OpenVINO™

# OpenVINO Tutorial

Suk-Hwan Lee

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

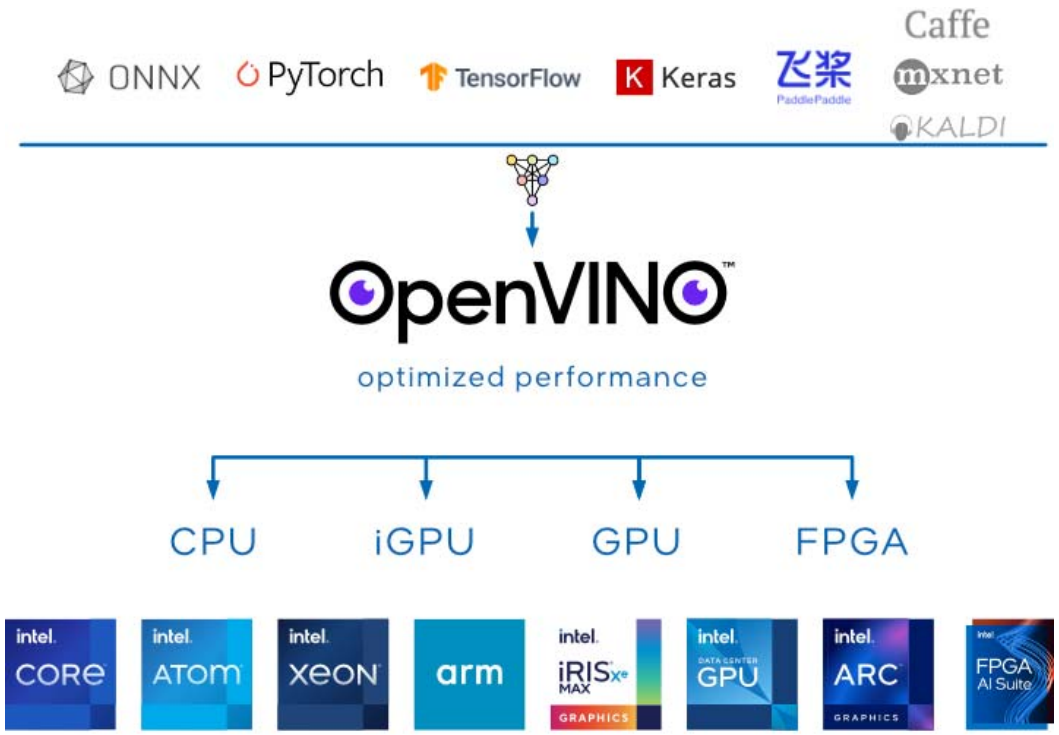
# References

OpenVINO tutorial; **OpenVINO 2023.0**

https://docs.openvino.ai/2023.0/home.html

Documentation : https://docs.openvino.ai/2023.0/documentation.html
Tutorials : https://docs.openvino.ai/2022.3/learn_openvino.html
**github** : https://github.com/openvinotoolkit/openvino_notebooks

LearnOpenCV : Introduction to Intel OpenVINO
https://learnopencv.com/introduction-to-intel-openvino-toolkit/




**BuildOpenCV4OpenVINO**
https://github.com/opencv/opencv/wiki/BuildOpenCV4OpenVINO#installing-on-windows

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## Get Started

Welcome to OpenVINO! This guide introduces installation and learning materials for Intel® Distribution of OpenVINO™ toolkit. The guide walks through the following steps:

| Quick Start Example | Install OpenVINO | Learn OpenVINO |
| --- | --- | --- |
| Estimate image depth (no install required) | View options | Tutorials, basics, and features |

## 1. Quick Start Example (No Installation Required)

Try out OpenVINO's capabilities with this quick start example that estimates depth in a scene using an OpenVINO monodepth model. Run the example in a Jupyter Notebook inside your web browser to quickly see how to load a model, prepare an image, inference the image, and display the result.



## 2. Install OpenVINO

See the installation overview page for options to install OpenVINO and set up a development environment on your device.

## 3. Learn OpenVINO

OpenVINO provides a wide array of examples and documentation showing how to work with models, run inference, and deploy applications. Step through the sections below to learn the basics of OpenVINO and explore its advanced optimization features. For further details, visit OpenVINO documentation.

OpenVINO users of all experience levels can try Intel® DevCloud, a free web-based service for developing, testing, and running OpenVINO applications on an online cluster of the latest Intel® hardware.

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## OpenVINO Basics

Learn the basics of working with models and inference in OpenVINO. Begin with "Hello World" Interactive Tutorials that show how to prepare models, run inference, and retrieve results using the OpenVINO API. Then, explore other examples from the Open Model Zoo and OpenVINO Code Samples that can be adapted for your own application.

### Interactive Tutorials - Jupyter Notebooks ¶

Start with interactive Python tutorials that show the basics of model inferencing, the OpenVINO API, how to convert models to OpenVINO format, and more.

- Hello Image Classification - Load an image classification model in OpenVINO and use it to apply a label to an image
- OpenVINO Runtime API Tutorial - Learn the basic Python API for working with models in OpenVINO
- Convert TensorFlow Models to OpenVINO
- Convert PyTorch Models to OpenVINO

### OpenVINO Code Samples

View sample code for various C++ and Python applications that can be used as a starting point for your own application. For C++ developers, step through the Get Started with C++ Samples to learn how to build and run an image classification program that uses OpenVINO's C++ API.

### Integrate OpenVINO With Your Application

Learn how to use the OpenVINO API to implement an inference pipeline in your application.

### Model Compression and Quantization

Use OpenVINO's model compression tools to reduce your model's latency and memory footprint while maintaining good accuracy.

- Tutorial - OpenVINO Post-Training Model Quantization
- Tutorial - Quantization-Aware Training in TensorFlow with OpenVINO NNCF
- Tutorial - Quantization-Aware Training in PyTorch with NNCF
- Model Optimization Guide

### Automated Device Configuration

OpenVINO's hardware device configuration options enable you to write an application once and deploy it anywhere with optimal performance.

- Increase application portability with Automatic Device Selection (AUTO)
- Perform parallel inference across processors with Multi-Device Execution (MULTI)
- Efficiently split inference between hardware cores with Heterogeneous Execution (HETERO)

### Flexible Model and Pipeline Configuration

Pipeline and model configuration features in OpenVINO Runtime allow you to easily optimize your application's performance on any target hardware.

- Automatic Batching performs on-the-fly grouping of inference requests to maximize utilization of the target hardware's memory and processing cores.
- Performance Hints automatically adjust runtime parameters to prioritize for low latency or high throughput
- Dynamic Shapes reshapes models to accept arbitrarily-sized inputs, increasing flexibility for applications that encounter different data shapes
- Benchmark Tool characterizes model performance in various hardware and pipeline configurations

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino



## Installing Intel® Distribution of OpenVINO™ Toolkit

Intel® Distribution of OpenVINO™ Toolkit is a comprehensive toolkit for developing applications and solutions based on deep learning tasks, such as computer vision, automatic speech recognition, natural language processing, recommendation systems, and more. It provides high-performance and rich deployment options, from edge to cloud. Some of its advantages are:

- Enables CNN-based and transformer-based deep learning inference on the edge or cloud.
- Supports various execution modes across Intel® technologies: Intel® CPU, Intel® Integrated Graphics, Intel® Discrete Graphics, Intel® Neural Compute Stick 2, and Intel® Vision Accelerator Design with Intel® Movidius™ VPUs.
- Speeds time-to-market via an easy-to-use library of computer vision functions and pre-optimized kernels.
- Compatible with models from a wide variety of frameworks, including TensorFlow, PyTorch, PaddlePaddle, ONNX, and more.

## Install OpenVINO

Check out the OpenVINO Download Page

OpenVINO installation package is distributed as two options: OpenVINO Runtime and OpenVINO Development Tools.

- **OpenVINO Runtime** contains the core set of libraries for running machine learning model inference on processor devices.
- **OpenVINO Development Tools** is a set of utilities for working with OpenVINO and OpenVINO models. It includes the following tools:
  - OpenVINO Runtime
  - Model Optimizer
  - Post-Training Optimization Tool
  - Benchmark Tool
  - Accuracy Checker and Annotation Converter
  - Model Downloader and other Open Model Zoo tools

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## OpenVINO Runtime

### Installing OpenVINO Runtime with Anaconda Package Manager

1. Set up the Anaconda environment (Python 3.7 used as an example):

```
conda create --name py37 python=3.7
```

```
conda activate py37
```

2. Update it to the latest version:

```
conda update --all
```

3. Install the OpenVINO Runtime package:

```
conda install -c conda-forge openvino=2022.3.1
```

Congratulations! You have finished installing OpenVINO Runtime.

### Uninstalling OpenVINO™ Runtime

Once OpenVINO Runtime is installed via Conda, you can remove it using the following command, with the proper OpenVINO version number:

```
conda remove openvino=2022.3.1
```

## Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

### OpenVINO Development Tools

# Install OpenVINO™ Development Tools

OpenVINO Development Tools is a set of utilities that make it easy to develop and optimize models and applications for OpenVINO. It provides the following tools:

- Model conversion API
- Benchmark Tool
- Accuracy Checker and Annotation Converter
- Post-Training Optimization Tool
- Model Downloader and other Open Model Zoo tools

The instructions on this page show how to install OpenVINO Development Tools. If you are a Python developer, it only takes a few simple steps to install the tools with PyPI. If you are developing in C++, OpenVINO Runtime must be installed separately before installing OpenVINO Development Tools.

In both cases, Python 3.7 - 3.11 needs to be installed on your machine before starting.

**Note**

From the 2022.1 release, the OpenVINO™ Development Tools can only be installed via PyPI.

## Installation into an Existing Environment with the Source Deep Learning Framework

To install OpenVINO Development Tools (see the Install the Package section of this article) into an existing environment with the deep learning framework used for the model training or creation, run the following command:

```
pip install openvino-dev
```

## Installation in a New Environment

If you do not have an environment with a deep learning framework for the input model or you encounter any compatibility issues between OpenVINO and your version of deep learning framework, you may install OpenVINO Development Tools with validated versions of frameworks into a new environment.

### Step 1. Set Up Python Virtual Environment

Create a virtual Python environment to avoid dependency conflicts. To create a virtual environment, use the following command:

Linux and macOS | Windows

```
python3 -m venv openvino_env
```

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## OpenVINO Development Tools

### Step 2. Activate Virtual Environment

Activate the newly created Python virtual environment by issuing this command:

Linux and macOS | Windows

```
source openvino_env/bin/activate
```

**Important**

The above command must be re-run every time a new command terminal window is opened.

### Step 3. Set Up and Update PIP to the Highest Version

Make sure *pip* is installed in your environment and upgrade it to the latest version by issuing the following command:

```
python -m pip install --upgrade pip
```

### Step 4. Install the Package

To install and configure the components of the development package together with validated versions of specific frameworks, use the commands below.

```
pip install openvino-dev[extras]
```

where the **extras** parameter specifies the source deep learning framework for the input model and is one or more of the following values separated with "," : **caffe**, **kaldi**, **mxnet**, **onnx**, **pytorch**, **tensorflow**, **tensorflow2**.

For example, to install and configure dependencies required for working with TensorFlow 2.x and ONNX models, use the following command:

```
pip install openvino-dev[tensorflow2,onnx]
```

**Note**

Model conversion API support for TensorFlow 1.x environment has been deprecated. Use the **tensorflow2** parameter to install a TensorFlow 2.x environment that can convert both TensorFlow 1.x and 2.x models. If your model isn't compatible with the TensorFlow 2.x environment, use the *tensorflow* parameter to install the TensorFlow 1.x environment. The TF 1.x environment is provided only for legacy compatibility reasons.

For more details on the openvino-dev PyPI package, see pypi.org .

# Get Started

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## OpenVINO Development Tools

### Step 5. Test the Installation

To verify the package is properly installed, run the command below (this may take a few seconds):

```
mo -h
```

You will see the help message for **mo** if installation finished successfully. If you get an error, refer to the Troubleshooting Guide for possible solutions.

Congratulations! You finished installing OpenVINO Development Tools with C++ capability. Now you can start exploring OpenVINO's functionality through example C++ applications. See the "What's Next?" section to learn more!

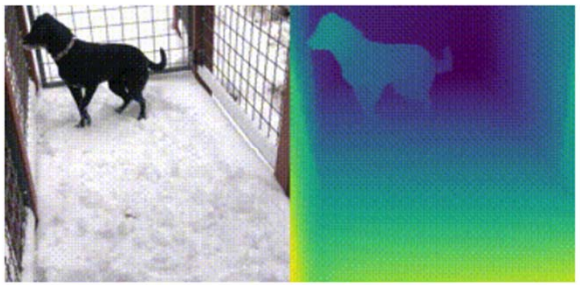
### Learn OpenVINO Development Tools

- Explore a variety of pre-trained deep learning models in the Open Model Zoo and deploy them in demo applications to see how they work.
- Want to import a model from another framework and optimize its performance with OpenVINO? Visit the Convert a Model page.
- Accelerate your model's speed even further with quantization and other compression techniques using Neural Network Compression Framework (NNCF).
- Benchmark your model's inference speed with one simple command using the Benchmark Tool.

## What's Next?

Learn more about OpenVINO and use it in your own application by trying out some of these examples!

### Get started with Python



Try the Python Quick Start Example to estimate depth in a scene using an OpenVINO monodepth model in a Jupyter Notebook inside your web browser.

Visit the Tutorials page for more Jupyter Notebooks to get you started with OpenVINO, such as:

- OpenVINO Python API Tutorial
- Basic image classification program with Hello Image Classification
- Convert a PyTorch model and use it for image background removal

# Learn OpenVINO



# Interactive Tutorials (Python)

This collection of Python tutorials are written for running on Jupyter notebooks. The tutorials provide an introduction to the OpenVINO™ toolkit and explain how to use the Python API and tools for optimized deep learning inference. You can run the code one section at a time to see how to integrate your application with OpenVINO libraries.

Notebooks with a launch binder button can be run without installing anything. Once you have found the tutorial of your interest, just click the button next to the name of it and Binder will start it in a new tab of a browser. Binder is a free online service with limited resources (for more information about it, see the Additional Resources section).

**Note**

For the best performance, more control and resources, you should run the notebooks locally. Follow the Installation Guide in order to get information on how to run and manage the notebooks on your machine.

## First steps with OpenVINO

Brief tutorials that demonstrate how to use Python API for inference in OpenVINO.

| Notebook | Description | Preview |
|---|---|---|
| 001-hello-world launch binder | Classify an image with OpenVINO. |  |
| 002-openvino-api launch binder | Learn the OpenVINO Python API. | |
| 003-hello-segmentation launch binder | Semantic segmentation with OpenVINO. |  |
| 004-hello-detection launch binder | Text detection with OpenVINO. |  |

## Convert & Optimize

Tutorials that explain how to optimize and quantize models with OpenVINO tools.

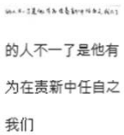
| Notebook | Description | Preview |
|---|---|---|
| 101-tensorflow-to-openvino launch binder | Convert TensorFlow models to OpenVINO IR. | |
| 102-pytorch-onnx-to-openvino | Convert PyTorch models to OpenVINO IR. | |
| 103-paddle-onnx-to-openvino launch binder | Convert PaddlePaddle models to OpenVINO IR. | |
| 104-model-tools launch binder | Download, convert and benchmark models from Open Model Zoo. | |

# Learn OpenVINO

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## Model Demos

Demos that demonstrate inference on a particular model.

| Notebook | Description | Preview |
|---|---|---|
| 205-vision-background-removal launch binder | Remove and replace the background in an image using salient object detection. |  |
| 209-handwritten-ocr launch binder | OCR for handwritten simplified Chinese and Japanese. |  |
| 211-speech-to-text launch binder | Run inference on speech-to-text recognition model. |  |
| 215-image-inpainting launch binder | Fill missing pixels with image in-painting. |  |
| 218-vehicle-detection-and-recognition launch binder | Use pre-trained models to detect and recognize vehicles and their attributes with OpenVINO. |  |

## Model Training

Tutorials that include code to train neural networks.

| Notebook | Description | Preview |
|---|---|---|
| 301-tensorflow-training-openvino | Train a flower classification model from TensorFlow, then convert to OpenVINO IR. |  |
| 301-tensorflow-training-openvino-nncf | Use Post-training Optimization Tool (POT) to quantize the flowers model. | |
| 302-pytorch-quantization-aware-training | Use Neural Network Compression Framework (NNCF) to quantize PyTorch model. | |
| 305-tensorflow-quantization-aware-training | Use Neural Network Compression Framework (NNCF) to quantize TensorFlow model. | |

# Learn OpenVINO

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## Live Demos

Live inference demos that run on a webcam or video files.

| Notebook | Description | Preview |
|---|---|---|
| 401-object-detection-webcam launch binder | Object detection with a webcam or video file. |  |
| 402-pose-estimation-webcam launch binder | Human pose estimation with a webcam or video file. |  |
| 403-action-recognition-webcam launch binder | Human action recognition with a webcam or video file. |  |
| 404-style-transfer-webcam launch binder | Style transfer with a webcam or video file. |  |
| 405-paddle-ocr-webcam launch binder | OCR with a webcam or video file. |  |
| 406-3D-pose-estimation-webcam launch binder | 3D display of human pose estimation with a webcam or video file. |  |
| 407-person-tracking-webcam launch binder | Person tracking with a webcam or video file. |  |

## Recommended Tutorials

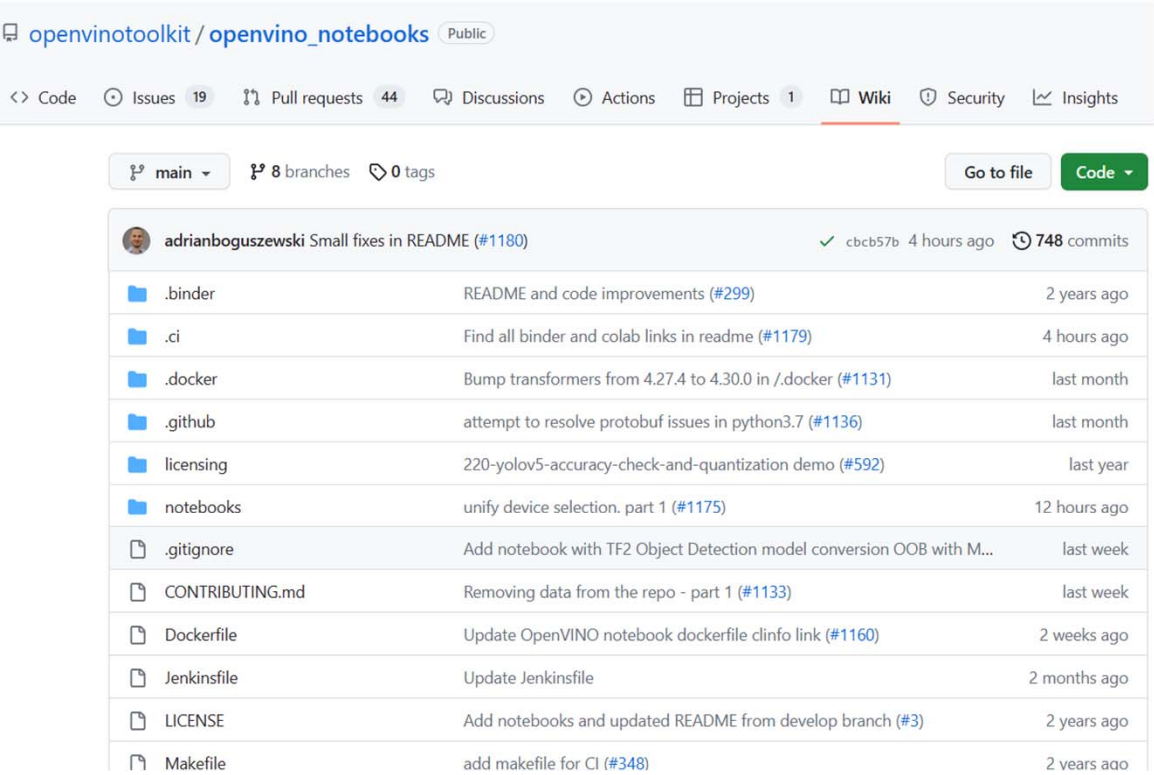
The following tutorials are guaranteed to provide a great experience with inference in OpenVINO:

| Notebook | | Preview |
|---|---|---|
| Vision-monodepth launch binder | Monocular depth estimation with images and video. |  |
| Vision-background-removal launch binder | Remove and replace the background in an image using salient object detection. |  |
| Object-detection-webcam launch binder | Object detection with a webcam or video file. |  |
| Pose-estimation-webcam launch binder | Human pose estimation with a webcam or video file. |  |
| Action-recognition-webcam launch binder | Human action recognition with a webcam or video file. |  |

# Learn OpenVINO

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

## Installation of OpenVINO™ Notebooks




### Installation Guide

OpenVINO Notebooks require Python and Git. To get started, select the guide for your operating system or environment:

| Windows | Ubuntu | macOS | Red Hat | CentOS | Azure ML | Docker | Amazon SageMaker |
|---|---|---|---|---|---|---|---|

### Getting Started

The Jupyter notebooks are categorized into four classes, select one related to your needs or give them all a try. Good Luck!

