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 >= 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

Install OpenVINO

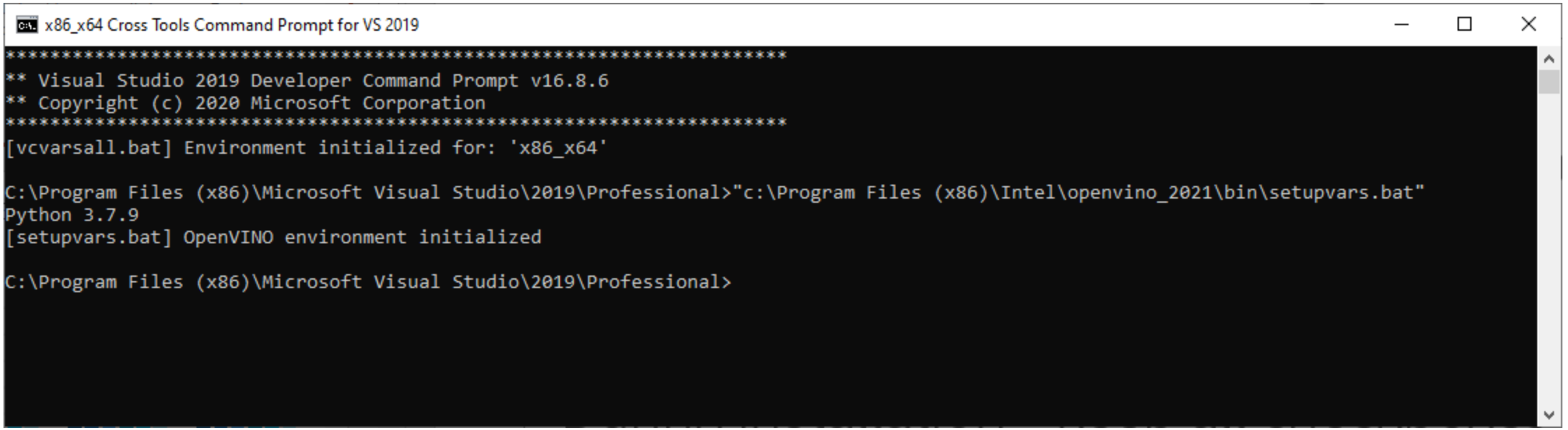
- Distribution: online and offline
- Version: 2021 4.2 LTS (This is the latest available version which is used for this practice)
- Type: offline
- During installation - keep all checkboxes on (Inference Engine, Model Optimizer, Open Model Zoo, OpenCV)

