

Performance Optimization CV Winter Camp 2022

# The OpenVINO toolkit practice

Mikhail Nosov



# Agenda

- Prerequisites
- Model Optimizer
- Inference Engine
  - Prepare data
  - Inference on MobileNet V2
- Performance improvements
  - Async mode
  - Config keys to increase throughput
  - Batched inputs
  - Performance counters

# Prerequisites, what we need

- [Python 3.6/3.7/3.8](#) 64-bit
- [MS Visual Studio](#) (2019 or Community version, C++)
- [CMake](#) (version  $\geq 3.13$ )
- [Git](#) for Windows
- [OpenVINO](#) version 2021.4.2 (next slide for details)

[https://docs.openvino.ai/2021.4/openvino\\_docs\\_install\\_guides\\_installing\\_openvino\\_windows.html](https://docs.openvino.ai/2021.4/openvino_docs_install_guides_installing_openvino_windows.html)

# Install OpenVINO

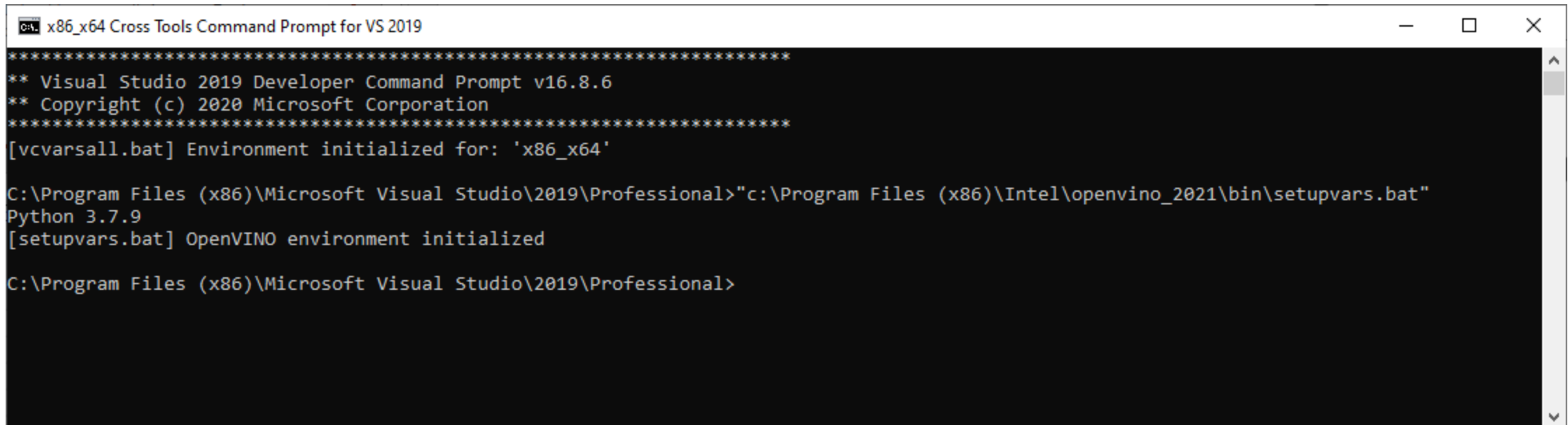
- Distribution: online and offline
- Version: 2021 4.2 LTS (This is the latest available version which is used for this practice)
- Type: offline
- During installation - keep all checkboxes on (Inference Engine, Model Optimizer, Open Model Zoo, OpenCV)

# Training code

[https://github.com/nosovmik/opencvino\\_basic\\_practice](https://github.com/nosovmik/opencvino_basic_practice)

# Setup environment variables

- Open Visual Studio command prompt
- Run “c:\Program Files (x86)\Intel\openvino\_2021\bin\setupvars.bat”