**NOTE: The main branch of this repository was updated to support the new OpenVINO 2023.0 release.** To upgrade to the new release version, please run `pip install --upgrade -r requirements.txt` in your `openvino_env` virtual environment. If you need to install for the first time, see the Installation Guide section below. If you wish to use the previous Long Term Support (LTS) version of OpenVINO check out the 2022.3 branch.

If you need help, please start a GitHub Discussion.

# Conda

Adrian Boguszewski edited this page on Apr 27, 2022 · 9 revisions

The steps to use the OpenVINO notebooks with Anaconda are slightly different than with Python from an installer version. This is a modified installation guide for Anaconda. It has been tested with Miniconda on Windows 10. If you run into an issue with these steps, please let us know.

On Windows, these steps should be executed inside an **Anaconda Prompt** (open Anaconda Prompt from the start menu, or press Windows-S and start typing *Anaconda*). Use the regular Anaconda Prompt, not the Powershell prompt.

## Step 1: Clone the Repository

```
git clone https://github.com/openvinotoolkit/openvino_notebooks.git
```

## Step 2: Create a Conda Environment with Python 3.8

Python 3.6 and 3.7 are also supported. On Ubuntu 18, Python 3.6 is recommended, as it does not require installation of additional Python libraries.

```
cd openvino_notebooks
conda create -n openvino_env python=3.8
```

## Step 3: Activate the Environment

```
conda activate openvino_env
```

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda install <package-name>
```

## Step 4: Install the Packages

Install OpenVINO, Jupyter, and other required packages to run the notebooks.

```
# Upgrade pip to the latest version to ensure compatibility with all dependencies
python -m pip install --upgrade pip==21.3.*
pip install -r requirements.txt
```

## Step 5 (conda): Add the OpenVINO library to your PATH

> This step is only for Windows. Skip this step for macOS and Linux.

Depending on your Conda installation, this step may be required at the moment for conda environments on Windows. In a future OpenVINO version this should no longer be necessary.

The command below assumes that Miniconda is installed in the default location: `C:\Users\<username>\Minoconda3`, where `<username>` is your Windows username. If you installed Anaconda, replace Miniconda3 with Anaconda3. If you installed Anaconda or Minoconda in a different location, you can run `python -c "import os,sys;print(os.path.dirname(sys.executable))"` to find the path to `openvino_env`.

Note that at the moment you need to run this command again if you open a new Anaconda Prompt to run the notebooks. You can add this folder to your PATH for every command prompt you open, by following these steps on Microsoft's website. Note however, that this may cause issues if you have multiple OpenVINO versions installed.

```
set PATH=C:\Users\<username>\Miniconda3\envs\openvino_env\Lib\site-packages\openvino\libs;%PATH%
```

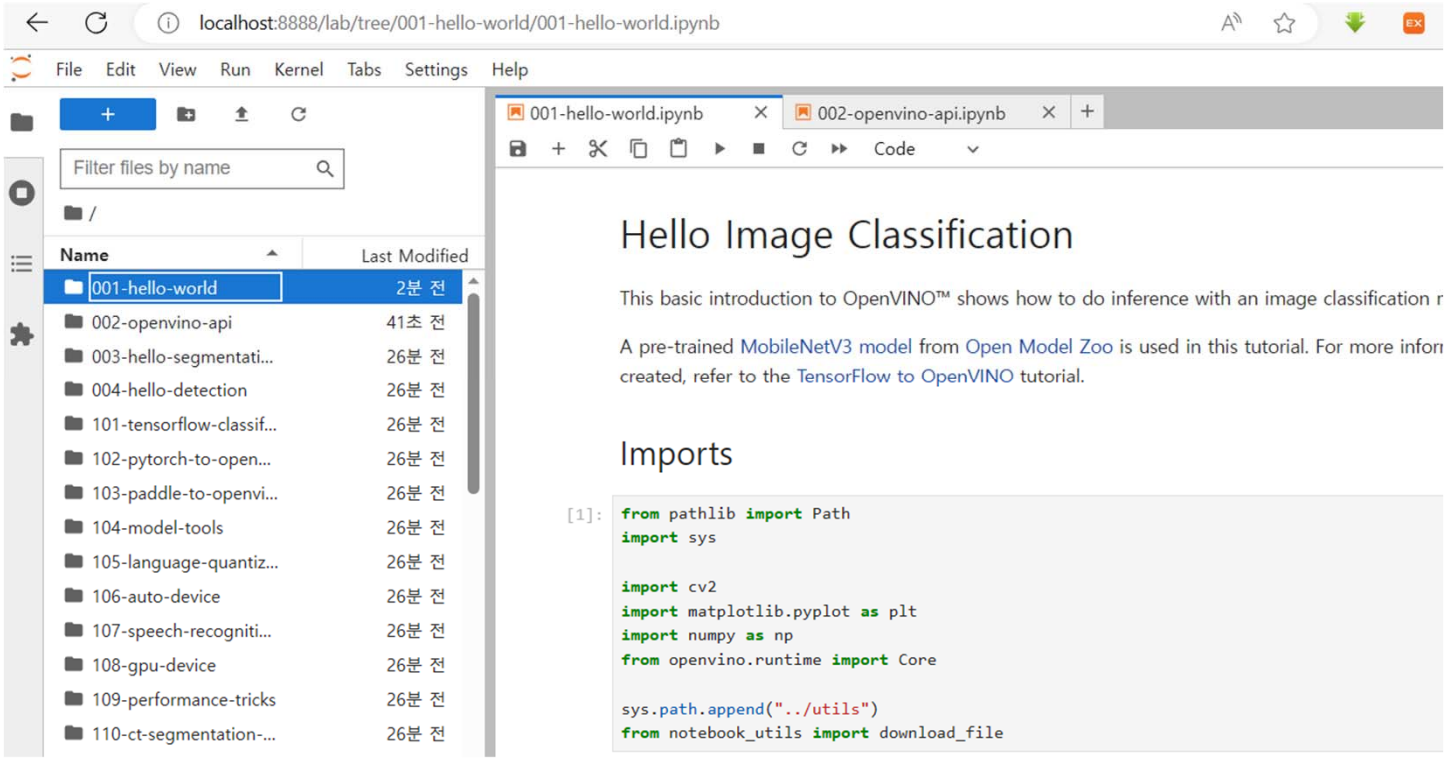
## Step 6 (conda): Launch the Notebooks!

```
# To launch a single notebook
jupyter notebook <notebook_filename>

# To launch all notebooks in Jupyter Lab
jupyter lab notebooks
```

# Learn OpenVINO

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

```
C:₩Users₩sky₩Documents₩Openvino₩openvino_notebooks 디렉터리

[.]                 [..]                [.binder]           [.ci]               [.docker]           [.github]
.gitignore          check_install.py    CONTRIBUTING.md     Dockerfile          Jenkinsfile         LICENSE
[licensing]         Makefile            [notebooks]         README.md           README_cn.md        requirements.txt
              10개 파일             146,317 바이트
               8개 디렉터리  72,714,678,272 바이트 남음

(openvino_env) C:₩Users₩sky₩Documents₩Openvino₩openvino_notebooks>jupyter lab notebooks
```



# Learn OpenVINO

https://docs.openvino.ai/2022.3/get_started.html#learn-openvino

Try it~

**Venv**

## Installing notebooks

WINDOWS | Linux Systems | macOS | Azure ML | Docker

1. **Create a Virtual Environment**

   If you already have installed *openvino-dev*, you may skip this step and proceed with the next one.

   ```
   python -m venv openvino_env
   ```

2. **Activate the Environment**

   ```
   openvino_env\Scripts\activate
   ```

3. **Clone the Repository**

   Using the –depth=1 option for git clone reduces download size.

   ```
   git clone --depth=1 https://github.com/openvinotoolkit/openvino_notebooks.git
   cd openvino_notebooks
   ```

4. **Upgrade PIP**

   ```
   python -m pip install --upgrade pip
   ```

5. **Install required packages**

   ```
   pip install -r requirements.txt
   ```

6. **Install the virtualenv Kernel in Jupyter**

   ```
   python -m ipykernel install --user --name openvino_env
   ```

## Run the Notebooks

### Launch a Single Notebook

If you want to launch only one notebook, such as the *Monodepth* notebook, run the command below.

```
jupyter 201-vision-monodepth.ipynb
```

### Launch All Notebooks

```
jupyter lab notebooks
```

In your browser, select a notebook from the file browser in Jupyter Lab, using the left sidebar. Each tutorial is located in a subdirectory within the **notebooks** directory.

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

# Hello Image Classification

This basic introduction to OpenVINO™ shows how to do inference with an image classification model.

A pre-trained MobileNetV3 model from Open Model Zoo is used in this tutorial. For more information about how OpenVINO IR models are created, refer to the TensorFlow to OpenVINO tutorial.

## Imports

```
from pathlib import Path
import sys

import cv2
import matplotlib.pyplot as plt
import numpy as np
from openvino.runtime import Core

sys.path.append("../utils")
from notebook_utils import download_file
```

## Download the Model and data samples

```
base_artifacts_dir = Path('./artifacts').expanduser()

model_name = "v3-small_224_1.0_float"
model_xml_name = f'{model_name}.xml'
model_bin_name = f'{model_name}.bin'

model_xml_path = base_artifacts_dir / model_xml_name

base_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/mobelinet-v3-tf/FP32/'

if not model_xml_path.exists():
    download_file(base_url + model_xml_name, model_xml_name, base_artifacts_dir)
    download_file(base_url + model_bin_name, model_bin_name, base_artifacts_dir)
else:
    print(f'{model_name} already downloaded to {base_artifacts_dir}')
```

맨 앞의 ~를 홈 디렉토리에 치환

```
model_xml_path
```

```
WindowsPath('artifacts/v3-small_224_1.0_float.xml')
```

```
v3-small_224_1.0_float already downloaded to artifacts
```

storage.openvinotoolkit.org/repositories/openvino_notebooks/models/mobelinet-v3-tf/

https://storage.openvinotoolkit.org / repositories / openvino_notebooks / models / mobelinet-v3-tf / FP32

| FILE NAME ↓ | FILE SIZE ↓ | DATE ↓ |
|---|---|---|
| ../ | - | |
| v3-small_224_1.0_float.bin | 4.8 MB | 2023-05-24 07:36 |
| v3-small_224_1.0_float.xml | 293.6 kB | 2023-05-24 07:36 |

# Hello Image Classification

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Select inference device

select device from dropdown list for running inference using OpenVINO

```
import ipywidgets as widgets

core = Core()
device = widgets.Dropdown(
    options=core.available_devices + ["AUTO"],
    value='AUTO',
    description='Device:',
    disabled=False,
)

device
```

Device: AUTO

```
core.available_devices
```

```
['CPU', 'GPU']
```

## Load the Model

```
core = Core()
model = core.read_model(model=model_xml_path)
compiled_model = core.compile_model(model=model, device_name=device.value)

output_layer = compiled_model.output(0)
```

```
model
```

```
<Model: 'v3-small_224_1.0_float'
inputs[
<ConstOutput: names[input:0] shape[1,224,224,3] type: f32>
]
outputs[
<ConstOutput: names[MobilenetV3/Predictions/Softmax:0] shape[1,1001] type: f32>
]>
```

```
compiled_model.output
```

```
<bound method PyCapsule.output of <CompiledModel:
inputs[
<ConstOutput: names[input:0] shape[1,224,224,3] type: f32>
]
outputs[
<ConstOutput: names[MobilenetV3/Predictions/Softmax:0] shape[1,1001] type: f32>
]>>
```

```
output_layer
```

```
<ConstOutput: names[MobilenetV3/Predictions/Softmax:0] shape[1,1001] type: f32>
```

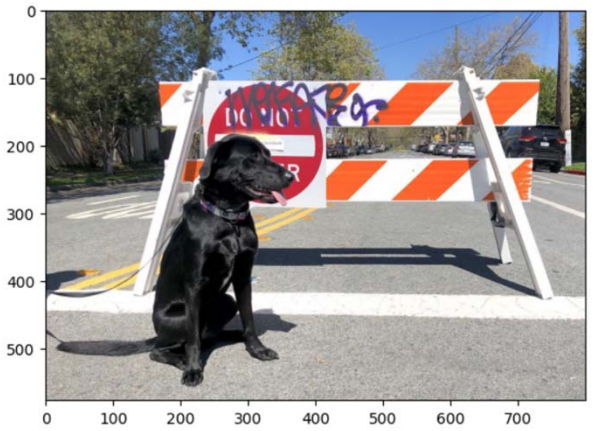
# Hello Image Classification

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Load an Image

```
# The MobileNet model expects images in RGB format.
image = cv2.cvtColor(cv2.imread(filename="../data/image/coco.jpg"), code=cv2.COLOR_BGR2RGB)

# Resize to MobileNet image shape.
input_image = cv2.resize(src=image, dsize=(224, 224))

# Reshape to model input shape.
input_image = np.expand_dims(input_image, 0)
print(input_image.shape)

plt.imshow(image);
```

```
(1, 224, 224, 3)
```




## Do Inference

```
result_infer = compiled_model([input_image])[output_layer]
result_index = np.argmax(result_infer)
```

```
print(result_infer)
print(result_index)
```

```
[[3.8589944e-05 1.9022416e-05 2.3226664e-04 ... 2.3366909e-05
  1.9658415e-05 1.5064743e-05]]
206
```

```
# Convert the inference result to a class name.
imagenet_classes = open("../data/datasets/imagenet/imagenet_2012.txt").read().splitlines()

# The model description states that for this model, class 0 is a background.
# Therefore, a background must be added at the beginning of imagenet_classes.
imagenet_classes = ['background'] + imagenet_classes

imagenet_classes[result_index]
```

```
'n02099267 flat-coated retriever'
```

## OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

# OpenVINO™ Runtime API Tutorial

This notebook explains the basics of the OpenVINO Runtime API. It covers:



The notebook is divided into sections with headers. The next cell contains global requirements installation and imports. Each section is standalone and does not depend on any previous sections. A segmentation and classification OpenVINO IR model and a segmentation ONNX model are provided as examples. These model files can be replaced with your own models. The exact outputs will be different, but the process is the same.

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
# Required imports. Please execute this cell first.
!pip install -q "openvino>=2023.0.0"
!pip install requests tqdm

# Fetch `notebook_utils` module
import urllib.request
urllib.request.urlretrieve(
    url='https://raw.githubusercontent.com/openvinotoolkit/openvino_notebooks/main/notebooks/utils/notebook_utils.py',
    filename='notebook_utils.py'
)

from notebook_utils import download_file
```

```
Requirement already satisfied: requests in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (2.31.0)
Requirement already satisfied: tqdm in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (4.65.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sky\documents\openvino\openvino_env\lib\site-packag
es (from requests) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (from req
uests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (fr
om requests) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (fr
om requests) (2023.5.7)
Requirement already satisfied: colorama in c:\users\sky\documents\openvino\openvino_env\lib\site-packages (from tqdm)
(0.4.6)
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Loading OpenVINO Runtime and Showing Info

Initialize OpenVINO Runtime with Core()

```
from openvino.runtime import Core

core = Core()
```

OpenVINO Runtime can load a network on a device. A device in this context means a CPU, an Intel GPU, a Neural Compute Stick 2, etc. The `available_devices` property shows the available devices in your system. The "FULL_DEVICE_NAME" option to `core.get_property()` shows the name of the device.

In this notebook, the CPU device is used. To use an integrated GPU, use `device_name="GPU"` instead. Be aware that loading a network on GPU will be slower than loading a network on CPU, but inference will likely be faster.

```
devices = core.available_devices

for device in devices:
    device_name = core.get_property(device, "FULL_DEVICE_NAME")
    print(f"{device}: {device_name}")
```

```
CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
GPU: NVIDIA GeForce RTX 2070 Super with Max-Q Design (dGPU)
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Loading a Model

After initializing OpenVINO Runtime, first read the model file with `read_model()`, then compile it to the specified device with the `compile_model()` method.

OpenVINO™ supports several model formats and enables developers to convert them to its own OpenVINO IR format using a tool dedicated to this task.

### OpenVINO IR Model

An OpenVINO IR (Intermediate Representation) model consists of an `.xml` file, containing information about network topology, and a `.bin` file, containing the weights and biases binary data. Models in OpenVINO IR format are obtained by using Model Optimizer tool. The `read_model()` function expects the `.bin` weights file to have the same filename and be located in the same directory as the `.xml` file: `model_weights_file == Path(model_xml).with_suffix(".bin")`. If this is the case, specifying the weights file is optional. If the weights file has a different filename, it can be specified using the `weights` parameter in `read_model()`.