Training code

https://github.com/nosovmik/openvino_basic_practice

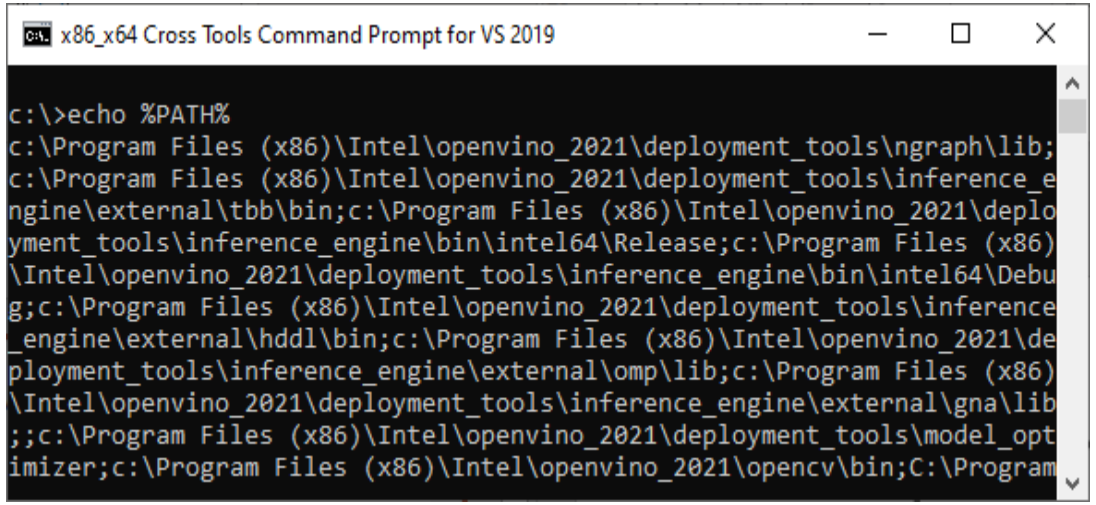
Setup environment variables

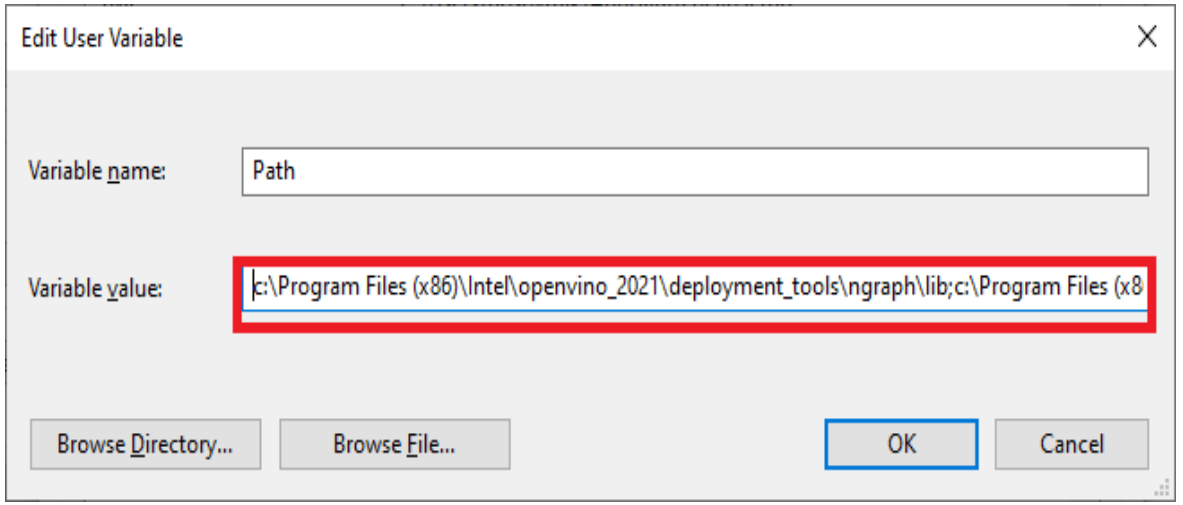
- Open Visual Studio command prompt
- Run “c:\Program Files (x86)\Intel\openvino_2021\bin\setupvars.bat”



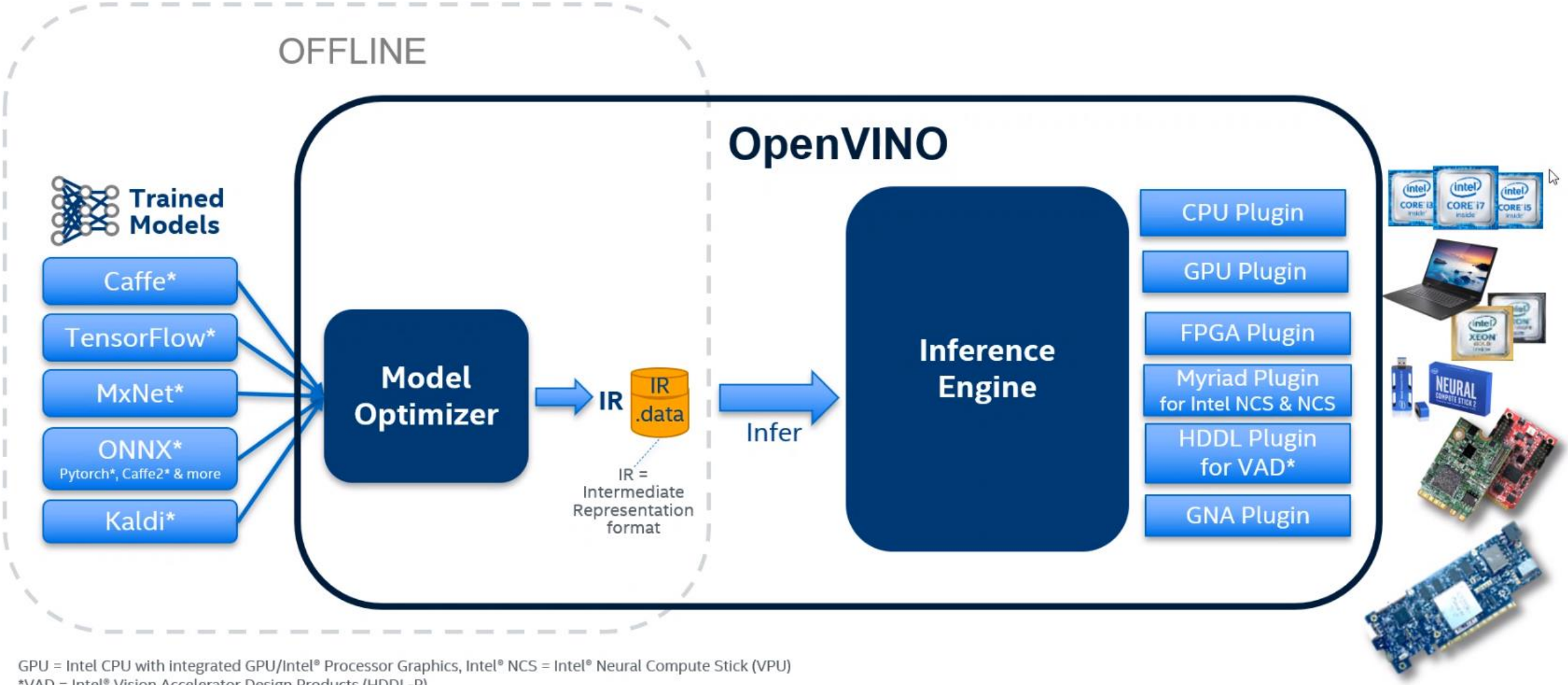
Setup environment variables (2, optional)

