Nizhny Novgorod State University
Institute of Information Technologies, Mathematics and Mechanics
Department of Computer software and supercomputer technologies

Educational course
«Modern methods and technologies of deep learning in computer vision»

Overview of the Intel Distribution of OpenVINO Toolkit

Supported by Intel

Vasiliev Evgeny

Content

- Goals
- Intel Distribution of OpenVINO Toolkit
- Components of the Intel Distribution of OpenVINO Toolkit
- Inference Engine
- DNN module of the OpenCV library
- Conclusion

Goals

- ***The goal*** is to study the features of the Intel Distribution of OpenVINO Toolkit for deep learning inference

INTEL DISTRIBUTION OF OPENVINO TOOLKIT

Intel Distribution of OpenVINO Toolkit (1)

- Intel Distribution of OpenVINO Toolkit is a toolkit for solving computer vision and deep learning tasks, it is developing by Intel
- The goal is to simplify using of various computer vision and deep learning algorithms on different Intel platforms
- Advantages:
  - High performance, minimal package size and few dependencies
  - High-performance inference of deep neural networks developed and trained using different deep learning libraries

Intel Distribution of OpenVINO Toolkit (2)

- License: EULA (also, there is an open-source OpenVINO toolkit licensed under Apache 2.0, https://01.org/openvinotoolkit)
- Documentation [https://docs.openvinotoolkit.org]
- Official page of the Intel Distribution of OpenVINO Toolkit [https://software.intel.com/en-us/openvino-toolkit]

Components of the Intel Distribution of OpenVINO Toolkit (1)

Deep Learning

Intel Deep Learning Deployment Toolkit

- **Model Optimizer** — Convert & Optimize
- IR
- **Inference Engine** — Optimized Inference

Model Optimizer → IR → Inference Engine

IR = Intermediate Representation file

Open Model Zoo

- **150+ Pretrained Models**
- **Demos**
- **Model Downloader**

Deep Learning Workbench

- **Benchmark App**
- **Accuracy Checker**
- **Post-Training Optimization Toolkit**

Traditional Computer Vision

Optimized Libraries & Code Samples

- **OpenCV**
- **Samples**

For Intel CPU & GPU/Intel Processor Graphics

Tools & Libraries

Increase Media/Video/Graphics Performance

- **Intel Media SDK** — Open Source version
- **OpenCL** — Drivers & Runtimes

For GPU/Intel Processor Graphics

Optimize Intel FPGA (Linux only)

- **FPGA Runtime Environment**
- **Bitstreams**

OS support: Windows 10 (64 bit), Ubuntu 18.04.3 LTS (64 bit), CentOS 7.4 (64 bit), macOS (64 bit)

Components of the Intel Distribution of OpenVINO Toolkit (2)

- **Deep Learning for Computer Vision**
  - Intel Deep Learning Deployment Toolkit (DLDT)
    - Model Optimizer is a tool for converting pre-trained deep models from the training framework format into the intermediate representation (IR) of the OpenVINO toolkit
    - Inference Engine is a component for high-performance inference of deep neural networks
  - Open Model Zoo is a public repository of pre-trained models for solving various problems, samples and demos
  - Deep Learning Workbench is a tool for calibrating models, measuring accuracy and benchmarking models

Components of the Intel Distribution of OpenVINO Toolkit (3)

- **Traditional Computer Vision**
  - OpenCV is a well-known and widely used computer vision library
- **Tools & Packages**
  - Set of tools for improving performance of processing graphics and video

Model Optimizer

- For high-performance inference of deep models using the OpenVINO toolkit you need convert them into ***the intermediate representation*** (IR)

- ***Model Optimizer*** is a tool for converting models from various formats to the intermediate representation
- Supported formats: ONNX, TensorFlow, Caffe, MXNet, Kaldi

- Model Optimizer documentation [https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html]

Inference Engine

- ***Inference Engine*** provides programming interface for deep learning inference on the following platforms:
  - Intel CPUs
  - Intel Processor Graphics
  - Intel FPGAs
  - Intel Movidius Neural Compute Stick, etc.
- Inference Engine supports heterogeneous inference of neural networks, which assumes the distribution of model layers between computational devices
- Inference Engine also supports multi-device inference, in which multiple requests are distributed among available devices