The OpenVINO Model Optimizer tool is used to convert models to OpenVINO IR format. Model Optimizer reads the original model and creates an OpenVINO IR model (.xml and .bin files) so inference can be performed without delays due to format conversion. Optionally, Model Optimizer can adjust the model to be more suitable for inference, for example, by alternating input shapes, embedding preprocessing and cutting training parts off. For information on how to convert your existing TensorFlow, PyTorch or ONNX model to OpenVINO IR format with Model Optimizer, refer to the tensorflow-to-openvino and pytorch-onnx-to-openvino notebooks.

## Supported Model Formats

**OpenVINO IR (Intermediate Representation)** - the proprietary format of OpenVINO™, benefiting from the full extent of its features.

**ONNX, PaddlePaddle, TensorFlow, TensorFlow Lite** - formats supported directly, which means they can be used with OpenVINO Runtime without any prior conversion. For a guide on how to run inference on ONNX, PaddlePaddle, or TensorFlow, see how to Integrate OpenVINO™ with Your Application.

**MXNet, Caffe, Kaldi** - formats supported indirectly, which means they need to be converted to OpenVINO IR before running inference. The conversion is done with Model Conversion API and in some cases may involve intermediate steps.

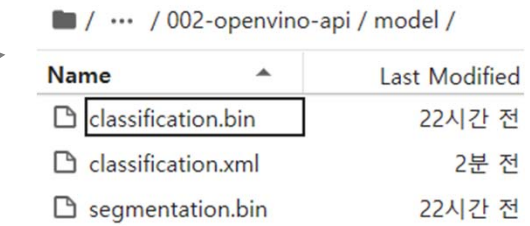
- How to convert ONNX
- How to convert PaddlePaddle
- How to convert TensorFlow
- How to convert TensorFlow Lite
- How to convert MXNet
- How to convert Caffe
- How to convert Kaldi
- Conversion examples for specific models

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
ir_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
ir_model_name_xml = 'classification.xml'
ir_model_name_bin = 'classification.bin'

download_file(ir_model_url + ir_model_name_xml, filename=ir_model_name_xml, directory='model')
download_file(ir_model_url + ir_model_name_bin, filename=ir_model_name_bin, directory='model')
```

```
model₩classification.xml: 100% 179k/179k [00:00<00:00, 199kB/s]
'model\classification.bin' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.bin')
```




```
from openvino.runtime import Core

core = Core()
classification_model_xml = "model/classification.xml"

model = core.read_model(model=classification_model_xml)
compiled_model = core.compile_model(model=model, device_name="CPU")
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## ONNX Model

ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers. OpenVINO supports reading models in ONNX format directly,that means they can be used with OpenVINO Runtime without any prior conversion.

Reading and loading an ONNX model, which is a single `.onnx` file, works the same way as with an OpenVINO IR model. The `model` argument points to the filename of an ONNX model.

```
onnx_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/segmentation.onnx'
onnx_model_name = 'segmentation.onnx'

download_file(onnx_model_url, filename=onnx_model_name, directory='model')
```

```
model₩segmentation.onnx: 100% 4.41M/4.41M [00:01<00:00, 5.06MB/s]
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/segmentation.onnx')
```

```
from openvino.runtime import Core

core = Core()
onnx_model_path = "model/segmentation.onnx"

model_onnx = core.read_model(model=onnx_model_path)
compiled_model_onnx = core.compile_model(model=model_onnx, device_name="CPU")
```

The ONNX model can be exported to OpenVINO IR with `serialize()` :

```
from openvino.runtime import serialize

serialize(model_onnx, xml_path="model/exported_onnx_model.xml")
```

**ONNX is an open format built to represent machine learning models.**

**ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers**

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## PaddlePaddle Model **PaddlePaddle: Baidu's DL Framework**

PaddlePaddle models saved for inference can also be passed to OpenVINO Runtime without any conversion step. Pass the filename with extension to `read_model` and exported an OpenVINO IR with `serialize`

```
paddle_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
paddle_model_name = 'inference.pdmodel'
paddle_params_name = 'inference.pdiparams'

download_file(paddle_model_url + paddle_model_name, filename=paddle_model_name, directory='model')
download_file(paddle_model_url + paddle_params_name, filename=paddle_params_name, directory='model')
```

```
model₩inference.pdmodel: 100% 1.03M/1.03M [00:01<00:00, 1.11MB/s]
model₩inference.pdiparams: 100% 21.0M/21.0M [00:13<00:00, 5.77MB/s]
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/inference.pdiparams')
```

```
from openvino.runtime import Core

core = Core()
paddle_model_path = 'model/inference.pdmodel'

model_paddle = core.read_model(model=paddle_model_path)
compiled_model_paddle = core.compile_model(model=model_paddle, device_name="CPU")
```

```
from openvino.runtime import serialize

serialize(model_paddle, xml_path="model/exported_paddle_model.xml")
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## TensorFlow Model

pb 파일

TensorFlow models saved in frozen graph format can also be passed to `read_model` starting in OpenVINO 2022.3.

> **NOTE**: Directly loading TensorFlow models is available as a preview feature in the OpenVINO 2022.3 release. Fully functional support will be provided in the upcoming 2023 releases. Currently support is limited to only frozen graph inference format. Other TensorFlow model formats must be converted to OpenVINO IR using Model Optimizer.

```
pb_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/classification.pb'
pb_model_name = 'classification.pb'

download_file(pb_model_url, filename=pb_model_name, directory='model')
```

```
model₩classification.pb: 100% 9.88M/9.88M [00:02<00:00, 9.79MB/s]

WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.pb')
```

```
from openvino.runtime import Core

core = Core()
tf_model_path = "model/classification.pb"

model_tf = core.read_model(model=tf_model_path)
compiled_model_tf = core.compile_model(model=model_tf, device_name="CPU")
```

```
from openvino.runtime import serialize

serialize(model_tf, xml_path="model/exported_tf_model.xml")
```

**ckpt 파일** : 일반적으로 이야기하는 ckpt파일은 .ckpt-data와 같으며, **딥러닝 모델을 제외한 학습한 가중치(weight)만 있는 파일**. 모델 구조(graph)는 저장하지 않는다.

- **ckpt-meta** : 모델(graph)만 있는 파일
- **ckpt-data** : 딥러닝 모델을 제외한 학습한 가중치(weight)만 있는 파일. 모델 구조(graph)는 저장하지 않는다.

**pb 파일** : **모델 구조와 가중치(weight) 모두 저장된 파일. freeze_graph.py**를 통해서 만들 수 있고,'그래프를 freezing시킨다.'라고 하면 pb파일을 만들 것이라는 뜻이다.

**h5 파일** : **Hierarchical Data Format (HDF)**형식으로 저장되는 데이터. Keras에서는 모델 및 가중치(weight) 모두를 가지고 있는 파일이다.

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## TensorFlow Lite Model

TensorFlow Lite는 모바일, 마이크로컨트롤러 및 기타 에지 기기에 모델을 배포하기 위한 모바일 라이브러리

TFLite models saved for inference can also be passed to OpenVINO Runtime. Pass the filename with extension `.tflite` to `read_model` and exported an OpenVINO IR with `serialize`.

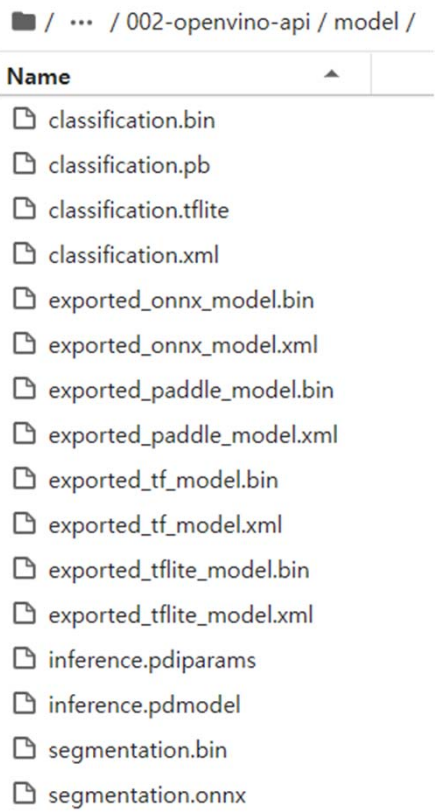
This tutorial uses the image classification model inception_v4_quant. It is pre-trained model optimized to work with TensorFlow Lite.

```
from pathlib import Path

tflite_model_url = 'https://tfhub.dev/tensorflow/lite-model/inception_v4_quant/1/default/1?lite-format=tflite'
tflite_model_path = Path('model/classification.tflite')

download_file(tflite_model_url, filename=tflite_model_path.name, directory=tflite_model_path.parent)
```

```
'model\classification.tflite' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.tflite')
```

```
from openvino.runtime import Core

ie = Core()

model_tflite = core.read_model(tflite_model_path)
compiled_model_tflite = core.compile_model(model=model_tflite, device_name="CPU")
```

```
from openvino.runtime import serialize

serialize(model_tflite, xml_path="model/exported_tflite_model.xml")
```



# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Getting Information about a Model

The OpenVINO Model instance stores information about the model. Information about the inputs and outputs of the model are in `model.inputs` and `model.outputs`. These are also properties of the CompiledModel instance. While using `model.inputs` and `model.outputs` in the cells below, you can also use `compiled_model.inputs` and `compiled_model.outputs`.

```
ir_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
ir_model_name_xml = 'classification.xml'
ir_model_name_bin = 'classification.bin'

download_file(ir_model_url + ir_model_name_xml, filename=ir_model_name_xml, directory='model')
download_file(ir_model_url + ir_model_name_bin, filename=ir_model_name_bin, directory='model')
```

```
'model\classification.xml' already exists.
'model\classification.bin' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.bin')
```

### Model Inputs

Information about all input layers is stored in the `inputs` dictionary.

```
from openvino.runtime import Core

core = Core()
classification_model_xml = "model/classification.xml"
model = core.read_model(model=classification_model_xml)
model.inputs
```

```
[<Output: names[input:0, input] shape[1,3,224,224] type: f32>]
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

The cell above shows that the loaded model expects one input with the name *input*. If you loaded a different model, you may see a different input layer name, and you may see more inputs. You may also obtain info about each input layer using `model.input(index)`, where index is a numeric index of the input layers in the model. If a model has only one input, index can be omitted.

```
input_layer = model.input(0)
input_layer
```

```
<Output: names[input:0, input] shape[1,3,224,224] type: f32>
```

It is often useful to have a reference to the name of the first input layer. For a model with one input, `model.input(0).any_name` gets this name.

```
input_layer.any_name
```

```
'input'
```

The next cell prints the input layout, precision and shape.

```
print(f"input precision: {input_layer.element_type}")
print(f"input shape: {input_layer.shape}")
```

```
input precision: <Type: 'float32'>
input shape: [1,3,224,224]
```

This cell shows that the model expects inputs with a shape of [1,3,224,224], and that this is in the `NCHW` layout. This means that the model expects input data with the batch size of 1 (`N`), 3 channels (`C`), and images with a height (`H`) and width (`W`) equal to 224. The input data is expected to be of `FP32` (floating point) precision.

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Model Outputs

```
from openvino.runtime import Core

ie = Core()
classification_model_xml = "model/classification.xml"
model = core.read_model(model=classification_model_xml)
model.outputs
```

```
[<Output: names[MobilenetV3/Predictions/Softmax] shape[1,1001] type: f32>]
```

Model output info is stored in `model.outputs`. The cell above shows that the model returns one output, with the `MobilenetV3/Predictions/Softmax` name. Loading a different model will result in different output layer name, and more outputs might be returned. Similar to input, you may also obtain information about each output separately using `model.output(index)`

Since this model has one output, follow the same method as for the input layer to get its name.

```
output_layer = model.output(0)
output_layer.any_name
```

```
'MobilenetV3/Predictions/Softmax'
```

Getting the output precision and shape is similar to getting the input precision and shape.

```
print(f"output precision: {output_layer.element_type}")
print(f"output shape: {output_layer.shape}")
```

```
output precision: <Type: 'float32'>
output shape: [1,1001]
```

This cell shows that the model returns outputs with a shape of [1, 1001], where 1 is the batch size (`N`) and 1001 is the number of classes (`C`). The output is returned as 32-bit floating point.

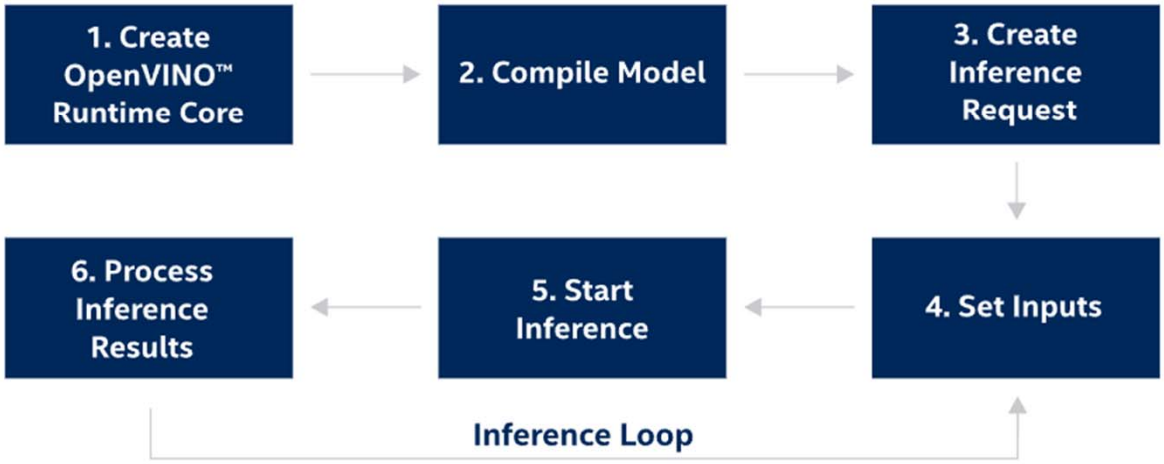
# OpenVINO Runtime API Tutorial

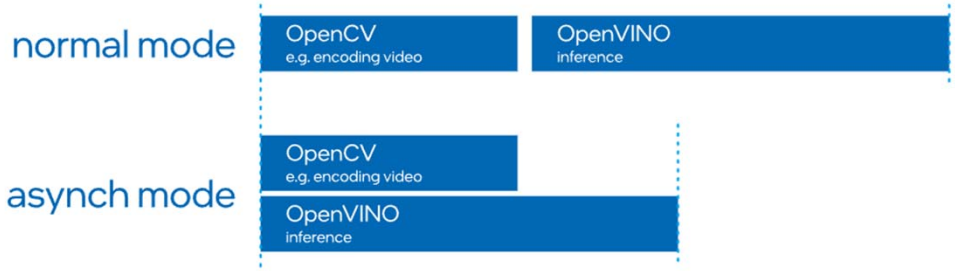
https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Doing Inference on a Model

> **NOTE** this notebook demonstrates only the basic synchronous inference API. For an async inference example, please refer to Async API notebook

The diagram below shows a typical inference pipeline with OpenVINO




Creating **OpenVINO Core** and **model compilation** is covered in the previous steps.

The next step is preparing an **inference request**.

- To do inference on a model, **first create an inference request by calling the create_infer_request() method of CompiledModel**, compiled_model that was loaded with **compile_model()**.
- Then, call the **infer()** method of **InferRequest**. It expects one argument: **inputs**. This is a dictionary that maps input layer names to input data or list of input data in np.ndarray format, where the position of the input tensor corresponds to input index. If a model has a single input, wrapping to a dictionary or list can be omitted.

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
# Install opencv package for image handling
!pip install -q opencv-python
```

**Load the network**

```
ir_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
ir_model_name_xml = 'classification.xml'
ir_model_name_bin = 'classification.bin'

download_file(ir_model_url + ir_model_name_xml, filename=ir_model_name_xml, directory='model')
download_file(ir_model_url + ir_model_name_bin, filename=ir_model_name_bin, directory='model')
```

```
'model\classification.xml' already exists.
'model\classification.bin' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.bin')
```

```
from openvino.runtime import Core

ie = Core()
classification_model_xml = "model/classification.xml"
model = core.read_model(model=classification_model_xml)
compiled_model = core.compile_model(model=model, device_name="CPU")
input_layer = compiled_model.input(0)
output_layer = compiled_model.output(0)
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

### Load an image and convert to the input shape

To propagate an image through the network, it needs to be loaded into an array, resized to the shape that the network expects, and converted to the input layout of the network.

```
import cv2

image_filename = download_file(
    "https://storage.openvinotoolkit.org/repositories/openvino_notebooks/data/data/image/coco_hollywood.jpg",
    directory="data"
)
image = cv2.imread(str(image_filename))
image.shape
```

```
data₩coco_hollywood.jpg: 100% 485k/485k [00:00<00:00, 459kB/s]
(663, 994, 3)
```

The image has a shape of (663,994,3). It is 663 pixels in height, 994 pixels in width, and has 3 color channels. A reference to the height and width expected by the network is obtained and the image is resized to these dimensions.

```
# N,C,H,W = batch size, number of channels, height, width.
N, C, H, W = input_layer.shape
# OpenCV resize expects the destination size as (width, height).
resized_image = cv2.resize(src=image, dsize=(W, H))
resized_image.shape
```

```
(224, 224, 3)
```

Now, the image has the width and height that the network expects. This is still in `HWC` format and must be changed to `NCHW` format. First, call the `np.transpose()` method to change to `CHW` and then add the `N` dimension (where `N` = 1) by calling the `np.expand_dims()` method. Next, convert the data to `FP32` with `np.astype()` method.

```
import numpy as np

input_data = np.expand_dims(np.transpose(resized_image, (2, 0, 1)), 0).astype(np.float32)
input_data.shape
```

```
(1, 3, 224, 224)
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

### Do inference

Now that the input data is in the right shape, run inference. The CompiledModel inference result is a dictionary where keys are the Output class instances (the same keys in `compiled_model.outputs` that can also be obtained with `compiled_model.output(index)`) and values - predicted result in np.array format.

```
# for single input models only
result = compiled_model(input_data)[output_layer]

# for multiple inputs in a list
result = compiled_model([input_data])[output_layer]

# or using a dictionary, where the key is input tensor name or index
result = compiled_model({input_layer.any_name: input_data})[output_layer]
```

You can also create `InferRequest` and run `infer` method on request.

```
request = compiled_model.create_infer_request()
request.infer(inputs={input_layer.any_name: input_data})
result = request.get_output_tensor(output_layer.index).data
```

The `.infer()` function sets output tensor, that can be reached, using `get_output_tensor()`. Since this network returns one output, and the reference to the output layer is in the `output_layer.index` parameter, you can get the data with `request.get_output_tensor(output_layer.index)`. To get a numpy array from the output, use the `.data` parameter.