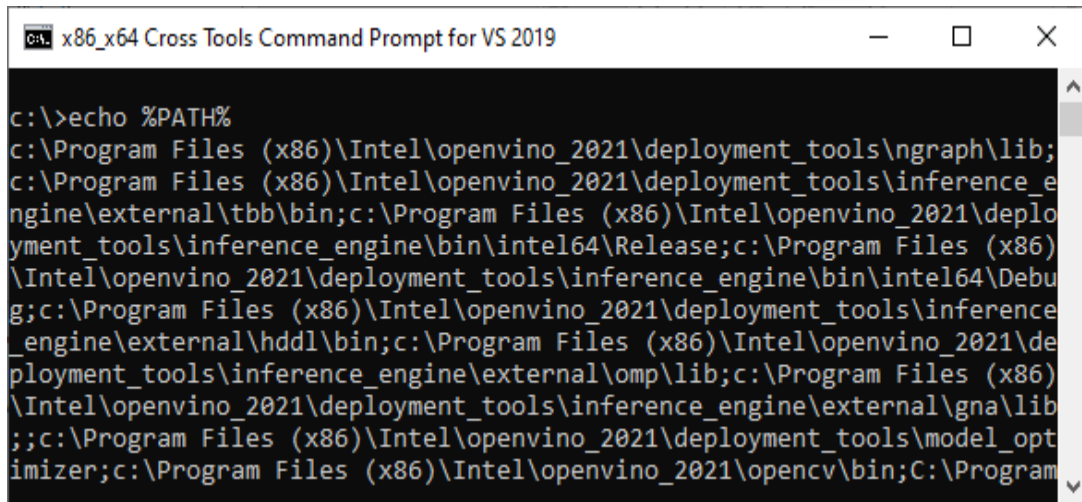
```
x86_x64 Cross Tools Command Prompt for VS 2019
*****
** Visual Studio 2019 Developer Command Prompt v16.8.6
** Copyright (c) 2020 Microsoft Corporation
*****
[vcvarsall.bat] Environment initialized for: 'x86_x64'

C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional>"c:\Program Files (x86)\Intel\openvino_2021\bin\setupvars.bat"
Python 3.7.9
[setupvars.bat] OpenVINO environment initialized

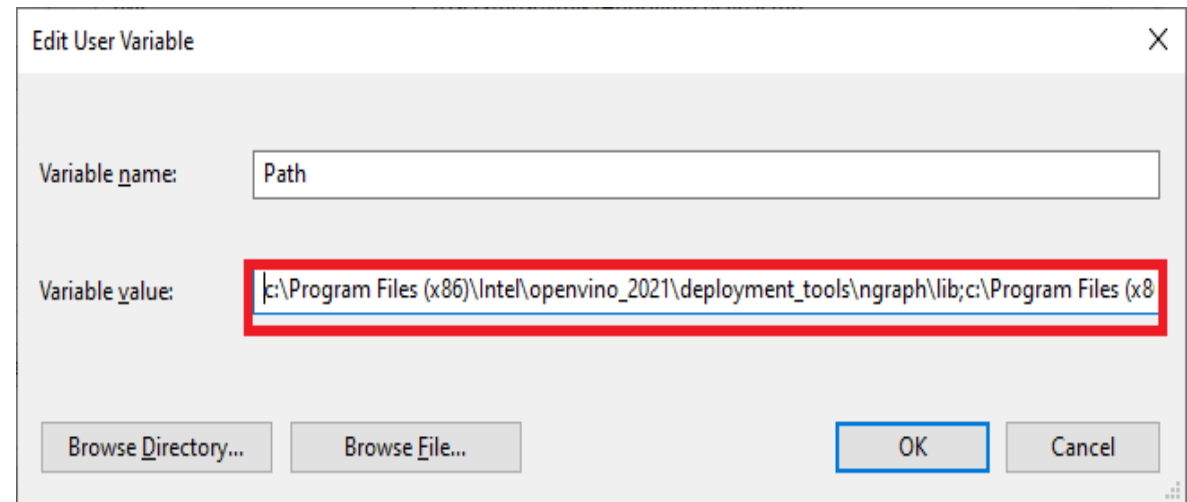
C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional>
```

# Setup environment variables (2, optional)

- Check new PATH, set it as global environment variable
- Can be useful for MS Visual Studio



```
c:\>echo %PATH%
c:\Program Files (x86)\Intel\openvino_2021\deployment_tools\ngraph\lib;
c:\Program Files (x86)\Intel\openvino_2021\deployment_tools\inference_e
ngine\external\tbb\bin;c:\Program Files (x86)\Intel\openvino_2021\depl
oyment_tools\inference_engine\bin\intel64\Release;c:\Program Files (x86)
\Intel\openvino_2021\deployment_tools\inference_engine\bin\intel64\Debu
g;c:\Program Files (x86)\Intel\openvino_2021\deployment_tools\inference
_engine\external\hddl\bin;c:\Program Files (x86)\Intel\openvino_2021\de
ployment_tools\inference_engine\external\omp\lib;c:\Program Files (x86)
\Intel\openvino_2021\deployment_tools\inference_engine\external\gna\lib
;c:\Program Files (x86)\Intel\openvino_2021\deployment_tools\model_opt
imizer;c:\Program Files (x86)\Intel\openvino_2021\opencv\bin;C:\Program
```



# Obtain Pre-trained model



OFFLINE



**Trained Models**

Caffe\*

TensorFlow\*

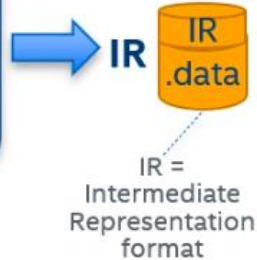
MxNet\*

ONNX\*

Pytorch\*, Caffe2\* & more

Kaldi\*

**Model Optimizer**



**OpenVINO**

**Inference Engine**

CPU Plugin

GPU Plugin

FPGA Plugin

Myriad Plugin  
for Intel NCS & NCS

HDDL Plugin  
for VAD\*

GNA Plugin



GPU = Intel CPU with integrated GPU/Intel® Processor Graphics, Intel® NCS = Intel® Neural Compute Stick (VPU)

\*VAD = Intel® Vision Accelerator Design Products (HDDL-R)

# Intel Open Model Zoo repository

- GitHub: [https://github.com/openvinotoolkit/open\\_model\\_zoo](https://github.com/openvinotoolkit/open_model_zoo)
- Also available at C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\open\_model\_zoo
- Overviews: [https://docs.openvino.ai/2021.4/omz\\_models\\_group\\_public.html](https://docs.openvino.ai/2021.4/omz_models_group_public.html)