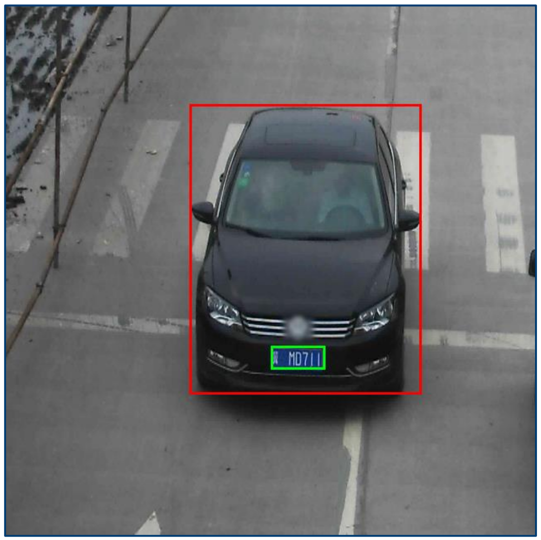
Open Model Zoo

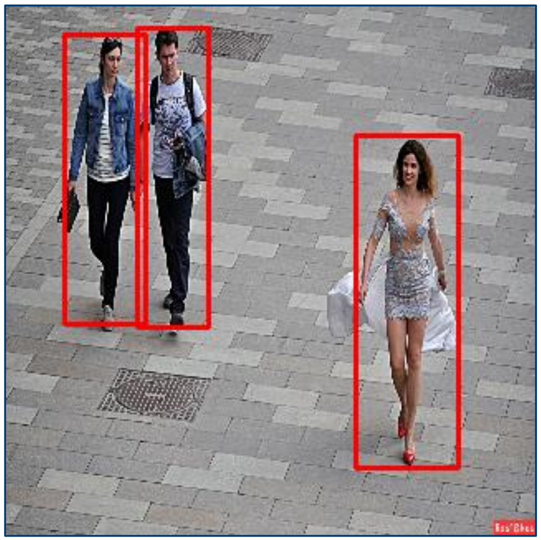
- ***Open Model Zoo***
  – Hundreds of trained deep models in various formats (public models and models trained by Intel engineers)
  – Library of samples and demo applications in C++ and Python
  – Model Downloader for downloading models and converting them to the intermediate representation

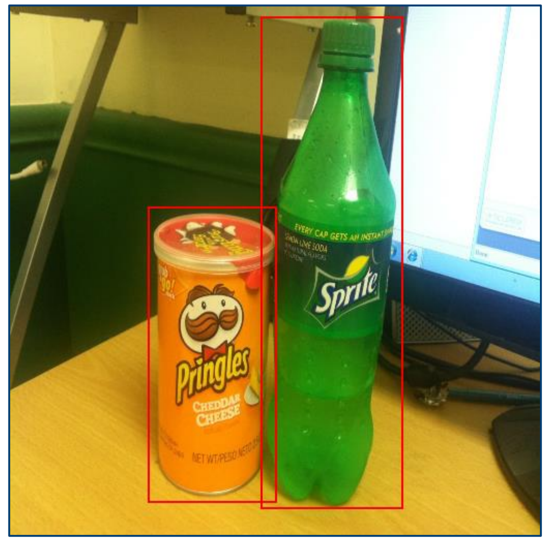
- Open Model Zoo [https://github.com/opencv/open_model_zoo]

Intel models (1)

- Object detection problem
  - Face detection and face recognition
  - Pedestrian detection
  - Vehicles detection (car model, color)
  - License plates detection and recognition



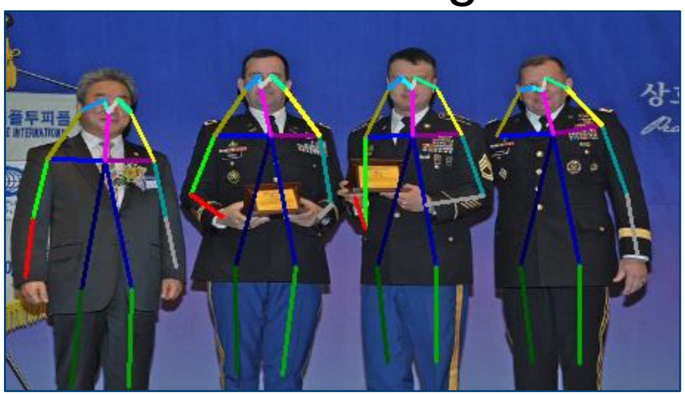


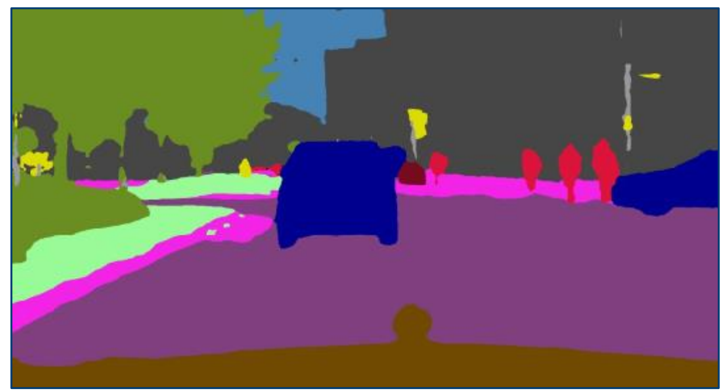


* Validation results for the selected Intel models [https://github.com/itlab-vision/openvino-dl-benchmark/blob/master/results/validation_results_intel_models.md].