```
result.shape
```

```
(1, 1001)
```

The output shape is (1,1001), which is the expected output shape. This shape indicates that the network returns probabilities for 1001 classes. To learn more about this notion, refer to the hello world notebook.

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

# Reshaping and Resizing

## Change Image Size

Instead of reshaping the image to fit the model, it is also possible to reshape the model to fit the image. Be aware that not all models support reshaping, and models that do, may not support all input shapes. The model accuracy may also suffer if you reshape the model input shape.

First check the input shape of the model, then reshape it to the new input shape.

```
ir_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
ir_model_name_xml = 'segmentation.xml'
ir_model_name_bin = 'segmentation.bin'

download_file(ir_model_url + ir_model_name_xml, filename=ir_model_name_xml, directory='model')
download_file(ir_model_url + ir_model_name_bin, filename=ir_model_name_bin, directory='model')
```

```
model₩segmentation.xml: 100% [progress bar] 1.38M/1.38M [00:01<00:00, 1.47MB/s]
'model\segmentation.bin' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/segmentation.bin')
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```python
from openvino.runtime import Core, PartialShape

core = Core()
segmentation_model_xml = "model/segmentation.xml"
segmentation_model = core.read_model(model=segmentation_model_xml)
segmentation_input_layer = segmentation_model.input(0)
segmentation_output_layer = segmentation_model.output(0)

print("~~~~ ORIGINAL MODEL ~~~~")
print(f"input shape: {segmentation_input_layer.shape}")
print(f"output shape: {segmentation_output_layer.shape}")

new_shape = PartialShape([1, 3, 544, 544])
segmentation_model.reshape({segmentation_input_layer.any_name: new_shape})
segmentation_compiled_model = core.compile_model(model=segmentation_model, device_name="CPU")
# help(segmentation_compiled_model)
print("~~~~ RESHAPED MODEL ~~~~")
print(f"model input shape: {segmentation_input_layer.shape}")
print(
    f"compiled_model input shape: "
    f"{segmentation_compiled_model.input(index=0).shape}"
)
print(f"compiled_model output shape: {segmentation_output_layer.shape}")
```

```
~~~~ ORIGINAL MODEL ~~~~
input shape: [1,3,512,512]
output shape: [1,1,512,512]
~~~~ RESHAPED MODEL ~~~~
model input shape: [1,3,544,544]
compiled_model input shape: [1,3,544,544]
compiled_model output shape: [1,1,544,544]
```

The input shape for the segmentation network is [1,3,512,512], with the `NCHW` layout: the network expects 3-channel images with a width and he
512 and a batch size of 1. Reshape the network with the `.reshape()` method of `IENetwork` to make it accept input images with a width and
544. This segmentation network always returns arrays with the input width and height of equal value. Therefore, setting the input dimensions to
also modifies the output dimensions. After reshaping, compile the network once again.

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Change Batch Size

Use the `.reshape()` method to set the batch size, by increasing the first element of `new_shape`. For example, to set a batch size of two, set `new_shape = (2,3,544,544)` in the cell above.

```
from openvino.runtime import Core, PartialShape

segmentation_model_xml = "model/segmentation.xml"
segmentation_model = core.read_model(model=segmentation_model_xml)
segmentation_input_layer = segmentation_model.input(0)
segmentation_output_layer = segmentation_model.output(0)
new_shape = PartialShape([2, 3, 544, 544])
segmentation_model.reshape({segmentation_input_layer.any_name: new_shape})
segmentation_compiled_model = core.compile_model(model=segmentation_model, device_name="CPU")

print(f"input shape: {segmentation_input_layer.shape}")
print(f"output shape: {segmentation_output_layer.shape}")
```

```
input shape: [2,3,544,544]
output shape: [2,1,544,544]
```

The output shows that by setting the batch size to 2, the first element (`N`) of the input and output shape has a value of 2. Propagate the input image through the network to see the result:

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
import numpy as np
from openvino.runtime import Core, PartialShape

core = Core()
segmentation_model_xml = "model/segmentation.xml"
segmentation_model = core.read_model(model=segmentation_model_xml)
segmentation_input_layer = segmentation_model.input(0)
segmentation_output_layer = segmentation_model.output(0)
new_shape = PartialShape([2, 3, 544, 544])
segmentation_model.reshape({segmentation_input_layer.any_name: new_shape})
segmentation_compiled_model = core.compile_model(model=segmentation_model, device_name="CPU")
input_data = np.random.rand(2, 3, 544, 544)

output = segmentation_compiled_model([input_data])

print(f"input data shape: {input_data.shape}")
print(f"result data data shape: {segmentation_output_layer.shape}")
```

```
input data shape: (2, 3, 544, 544)
result data data shape: [2,1,544,544]
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Caching a Model

For some devices, like GPU, loading a model can take some time. Model Caching solves this issue by caching the model in a cache directory. If `core.compile_model(model=net, device_name=device_name, config=config_dict)` is set, caching will be used. This option checks if a model exists in the cache. If so, it loads it from the cache. If not, it loads the model regularly, and stores it in the cache, so that the next time the model is loaded when this option is set, the model will be loaded from the cache.

In the cell below, we create a *model_cache* directory as a subdirectory of *model*, where the model will be cached for the specified device. The model will be loaded to the GPU. After running this cell once, the model will be cached, so subsequent runs of this cell will load the model from the cache.

*Note: Model Caching is also available on CPU devices*

```
ir_model_url = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/models/002-example-models/'
ir_model_name_xml = 'classification.xml'
ir_model_name_bin = 'classification.bin'

download_file(ir_model_url + ir_model_name_xml, filename=ir_model_name_xml, directory='model')
download_file(ir_model_url + ir_model_name_bin, filename=ir_model_name_bin, directory='model')
```

```
'model\classification.xml' already exists.
'model\classification.bin' already exists.
```

```
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/002-openvino-api/model/classification.bin')
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
import time
from pathlib import Path

from openvino.runtime import Core

core = Core()

#device_name = "GPU"
device_name = "CPU"

if device_name in core.available_devices:
    cache_path = Path("model/model_cache")
    cache_path.mkdir(exist_ok=True)
    # Enable caching for OpenVINO Runtime. To disable caching set enable_caching = False
    enable_caching = True
    config_dict = {"CACHE_DIR": str(cache_path)} if enable_caching else {}

    classification_model_xml = "model/classification.xml"
    model = core.read_model(model=classification_model_xml)

    start_time = time.perf_counter()
    compiled_model = core.compile_model(model=model, device_name=device_name, config=config_dict)
    end_time = time.perf_counter()
    print(f"Loading the network to the {device_name} device took {end_time-start_time:.2f} seconds.")
```

```
Loading the network to the CPU device took 0.15 seconds.
```

After running the previous cell, we know the model exists in the cache directory. Then, we delete the compiled model and load it again. Now, we measure the time it takes now.

```
if device_name in core.available_devices:
    del compiled_model
    start_time = time.perf_counter()
    compiled_model = core.compile_model(model=model, device_name=device_name, config=config_dict)
    end_time = time.perf_counter()
    print(f"Loading the network to the {device_name} device took {end_time-start_time:.2f} seconds.")
```

```
Loading the network to the CPU device took 0.09 seconds.
```

# OpenVINO Runtime API Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
if device_name in core.available_devices:
    del compiled_model
    start_time = time.perf_counter()
    compiled_model = core.compile_model(model=model, device_name=device_name, config=config_dict)
    end_time = time.perf_counter()
    print(f"Loading the network to the {device_name} device took {end_time-start_time:.2f} seconds.")
```

```
Loading the network to the CPU device took 0.09 seconds.
```

# First steps : Hello Image Segmentation

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Hello Image Segmentation

A very basic introduction to using segmentation models with OpenVINO™.

In this tutorial, a pre-trained road-segmentation-adas-0001 model from the Open Model Zoo is used. ADAS stands for Advanced Driver Assistance Services. The model recognizes four classes: background, road, curb and mark.

## Imports

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import sys
from openvino.runtime import Core

sys.path.append("../utils")
from notebook_utils import segmentation_map_to_image, download_file
```

## Download model weights

```
from pathlib import Path

base_model_dir = Path("./model").expanduser()

model_name = "road-segmentation-adas-0001"
model_xml_name = f'{model_name}.xml'
model_bin_name = f'{model_name}.bin'

model_xml_path = base_model_dir / model_xml_name

if not model_xml_path.exists():
    model_xml_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo
    model_bin_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo

    download_file(model_xml_url, model_xml_name, base_model_dir)
    download_file(model_bin_url, model_bin_name, base_model_dir)
else:
    print(f'{model_name} already downloaded to {base_model_dir}')
```

```
model₩road-segmentation-adas-0001.xml: 100% 389k/389k [00:00<00:00, 319kB/s]
model₩road-segmentation-adas-0001.bin: 100% 720k/720k [00:01<00:00, 596kB/s]
```

model_xml_url =
"https://storage.openvinotoolkit.org/repositories/open_model_zoo/2023.0/models_bin/1/road-segmentation-adas-0001/FP32/road-segmentation-adas-0001.xml"
model_bin_url =
"https://storage.openvinotoolkit.org/repositories/open_model_zoo/2023.0/models_bin/1/road-segmentation-adas-0001/FP32/road-segmentation-adas-0001.bin"

# First steps : Hello Image Segmentation

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Select inference device

select device from dropdown list for running inference using OpenVINO

```
import ipywidgets as widgets

ie = Core()
device = widgets.Dropdown(
    options=ie.available_devices + ["AUTO"],
    value='AUTO',
    description='Device:',
    disabled=False,
)

device
```

Device: AUTO

## Load the Model

```
core = Core()

model = core.read_model(model=model_xml_path)
compiled_model = core.compile_model(model=model, device_name=device.value)

input_layer_ir = compiled_model.input(0)
output_layer_ir = compiled_model.output(0)
```

## Load an Image

A sample image from the Mapillary Vistas dataset is provided.

```
# The segmentation network expects images in BGR format.
image = cv2.imread("../data/image/empty_road_mapillary.jpg")

rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_h, image_w, _ = image.shape

# N,C,H,W = batch size, number of channels, height, width.
N, C, H, W = input_layer_ir.shape

# OpenCV resize expects the destination size as (width, height).
resized_image = cv2.resize(image, (W, H))

# Reshape to the network input shape.
input_image = np.expand_dims(
    resized_image.transpose(2, 0, 1), 0
)
plt.imshow(rgb_image)
```

```
<matplotlib.image.AxesImage at 0x25f8b973790>
```

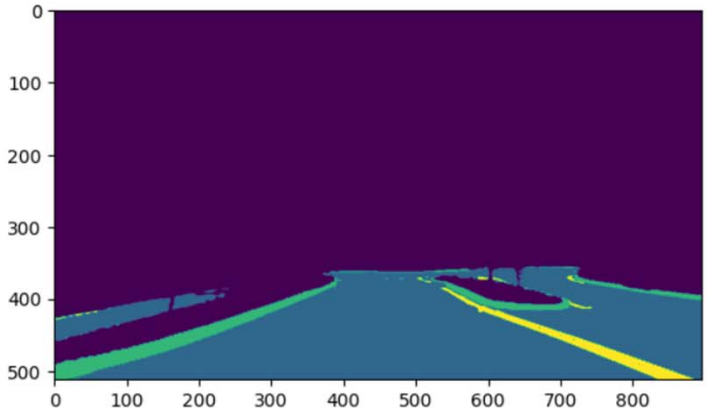
# First steps : Hello Image Segmentation

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Do Inference

```
# Run the inference.
result = compiled_model([input_image])[output_layer_ir]

# Prepare data for visualization.
segmentation_mask = np.argmax(result, axis=1)
plt.imshow(segmentation_mask.transpose(1, 2, 0))
```

```
<matplotlib.image.AxesImage at 0x25f8bfce6a0>
```




```
print(result.shape, segmentation_mask.shape)
```

```
(1, 4, 512, 896) (1, 512, 896)
```

```
result
```

```
array([[[[9.45851862e-01, 9.85596180e-01, 9.96280968e-01, ...,
          9.98552859e-01, 9.92435753e-01, 9.61370587e-01],
         [9.89936948e-01, 9.95530784e-01, 9.97918785e-01, ...,
          9.99729097e-01, 9.98387814e-01, 9.90465522e-01],
         [9.97875929e-01, 9.98562515e-01, 9.98833716e-01, ...,
          9.99948502e-01, 9.99655247e-01, 9.97691751e-01],
         ...,
         [2.55799061e-03, 1.03500288e-03, 4.17858158e-04, ...,
          1.85598768e-02, 3.61758284e-02, 6.87654689e-02],
         [3.91959259e-03, 1.58560742e-03, 6.39521284e-04, ...,
          2.48311460e-02, 5.42409644e-02, 1.12591781e-01],
         [5.98789379e-03, 2.42358982e-03, 9.77034331e-04, ...,
          3.23465504e-02, 7.94906467e-02, 1.75720572e-01]],

        [[5.35648898e-04, 1.30819040e-04, 3.09932984e-05, ...,
          7.26012740e-06, 4.14591086e-05, 2.30731224e-04],
         [2.82973866e-04, 8.51551667e-05, 2.55428022e-05, ...,
          1.47494552e-06, 1.01945816e-05, 6.99897864e-05],
         [1.43978454e-04, 5.50447585e-05, 2.10355574e-05, ...,
          2.99358589e-07, 2.49501659e-06, 2.07573703e-05],
         ...,
         [7.86862336e-03, 4.70554736e-03, 2.80780043e-03, ...,
          9.38945353e-01, 9.14230347e-01, 8.68156135e-01],
         [1.20853987e-02, 7.75631098e-03, 4.96310741e-03, ...,
          9.02305245e-01, 8.78641665e-01, 8.13093185e-01],
         [1.85060911e-02, 1.27558457e-02, 8.75728950e-03, ...,
          8.44260156e-01, 8.25368404e-01, 7.25875974e-01]],

        [[1.95621662e-02, 5.05533116e-03, 1.26732851e-03, ...,
          2.52263184e-04, 1.17486890e-03, 5.33262827e-03],
         [5.40651847e-03, 1.93898252e-03, 6.93143462e-04, ...,
          3.42686435e-05, 1.98079128e-04, 1.13725290e-03],
         [1.43914041e-03, 7.38520641e-04, 3.78827506e-04, ...,
          4.65075664e-06, 3.32385680e-05, 2.37127286e-04],
         ...,
         [9.89532113e-01, 9.94224131e-01, 9.96744156e-01, ...,
          2.58126226e-03, 7.07296003e-03, 1.89003441e-02],
         [9.83945906e-01, 9.90618408e-01, 9.94365454e-01, ...,
          6.51220698e-03, 1.58535931e-02, 3.66752893e-02],
         [9.75447774e-01, 9.84776199e-01, 9.90231931e-01, ...,
          1.59968138e-02, 3.47323567e-02, 6.78354800e-02]],

        [[3.40501890e-02, 9.21776984e-03, 2.42068991e-03, ...,
          1.18767295e-03, 6.34795101e-03, 3.30660827e-02],
         [4.37356625e-03, 2.44503305e-03, 1.36247056e-03, ...,
          2.35087951e-04, 1.40387169e-03, 8.32721870e-03],
         [5.41048823e-04, 6.44033600e-04, 7.66301295e-04, ...,
          4.64887489e-05, 3.09012103e-04, 2.05033948e-03],
         ...,
         [4.13225534e-05, 3.53332289e-05, 3.01456166e-05, ...,
          3.99135649e-02, 4.25208211e-02, 4.41781022e-02],
         [4.91195315e-05, 3.96596442e-05, 3.19261817e-05, ...,
          6.63514510e-02, 5.12637757e-02, 3.76397744e-02],
         [5.82119901e-05, 4.44143116e-05, 3.37519778e-05, ...,
          1.07396476e-01, 6.04086705e-02, 3.05679645e-02]]]],
dtype=float32)
```

```
segmentation_mask
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [2, 2, 2, ..., 1, 1, 1],
        [2, 2, 2, ..., 1, 1, 1],
        [2, 2, 2, ..., 1, 1, 1]]], dtype=int64)
```

4개 영역
분할

# First steps : Hello Image Segmentation

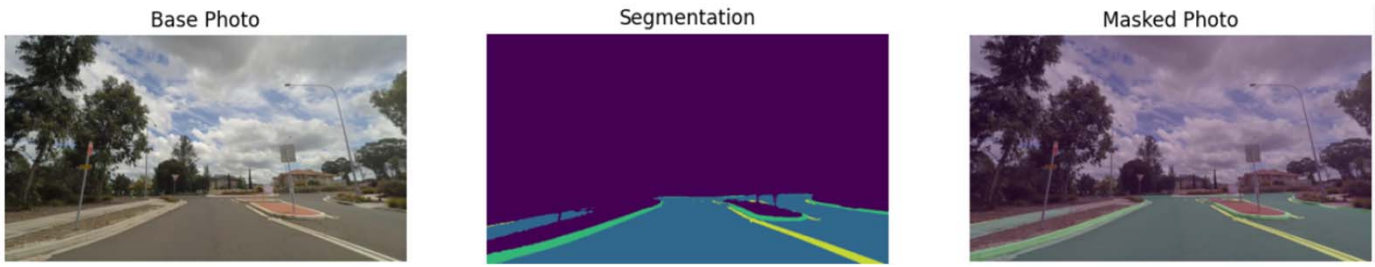
https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Prepare Data for Visualization

```
# Define colormap, each color represents a class.
colormap = np.array([[68, 1, 84], [48, 103, 141], [53, 183, 120], [199, 216, 52]])

# Define the transparency of the segmentation mask on the photo.
alpha = 0.3

# Use function from notebook_utils.py to transform mask to an RGB image.
mask = segmentation_map_to_image(segmentation_mask, colormap)
resized_mask = cv2.resize(mask, (image_w, image_h))

# Create an image with mask.
image_with_mask = cv2.addWeighted(resized_mask, alpha, rgb_image, 1 - alpha, 0)
```

## Visualize data

```
# Define titles with images.
data = {"Base Photo": rgb_image, "Segmentation": mask, "Masked Photo": image_with_mask}

# Create a subplot to visualize images.
fig, axs = plt.subplots(1, len(data.items()), figsize=(15, 10))

# Fill the subplot.
for ax, (name, image) in zip(axs, data.items()):
    ax.axis('off')
    ax.set_title(name)
    ax.imshow(image)

# Display an image.
plt.show(fig)
```



# First steps : Hello Object Detection

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Hello Object Detection

A very basic introduction to using object detection models with OpenVINO™.

The horizontal-text-detection-0001 model from Open Model Zoo is used. It detects horizontal text in images and returns a blob of data in the shape of `[100, 5]`. Each detected text box is stored in the `[x_min, y_min, x_max, y_max, conf]` format, where the `(x_min, y_min)` are the coordinates of the top left bounding box corner, `(x_max, y_max)` are the coordinates of the bottom right bounding box corner and `conf` is the confidence for the predicted class.

## Imports

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from openvino.runtime import Core
from pathlib import Path
import sys

sys.path.append("../utils")
from notebook_utils import download_file
```

model_xml_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo/2022.3/models_bin/1/horizontal-text-detection-0001/FP32/horizontal-text-detection-0001.xml"
model_bin_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo/2022.3/models_bin/1/horizontal-text-detection-0001/FP32/horizontal-text-detection-0001.bin"