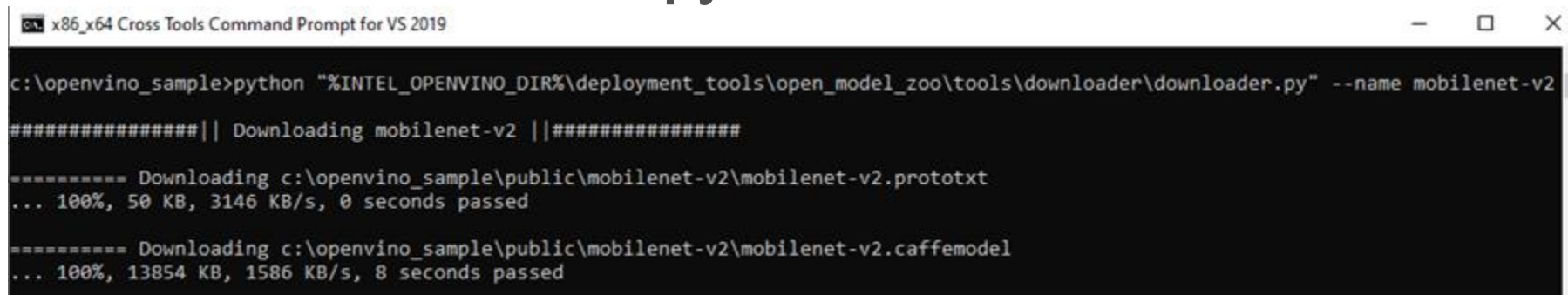
## Install dependencies

```
python -m pip install -r  
"%INTEL_OPENVINO_DIR%\deployment_tools\open_model_zoo  
\tools\downloader\requirements.in"
```

# Fetch Pre-Trained model (mobilenet-v2)

## Download command (from your working directory)

```
c:\openvino_sample>python  
"%INTEL_OPENVINO_DIR%\deployment_tools\open_model_zoo\tools  
s\downloader\downloader.py" --name mobilenet-v2
```



```
x86_x64 Cross Tools Command Prompt for VS 2019  
c:\openvino_sample>python "%INTEL_OPENVINO_DIR%\deployment_tools\open_model_zoo\tools\downloader\downloader.py" --name mobilenet-v2  
#####|| Downloading mobilenet-v2 ||#####  
----- Downloading c:\openvino_sample\public\mobilenet-v2\mobilenet-v2.prototxt  
... 100%, 50 KB, 3146 KB/s, 0 seconds passed  
----- Downloading c:\openvino_sample\public\mobilenet-v2\mobilenet-v2.caffemodel  
... 100%, 13854 KB, 1586 KB/s, 8 seconds passed
```

It is not an IR (.caffemodel) – need Model Optimizer to convert

[https://docs.openvino.ai/2021.4/omz\\_tools\\_downloader.html](https://docs.openvino.ai/2021.4/omz_tools_downloader.html)

[https://docs.openvino.ai/2021.4/omz\\_models\\_model\\_mobilenet\\_v2.html](https://docs.openvino.ai/2021.4/omz_models_model_mobilenet_v2.html)

# Model Optimizer

- Located offline at <intel\_openvino\_dir>\deployment\_tools\model\_optimizer
- Converts models to IR format from framework files
- Performs device-independent optimizations
- Many other things
- Entry point: “mo.py”

## Install python dependencies

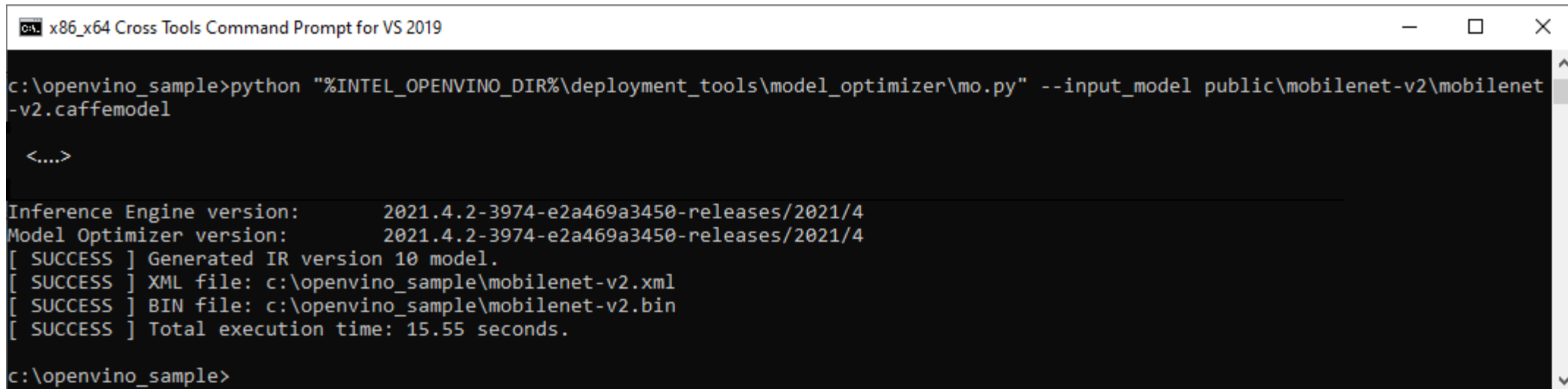
```
python -m pip install -r  
"%INTEL_OPENVINO_DIR%\deployment_tools\model_optimizer\  
requirements_caffe.txt
```