Intel models (2)

- ❑ Object recognition problem
  - – Age and gender recognition models
  - – Emotions recognition
- ❑ Human pose estimation
- ❑ Segmentation problem
  - – Semantic segmentation
  - – Instance segmentation

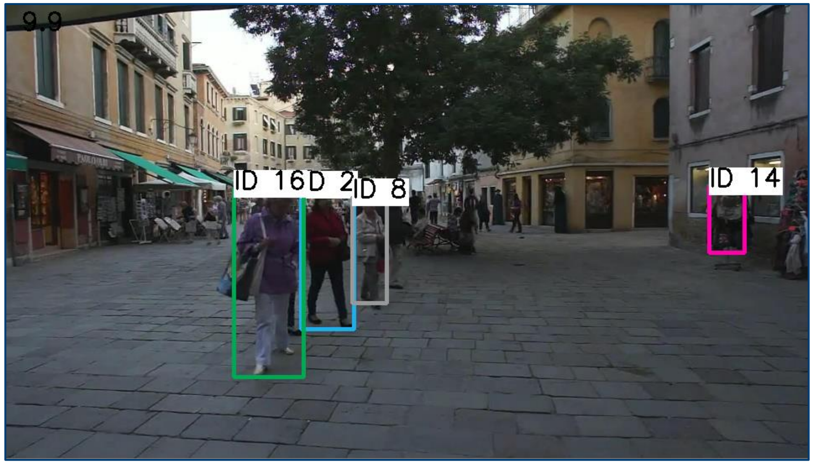


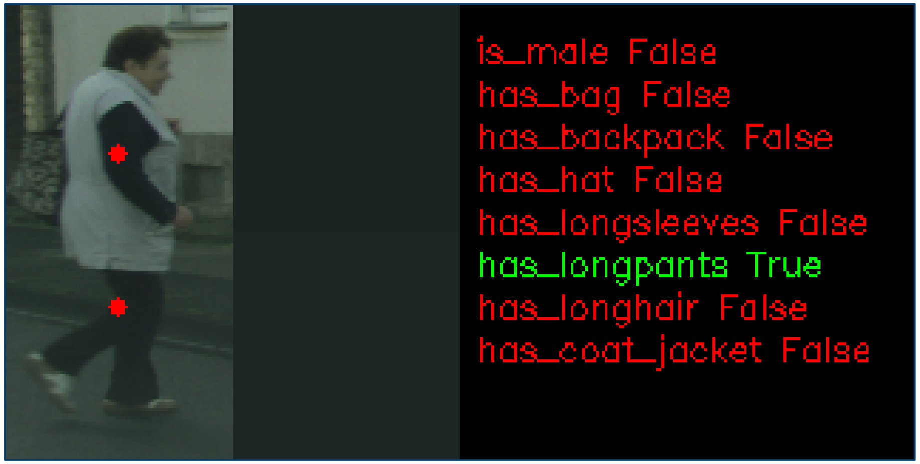


* Validation results for the selected Intel models [https://github.com/itlab-vision/openvino-dl-benchmark/blob/master/results/validation_results_intel_models.md].

Intel models (3)

- Object tracking
  - Pedestrian tracking
  - Person identification
- Image processing
  - Super-resolution

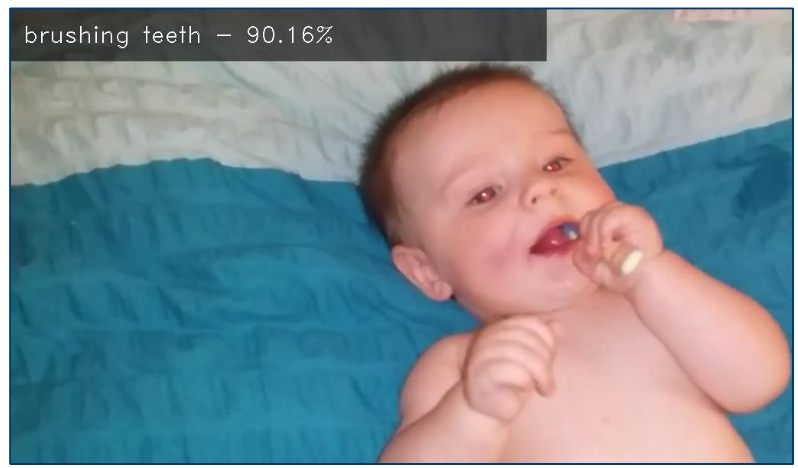




* Validation results for the selected Intel models [https://github.com/itlab-vision/openvino-dl-benchmark/blob/master/results/validation_results_intel_models.md].