## Download model weights

```
base_model_dir = Path("./model").expanduser()

model_name = "horizontal-text-detection-0001"
model_xml_name = f'{model_name}.xml'
model_bin_name = f'{model_name}.bin'

model_xml_path = base_model_dir / model_xml_name
model_bin_path = base_model_dir / model_bin_name

if not model_xml_path.exists():
    model_xml_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo/
    model_bin_url = "https://storage.openvinotoolkit.org/repositories/open_model_zoo/

    download_file(model_xml_url, model_xml_name, base_model_dir)
    download_file(model_bin_url, model_bin_name, base_model_dir)
else:
    print(f'{model_name} already downloaded to {base_model_dir}')
```

```
model₩horizontal-text-detection-0001.xml: 100% 680k/680k [00:05<00:00, 283kB/s]
model₩horizontal-text-detection-0001.bin: 100% 7.39M/7.39M [00:02<00:00, 7.60MB/s]
```

# First steps : Hello Object Detection

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Select inference device

select device from dropdown list for running inference using OpenVINO

```
import ipywidgets as widgets

ie = Core()
device = widgets.Dropdown(
    options=ie.available_devices + ["AUTO"],
    value='AUTO',
    description='Device:',
    disabled=False,
)

device
```

Device: AUTO

## Load the Model

```
ie = Core()

model = ie.read_model(model=model_xml_path)
compiled_model = ie.compile_model(model=model, device_name="CPU")

input_layer_ir = compiled_model.input(0)
output_layer_ir = compiled_model.output("boxes")
```

```
output_layer_ir
```

```
<ConstOutput: names[boxes] shape[..100,5] type: f32>
```

## Load an Image

```
# Text detection models expect an image in BGR format.
image = cv2.imread("../data/image/intel_rnb.jpg")

# N,C,H,W = batch size, number of channels, height, width.
N, C, H, W = input_layer_ir.shape

# Resize the image to meet network expected input sizes.
resized_image = cv2.resize(image, (W, H))

# Reshape to the network input shape.
input_image = np.expand_dims(resized_image.transpose(2, 0, 1), 0)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB));
```

# First steps : Hello Object Detection

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Do Inference

```
# Create an inference request.
boxes = compiled_model([input_image])[output_layer_ir]

# Remove zero only boxes.
boxes = boxes[~np.all(boxes == 0, axis=1)]
```

```
boxes
```

```
array([[3.99946808e+02, 8.11520157e+01, 5.61839111e+02, 1.36155273e+02,
        5.36778033e-01],
       [2.61954102e+02, 6.84256058e+01, 3.85380066e+02, 1.20920616e+02,
        4.74904537e-01],
       [6.16452637e+02, 2.80120209e+02, 6.66263855e+02, 3.11769623e+02,
        4.49359655e-01],
       [2.07640488e+02, 6.28020172e+01, 2.34421341e+02, 1.07031281e+02,
        3.74582469e-01],
       [5.17440063e+02, 5.56030579e+02, 5.49294922e+02, 5.87316589e+02,
        3.26068223e-01],
       [2.23853302e+01, 4.54875183e+01, 1.88720001e+02, 1.02219788e+02,
        3.04750621e-01]], dtype=float32)
```

# First steps : Hello Object Detection

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Visualize Results

```python
# For each detection, the description is in the [x_min, y_min, x_max, y_max, conf] format:
# The image passed here is in BGR format with changed width and height. To display it in colors expected by matplotlib, use cvtColor function
def convert_result_to_image(bgr_image, resized_image, boxes, threshold=0.3, conf_labels=True):
    # Define colors for boxes and descriptions.
    colors = {"red": (255, 0, 0), "green": (0, 255, 0)}

    # Fetch the image shapes to calculate a ratio.
    (real_y, real_x), (resized_y, resized_x) = bgr_image.shape[:2], resized_image.shape[:2]
    ratio_x, ratio_y = real_x / resized_x, real_y / resized_y

    # Convert the base image from BGR to RGB format.
    rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)

    # Iterate through non-zero boxes.
    for box in boxes:
        # Pick a confidence factor from the last place in an array.
        conf = box[-1]
        if conf > threshold:
            # Convert float to int and multiply corner position of each box by x and y ratio.
            # If the bounding box is found at the top of the image,
            # position the upper box bar little lower to make it visible on the image.
            (x_min, y_min, x_max, y_max) = [
                int(max(corner_position * ratio_y, 10)) if idx % 2
                else int(corner_position * ratio_x)
                for idx, corner_position in enumerate(box[:-1])
            ]

            # Draw a box based on the position, parameters in rectangle function are: image, start_point, end_point, color, thickness.
            rgb_image = cv2.rectangle(rgb_image, (x_min, y_min), (x_max, y_max), colors["green"], 3)
```

# First steps : Hello Object Detection

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
            # Add text to the image based on position and confidence.
            # Parameters in text function are: image, text, bottom-left_corner_textfield, font, font_scale, color, thickness, line_type.
            if conf_labels:
                rgb_image = cv2.putText(
                    rgb_image,
                    f"{conf:.2f}",
                    (x_min, y_min - 10),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.8,
                    colors["red"],
                    1,
                    cv2.LINE_AA,
                )

    return rgb_image
```

```
plt.figure(figsize=(10, 6))
plt.axis("off")
plt.imshow(convert_result_to_image(image, resized_image, boxes, conf_labels=False));
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Convert a PyTorch Model to OpenVINO™ IR

This tutorial demonstrates step-by-step instructions on how to do inference on a PyTorch classification model using OpenVINO Runtime. Starting from OpenVINO 2023.0 release, OpenVINO supports direct PyTorch model conversion without an intermediate step to convert them into ONNX format. In order, if you try to use the lower OpenVINO version or prefer to use ONNX, please check this tutorial.

In this tutorial, we will use the RegNetY_800MF model from torchvision to demonstrate how to convert PyTorch models to OpenVINO Intermediate Representation.

The RegNet model was proposed in Designing Network Design Spaces by Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, Piotr Dollár. The authors design search spaces to perform Neural Architecture Search (NAS). They first start from a high dimensional search space and iteratively reduce the search space by empirically applying constraints based on the best-performing models sampled by the current search space. Instead of focusing on designing individual network instances, authors design network design spaces that parametrize populations of networks. The overall process is analogous to the classic manual design of networks but elevated to the design space level. The RegNet design space provides simple and fast networks that work well across a wide range of flop regimes.

## Prerequisites

Install notebook dependecies

```
!pip install -q "openvino-dev>=2023.0.0"
```

Download input data and label map

```
import requests
from pathlib import Path
from PIL import Image

MODEL_DIR = Path("model")
DATA_DIR = Path("data")

MODEL_DIR.mkdir(exist_ok=True)
DATA_DIR.mkdir(exist_ok=True)
MODEL_NAME = "regnet_y_800mf"

image = Image.open(requests.get("https://farm9.staticflickr.com/8225/8511402100_fea1

labels_file = DATA_DIR / "imagenet_2012.txt"

if not labels_file.exists():
    resp = requests.get("https://raw.githubusercontent.com/openvinotoolkit/open_mode
    with labels_file.open("wb") as f:
        f.write(resp.content)

imagenet_classes = labels_file.open("r").read().splitlines()
```

resp = requests.get("https://raw.githubusercontent.com/openvinotoolkit/open_model_zoo/master/data/dataset_classes/imagenet_2012.txt")

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
resp.content[0:500]
```

```
b'n01440764 tench, Tinca tinca\nn01443537 goldfish, Carassius auratus\nn01484850 gre
at white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias\nn0
1491361 tiger shark, Galeocerdo cuvieri\nn01494475 hammerhead, hammerhead shark\nn01
496331 electric ray, crampfish, numbfish, torpedo\nn01498041 stingray\nn01514668 coc
k\nn01514859 hen\nn01518878 ostrich, Struthio camelus\nn01530575 brambling, Fringill
a montifringilla\nn01531178 goldfinch, Carduelis carduelis\nn01532829 house finch, l
innet, Car'
```

```
imagenet_classes[0:10]
```

```
['n01440764 tench, Tinca tinca',
 'n01443537 goldfish, Carassius auratus',
 'n01484850 great white shark, white shark, man-eater, man-eating shark, Carcharodon
carcharias',
 'n01491361 tiger shark, Galeocerdo cuvieri',
 'n01494475 hammerhead, hammerhead shark',
 'n01496331 electric ray, crampfish, numbfish, torpedo',
 'n01498041 stingray',
 'n01514668 cock',
 'n01514859 hen',
 'n01518878 ostrich, Struthio camelus']
```

## Load PyTorch Model

Generally, PyTorch models represent an instance of the `torch.nn.Module` class, initialized by a state dictionary with model weights. Typical steps for getting a pre-trained model:

1. Create an instance of a model class
2. Load checkpoint state dict, which contains pre-trained model weights
3. Turn the model to evaluation for switching some operations to inference mode

The `torchvision` module provides a ready-to-use set of functions for model class initialization. We will use `torchvision.models.regnet_y_800mf`. You can directly pass pre-trained model weights to the model initialization function using the weights enum `RegNet_Y_800MF_Weights.DEFAULT`.

```
import torchvision

# get default weights using available weights Enum for model
weights = torchvision.models.RegNet_Y_800MF_Weights.DEFAULT

# create model topology and load weights
model = torchvision.models.regnet_y_800mf(weights=weights)

# switch model to inference mode
model.eval();
```

```
Downloading: "https://download.pytorch.org/models/regnet_y_800mf-58fc7688.pth" to
C:\Users\sky/.cache\torch\hub\checkpoints\regnet_y_800mf-58fc7688.pth
```

100% 24.8M/24.8M [00:02<00:00, 11.8MB/s]

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Prepare Input Data

The code below demonstrates how to preprocess input data using a model-specific transforms module from `torchvision`. After transformation, we should concatenate images into batched tensor, in our case, we will run the model with batch 1, so we just unsqueeze input on the first dimension.

```
import torch

# Initialize the Weight Transforms
preprocess = weights.transforms()

# Apply it to the input image
img_transformed = preprocess(image)

# Add batch dimension to image tensor
input_tensor = img_transformed.unsqueeze(0)
```

```
preprocess
```

```
ImageClassification(
    crop_size=[224]
    resize_size=[232]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BILINEAR
)
```

```
print(image.size)
print(img_transformed.shape)
print(input_tensor.shape)
```

```
(640, 480)
torch.Size([3, 224, 224])
torch.Size([1, 3, 224, 224])
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Run PyTorch Model Inference (1)

The model returns a vector of probabilities in raw logits format, softmax can be applied to get normalized values in the [0, 1] range. For a demonstration that the output of the original model and OpenVINO converted is the same, we defined a common postprocessing function which can be reused later.

```
import numpy as np
from scipy.special import softmax

# Perform model inference on input tensor
result = model(input_tensor)

# Postprocessing function for getting results in the same way for both PyTorch model inference and OpenVINO
def postprocess_result(output_tensor:np.ndarray, top_k:int = 5):
    """
    Posprocess model results. This function applied sofrmax on output tensor and returns specified top_k number of labels with highest probabil
    Parameters:
      output_tensor (np.ndarray): model output tensor with probabilities
      top_k (int, *optional*, default 5): number of labels with highest probability for return
    Returns:
      topk_labels: label ids for selected top_k scores
      topk_scores: selected top_k highest scores predicted by model
    """
    softmaxed_scores = softmax(output_tensor, -1)[0]
    topk_labels = np.argsort(softmaxed_scores)[-top_k:][::-1]
    topk_scores = softmaxed_scores[topk_labels]
    return topk_labels, topk_scores

# Postprocess results
top_labels, top_scores = postprocess_result(result.detach().numpy())
```

```
print(top_labels, top_scores)
```

```
[282 285 508 281 588] [0.2590592  0.10256905 0.09220967 0.09086117 0.02345678]
```

파이토치에서 Tensor 객체의 detach() 메소드는 현재 Tensor 객체와 동일한 데이터를 가지지만 연산 그래프(Computational Graph)에서 분리된 새로운 Tensor 객체를 생성합니다.

이 메소드는 일반적으로 Tensor 객체를 다른 Tensor 객체로 변환하고자 할 때 사용됩니다.

예를 들어, 주어진 Tensor 객체에 대한 연산의 결과로 생성된 새로운 Tensor 객체가 있을 때, 이 새로운 Tensor 객체를 사용하여 추가적인 계산을 수행하고자 할 때, 기존 Tensor 객체의 연산 그래프와의 의존성을 제거하여 메모리 사용량을 줄이고 계산 속도를 향상시키는 데 유용합니다.

detach() 메소드는 requires_grad 속성을 False로 설정하여 기존 Tensor 객체와 다르게 자동 미분 기능에서 제외됩니다.

따라서 detach() 메소드를 사용하여 생성된 Tensor 객체는 그라디언트(gradient) 계산에 사용되지 않으며, 그라디언트를 계산하지 않는 모델에서 중간 출력 값을 얻을 때 특히 유용합니다.

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

## Benchmark PyTorch Model Inference

```
%%timeit

# Run model inference
model(input_tensor)
```

```
29.1 ms ± 729 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

## (2) Convert PyTorch Model to OpenVINO Intermediate Representation

Starting from the 2023.0 release OpenVINO supports direct PyTorch models conversion to OpenVINO Intermediate Representation (IR) format. Model Optimizer Python API should be used for these purposes. More details regarding PyTorch model conversion can be found in OpenVINO documentation

> **Note**: Please, take into account that direct support PyTorch models conversion is an experimental feature. Model coverage will be increased in the next releases. For cases, when PyTorch model conversion failed, you still can try to export the model to ONNX format. Please refer to this tutorial which explains how to convert PyTorch model to ONNX, then to OpenVINO

The `convert_model` function accepts the PyTorch model object and returns the `openvino.runtime.Model` instance ready to load on a device using `core.compile_model` or save on disk for next usage using `openvino.runtime.serialize`. Optionally, we can provide additional parameters, such as:

- `compress_to_fp16` - flag to perform model weights compression into FP16 data format. It may reduce the required space for model storage on disk and give speedup for inference devices, where FP16 calculation is supported.
- `example_input` - input data sample which can be used for model tracing.
- `input_shape` - the shape of input tensor for conversion

and any other advanced options supported by Model Optimizer Python API. More details can be found on this page

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

```
from openvino.tools import mo
from openvino.runtime import Core, serialize

# Create OpenVINO Core object instance
core = Core()

# Convert model to openvino.runtime.Model object
ov_model = mo.convert_model(model)

# Save openvino.runtime.Model object on disk
serialize(ov_model, MODEL_DIR / f"{MODEL_NAME}_dynamic.xml")

ov_model
```

```
<Model: 'Model150'
inputs[
<ConstOutput: names[1, x.1, x] shape[?,3,?,?] type: f32>
]
outputs[
<ConstOutput: names[401, x.21] shape[?,1000] type: f32>
]>
```

## Select inference device

select device from dropdown list for running inference using OpenVINO

```
import ipywidgets as widgets

device = widgets.Dropdown(
    options=core.available_devices + ["AUTO"],
    value='AUTO',
    description='Device:',
    disabled=False,
)

device
```

Device: AUTO

```
# Load OpenVINO model on device
compiled_model = core.compile_model(ov_model, device.value)
compiled_model
```

```
<CompiledModel:
inputs[
<ConstOutput: names[1, x.1, x] shape[?,3,?,?] type: f32>
]
outputs[
<ConstOutput: names[401, x.21] shape[?,1000] type: f32>
]>
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Run OpenVINO Model Inference

```
# Run model inference
result = compiled_model(input_tensor)[0]

# Posptorcess results
top_labels, top_scores = postprocess_result(result)

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

## (3) Convert PyTorch Model with Static Input Shape

The default conversion path preserves dynamic input shapes, in order if you want to convert the model with static shapes, you can explicitly specify it during conversion using the `input_shape` parameter or reshape the model into the desired shape after conversion. For the model reshaping example please check the following tutorial.

```
# Convert model to openvino.runtime.Model object
ov_model = mo.convert_model(model, input_shape=[[1,3,224,224]])
# Save openvino.runtime.Model object on disk
serialize(ov_model, MODEL_DIR / f"{MODEL_NAME}_static.xml")
ov_model
```

```
<Model: 'Model120'
inputs[
<ConstOutput: names[1, x.1, x] shape[1,3,224,224] type: f32>
]
outputs[
<ConstOutput: names[355] shape[1,1000] type: f32>
]>
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Select inference device *Same*

select device from dropdown list for running inference using OpenVINO

```
device
```

Device: AUTO

```
# Load OpenVINO model on device
compiled_model = core.compile_model(ov_model, device.value)
compiled_model
```

```
<CompiledModel:
inputs[
<ConstOutput: names[1, x.1, x] shape[1,3,224,224] type: f32>
]
outputs[
<ConstOutput: names[355] shape[1,1000] type: f32>
]>
```

Now, we can see that input of our converted model is tensor of shape [1, 3, 224, 224] instead of [?, 3, ?, ?] reported by previously converted model.

## Benchmark OpenVINO Model Inference with Static Input Shape

```
%%timeit

compiled_model(input_tensor)
```

```
5.83 ms ± 815 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

## Run OpenVINO Model Inference with Static Input Shape *Same*

```
# Run model inference
result = compiled_model(input_tensor)[0]

# Posptorcess results
top_labels, top_scores = postprocess_result(result)

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Convert TorchScript Model to OpenVINO Intermediate Representation (4)

TorchScript is a way to create serializable and optimizable models from PyTorch code. Any TorchScript program can be saved from a Python process and loaded in a process where there is no Python dependency. More details about TorchScript can be found in PyTorch documentation.

There are 2 possible ways to convert the PyTorch model to TorchScript:

- `torch.jit.script` - Scripting a function or `nn.Module` will inspect the source code, compile it as TorchScript code using the TorchScript compiler, and return a `ScriptModule` or `ScriptFunction`.
- `torch.jit.trace` - Trace a function and return an executable or `ScriptFunction` that will be optimized using just-in-time compilation.

Let's consider both approaches and their conversion into OpenVINO IR.

### Scriped Model (4-1)

`torch.jit.script` inspects model source code and compiles it to `ScriptModule`. After compilation model can be used for inference or saved on disk using the `torch.jit.save` function and after that restored with `torch.jit.load` in any other environment without the original PyTorch model code definitions.

