in the main pane. It would be nice to have “Python mode” when all the specifications are automatically replaced with Python versions. Or at least have them side by side, as in the old Sphinx-based documentation. A special tool (initially and @pre-commit) should check the link consistency. These and several other important things to improve are already outlined in a separate document: https://github.com/opencv/opencv/wiki/Documentation-improvement-plan. It’s probably a good time to start implementing it. The plan is to continue to use Doxygen or at least a tool that can parse headers and is compatible with Doxygen, but to heavily customize the produced HTML files.

See also https://github.com/opencv/opencv/pull/18712.