Intel models (4)

- Text processing
  - Text detection
  - Text recognition
- Action recognition
  - Driver action recognition
  - General action recognition



* Validation results for the selected Intel models [https://github.com/itlab-vision/openvino-dl-benchmark/blob/master/results/validation_results_intel_models.md].

Samples and demos

- The OpenVINO toolkit contains samples and demos for inferring models from Open Model Zoo
- Demo applications solve specific computer vision problems
  - Smart Classroom Demo allows to detect students and recognize their actions
  - Security Barrier Camera Demo allows to detect vehicles and recognize vehicle license plates
- In the demo application, you can use different deep neural networks that provide the solution of the same task. Different architectures may provide different accuracy and speed
- Demo applications are available by the link [https://github.com/opencv/open_model_zoo/tree/master/demos]

Deep Learning Workbench

- ***Deep Learning Workbench*** is a toolkit to measure model accuracy and increase model performance
  - Deep Learning Workbench provides a graphical application for the OpenVINO components (Model Optimizer, Accuracy Checker Tool, etc.). It also allows to collect and visualize model inference statistics
  - Benchmark App is a tool for evaluating performance of deep models on various devices
  - Accuracy Checker Tool is a tool for measuring model accuracy on the provided dataset
  - Post-Training Optimization Toolkit is a toolkit for optimizing models by converting them into a hardware-friendly representation

OpenCV

- ***OpenCV*** is a library of computer vision, image processing and general-purpose numerical algorithms
- OpenCV is an open-source library licensed under BSD 3-Clause License, the library can be used in commercial projects
- OpenCV is developed in C/C++ programming language, and it provides programming interfaces for Python, Java, and other languages

- OpenCV [https://opencv.org]

OpenCV modules

- The selected modules of the OpenCV library:
  - ***core*** is a library core containing basic data types and math functions
  - ***imgproc*** is an image processing module (image filtering, drawing functions, color spaces)
  - ***video*** is a video analytics module
  - ***features2d*** is a module containing the implementation of keypoints detectors and descriptors
  - ***objdetect*** is a module for detecting objects using cascade classifiers
  - ***ml*** is a module of classical machine learning algorithms (clustering, regression, statistical classification)
  - ***dnn*** is a module for deep learning inference

Components discussed below

- Further, the components of the Intel Distribution of OpenVINO Toolkit providing the deep learning inference will be discussed
  - Inference Engine
  - DNN module of the OpenCV library
- Study sequence:
  - Purpose and features of the component
  - Application programming interface (API)
  - Example