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





Obtain Pre-trained model



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
- 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

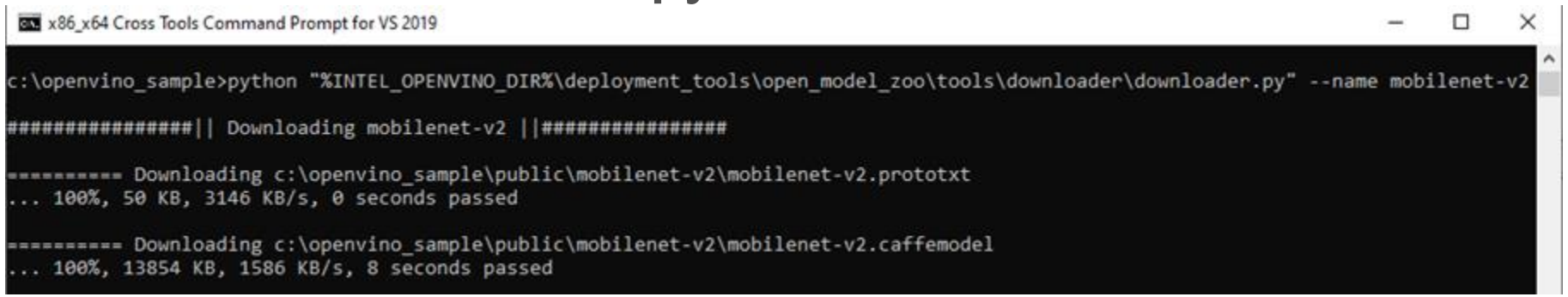
Install dependencies

```
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\downloader**downloader.py**" **--name mobilenet-v2**



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_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”

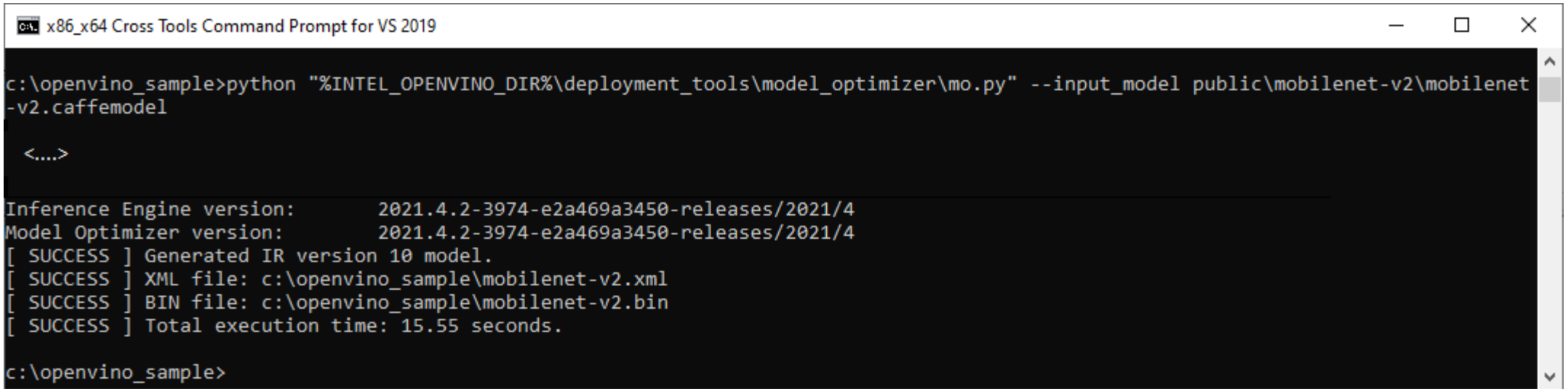
Install python dependencies