# Model Optimizer – convert mobilenet-v2

## Convert mobilenet-v2.caffemodel

```
c:\openvino_sample>python
```

```
"%INTEL_OPENVINO_DIR%\deployment_tools\model_optimizer\mo.py"  
" --input_model public\mobilenet-v2\mobilenet-v2.caffemodel
```



```
x86_x64 Cross Tools Command Prompt for VS 2019  
c:\openvino_sample>python "%INTEL_OPENVINO_DIR%\deployment_tools\model_optimizer\mo.py" --input_model public\mobilenet-v2\mobilenet-v2.caffemodel  
<....>  
Inference Engine version:      2021.4.2-3974-e2a469a3450-releases/2021/4  
Model Optimizer version:      2021.4.2-3974-e2a469a3450-releases/2021/4  
[ SUCCESS ] Generated IR version 10 model.  
[ SUCCESS ] XML file: c:\openvino_sample\mobilenet-v2.xml  
[ SUCCESS ] BIN file: c:\openvino_sample\mobilenet-v2.bin  
[ SUCCESS ] Total execution time: 15.55 seconds.  
c:\openvino_sample>
```

# Create CMake project

# Create project

- Create empty files
  - “ov\_practice1.cpp”
  - CMakeLists.txt

# CMakeLists.txt

```
cmake_minimum_required (VERSION 3.13)
```

```
project(ov_practice)
```

```
find_package(OpenCV REQUIRED)
```

```
find_package(InferenceEngine 2021.4.2 REQUIRED)
```

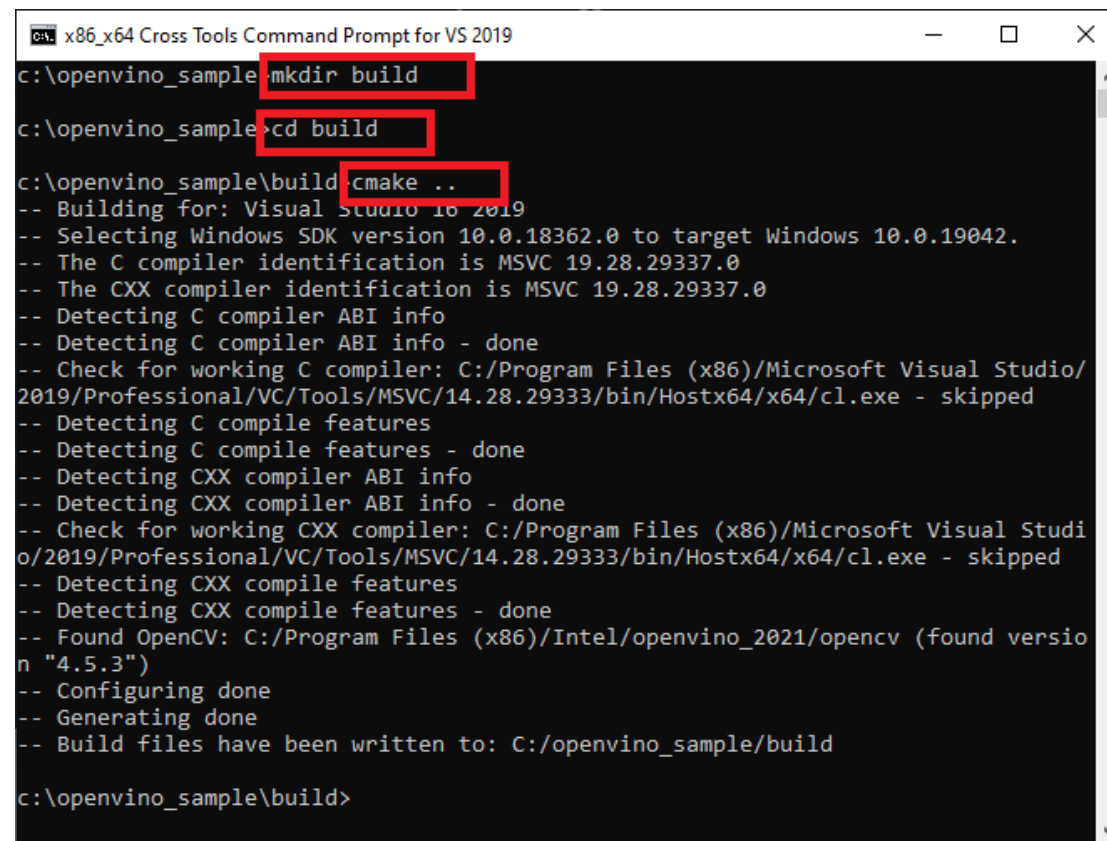
```
add_executable(ov_practice ov_practice1.cpp)
```

```
target_link_libraries(ov_practice ${OpenCV_LIBRARIES}  
${InferenceEngine_LIBRARIES})
```



# Create MS Visual Studio solution (.sln) file

- Create 'build' folder
- From 'build', run 'run cmake ..'
- Open "\*.sln" from Visual Studio
  - > devenv ov\_practice.sln



```
x86_x64 Cross Tools Command Prompt for VS 2019
c:\openvino_sample>mkdir build
c:\openvino_sample>cd build
c:\openvino_sample\build>cmake ..
-- Building for: Visual Studio 16 2019
-- Selecting Windows SDK version 10.0.18362.0 to target Windows 10.0.19042.
-- The C compiler identification is MSVC 19.28.29337.0
-- The CXX compiler identification is MSVC 19.28.29337.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Tools/MSVC/14.28.29333/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Tools/MSVC/14.28.29333/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: C:/Program Files (x86)/Intel/openvino_2021/opencv (found version "4.5.3")
-- Configuring done
-- Generating done
-- Build files have been written to: C:/openvino_sample/build
c:\openvino_sample\build>
```