TorchScript itself is a subset of the Python language, so not all features in Python work, but TorchScript provides enough functionality to compute on tensors and do control-dependent operations. For a complete guide, see the TorchScript Language Reference.

```
# Get model path
scripted_model_path = MODEL_DIR / f"{MODEL_NAME}_scripted.pth"

# Compile and save model if it has not been compiled before or load compiled model
if not scripted_model_path.exists():
    scripted_model = torch.jit.script(model)
    torch.jit.save(scripted_model, scripted_model_path)
else:
    scripted_model = torch.jit.load(scripted_model_path)

# Run scripted model inference
result = scripted_model(input_tensor)

# Postprocess results
top_labels, top_scores = postprocess_result(result.detach().numpy())

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

### Benchmark Scripted Model Inference

```
%%timeit

scripted_model(input_tensor)
```

```
31 ms ± 10.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Convert PyTorch Scripted Model to OpenVINO Intermediate Representation

The conversion step for the scripted model to OpenVINO IR is similar to the original PyTorch model.

```
# Convert model to openvino.runtime.Model object
ov_model = mo.convert_model(scripted_model)

# Load OpenVINO model on device
compiled_model = core.compile_model(ov_model, device.value)

# Run OpenVINO model inference
result = compiled_model(input_tensor, device.value)[0]

# Postprocess results
top_labels, top_scores = postprocess_result(result)

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

## Benchmark OpenVINO Model Inference Converted From Scripted Model

```
%%timeit

compiled_model(input_tensor)
```

```
5.93 ms ± 312 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Traced Model (4-2)

Using `torch.jit.trace`, you can turn an existing module or Python function into a TorchScript `ScriptFunction` or `ScriptModule`. You must provide example inputs, and model will be executed, recording the operations performed on all the tensors.

- The resulting recording of a standalone function produces ScriptFunction.
- The resulting recording of nn.Module.forward or nn.Module produces ScriptModule.

In the same way like scripted model, traced model can be used for inference or saved on disk using `torch.jit.save` function and after that restored with `torch.jit.load` in any other environment without original PyTorch model code definitions.

```
# Get model path
traced_model_path = MODEL_DIR / f"{MODEL_NAME}_traced.pth"

# Trace and save model if it has not been traced before or load traced model
if not traced_model_path.exists():
    traced_model = torch.jit.trace(model, example_inputs=input_tensor)
    torch.jit.save(traced_model, traced_model_path)
else:
    traced_model = torch.jit.load(traced_model_path)

# Run traced model inference
result = traced_model(input_tensor)

# Postprocess results
top_labels, top_scores = postprocess_result(result.detach().numpy())

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

## Benchmark Traced Model Inference

```
%%timeit

traced_model(input_tensor)
```

```
31.9 ms ± 14.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

# PyTorch to OpenVINO™ IR Tutorial

https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks

## Convert PyTorch Traced Model to OpenVINO Intermediate Representation

The conversion step for a traced model to OpenVINO IR is similar to the original PyTorch model.

```
# Convert model to openvino.runtime.Model object
ov_model = mo.convert_model(traced_model)

# Load OpenVINO model on device
compiled_model = core.compile_model(ov_model, device.value)

# Run OpenVINO model inference
result = compiled_model(input_tensor)[0]

# Postprocess results
top_labels, top_scores = postprocess_result(result)

# Show results
display(image)
for idx, (label, score) in enumerate(zip(top_labels, top_scores)):
    _, predicted_label = imagenet_classes[label].split(" ", 1)
    print(f"{idx + 1}: {predicted_label} - {score * 100 :.2f}%")
```

```
1: tiger cat - 25.91%
2: Egyptian cat - 10.26%
3: computer keyboard, keypad - 9.22%
4: tabby, tabby cat - 9.09%
5: hamper - 2.35%
```

## Benchmark OpenVINO Model Inference Converted From Traced Model

```
%%timeit

compiled_model(input_tensor)[0]
```

```
6.88 ms ± 587 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

# Convert and Optimize YOLOv8 with OpenVINO™

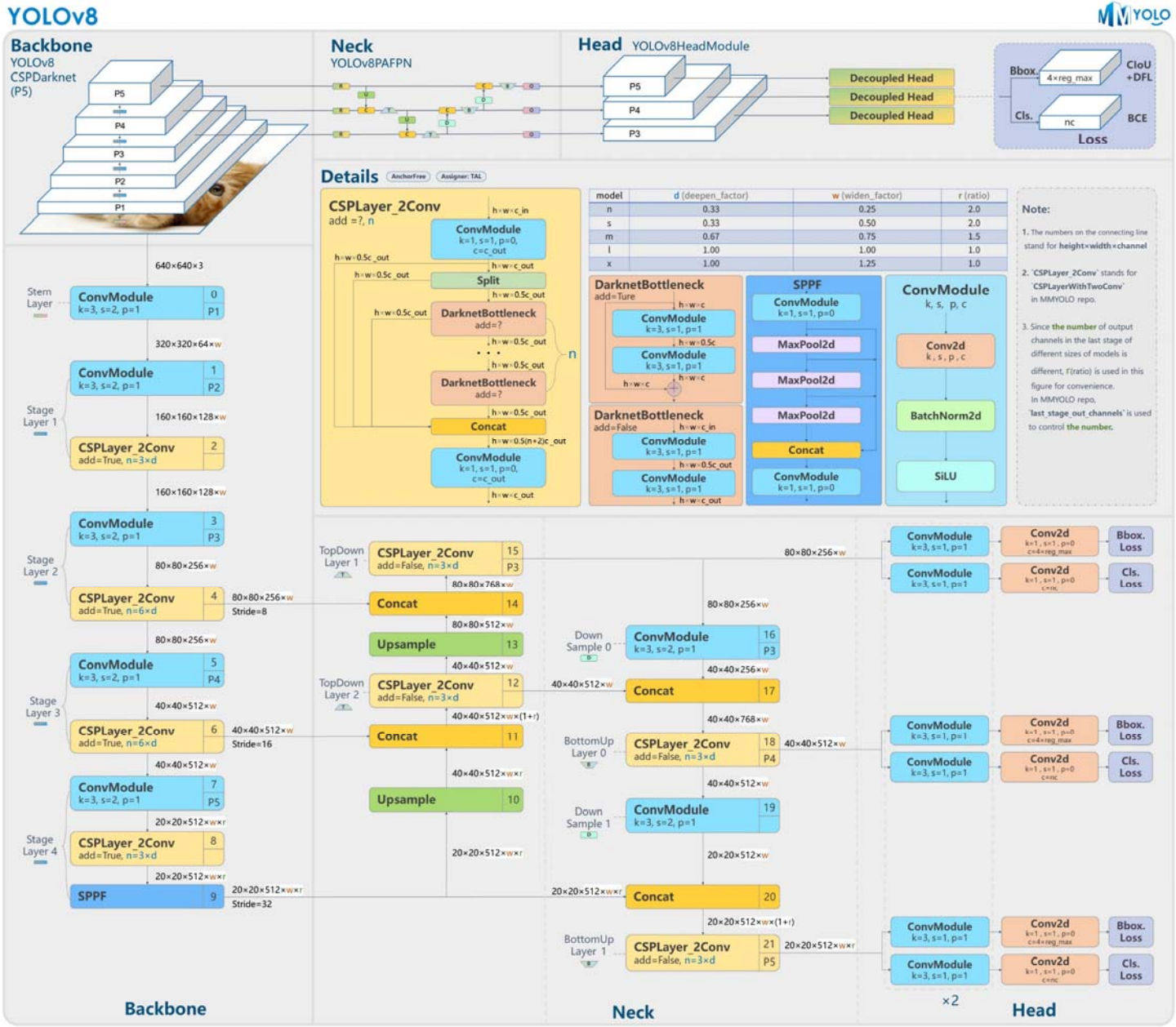
https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

- **YOLOv8** is the next major update from **YOLOv5**, open sourced by **ultralytics** on 2023.1.10, and now supports **image classification**, **object detection** and **instance segmentation** tasks.
- According to the official description, **Ultralytics YOLOv8** is the latest version of the YOLO object detection and image segmentation model developed by Ultralytics.
- YOLOv8 is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. These include a **new backbone network**, a **new anchor-free detection head**, and a **new loss function**.
- YOLOv8 is also highly efficient and can be run on a variety of hardware platforms, from CPUs to GPUs.

https://github.com/ultralytics/ultralytics


- **Core features and modifications of YOLOv8**

1) A new state-of-the-art (SOTA) model is proposed, featuring an **object detection model** for **P5 640** and **P6 1280** resolutions, as well as a **YOLACT-based instance segmentation** model. The model also includes different size options with **N/S/M/L/X scales**, similar to **YOLOv5**, to cater to various scenarios.
2) The **backbone network and neck module** are based on the **YOLOv7 ELAN design concept**, replacing the C3 module of YOLOv5 with the C2f module. However, there are a lot of operations such as Split and Concat in this C2f module that are not as deployment-friendly as before.
3) The **Head module** has been updated to the current mainstream decoupled structure, **separating the classification and detection heads**, and **switching from Anchor-Based to Anchor-Free**.
4) The **loss calculation** adopts the **TaskAlignedAssigner in TOOD** and introduces the **Distribution Focal Loss** to the regression loss.
5) In the **data augmentation** part, **Mosaic** is closed in **the last 10 training epoch**, which is the same as YOLOX training part.


YOLOv8
MMYOLO
Backbone
YOLOv8
CSPDarknet
(P5)
Neck
YOLOv8PAFPN
Head YOLOv8HeadModule
Decoupled Head
Decoupled Head
Decoupled Head
Bbox.
4×reg_max
CIoU
+DFL
Cls.
nc
BCE
Loss
Details
AnchorFree
Assigner: TAL
CSPLayer_2Conv
add =?, n
ConvModule
k=1, s=1, p=0,
c=c_out
Split
DarknetBottleneck
add=?
Concat
ConvModule
k=1, s=1, p=0,
c=c_out
model
d (deepen_factor)
w (widen_factor)
r (ratio)
n 0.33 0.25 2.0
s 0.33 0.50 2.0
m 0.67 0.75 1.5
l 1.00 1.00 1.0
x 1.00 1.25 1.0
DarknetBottleneck
add=Ture
ConvModule
k=3, s=1, p=1
DarknetBottleneck
add=False
SPPF
ConvModule
k=1, s=1, p=0
MaxPool2d
MaxPool2d
MaxPool2d
Concat
ConvModule
k=1, s=1, p=0
ConvModule
k, s, p, c
Conv2d
k , s , p , c
BatchNorm2d
SiLU
Note:
1. The numbers on the connecting line stand for height×width×channel
2. `CSPLayer_2Conv` stands for `CSPLayerWithTwoConv` in MMYOLO repo.
3. Since the number of output channels in the last stage of different sizes of models is different, r(ratio) is used in this figure for convenience. In MMYOLO repo, `last_stage_out_channels` is used to control the number.
640×640×3
Stem Layer
ConvModule
k=3, s=2, p=1
0
P1
320×320×64×w
ConvModule
k=3, s=2, p=1
1
P2
Stage Layer 1
160×160×128×w
CSPLayer_2Conv
add=True, n=3×d
2
160×160×128×w
ConvModule
k=3, s=2, p=1
3
P3
Stage Layer 2
80×80×256×w
CSPLayer_2Conv
add=True, n=6×d
4
80×80×256×w
Stride=8
80×80×256×w
ConvModule
k=3, s=2, p=1
5
P4
Stage Layer 3
40×40×512×w
CSPLayer_2Conv
add=True, n=6×d
6
40×40×512×w
Stride=16
40×40×512×w
ConvModule
k=3, s=2, p=1
7
P5
20×20×512×w×r
Stage Layer 4
CSPLayer_2Conv
add=True, n=3×d
8
20×20×512×w×r
SPPF
9
20×20×512×w×r
Stride=32
TopDown Layer 1
CSPLayer_2Conv
add=False, n=3×d
15
P3
80×80×768×w
Concat
14
80×80×512×w
Upsample
13
40×40×512×w
TopDown Layer 2
CSPLayer_2Conv
add=False, n=3×d
12
40×40×512×w×(1+r)
Concat
11
40×40×512×w×r
Upsample
10
20×20×512×w×r
80×80×256×w
Down Sample 0
ConvModule
k=3, s=2, p=1
16
P3
40×40×256×w
40×40×512×w
Concat
17
40×40×768×w
BottomUp Layer 0
CSPLayer_2Conv
add=False, n=3×d
18
P4
40×40×512×w
40×40×512×w
Down Sample 1
ConvModule
k=3, s=2, p=1
19
20×20×512×w
20×20×512×w×r
Concat
20
20×20×512×w×(1+r)
BottomUp Layer 1
CSPLayer_2Conv
add=False, n=3×d
21
P5
20×20×512×w×r
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=4×reg_max
Bbox. Loss
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=nc
Cls. Loss
×2
Backbone
Neck
Head

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

▼ Install

Pip install the ultralytics package including all requirements in a Python>=3.8 environment with PyTorch>=1.7.


```
pip install ultralytics
```

pip install ultralytics==**8.0.43**

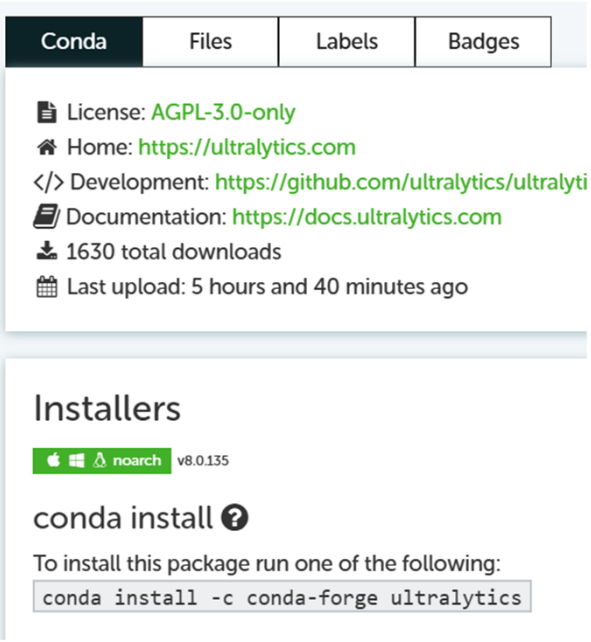
**openvino notebooks : YOLOv8 상에서 버전 8.0.43 으로 설치**

**Python**

YOLOv8 may also be used directly in a Python environment, and accepts the same arguments as in the CLI example above:

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
path = model.export(format="onnx")  # export the model to ONNX format
```

Models download automatically from the latest Ultralytics release. See YOLOv8 Python Docs for more examples.



https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

# Convert and Optimize YOLOv8 with OpenVINO™

The YOLOv8 algorithm developed by Ultralytics is a cutting-edge, state-of-the-art (SOTA) model that is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection, image segmentation, and image classification tasks.

YOLO stands for "You Only Look Once", it is a popular family of real-time object detection algorithms. The original YOLO object detector was first released in 2016. Since then, different versions and variants of YOLO have been proposed, each providing a significant increase in performance and efficiency. YOLOv8 builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. More details about its realization can be found in the original model repository.

Real-time object detection and instance segmentation are often used as key components in computer vision systems. Applications that use real-time object detection models include video analytics, robotics, autonomous vehicles, multi-object tracking and object counting, medical image analysis, and many others.

This tutorial demonstrates step-by-step instructions on how to run and optimize PyTorch YOLOv8 with OpenVINO. We consider the steps required for object detection and instance segmentation scenarios.

The tutorial consists of the following steps:

- Prepare the PyTorch model.
- Download and prepare a dataset.
- Validate the original model.
- Convert the PyTorch model to OpenVINO IR.
- Validate the converted model.
- Prepare and run optimization pipeline.
- Compare performance of the FP32 and quantized models.
- Compare accuracy of the FP32 and quantized models.

## 1) Get Pytorch model

Generally, PyTorch models represent an instance of the torch.nn.Module class, initialized by a state dictionary with model weights. We will use the YOLOv8 nano model (also known as `yolov8n`) pre-trained on a COCO dataset, which is available in this repo. Similar steps are also applicable to other YOLOv8 models. Typical steps to obtain a pre-trained model:

1. Create an instance of a model class.
2. Load a checkpoint state dict, which contains the pre-trained model weights.
3. Turn the model to evaluation for switching some operations to inference mode.

In this case, the creators of the model provide an API that enables converting the YOLOv8 model to ONNX and then to OpenVINO IR. Therefore, we do not need to do these steps manually.

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Prerequisites

Install necessary packages.

```
!pip install -q 'openvino-dev>=2023.0.0' 'nncf>=2.5.0'
!pip install -q 'ultralytics==8.0.43' onnx
```

```
ERROR: Invalid requirement: "'openvino-dev"
ERROR: Invalid requirement: "'ultralytics==8.0.43'"
```

```
from pathlib import Path

# Fetch the notebook utils script from the openvino_notebooks repo
import urllib.request
urllib.request.urlretrieve(
    url='https://raw.githubusercontent.com/openvinotoolkit/openvino_notebooks/main/
    filename='notebook_utils.py'
)

from notebook_utils import download_file, VideoPlayer
```

url='https://raw.githubusercontent.com/openvinotoolkit/openvino_notebooks/main/notebooks/utils/notebook_utils.py'

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

Define utility functions for drawing results

```python
from typing import Tuple, Dict
import cv2
import numpy as np
from PIL import Image
from ultralytics.yolo.utils.plotting import colors

def plot_one_box(box:np.ndarray, img:np.ndarray, color:Tuple[int, int, int] = None, mask:np.ndarray = None,
                 label:str = None, line_thickness:int = 5):
    """
    Helper function for drawing single bounding box on image
    Parameters:
        x (np.ndarray): bounding box coordinates in format [x1, y1, x2, y2]
        img (no.ndarray): input image
        color (Tuple[int, int, int], *optional*, None): color in BGR format for drawing box,
            if not specified will be selected randomly
        mask (np.ndarray, *optional*, None): instance segmentation mask polygon in format [N, 2],
            where N - number of points in contour, if not provided, only box will be drawn
        label (str, *optonal*, None): box label string, if not provided will not be provided as drowing result
        line_thickness (int, *optional*, 5): thickness for box drawing lines
    """
    # Plots one bounding box on image img
    tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1  # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))
    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA)  # filled
        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255], thickness=tf, lineType=cv2.LINE_AA)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
    if mask is not None:
        image_with_mask = img.copy()
        mask
        cv2.fillPoly(image_with_mask, pts=[mask.astype(int)], color=color)
        img = cv2.addWeighted(img, 0.5, image_with_mask, 0.5, 1)
    return img
```

```
def draw_results(results:Dict, source_image:np.ndarray, label_map:Dict):
    """
    Helper function for drawing bounding boxes on image
    Parameters:
        image_res (np.ndarray): detection predictions in format [x1, y1, x2, y2, score, label_id]
        source_image (np.ndarray): input image for drawing
        label_map; (Dict[int, str]): label_id to class name mapping
    Returns:

    """
    boxes = results["det"]
    masks = results.get("segment")
    h, w = source_image.shape[:2]
    for idx, (*xyxy, conf, lbl) in enumerate(boxes):
        label = f'{label_map[int(lbl)]} {conf:.2f}'
        mask = masks[idx] if masks is not None else None
        source_image = plot_one_box(xyxy, source_image, mask=mask, label=label, color=colors(int(lbl)), line_thickness=1)
    return source_image
```

```
WARNING  Ultralytics settings reset to defaults. This is normal and may be due to a recent ultralytics package update, but may h
ave overwritten previous settings.
View and update settings with 'yolo settings' or at 'C:\Users\sky\AppData\Roaming\Ultralytics\settings.yaml'
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
# Download a test sample
IMAGE_PATH = Path('./data/coco_bike.jpg')
download_file(
    url='https://storage.openvinotoolkit.org/repositories/openvino_notebooks/data/
    filename=IMAGE_PATH.name,
    directory=IMAGE_PATH.parent
)
```

```
'data\coco_bike.jpg' already exists.
WindowsPath('C:/Users/sky/Documents/Openvino/openvino_notebooks/notebooks/230-yolo
v8-optimization/data/coco_bike.jpg')
```

url='https://storage.openvinotoolkit.org/repositories/openvino_notebooks/data/data/image/coco_bike.jpg'

## 2) Instantiate model

There are several models available in the original repository, targeted for different tasks. For loading the model, required to specify a path to the model checkpoint. It can be some local path or name available on models hub (in this case model checkpoint will be downloaded automatically).

