The OpenVINO toolkit tutorial

Performance Optimization CV Winter Camp 2021

Eduard Zamaliev

What we need

- Microsoft Visual Studio 2015/2017
- CMake
- The OpenVINO toolkit (contains OpenCV)
- Open Model Zoo
- Segmentation model (e.g. DeepLab V3)

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
   add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
   add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings
                /W3  # Specify the level of warnings to be generated by the compiler
                /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)


add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)


find_package(OpenCV  REQUIRED)

add_definitions(-DUSE_OPENCV)

find_package(InferenceEngine 2.0 REQUIRED)


if(MSVC)

   add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)

   add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings

                  /W3  # Specify the level of warnings to be generated by the compiler

                  /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw

endif()


add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)

add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)



add_executable(blur_background_demo blur_background_demo.cpp)

target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
   add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
   add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings
                  /W3  # Specify the level of warnings to be generated by the compiler
                  /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)


add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
  add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
  add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings
             /W3  # Specify the level of warnings to be generated by the compiler
             /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)


add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
   add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
   add_compile_options(/wd4251 /wd4275 /wd4267 /W3 /EHsc)
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)


add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
  add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
  add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings
          /W3  # Specify the level of warnings to be generated by the compiler
          /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)

add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES} ${InferenceEngine_LIBRARIES} models utils gflags)
```

intel.

Build the project

```
cmake_minimum_required (VERSION 3.10)

project(blur_background_demo)

find_package(OpenCV  REQUIRED)
add_definitions(-DUSE_OPENCV)
find_package(InferenceEngine 2.0 REQUIRED)

if(MSVC)
  add_definitions(-D_CRT_SECURE_NO_WARNINGS -DSCL_SECURE_NO_WARNINGS)
  add_compile_options(/wd4251 /wd4275 /wd4267  # disable some warnings
              /W3  # Specify the level of warnings to be generated by the compiler
              /EHsc)  # Enable standard C++ stack unwinding, assume functions with extern "C" never throw
endif()

add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/models models)
add_subdirectory(${OMZ_DEMO_DIR}/common/cpp/utils utils)
add_subdirectory(${OMZ_DEMO_DIR}/thirdparty/gflags gflags EXCLUDE_FROM_ALL)

add_executable(blur_background_demo blur_background_demo.cpp)
target_link_libraries(blur_background_demo ${OpenCV_LIBRARIES}
${InferenceEngine_LIBRARIES} models utils gflags)
```

Build the project

- Set the OpenVINO's environment:

```
<path/to/openvino>/bin/setupvars.bat
```

OR

```
<path/to/openvino>/bin/setupvars.sh
```

- Run Cmake:

```
cmake -B <path/for/build> -DOMZ_DEMO_ZOO <path/to/omz>
```

Code: what we use

```
#include <iostream>
#include <string>

#include <opencv2/opencv.hpp>

#include <inference_engine.hpp>

#include <models/segmentation_model.h>
#include <utils/ocv_common.hpp>
#include <utils/performance_metrics.hpp>
```

Code: application parameters

```cpp
int main(int argc, char *argv[])
{
	std::string input = argv[1];
	std::string backgroundPath = argv[2];
	std::string model_path = argv[3];

	…
}
```

Code: open camera

```
int main(int argc, char *argv[])
{
    …

    cv::VideoCapture cap;
    if (cap.open(std::stoi(input)))
    {
        cap.set(cv::CAP_PROP_FRAME_WIDTH, 1280);
        cap.set(cv::CAP_PROP_FRAME_HEIGHT, 720);
        cap.set(cv::CAP_PROP_BUFFERSIZE, 1);
        cap.set(cv::CAP_PROP_AUTOFOCUS, true);
        cap.set(cv::CAP_PROP_FOURCC, cv::VideoWriter::fourcc('M','J','P','G'));
    }

    …
}
```

Code: prepare engine and model

```
int main(int argc, char *argv[])
{
   …

   InferenceEngine::Core engine;