Write some C++ code

# Include necessary files

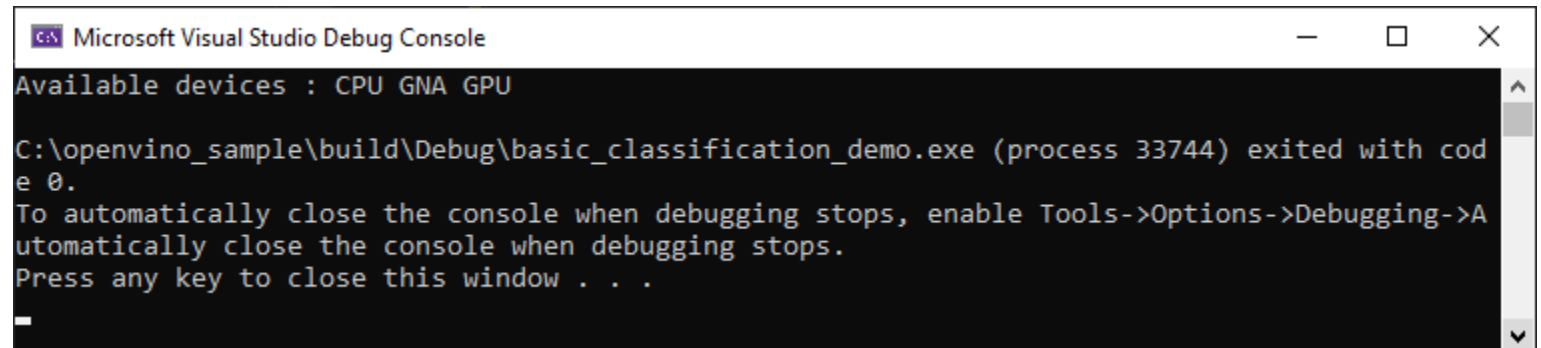
```
#include <iostream>
```

```
#include <opencv2/opencv.hpp>
```

```
#include <inference_engine.hpp>
```

# Check available devices for inference

```
int main(int argc, char *argv[])
{
    InferenceEngine::Core engine;
    auto devices = engine.GetAvailableDevices();
    std::cout << "Available devices :";
    for (const auto& device: devices) {
        std::cout << " " << device;
    }
    std::cout << std::endl;
}
```

A screenshot of the Microsoft Visual Studio Debug Console window. The window title is "Microsoft Visual Studio Debug Console". The output text is: "Available devices : CPU GNA GPU", followed by a message that the program exited with code 0, and instructions on how to automatically close the console when debugging stops. The text "Press any key to close this window . . ." is at the bottom. The console has a black background with white text and a scrollbar on the right.

```
Microsoft Visual Studio Debug Console
Available devices : CPU GNA GPU
C:\openvino_sample\build\Debug\basic_classification_demo.exe (process 33744) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

# Read model, load on CPU, prepare for inference

```
std::cout << "Reading...";  
  
// Read model from IR file (XML/BIN)  
InferenceEngine::CNNNetwork cnnNetwork = engine.ReadNetwork(  
    "c:\\opencvino_sample\\mobilenet-v2.xml",  
    "c:\\opencvino_sample\\mobilenet-v2.bin");  
  
std::cout << "Done\\n";  
  
// Load network to CPU device  
std::cout << "Loading...";  
InferenceEngine::ExecutableNetwork execNetwork = engine.LoadNetwork(cnnNetwork, "CPU");  
std::cout << "Done\\n";  
  
// Create Infer Request  
std::cout << "Creating infer request...";  
InferenceEngine::InferRequest inferRequest = execNetwork.CreateInferRequest();  
std::cout << "Done\\n";
```

# Have a look on the [model](#)

## Input

### Original Model

Image, name: data, shape: 1, 3, 224, 224, format: B, C, H, W, where:

- B - batch size
- C - channel
- H - height
- W - width

Channel order is BGR. Mean values: [103.94, 116.78, 123.68], scale value: 58.8235294117647.

## Output

### Original Model

Object classifier according to ImageNet classes, name: prob, shape: 1, 1000, output data format is B, C, where:

- B - batch size
- C - predicted probabilities for each class in a range [0, 1]

## XML:

```
<layer id="0" name="data" type="Parameter" version="opset1">
```

```
<data element_type="f32" shape="1, 3, 224, 224"/>
```

# Read image using OpenCV