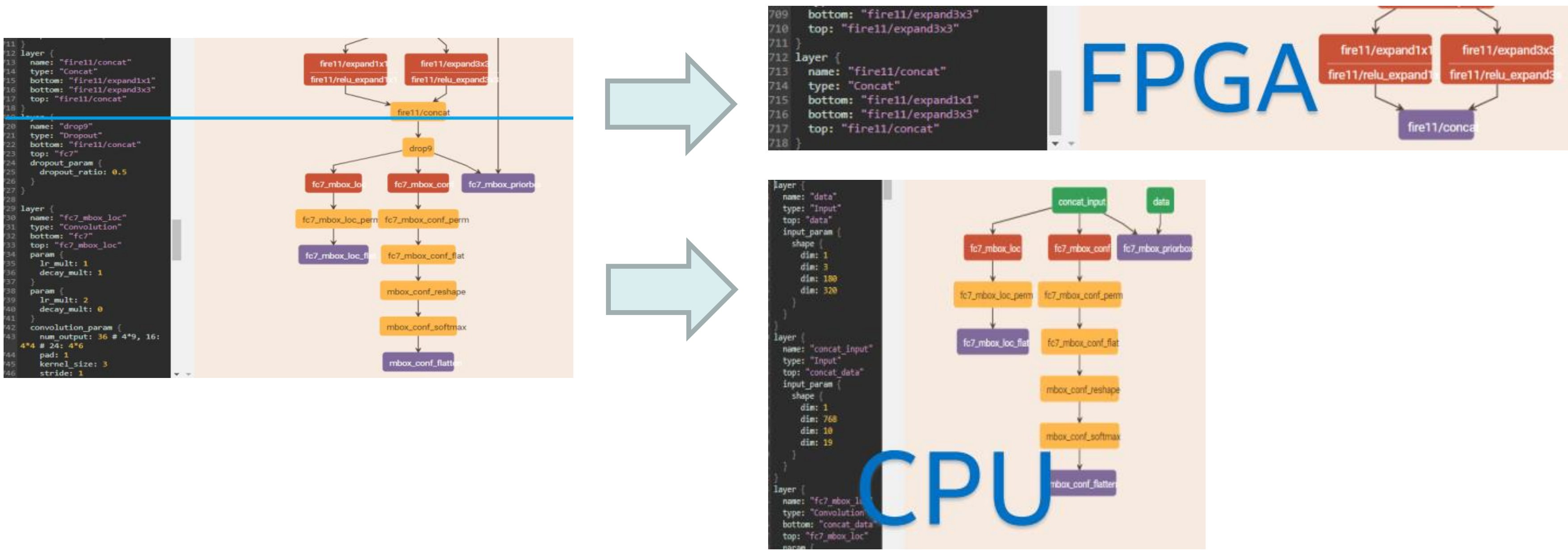
INFERENCE ENGINE

Inference Engine

- ***Inference Engine*** is a component that provides a high-level programming interface (C++, C, Python) for inference of deep neural networks in the intermediate representation on various Intel platforms due to the plugins
  - CPU (for Intel Xeon, Intel Core Processors, Intel Atom Processors, it is based on MKL-DNN)
  - GPU (for Intel Processor Graphics, it is based on clDNN (OpenCL))
  - FPGA (for Intel Programmable Acceleration Card)
  - MYRIAD (for Intel Movidius Neural Compute Stick, OpenCL)
  - Heterogeneous plugin
  - Multi-device plugin

Heterogeneous inference

- Inference Engine supports automatic splitting of a network inference between several devices, for example, CPU+GPU, CPU+FPGA



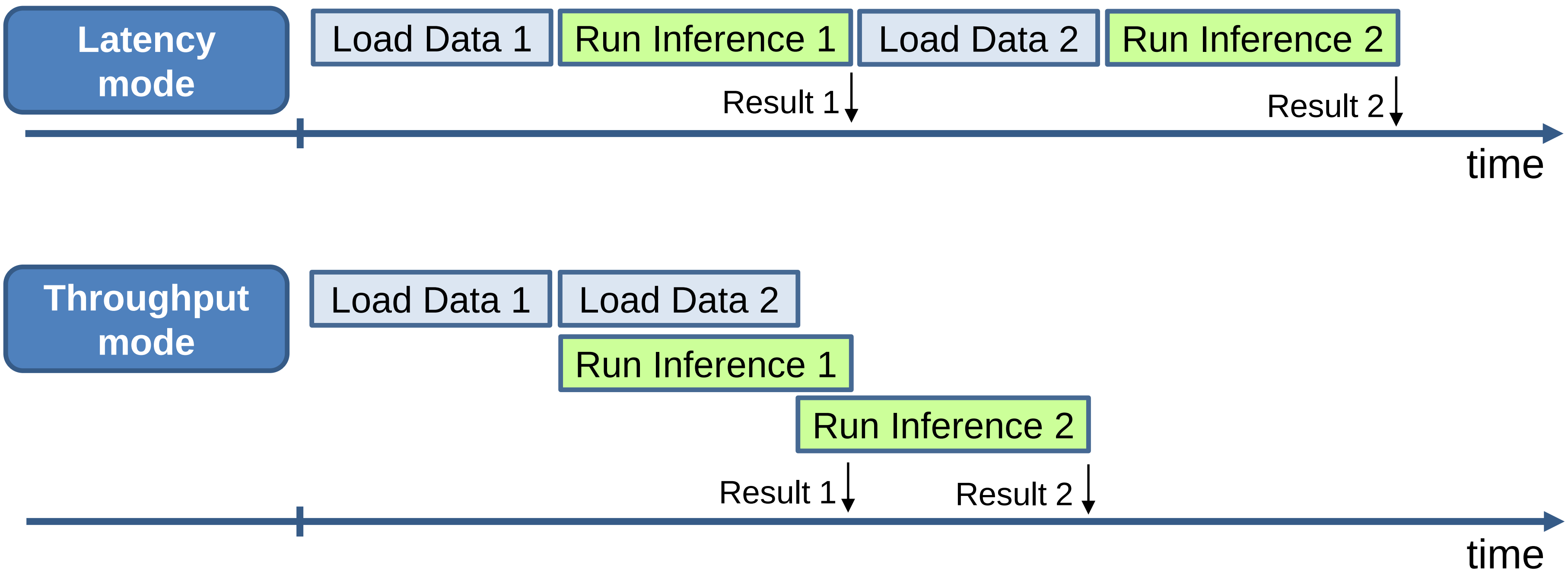
* Belova A. Introduction to the Intel Distribution of OpenVINO Toolkit. Tutorial “Object detection with deep learning: Performance optimization of neural network inference using the Intel OpenVINO toolkit” on PPAM 2019.