   ModelBase *model = new SegmentationModel(model_path, true);
   InferenceEngine::CNNNetwork cnnNetwork =
      engine.ReadNetwork(model->getModelFileName());


   …
}
```

Code: prepare engine and model

```
int main(int argc, char *argv[])
{
    …

    model->prepareInputsOutputs(cnnNetwork);

    std::string inputName  = model->getInputsNames()[0];
    std::string outputName = model->getOutputsNames()[0];

    …
}
```

Code: prepare engine and model

```
int main(int argc, char *argv[])
{
   …

   InferenceEngine::ExecutableNetwork execNetwork =
      engine.LoadNetwork(cnnNetwork, "GPU");

   InferenceEngine::InferRequest inferRequest =
      execNetwork.CreateInferRequest();


   …
}
```

Code: main work cycle

```
while (cap.isOpened())
{
    auto startTime = std::chrono::steady_clock::now();

    cv::Mat frame;
    cap.read(frame);

    …
}
```

Code: main work cycle

```
while (cap.isOpened())
{
    …


    InferenceEngine::Blob::Ptr imgBlob = wrapMat2Blob(frame);
    inferRequest.SetBlob(inputName, imgBlob);

    inferRequest.Infer();
    InferenceEngine::Blob::Ptr result = inferRequest.GetBlob(outputName);


    …
}
```

Code: main work cycle

```
while (cap.isOpened())
{
    …

    InferenceResult inferenceResult;
    inferenceResult.outputsData.emplace(outputName,
        std::make_shared<InferenceEngine::TBlob<float>>(
                *InferenceEngine::as<InferenceEngine::TBlob<float>>(result)));
    inferenceResult.internalModelData =
        std::shared_ptr<InternalImageModelData>(
                new InternalImageModelData(frame.size[1], frame.size[0]));

    std::unique_ptr<ResultBase> segmentationResult = model->postprocess(inferenceResult);

    …
}
```

Code: main work cycle

```
while (cap.isOpened())
{
    …

    cv::Mat outFrame;
    switch (type)
    {
    case DELETE:
        outFrame = remove_background(frame, segmentationResult->asRef<SegmentationResult>());
        break;
    case BACKGROUND:
        outFrame = replace_background(frame, background,
                                      segmentationResult->asRef<SegmentationResult>());
        break;
    }

    …
}
```

Code: main work cycle

```cpp
while (cap.isOpened())
{
    …

    metrics.update(startTime, outFrame, { 10, 22 }, cv::FONT_HERSHEY_COMPLEX,
    0.65);
    cv::imshow("Video", outFrame);
    int key = cv::waitKey(1);
    if (key == 27)
        break;
    if (key == 9)
    {
        type++;
        if (type == NONE)
                type = 0;
    }
}
```

Image transformation

```
cv::Mat replace_background(
      cv::Mat frame,
      cv::Mat background,
      SegmentationResult& segmentationResult)
{
   auto mask = segmentationResult.mask;

   cv::resize(background, background, frame.size());

   …
}
```

Image transformation

```
cv::Mat replace_background(…)
{
    …

    const int personLabel = 15;
    cv::Mat personMask = cv::Mat(mask.size(), mask.type(), 15);
    cv::compare(mask, personMask, personMask, cv::CMP_EQ);

    …
}
```

Image transformation

```
cv::Mat replace_background(…)
{
    …

    cv::Mat maskedFrame;
    cv::bitwise_or(frame, frame, maskedFrame, personMask);

    …
}
```

Image transformation

```
cv::Mat replace_background(…)
{
    …

    cv::Mat backgroundMask;
    cv::bitwise_not(personMask, backgroundMask);

    cv::Mat maskedBackground;
    cv::bitwise_or(background, background, maskedBackground,
    backgroundMask);

    …
}
```

Image transformation

```
cv::Mat replace_background(…)
{
   …

   cv::bitwise_or(maskedFrame, maskedBackground, frame);

   return frame;
}
```