Making prediction, the model accepts a path to input image and returns list with Results class object. Results contains boxes for object detection model and boxes and masks for segmentation model. Also it contains utilities for processing results, for example, `plot()` method for drawing.

Let us consider the examples:

```
models_dir = Path('./models')
models_dir.mkdir(exist_ok=True)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Object detection

```
from ultralytics import YOLO

DET_MODEL_NAME = "yolov8n"

det_model = YOLO(models_dir / f'{DET_MODEL_NAME}.pt')
label_map = det_model.model.names

res = det_model(IMAGE_PATH)
Image.fromarray(res[0].plot()[:, :, ::-1])
```

```
Ultralytics YOLOv8.0.43  Python-3.8.6 torch-1.13.1+cpu CPU
YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFLOPs

image 1/1 C:\Users\sky\Documents\Openvino\openvino_notebooks\notebooks\230-yolov8-optimization\data\coco_bike.jpg: 480x640 2 bicycles, 2 cars, 1 dog, 98.0ms
Speed: 1.0ms preprocess, 98.0ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)
```

## Instance Segmentation:

```
SEG_MODEL_NAME = "yolov8n-seg"

seg_model = YOLO(models_dir / f'{SEG_MODEL_NAME}.pt')
res = seg_model(IMAGE_PATH)
Image.fromarray(res[0].plot()[:, :, ::-1])
```

```
Ultralytics YOLOv8.0.43  Python-3.8.6 torch-1.13.1+cpu CPU
YOLOv8n-seg summary (fused): 195 layers, 3404320 parameters, 0 gradients, 12.6 GFLOPs

image 1/1 C:\Users\sky\Documents\Openvino\openvino_notebooks\notebooks\230-yolov8-optimization\data\coco_bike.jpg: 480x640 1 bicycle, 2 cars, 1 dog, 103.0ms
Speed: 1.0ms preprocess, 103.0ms inference, 4.0ms postprocess per image at shape (1, 3, 640, 640)
```

/ ··· / 230-yolov8-optimization / models /

| Name |
|---|
| yolov8n_openvino_int8_model |
| yolov8n_openvino_model |
| yolov8n-seg_openvino_int8_model |
| yolov8n-seg_openvino_model |
| yolov8n-seg.onnx |
| yolov8n-seg.pt |
| yolov8n.onnx |
| yolov8n.pt |

/ ··· / models / yolov8n_openvino_int8_model /

| Name |
|---|
| yolov8n_with_preprocess.bin |
| yolov8n_with_preprocess.xml |
| yolov8n.bin |
| yolov8n.xml |

/ ··· / models / yolov8n_openvino_model /

| Name |
|---|
| metadata.yaml |
| yolov8n.bin |
| yolov8n.xml |

/ ··· / models / yolov8n-seg_openvino_int8_model /

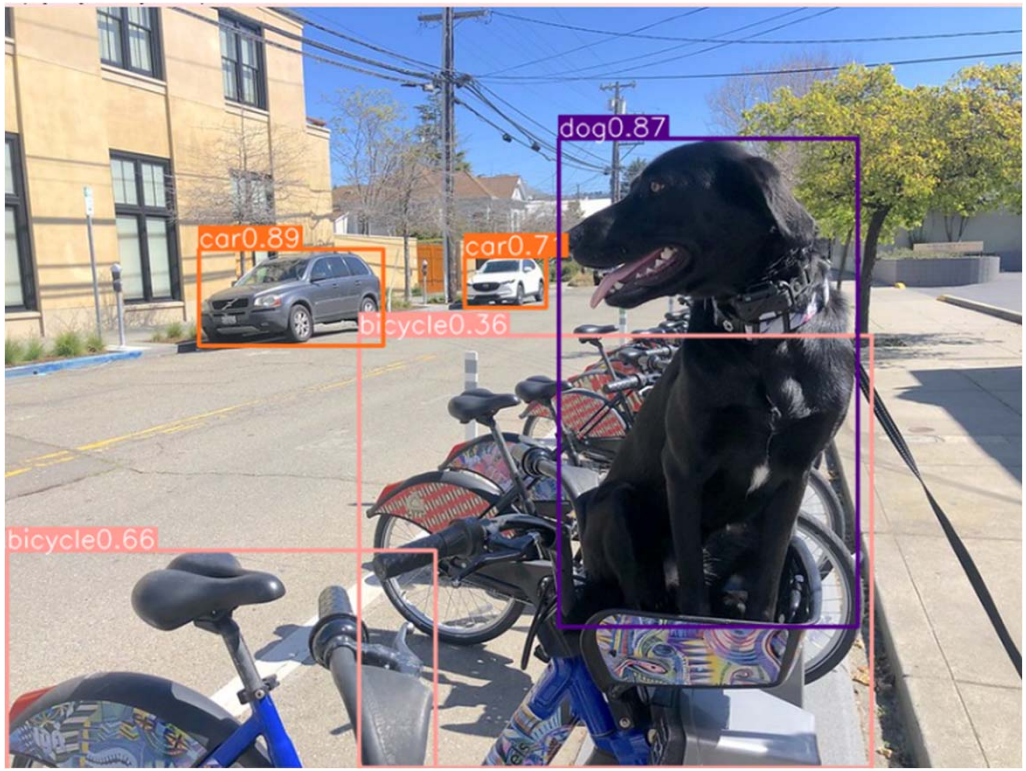
| Name |
|---|
| yolov8n-seg.bin |
| yolov8n-seg.xml |

/ ··· / models / yolov8n-seg_openvino_model /

| Name | Last Modified |
|---|---|
| metadata.yaml | 어제 |
| yolov8n-seg.bin | 어제 |
| yolov8n-seg.xml | 어제 |

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1




Obejct Detection


Instance Segmentation

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Convert model to OpenVINO IR 3)

YOLOv8 provides API for convenient model exporting to different formats including OpenVINO IR. `model.export` is responsible for model conversion. We need to specify the format, and additionally, we can preserve dynamic shapes in the model.

det_model_path = models\yolov8n_openvino_model\yolov8n.xml

seg_model_path = models\yolov8n-seg_openvino_model\yolov8n-seg.xml

```
# object detection model
det_model_path = models_dir / f"{DET_MODEL_NAME}_openvino_model/{DET_MODEL_NAME}.xml"
if not det_model_path.exists():
    det_model.export(format="openvino", dynamic=True, half=False)
```

```
Ultralytics YOLOv8.0.43  Python-3.8.6 torch-1.13.1+cpu CPU

PyTorch: starting from models\yolov8n.pt with input shape (1, 3, 640, 640) BCHW and output shape(s) (1, 84, 8400) (6.2 MB)

ONNX: starting export with onnx 1.14.0...
ONNX: export success  1.0s, saved as models\yolov8n.onnx (12.1 MB)

OpenVINO: starting export with openvino 2023.0.0-10926-b4452d56304-releases/2023/0...
OpenVINO: export success  2.4s, saved as models\yolov8n_openvino_model\ (12.3 MB)

Export complete (3.6s)
Results saved to C:\Users\sky\Documents\Openvino\openvino_notebooks\notebooks\230-yolov8-optimization\models
Predict:         yolo predict task=detect model=models\yolov8n_openvino_model imgsz=640
Validate:        yolo val task=detect model=models\yolov8n_openvino_model imgsz=640 data=coco.yaml
Visualize:       https://netron.app
```

```
# instance segmentation model
seg_model_path = models_dir / f"{SEG_MODEL_NAME}_openvino_model/{SEG_MODEL_NAME}.xml"
if not seg_model_path.exists():
    seg_model.export(format="openvino", dynamic=True, half=False)
```

```
Ultralytics YOLOv8.0.43  Python-3.8.6 torch-1.13.1+cpu CPU

PyTorch: starting from models\yolov8n-seg.pt with input shape (1, 3, 640, 640) BCHW and output shape(s) ((1, 116, 8400), (1, 32, 160, 160)) (6.7 MB)

ONNX: starting export with onnx 1.14.0...
ONNX: export success  1.2s, saved as models\yolov8n-seg.onnx (13.1 MB)

OpenVINO: starting export with openvino 2023.0.0-10926-b4452d56304-releases/2023/0...
OpenVINO: export success  0.3s, saved as models\yolov8n-seg_openvino_model\ (13.3 MB)

Export complete (1.9s)
Results saved to C:\Users\sky\Documents\Openvino\openvino_notebooks\notebooks\230-yolov8-optimization\models
Predict:         yolo predict task=segment model=models\yolov8n-seg_openvino_model imgsz=640
Validate:        yolo val task=segment model=models\yolov8n-seg_openvino_model imgsz=640 data=coco.yaml
Visualize:       https://netron.app
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Verify model inference 4)

To test model work, we create inference pipeline similar to `model.predict` method. The pipeline consists of preprocessing step, inference of OpenVINO model and results post-processing to get results. The main difference in models for object detection and instance segmentation is postprocessing part. Input specification and preprocessing are common for both cases.

## Preprocessing 5)

Model input is a tensor with the `[-1, 3, -1, -1]` shape in the `N, C, H, W` format, where

- `N` - number of images in batch (batch size)
- `C` - image channels
- `H` - image height
- `W` - image width

The model expects images in RGB channels format and normalized in [0, 1] range. Although the model supports dynamic input shape with preserving input divisibility to 32, it is recommended to use static shapes, for example, 640x640 for better efficiency. To resize images to fit model size `letterbox`, resize approach is used, where the aspect ratio of width and height is preserved.

To keep a specific shape, preprocessing automatically enables padding.

```python
from typing import Tuple
from ultralytics.yolo.utils import ops
import torch
import numpy as np

def letterbox(img: np.ndarray, new_shape:Tuple[int, int] = (640, 640),
              color:Tuple[int, int, int] = (114, 114, 114), auto:bool = False,
              scale_fill:bool = False, scaleup:bool = False, stride:int = 32):
    """
    Resize image and padding for detection. Takes image as input,
    resizes image to fit into new shape with saving original aspect ratio and
    pads it to meet stride-multiple constraints

    Parameters:
      img (np.ndarray): image for preprocessing
      new_shape (Tuple(int, int)): image size after preprocessing in
                                   format [height, width]
      color (Tuple(int, int, int)): color for filling padded area
      auto (bool): use dynamic input size, only padding for stride constrins applied
      scale_fill (bool): scale image to fill new_shape
      scaleup (bool): allow scale image if it is lower then desired input size,
                      can affect model accuracy
      stride (int): input padding stride
    Returns:
      img (np.ndarray): image after preprocessing
      ratio (Tuple(float, float)): hight and width scaling ratio
      padding_size (Tuple(int, int)): height and width padding size
    """
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
    # Resize and pad image while meeting stride-multiple constraints
    shape = img.shape[:2]  # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    if not scaleup:  # only scale down, do not scale up (for better test mAP)
        r = min(r, 1.0)

    # Compute padding
    ratio = r, r  # width, height ratios
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding
    if auto:  # minimum rectangle
        dw, dh = np.mod(dw, stride), np.mod(dh, stride)  # wh padding
    elif scale_fill:  # stretch
        dw, dh = 0.0, 0.0
        new_unpad = (new_shape[1], new_shape[0])
        ratio = new_shape[1] / shape[1], new_shape[0] / shape[0]  # width, height rati

    dw /= 2  # divide padding into 2 sides
    dh /= 2

    if shape[::-1] != new_unpad:  # resize
        img = cv2.resize(img, new_unpad, interpolation=cv2.INTER_LINEAR)
    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
    img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT, value
    return img, ratio, (dw, dh)
```

```
def preprocess_image(img0: np.ndarray):
    """
    Preprocess image according to YOLOv8 input requirements.
    Takes image in np.array format, resizes it to specific size using
    letterbox resize and changes data layout from HWC to CHW.

    Parameters:
      img0 (np.ndarray): image for preprocessing
    Returns:
      img (np.ndarray): image after preprocessing
    """
    # resize
    img = letterbox(img0)[0]

    # Convert HWC to CHW
    img = img.transpose(2, 0, 1)
    img = np.ascontiguousarray(img)
    return img
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```python
def image_to_tensor(image:np.ndarray):
    """
    Preprocess image according to YOLOv8 input requirements.
    Takes image in np.array format, resizes it to specific size using
    letterbox resize and changes data layout from HWC to CHW.

    Parameters:
      img (np.ndarray): image for preprocessing
    Returns:
      input_tensor (np.ndarray): input tensor in NCHW format
      with float32 values in [0, 1] range
    """
    input_tensor = image.astype(np.float32)  # uint8 to fp32
    input_tensor /= 255.0  # 0 - 255 to 0.0 - 1.0

    # add batch dimension
    if input_tensor.ndim == 3:
        input_tensor = np.expand_dims(input_tensor, 0)
    return input_tensor
```

## Postprocessing 5)

The model output contains detection boxes candidates, it is a tensor with the `[-1,84,-1]` shape in the `B,84,N` format, where:

- `B` - batch size
- `N` - number of detection boxes

Detection box has the `[ x , y , h , w , class_no_1 , ..., class_no_80 ]` format, where:

- `( x , y )` - raw coordinates of box center
- `h , w` - raw height and width of the box
- `class_no_1 , ..., class_no_80` - probability distribution over the classes.

For getting the final prediction, we need to apply a non-maximum suppression algorithm and rescale box coordinates to the original image size.

The instance segmentation model, additionally, has an output that contains proto mask candidates for instance segmentation. It should be decoded by using box coordinates. It is a tensor with the `[-1 32, -1, -1]` shape in the `B,C H,W` format, where:

- `B` - batch size
- `C` - number of candidates
- `H` - mask height
- `W` - mask width

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

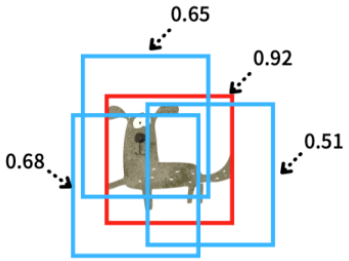
```python
def postprocess(
    pred_boxes:np.ndarray,
    input_hw:Tuple[int, int],
    orig_img:np.ndarray,
    min_conf_threshold:float = 0.25,
    nms_iou_threshold:float = 0.7,
    agnosting_nms:bool = False,
    max_detections:int = 300,
    pred_masks:np.ndarray = None,
    retina_mask:bool = False
):
    """
    YOLOv8 model postprocessing function. Applied non maximum supression algorithm
        to detections and rescale boxes to original image size
    Parameters:
        pred_boxes (np.ndarray): model output prediction boxes
        input_hw (np.ndarray): preprocessed image
        orig_image (np.ndarray): image before preprocessing
        min_conf_threshold (float, *optional*, 0.25): minimal accepted confidence
            for object filtering
        nms_iou_threshold (float, *optional*, 0.45): minimal overlap score
            for removing objects duplicates in NMS
        agnostic_nms (bool, *optiona*, False): apply class agnostinc NMS approach or n
        max_detections (int, *optional*, 300):  maximum detections after NMS
        pred_masks (np.ndarray, *optional*, None): model ooutput prediction masks,
            if not provided only boxes will be postprocessed
        retina_mask (bool, *optional*, False): retina mask postprocessing instead of n
    Returns:
       pred (List[Dict[str, np.ndarray]]): list of dictionary with det
            - detected boxes in format [x1, y1, x2, y2, score, label] and segment
            - segmentation polygons for each element in batch
    """
```

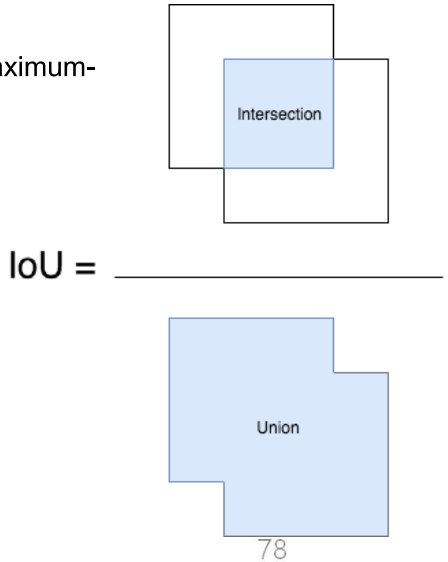
```python
    nms_kwargs = {"agnostic": agnosting_nms, "max_det":max_detections}
    # if pred_masks is not None:
    #     nms_kwargs["nm"] = 32
    preds = ops.non_max_suppression(
        torch.from_numpy(pred_boxes),
        min_conf_threshold,
        nms_iou_threshold,
        nc=80,
        **nms_kwargs
    )
    results = []
    proto = torch.from_numpy(pred_masks) if pred_masks is not None else None
```

native decoding

Non-maximum Suppression (NMS)

Ref : https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c



# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
for i, pred in enumerate(preds):
    shape = orig_img[i].shape if isinstance(orig_img, list) else orig_img.shape
    if not len(pred):
        results.append({"det": [], "segment": []})
        continue
    if proto is None:
        pred[:, :4] = ops.scale_boxes(input_hw, pred[:, :4], shape).round()
        results.append({"det": pred})
        continue
    if retina_mask:
        pred[:, :4] = ops.scale_boxes(input_hw, pred[:, :4], shape).round()
        masks = ops.process_mask_native(proto[i], pred[:, 6:], pred[:, :4], shape[:2])  # HWC
        segments = [ops.scale_segments(input_hw, x, shape, normalize=False) for x in ops.masks2segments(masks)]
    else:
        masks = ops.process_mask(proto[i], pred[:, 6:], pred[:, :4], input_hw, upsample=True)
        pred[:, :4] = ops.scale_boxes(input_hw, pred[:, :4], shape).round()
        segments = [ops.scale_segments(input_hw, x, shape, normalize=False) for x in ops.masks2segments(masks)]
    results.append({"det": pred[:, :6].numpy(), "segment": segments})
return results
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Test on single image 6)

Now, once we have defined preprocessing and postprocessing steps, we are ready to check model prediction.