Inference modes (1)

- Inference Engine supports two inference modes:
  - ***Latency mode.*** Supposed the next inference request is executed after the completion of the previous one. This mode minimizes inference time of a single request due to parallelizing calculations during forward propagation
  - ***Throughput mode.*** Assumed constructing a queue of inference requests, several requests can be executed in parallel. This mode maximizes the number of completed requests (as a rule, minimizes a total time)

Inference modes (2)

- Illustration of different modes:

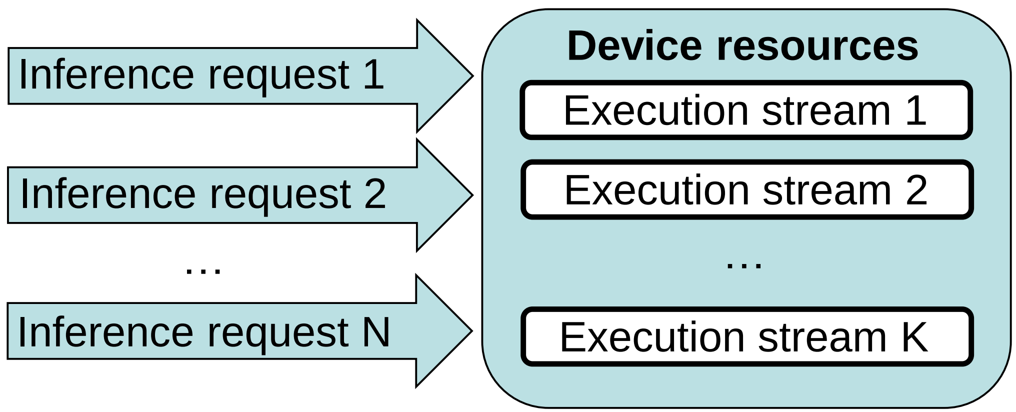


Latency mode

- Latency mode is used to minimize the time of a single inference request
- Speedup on CPUs is achieved due to the parallelism on shared-memory systems
- Parallelism on CPUs is implemented using ***threads***
- The number of threads is a parameter which can be set manually. By default, the optimal number of threads is equal to the number of physical cores

Throughput mode

- Throughput mode maximizes performance due to the parallel processing of several inference requests
- This mode allows you to increase overall throughput
- Throughput mode supposes the physical threads are divided into logical groups called ***streams***, in which calculations can be performed simultaneously and independently. Each stream processes one inference request



Inference Engine API for Python

- To infer deep neural networks using the OpenVINO toolkit, the following classes of the **`openvino.inference_engine`** module are used:
  - **`IECore`** represents an Inference Engine entity and allows you to manipulate with plugins using unified interfaces
  - **`IENetwork`** contains the information about the network model read from intermediate representation and allows you to manipulate with some model parameters such as output layers
  - **`ExecutableNetwork`** represents a network instance loaded to the plugin and ready for inference
  - **`InferRequest`** provides an interface to inference requests of **`ExecutableNetwork`** and serves to handle inference requests execution and to set and get output data

* Inference Engine Python API Overview [https://docs.openvinotoolkit.org/latest/_inference_engine_ie_bridges_python_docs_api_overview.html].

General outline of deep learning inference

1. Loading a deep neural network
2. Loading input images and converting to the format of the deep model input
3. Deep model inference
4. Output processing

1. Loading a deep model

- Initialize Inference Engine using **`IECore`**
- Create an object of the **`IENetwork`** class
- Load a deep model into the plugin and create an object for inference on the device using the **`load_network`** method of the **`IECore`** object

```
from openvino.inference_engine import IENetwork, IECore


configPath = 'path_to_model_config.xml'
weightsPath = 'path_to_model_weights.bin'

ie = IECore()
net = IENetwork(model = configPath, weights = weightsPath)
exec_net = ie.load_network(network = net, device_name = 'CPU')
```

2. Loading input images and converting to the format of the deep model input (1)

- As a rule, an input of the model is a 4-dimensional tensor of the size $[B \times C \times H \times W]$
  - B is a number of images
  - C is a number of channels for the image
  - H is an image height
  - W is an image width
- If we read images using the OpenCV library, then it is required to convert the tensor from the format **{BGRBGR…}** to the format **{RRR…GGG…BBB…}**, and change its shape in accordance with the model input shape