```
std::string img_path = "c:\\opencvino_sample\\car.png";  
size_t size = 224;  
cv::Mat image = cv::imread(img_path);  
if (!image.ptr()) { std::cout << "Image load failed\n"; return -1; }  
cv::resize(image, image, cv::Size(size, size));  
std::cout << image.channels() << " " << image.rows << " " << image.cols << "\n";
```

Now we have image, 224x224x3 (H=224, W=224, C=3)

# Convert image buffer to model's layout

## What we have

- 1x3x224x224: No
- Float: No
- BGR: Yes
- Mean values: No
- Scale value: No

## Original Model

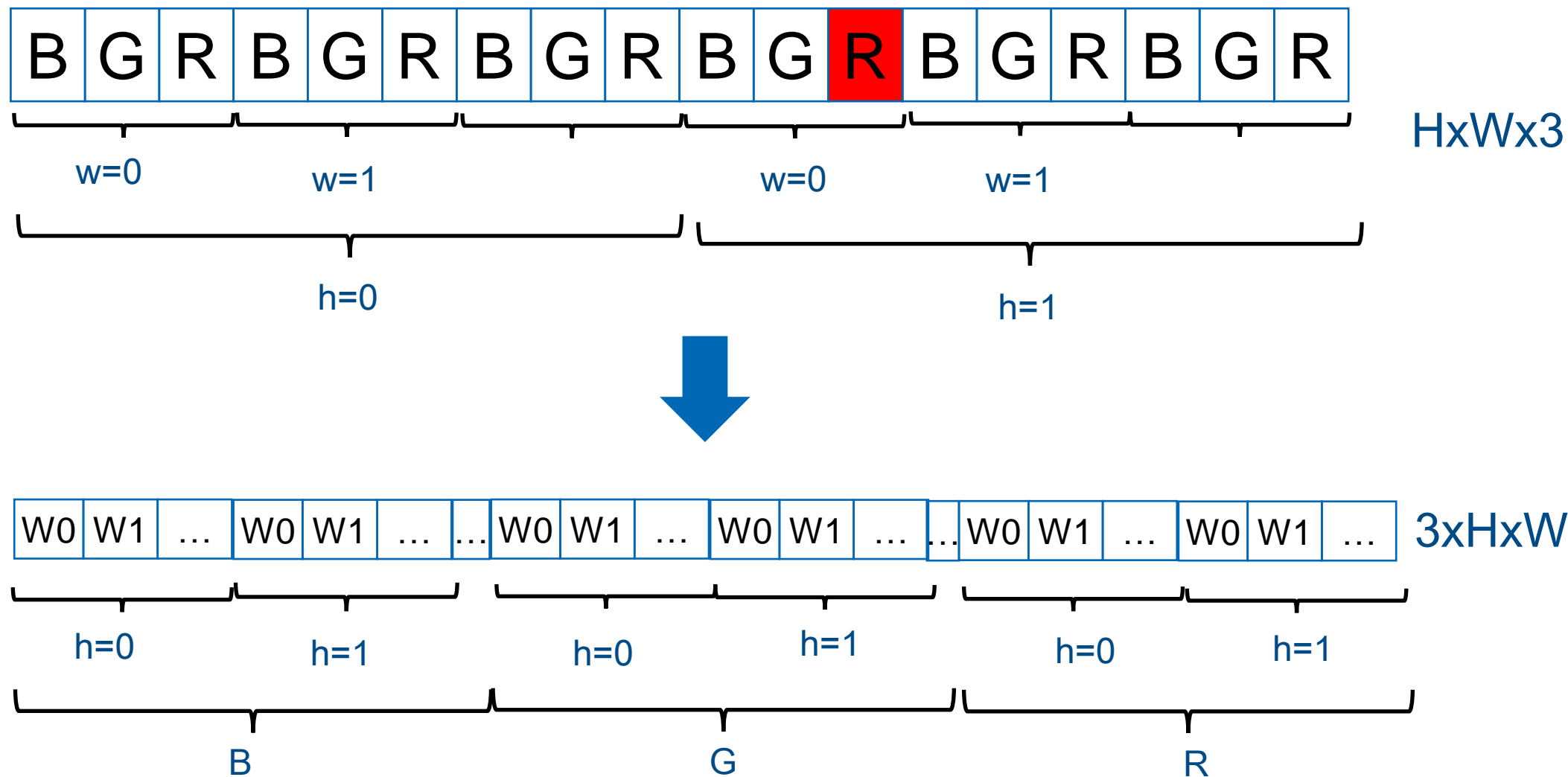
Image, name: `data`, shape: `1, 3, 224, 224`, format: `B, C, H, W`, where:

- `B` - batch size
- `C` - channel
- `H` - height
- `W` - width

Channel order is `BGR`. Mean values: `[103.94, 116.78, 123.68]`, scale value: `58.8235294117647`.



# Convert image buffer to model's layout



# Convert image buffer to model's layout (naive)

```
// Get pointer to allocated input buffer (already float32)
auto* blob = inferRequest.GetBlob("data")->as<InferenceEngine::MemoryBlob>();
float* blobData = blob->rmap().as<float*>();
// Do conversion manually
for (int h = 0; h < size; h++) {
    for (int w = 0; w < size; w++) {
        for (int c = 0; c < 3; c++) {
            int src_index = c + w * 3 + h * size * 3;
            int dst_index = c * size * size + h * size + w;
            blobData[dst_index] = image.ptr()[src_index];
        }
    }
}
```

# Apply normalization (mean/scales)

```
// Apply mean/scales
float means[3] = {103.94, 116.78, 123.68};
float scales[3] = {58.8235, 58.8235, 58.8235};
for (int c = 0; c < 3; c++) {
    for (int h = 0; h < size; h++) {
        for (int w = 0; w < size; w++) {
            int dst_index = c * size * size + h * size + w;
            blobData[dst_index] = (blobData[dst_index] - means[c]) / scales[c];
        }
    }
}
```

# Ready to start inference

```
std::cout << "Running inference...\n";

inferRequest.Infer();

std::cout << "Done\n";

// Get results array (1x1000)
auto* result = inferRequest.GetBlob("prob")->as<InferenceEngine::MemoryBlob>();
float* resultData = result->rmap().as<float*>();
```

# Get results