```
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**



Create CMake project

Create project

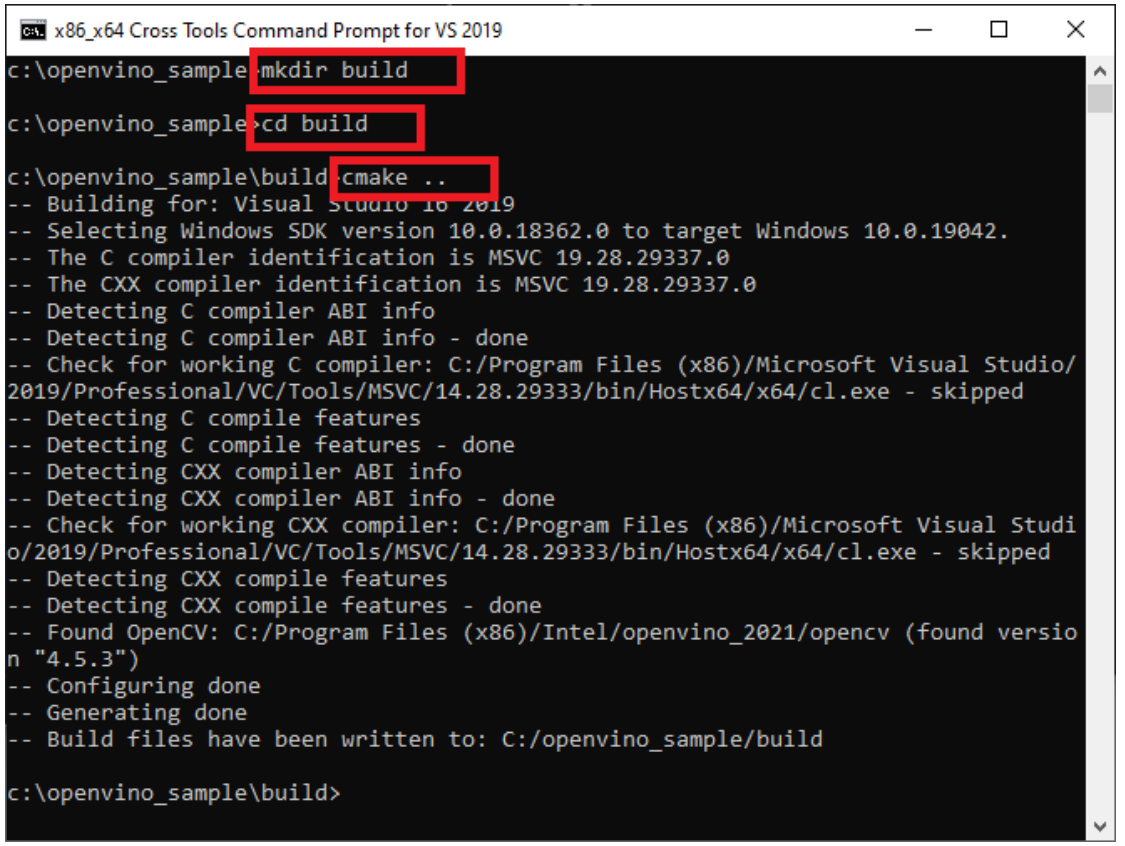
- Create empty files
  - “ov_practice1.cpp”
  - CMakeLists.txt