2. Loading input images and converting to the format of the deep model input (2)

- Read one or more images using the **`imread`** function
- Resize images using the **`resize`** function
- Reorder channels BGR -> RGB (if it is required) in images using the **`cvtColor`** function
- Reorder dimensions using the **`transpose`** function
- Expand tensor dimension if only one image is loaded using the **`expand_dims`** function of the **`numpy`** package

```
def prepare_image(imagePath, h, w):
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (w, h))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = image.transpose((2, 0, 1))
    blob = np.expand_dims(image, axis = 0)
    return blob
```

3. Deep model inference (1)

- There are synchronous (Sync API) and asynchronous programming interface (Async API) in the OpenVINO toolkit for deep learning inference:
  - ***Synchronous call*** blocks an application until the completion of inference request; it is not required to track the request completion. Synchronous API is used to implement the latency mode
  - ***Asynchronous call*** does not block an application until the completion of inference request; it is required to track the request completion. Asynchronous API can be used to implement both the latency and throughput modes

3. Deep model inference (2)

- ***Synchronous API***
  - To run the deep model inference in a synchronous mode, it is required to set the input tensor as the input of the deep model loaded to the plugin and call the **`infer()`** function

```
input_blob = next(iter(net.inputs))
out_blob = next(iter(net.outputs))
n, c, h, w = net.inputs[input_blob].shape

# Load, transpose, expand operations
blob = prepare_image(imagePath, h, w)

# Execute
output = exec_net.infer(inputs = {input_blob: blob})
output = output[out_blob]
```

3. Deep model inference (3)

- ***Asynchronous API***
  - To run the deep model inference in an asynchronous mode, it is required to set the input tensor as the input of the deep model loaded to the plugin, call the **`infer_async()`** function and wait for the request completion to extract the model output
  - There are two ways to check request completion:
    - Using the **`wait()`** function to check request status or wait for the request completion
    - Creating a callback function that will be called after the request completion

3. Deep model inference (4)

- ***Asynchronous API***
  - Sample of creating three inference requests and checking their completion using the **`wait()`** function

```
# Image loading similar to sync version
blobs = [blob1, blob2, blob3] # Images for independent requests

# Start async requests
for request_id in range(len(blobs)):
    exec_net.start_async(request_id = request_id,
        inputs = blobs[request_id])
# Wait for completing requests
for request_id in range(requests_counter):
    exec_net.requests[request_id].wait(-1)
# Copy results
list = [copy(exec_net.requests[request_id].outputs)
    for request_id in range(len(blobs))]
```

3. Deep model inference (5)

- To run several requests on CPU simultaneously, it is required to set the number of requests that can be simultaneously executed when the network is loaded to the device

```
exec_net = ie.load_network(network = net, device_name = 'CPU',
                           num_requests = YOUR_REQUESTS_NUMBER)
```

- You can get the available number of requests using the following command:

```
requests_number = len(exec_net.requests)
```

- If the number of input batches is greater than the available number of requests, then it is required to implement the queue of requests and set input batches from the queue to the pending requests

4. Output processing

- To process the network output, it is supposed the understanding of the format of the model output tensors

- The output tensors differ by task and model architecture

- For public models of Open Model Zoo, which solve the image classification problem on the ImageNet dataset, the output tensor shape is $[B \times 1000]$ as usual, where 1000 corresponds to the number of image categories

DNN MODULE OF THE OPENCV LIBRARY

DNN module of the OpenCV library

- DNN module of the OpenCV library supports the inference of deep neural networks on various hardware, including ARM processors

- DNN module submitted to OpenCV, starting with the version 3.3

- OpenCV supports models in the following formats: Caffe, TensorFlow, Darknet, ONNX
- Models trained using MXNet, Pytorch, and CNTK are supported by converting to the ONNX format

- OpenCV. Deep Neural Networks (dnn module) [https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html]

DNN backends

- OpenCV supports several ***backends*** for deep learning inference:
  - OpenCV (the easiest backend)
  - Inference Engine (the high-performance backend)
  - Halide [https://halide-lang.org]. Halide is a programming language that is designed to develop high-performance applications for image and array processing

DNN targets

- The parameter describing the device for deep learning inference is called the ***target***
- The DNN module supports the following targets with various backends:
  - CPU – OpenCV, Inference Engine, Halide
  - OpenCL – OpenCV, Inference Engine, Halide
  - OpenCL FP16 – OpenCV, Inference Engine
  - Intel Movidius Neural Compute Stick – Inference Engine
  - FPGA – Inference Engine