```
int bestId = -1;
float bestVal = 0;
for (int i = 0; i < 1000; i++) {
    if (bestVal < resultData[i]) {
        bestVal = resultData[i];
        bestId = i + 1;
    }
}
std::cout << "BEST: class = " << bestId << ", prob = " << bestVal << std::endl;
```

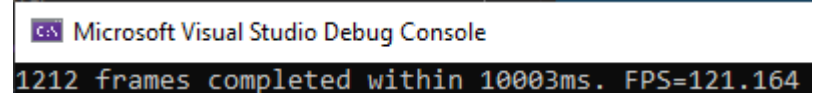
## Class names:

C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\demo\squeezenet1.1.labels

# Performance analysis

# Check some performance

```
#include <chrono>
...
int count = 0;
using namespace std::chrono;
auto start_time = steady_clock::now();
auto end_time = steady_clock::now() + seconds(10);
while (steady_clock::now() < end_time) {
    ...
    count++;
}
end_time = steady_clock::now();
auto diff = duration_cast<milliseconds>(end_time - start_time).count();
auto fps = 1000.f * count / diff;
std::cout << count << " frames completed within " << diff << "ms. FPS=" << fps << "\n";
```



Microsoft Visual Studio Debug Console

1212 frames completed within 10003ms. FPS=121.164

# Setup OpenVINO preprocessing

What we did manually

- Convert U8 to FP32
- Convert from {1,224,224,3} to {1,3,224,224}
- Apply mean/scale values
- Resize



# Use OpenVINO pre-processing API

```
// Modify inputs format before 'LoadNetwork'
auto input_info = cnnNetwork.getInputsInfo()["data"];
input_info->setLayout(InferenceEngine::Layout::NHWC);
input_info->setPrecision(InferenceEngine::Precision::U8);
input_info->getPreProcess().init(3);
input_info->getPreProcess().setVariant(InferenceEngine::MEAN_VALUE);
for (int c = 0; c < 3; c++) {
    input_info->getPreProcess()[c]->meanValue = means[c];
    input_info->getPreProcess()[c]->stdScale = scales[c];
}
```

# Setup OpenVINO preprocessing

// After 'LoadNetwork'

```
InferenceEngine::TensorDesc tDesc(InferenceEngine::Precision::U8,  
                                   {1, 3, size, size},  
                                   InferenceEngine::Layout::NHWC);  
  
auto blob = InferenceEngine::make_shared_blob<uint8_t>(tDesc, image.ptr());  
inferRequest.SetBlob("data", blob);  
  
...  
// Infer loop will look just like  
while (steady_clock::now() < end_time) {  
    inferRequest.Infer();  
    count++;  
}
```

 Microsoft Visual Studio Debug Console

1221 frames completed within 10007ms. FPS=122.015

# Mean/Scales to execution graph

```
c:\openvino_sample>python "%INTEL_OPENVINO_DIR%\deployment_tools\model_optimizer\mo.py" --input_model
public\mobilenet-v2\mobilenet-v2.caffemodel --mean_values [103.94,116.78,123.68]
--scale 58.8235 --model_name m2
```

```
// Modify inputs format before 'LoadNetwork'
auto input_info = cnnNetwork.getInputsInfo()["data"];
input_info->setLayout(InferenceEngine::Layout::NHWC);
input_info->setPrecision(InferenceEngine::Precision::U8);
//input_info->getPreProcess().init(3);
//input_info->getPreProcess().setVariant(InferenceEngine::MEAN_VALUE);
//for (int c = 0; c < 3; c++) {
//    input_info->getPreProcess()[c]->meanValue = means[c];
//    input_info->getPreProcess()[c]->stdScale = scales[c];
//}
```

```
C:\openvino_sample\build\Release\basic_classification_demo.exe
1237 frames completed within 10001ms. FPS=123.688
```

# Summary code snippet

```
auto cnnNetwork = engine.ReadNetwork("c:\\openvino_sample\\m2.xml");

// Set up preprocessing
const auto& input_info = cnnNetwork.getInputsInfo()["data"];
input_info->getPreProcess().setResizeAlgorithm(InferenceEngine::RESIZE_BILINEAR);
input_info->setLayout(InferenceEngine::Layout::NHWC);
input_info->setPrecision(InferenceEngine::Precision::U8);

// Load network, create InferRequest
InferenceEngine::ExecutableNetwork execNetwork = engine.LoadNetwork(cnnNetwork, "CPU");
InferenceEngine::InferRequest inferRequest = execNetwork.CreateInferRequest();

// Create memory blob and pass it to inferRequest
InferenceEngine::TensorDesc tDesc(InferenceEngine::Precision::U8,
                                   {1, 3, size, size}, InferenceEngine::Layout::NHWC);
auto blob = InferenceEngine::make_shared_blob<uint8_t>(tDesc, image.ptr());
inferRequest.SetBlob("data", blob)
```

# Performance considerations (1/2)

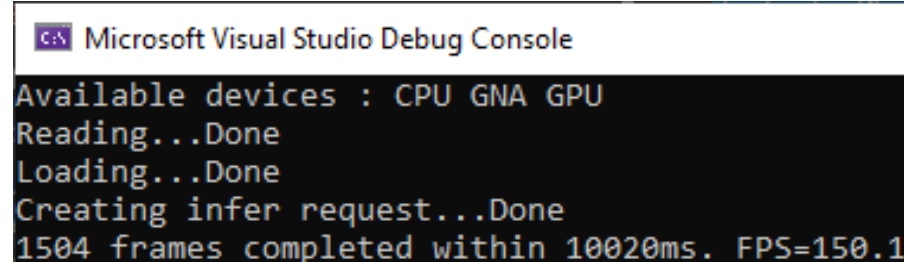
Execute several requests simultaneously