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



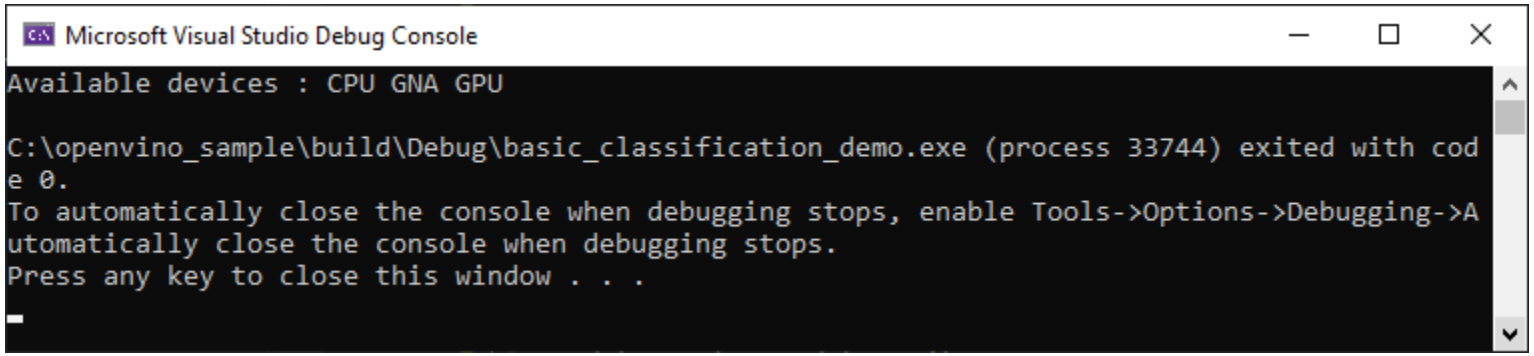
Write some C++ code

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;
}
```



Read model, load on CPU, prepare for inference

```
std::cout << "Reading...";
// Read model from IR file (XML/BIN)
InferenceEngine::CNNNetwork cnnNetwork = engine.ReadNetwork(
    "c:\\openvino_sample\\mobilenet-v2.xml",
    "c:\\openvino_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:\\openvino_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