General outline of deep learning inference

1. Loading a deep model
2. Loading input images
3. Converting images to the deep model input
4. Deep model inference
5. Output processing

1. Loading a deep model (1)

- Deep model usually consists of one or two files, the first one corresponds to the model architecture, the second one contains model weights
- To read a model, the **`readNet`** function is used, its parameters is a path (or two paths) to the model file, in any order
- Example of loading the model in the Caffe format, setting the backend and the target device by calling the **`setPreferableBackend`** and **`setPreferableTarget`** methods:

```
model = "deploy.prototxt"
weights = "bvlc_alexnet.caffemodel"

net = cv2.dnn.readNet(model, config)
net.setPreferableBackend(backend)
net.setPreferableTarget(target)
```

1. Loading a deep model (2)

- Available backends:

```
backend = cv2.dnn.DNN_BACKEND_DEFAULT
backend = cv2.dnn.DNN_BACKEND_HALIDE
backend = cv2.dnn.DNN_BACKEND_INFERENCE_ENGINE
backend = cv2.dnn.DNN_BACKEND_OPENCV
```

- Available targets:

```
target = cv2.dnn.DNN_TARGET_CPU
target = cv2.dnn.DNN_TARGET_OPENCL
target = cv2.dnn.DNN_TARGET_OPENCL_FP16
target = cv2.dnn.DNN_TARGET_MYRIAD
```

2. Loading images

- The image is loaded using the **`imread`** function, the parameter is the path to the image

```
image = cv2.imread(imagePath)
```

3. Converting images to the deep model input (1)

- As a rule, an input of the model is a 4-dimensional tensor of the size $[B \times C \times H \times W]$
  - B is a number of images
  - C is a number of channels for the image
  - H is an image height
  - W is an image width
- If we read images using the OpenCV library, then it is required to convert the tensor from the format **`{BGRBGR…}`** to the format **`{RRR…GGG…BBB…}`**, and change its shape in accordance with the model input shape

3. Converting images to the deep model input (2)

- To convert a single image, the **`blobFromImage`** function is used
- Sample of converting an image into the input format of the deep model is shown below

```
scalefactor = 1.0
# mean intensity
mean = (104, 117, 123)
# input size
size = (224, 224)

blob = cv2.dnn.blobFromImage(image, scalefactor = 1.0, size,
    mean, swapRB = True)
```

4. Deep model inference

- To run the deep model inference, it is required to set the input tensor as the input of the deep model and execute the **`forward()`** method

```
net.setInput(blob)
preds = net.forward()
```

5. Output processing

- To process the network output, it is supposed the understanding of the format of the model output tensors
- For public models of Open Model Zoo which solve the image classification problem on the ImageNet dataset, the output tensor shape is $[B \times 1000]$ as usual, where 1000 corresponds to the number of image categories

```
# output shape [1, 1000] for one input image
prob = preds[0]
classid = np.argmax(prob)
classprob = np.max(prob)
print('Class {}, probability {}'.format(classid, classprob))
```

Conclusion

- Components of the Intel Distribution of OpenVINO Toolkit were overviewed
- Possible ways to implement deep learning inference using Inference Engine and OpenCV were described
- Solving the practical tasks of the course, it is supposed to use one of the considered components
- Tutorials for solving the practical tasks prepared by the authors of the course are based on the Inference Engine component

Literature

- Intel Distribution of OpenVINO Toolkit [https://software.intel.com/en-us/openvino-toolkit].
- OpenVINO documentation website [https://docs.openvinotoolkit.org].
- OpenVINO – Open Sourced version [01.org/openvinotoolkit].
- OpenVINO performance topics [https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Intro_to_Performance.html].
- CPU Inference Performance Boost with “Throughput” Mode in the Intel Distribution of OpenVINO Toolkit [https://www.intel.ai/cpu-inference-performance-boost-openvino].
- OpenCV [https://opencv.org].
- Open Model Zoo [https://github.com/opencv/open_model_zoo].

Authors

- **Turlapov Vadim Evgenievich,** Dr., Prof., department of computer software and supercomputer technologies
  vadim.turlapov@itmm.unn.ru
- **Vasiliev Engeny Pavlovich,** lecturer, department of computer software and supercomputer technologies
  evgeny.vasiliev@itmm.unn.ru
- **Getmanskaya Alexandra Alexandrovna,** lecturer, department of computer software and supercomputer technologies
  alexandra.getmanskaya@itmm.unn.ru
- **Kustikova Valentina Dmitrievna**
  Phd, assistant professor, department of computer software and supercomputer technologies
  valentina.kustikova@itmm.unn.ru