```
int numReq = 8;
while (steady_clock::now() < end_time) {
    for (int i = 0; i < numReq; i++) {
        inferRequests[i].StartAsync();
    }
    for (int i = 0; i < numReq; i++) {
        inferRequests[i].Wait();
        count++;
    } // TODO: Не хотелось бы ждать Request №8 для того, чтобы перезапустить №1
}
```

[https://docs.openvino.ai/2021.4/openvino\\_docs\\_optimization\\_guide\\_dldt\\_optimization\\_guide.html](https://docs.openvino.ai/2021.4/openvino_docs_optimization_guide_dldt_optimization_guide.html)

# Performance considerations (2/2)

THREADS	STREAMS	NUM REQUESTS	FPS
1	0	<Any>	60
1	AUTO	16	130
AUTO	0	<Any>	110
AUTO	32	32	160
AUTO	32	1	70



Microsoft Visual Studio Debug Console

```
Available devices : CPU GNA GPU
Reading...Done
Loading...Done
Creating infer request...Done
1504 frames completed within 10020ms. FPS=150.1
```

```
engine.SetConfig({{"CPU_THREADS_NUM", "4"}}, "CPU");
```

```
engine.SetConfig({{"CPU_THROUGHPUT_STREAMS", "CPU_THROUGHPUT_AUTO"}}, "CPU");
```

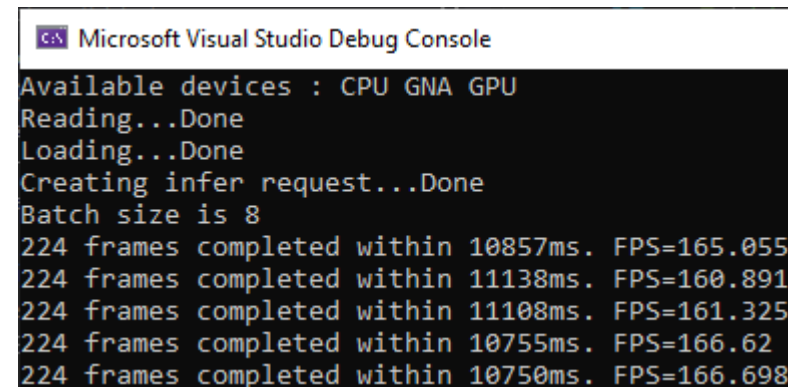
[https://docs.openvino.ai/2021.4/openvino\\_docs\\_IE\\_DG\\_supported\\_plugins\\_CPU.html](https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_supported_plugins_CPU.html)

# Batched inputs

- Transform model from 1x3x224x224 to 8x3x224x224

```
c:\openvino_sample>python "%INTEL_OPENVINO_DIR%\deployment_tools\model_optimizer\mo.py" --input_model
public\mobilenet-v2\mobilenet-v2.caffemodel --mean_values [103.94,116.78,123.68] --scale 58.8235 -b 8
```

```
size_t batch = input_info->getTensorDesc().getDims()[0];
std::cout << "Batch size is " << batch << "\n";
InferenceEngine::TensorDesc tDesc(InferenceEngine::Precision::U8,
                                   {batch, 3, size, size},
                                   InferenceEngine::Layout::NHWC);
std::vector<uint8_t> buffer(batch * 3 * size * size);
auto blob = InferenceEngine::make_shared_blob<uint8_t>(
    tDesc, batched_buffer.data());
```



Microsoft Visual Studio Debug Console

```
Available devices : CPU GNA GPU
Reading...Done
Loading...Done
Creating infer request...Done
Batch size is 8
224 frames completed within 10857ms. FPS=165.055
224 frames completed within 11138ms. FPS=160.891
224 frames completed within 11108ms. FPS=161.325
224 frames completed within 10755ms. FPS=166.62
224 frames completed within 10750ms. FPS=166.698
```

# Batched inputs (2/2)

THREADS	STREAMS	NUM REQUESTS	FPS (Batch = 1)	FPS (Batch = 2)
1	0	<Any>	60	60
1	AUTO	16	130	142
AUTO	0	<Any>	110	115
AUTO	32	32	160	167
AUTO	32	1	70	70



# Profiling - Performance Counters

Check performance counters (see benchmark\_app)

```
engine.SetConfig({{"PERF_COUNT", "YES"}}); // After IE creation
...<Load, do inference>
auto perf_map = inferRequest.GetPerformanceCounts();
for (const auto& it : perf_map) {
    if (it.second.status == InferenceEngine::InferenceEngineProfileInfo::EXECUTED) {
        std::cout << it.first << ": "; // layer name
        std::cout << std::to_string(it.second.cpu_uSec) << "mcs ";
        std::cout << it.second.exec_type << "\n";
    }
}
```

[https://docs.openvino.ai/2021.4/openvino\\_docs\\_MO\\_DG\\_Getting\\_Performance\\_Numbers.html](https://docs.openvino.ai/2021.4/openvino_docs_MO_DG_Getting_Performance_Numbers.html)