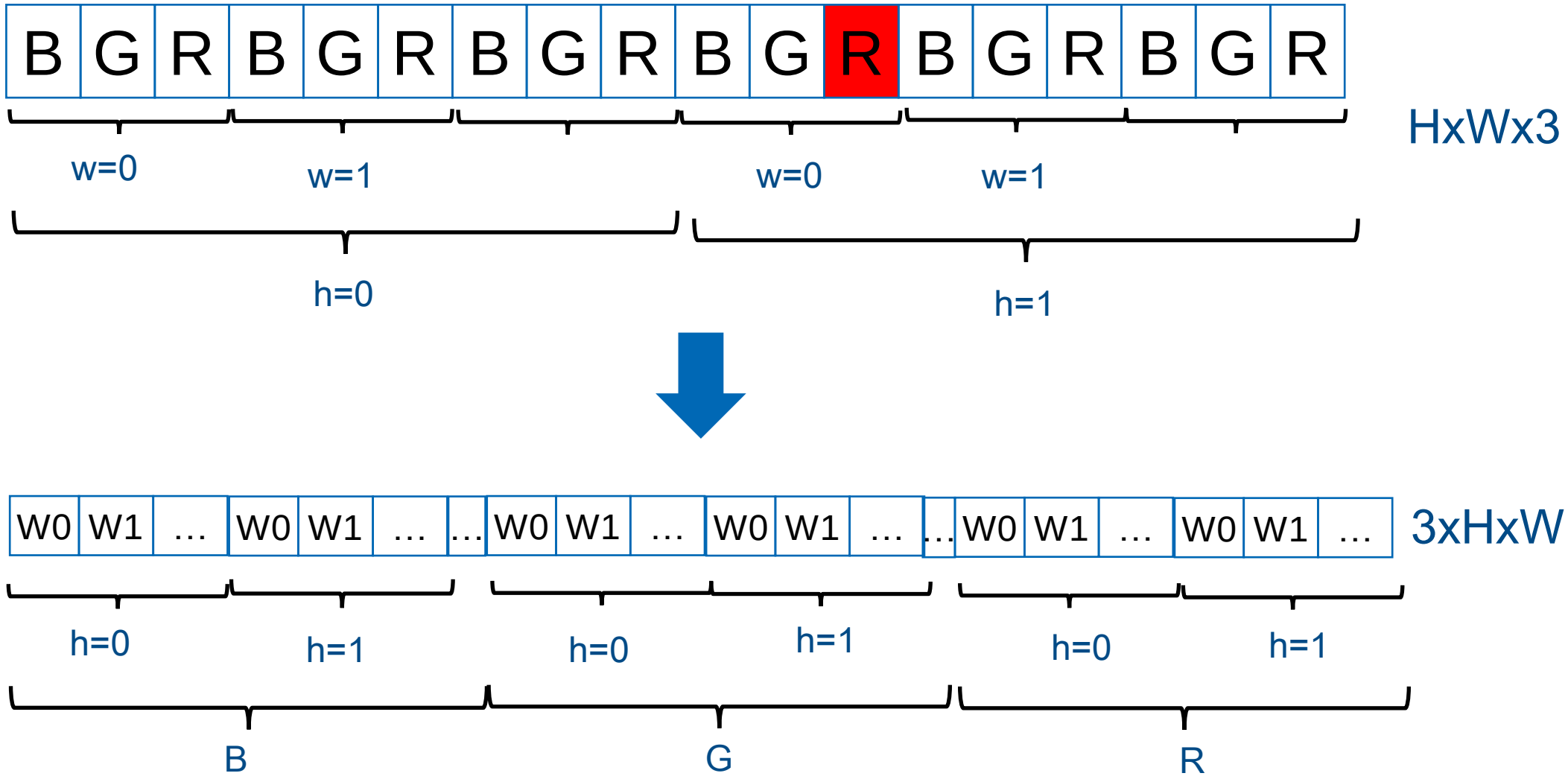
Original Model

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->rwmap().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;
```

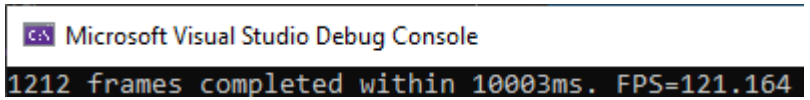
Class names:

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";
```



Setup OpenVINO preprocessing

What we did manually

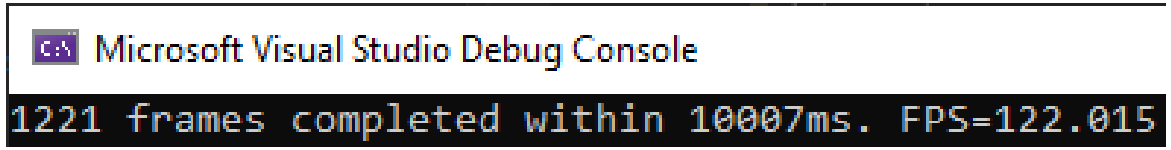
- Convert U8 to FP32
- Convert from {1,224,224,3} to {1,3,224,224}
- Apply mean/scale values
- Resize

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++;
}
```



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)

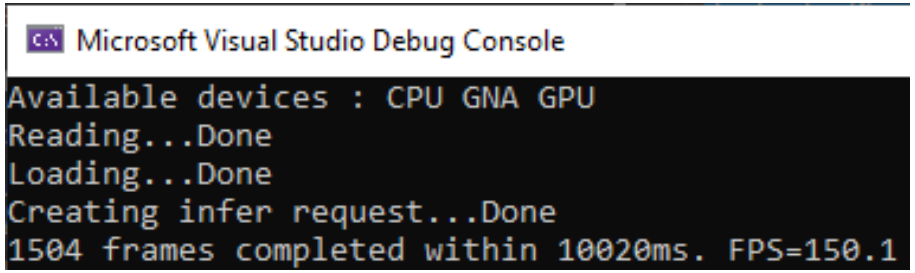
Execute several requests simultaneously

```
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

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




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

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

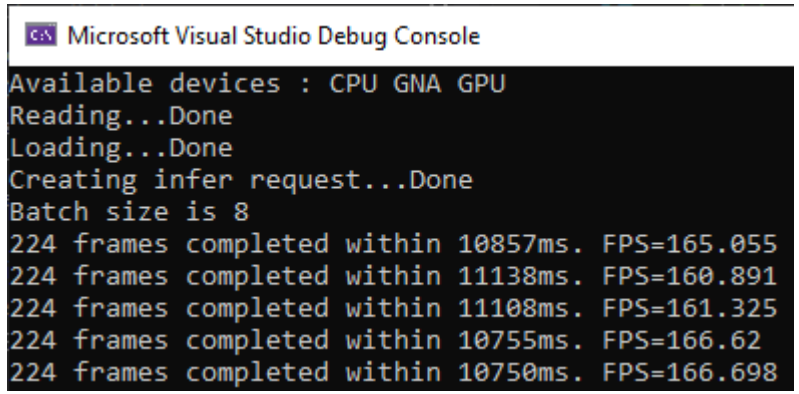
https://docs.openvino.ai/2021.4/openvino_docs_IE_DG_supported_plugins_CPU.html

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());
```



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


intel