First, object detection:

det_model_path = models\yolov8n_openvino_model\yolov8n.xml

```
from openvino.runtime import Core, Model

core = Core()
det_ov_model = core.read_model(det_model_path)
device = "CPU"  # "GPU"
if device != "CPU":
    det_ov_model.reshape({0: [1, 3, 640, 640]})
det_compiled_model = core.compile_model(det_ov_model, device)
```

```
def detect(image:np.ndarray, model:Model):
    """
    OpenVINO YOLOv8 model inference function.
    Preprocess image, runs model inference and postprocess results using NMS.
    Parameters:
        image (np.ndarray): input image.
        model (Model): OpenVINO compiled model.
    Returns:
        detections (np.ndarray): detected boxes in format [x1, y1, x2, y2,
                                                           score, label]
    """
    num_outputs = len(model.outputs)
    preprocessed_image = preprocess_image(image)
    input_tensor = image_to_tensor(preprocessed_image)
    result = model(input_tensor)
    boxes = result[model.output(0)]
    masks = None
    if num_outputs > 1:
        masks = result[model.output(1)]
    input_hw = input_tensor.shape[2:]
    detections = postprocess(pred_boxes=boxes, input_hw=input_hw,
                             orig_img=image, pred_masks=masks)
    return detections
```

```
input_image = np.array(Image.open(IMAGE_PATH))
detections = detect(input_image, det_compiled_model)[0]
image_with_boxes = draw_results(detections, input_image, label_map)

Image.fromarray(image_with_boxes)
```

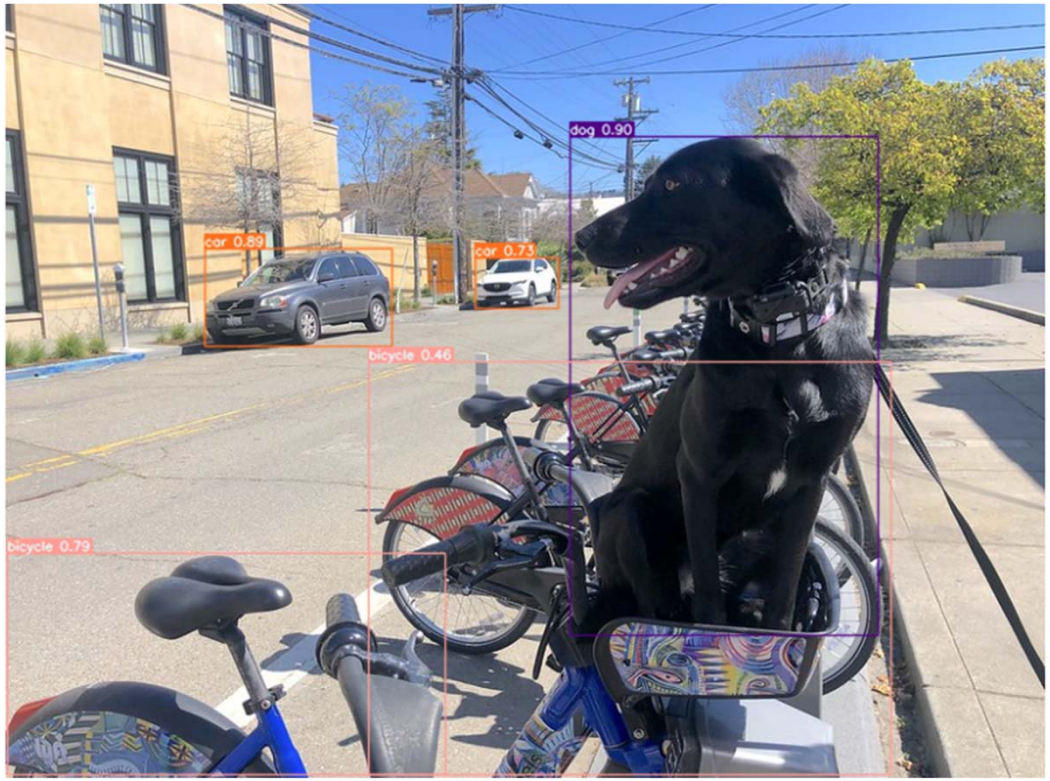
Then, instance segmentation:

seg_model_path = models\yolov8n-seg_openvino_model\yolov8n-seg.xml

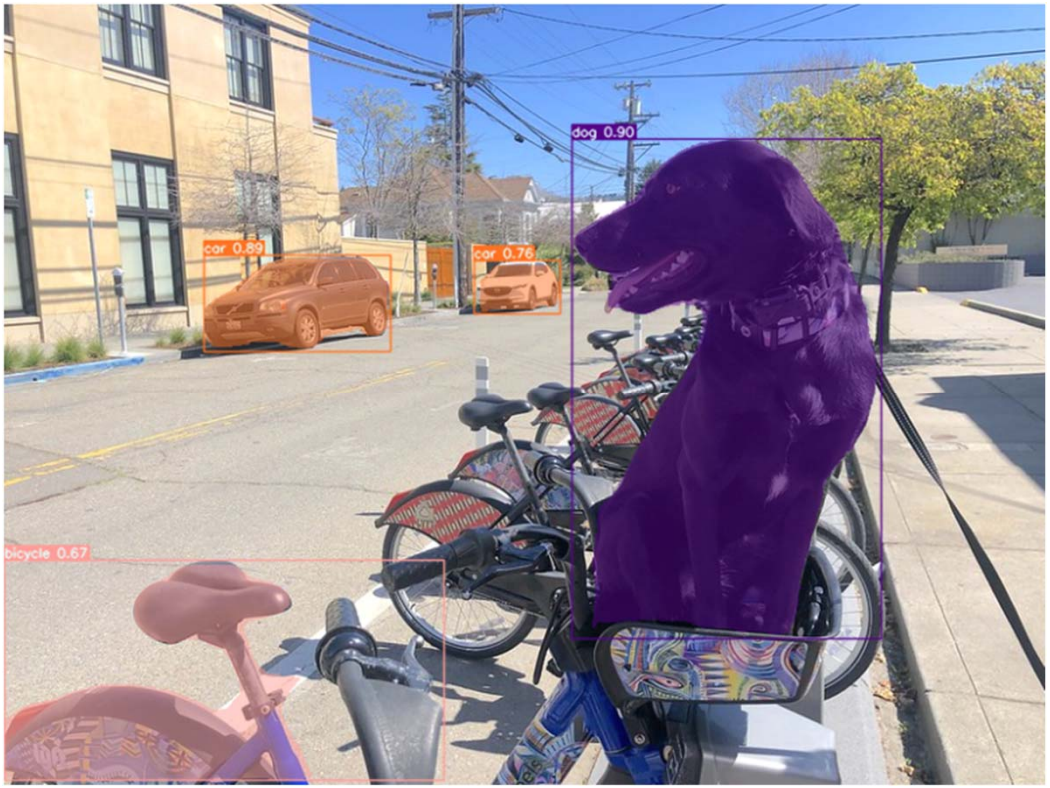
```
seg_ov_model = core.read_model(seg_model_path)
device = "CPU"  # GPU
if device != "CPU":
    seg_ov_model.reshape({0: [1, 3, 640, 640]})
seg_compiled_model = core.compile_model(seg_ov_model, device)

input_image = np.array(Image.open(IMAGE_PATH))
detections = detect(input_image, seg_compiled_model)[0]
image_with_masks = draw_results(detections, input_image, label_map)

Image.fromarray(image_with_boxes)
Image.fromarray(image_with_masks)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1




Obejct Detection




Instance Segmentation

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Check model accuracy on the dataset 7)

For comparing the optimized model result with the original, it is good to know some measurable results in terms of model accuracy on the validation dataset.

### Download the validation dataset

YOLOv8 is pre-trained on the COCO dataset, so to evaluate the model accuracy we need to download it. According to the instructions provided in the YOLOv8 repo, we also need to download annotations in the format used by the author of the model, for use with the original model evaluation function.

> **Note**: The initial dataset download may take a few minutes to complete. The download speed will vary depending on the quality of your internet connection.

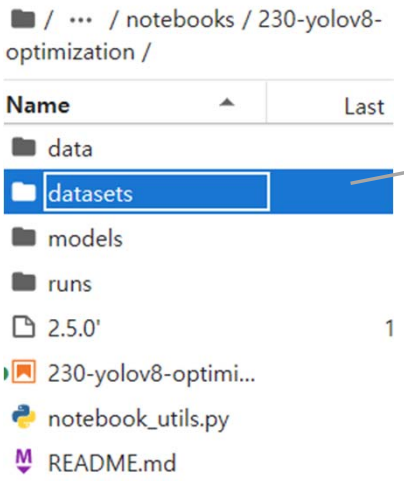
DATA_URL = "http://images.cocodataset.org/zips/val2017.zip"
LABELS_URL = "https://github.com/ultralytics/yolov5/releases/download/v1.0/coco2017labels-segments.zip"
CFG_URL = "https://raw.githubusercontent.com/ultralytics/ultralytics/main/ultralytics/datasets/coco.yaml"

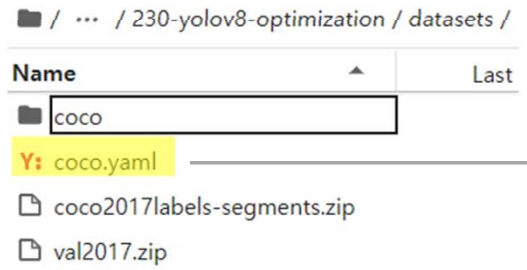
```
from zipfile import ZipFile

DATA_URL = "http://images.cocodataset.org/zips/val2017.zip"
LABELS_URL = "https://github.com/ultralytics/yolov5/releases/download/v1.0/coco2017l
CFG_URL = "https://raw.githubusercontent.com/ultralytics/ultralytics/main/ultralytic

OUT_DIR = Path('./datasets')

DATA_PATH = OUT_DIR / "val2017.zip"
LABELS_PATH = OUT_DIR / "coco2017labels-segments.zip"
CFG_PATH = OUT_DIR / "coco.yaml"

download_file(DATA_URL, DATA_PATH.name, DATA_PATH.parent)
download_file(LABELS_URL, LABELS_PATH.name, LABELS_PATH.parent)
download_file(CFG_URL, CFG_PATH.name, CFG_PATH.parent)

if not (OUT_DIR / "coco/labels").exists():
    with ZipFile(LABELS_PATH , "r") as zip_ref:
        zip_ref.extractall(OUT_DIR)
    with ZipFile(DATA_PATH , "r") as zip_ref:
        zip_ref.extractall(OUT_DIR / 'coco/images')
```

```
'datasets\val2017.zip' already exists.
'datasets\coco2017labels-segments.zip' already exists.
datasets₩coco.yaml: 2.47k/? [00:00<00:00, 361kB/s
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1



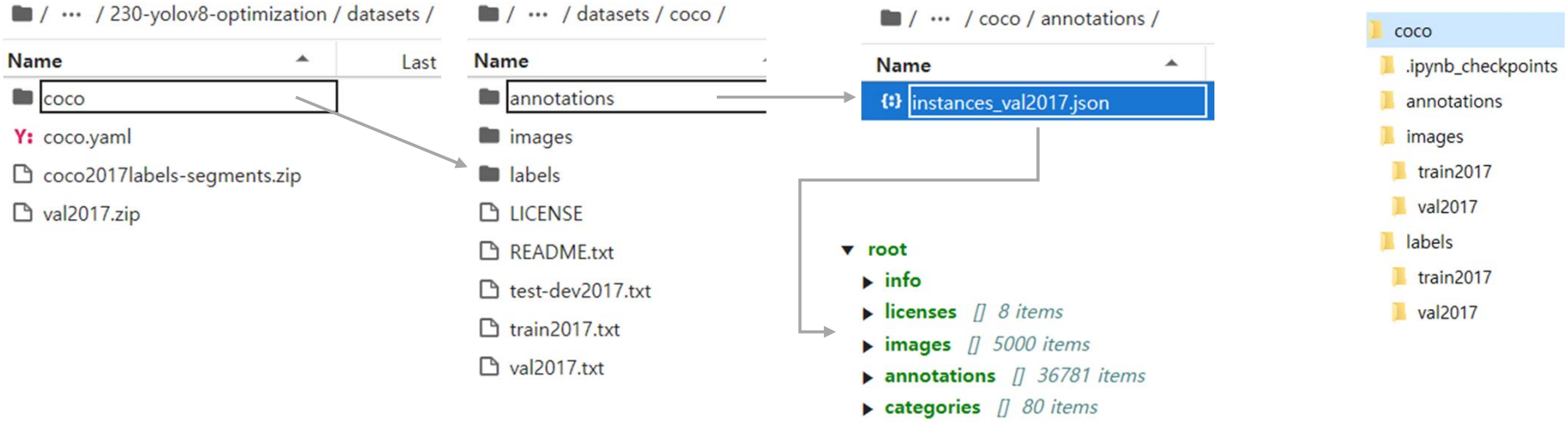

coco.yaml

```yaml
# Ultralytics YOLO 🚀, AGPL-3.0 license
# COCO 2017 dataset http://cocodataset.org by Microsoft
# Example usage: yolo train data=coco.yaml
# parent
# ├── ultralytics
# └── datasets
#     └── coco  ← downloads here (20.1 GB)


# Train/val/test sets as 1) dir: path/to/imgs, 2) file: p
path: ../datasets/coco  # dataset root dir
train: train2017.txt  # train images (relative to 'path')
val: val2017.txt  # val images (relative to 'path') 5000
test: test-dev2017.txt  # 20288 of 40670 images, submit t

# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
```

```
  78: hair drier
  79: toothbrush


# Download script/URL (optional)
download: |
  from ultralytics.yolo.utils.downloads import download
  from pathlib import Path

  # Download labels
  segments = True  # segment or box labels
  dir = Path(yaml['path'])  # dataset root dir
  url = 'https://github.com/ultralytics/yolov5/releases/download/v1.0/'
  urls = [url + ('coco2017labels-segments.zip' if segments else 'coco2017labels.zip')]  # labels
  download(urls, dir=dir.parent)
  # Download data
  urls = ['http://images.cocodataset.org/zips/train2017.zip',  # 19G, 118k images
          'http://images.cocodataset.org/zips/val2017.zip',  # 1G, 5k images
          'http://images.cocodataset.org/zips/test2017.zip']  # 7G, 41k images (optional)
  download(urls, dir=dir / 'images', threads=3)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1




# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

Define validation function

```
from tqdm.notebook import tqdm
from ultralytics.yolo.utils.metrics import ConfusionMatrix


def test(model:Model, core:Core, data_loader:torch.utils.data.DataLoader, validator,
         num_samples:int = None):
    """
    OpenVINO YOLOv8 model accuracy validation function.
        Runs model validation on dataset and returns metrics
    Parameters:
        model (Model): OpenVINO model
        data_loader (torch.utils.data.DataLoader): dataset loader
        validato: instalce of validator class
        num_samples (int, *optional*, None):
            validate model only on specified number samples, if provided
    Returns:
        stats: (Dict[str, float]) - dictionary with aggregated accuracy metrics statis
                                    key is metric name, value is metric value
    """
    validator.seen = 0
    validator.jdict = []
    validator.stats = []
    validator.batch_i = 1
    validator.confusion_matrix = ConfusionMatrix(nc=validator.nc)
    model.reshape({0: [1, 3, -1, -1]})
    num_outputs = len(model.outputs)
    compiled_model = core.compile_model(model)
    for batch_i, batch in enumerate(tqdm(data_loader, total=num_samples)):
        if num_samples is not None and batch_i == num_samples:
            break
        batch = validator.preprocess(batch)
        results = compiled_model(batch["img"])
        if num_outputs == 1:
            preds = torch.from_numpy(results[compiled_model.output(0)])
        else:
            preds = [torch.from_numpy(results[compiled_model.output(0)]),
                     torch.from_numpy(results[compiled_model.output(1)])]
        preds = validator.postprocess(preds)
        validator.update_metrics(preds, batch)
    stats = validator.get_stats()
    return stats
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```python
def print_stats(stats:np.ndarray, total_images:int, total_objects:int):
    """
    Helper function for printing accuracy statistic
    Parameters:
        stats: (Dict[str, float]) - dictionary with aggregated accuracy metrics statistics,
            key is metric name, value is metric value
        total_images (int) -  number of evaluated images
        total objects (int)
    Returns:
        None
    """
    print("Boxes:")
    mp, mr, map50, mean_ap = stats['metrics/precision(B)'], stats['metrics/recall(B)'], stats['metrics/mAP50(B)'], stats['metrics/mAP50-95(B)']
    # Print results
    s = ('%20s' + '%12s' * 6) % ('Class', 'Images', 'Labels', 'Precision', 'Recall', 'mAP@.5', 'mAP@.5:.95')
    print(s)
    pf = '%20s' + '%12i' * 2 + '%12.3g' * 4  # print format
    print(pf % ('all', total_images, total_objects, mp, mr, map50, mean_ap))
    if 'metrics/precision(M)' in stats:
        s_mp, s_mr, s_map50, s_mean_ap = stats['metrics/precision(M)'], stats['metrics/recall(M)'], stats['metrics/mAP50(M)'], stats['metrics/mAP50-95(M)']
        # Print results
        s = ('%20s' + '%12s' * 6) % ('Class', 'Images', 'Labels', 'Precision', 'Recall', 'mAP@.5', 'mAP@.5:.95')
        print(s)
        pf = '%20s' + '%12i' * 2 + '%12.3g' * 4  # print format
        print(pf % ('all', total_images, total_objects, s_mp, s_mr, s_map50, s_mean_ap))
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Configure Validator helper and create DataLoader

The original model repository uses a `Validator` wrapper, which represents the accuracy validation pipeline. It creates dataloader and evaluation metrics and updates metrics on each data batch produced by the dataloader. Besides that, it is responsible for data preprocessing and results postprocessing. For class initialization, the configuration should be provided. We will use the default setup, but it can be replaced with some parameters overriding to test on custom data. The model has connected the `ValidatorClass` method, which creates a validator class instance.

```
from ultralytics.yolo.utils import DEFAULT_CFG
from ultralytics.yolo.cfg import get_cfg
from ultralytics.yolo.data.utils import check_det_dataset

args = get_cfg(cfg=DEFAULT_CFG)
args.data = str(CFG_PATH)
```

CFG_PATH = OUT_DIR / "coco.yaml"

```
det_validator = det_model.ValidatorClass(args=args)
```

```
det_validator.data = check_det_dataset(args.data)
det_data_loader = det_validator.get_dataloader("datasets/coco", 1)
```

```
val: Scanning datasets\coco\labels\val2017.cache... 4952 images, 48 backgrounds, 0 c
orrupt: 100%|██████████| 5000/5000
```

```
args
```

```
namespace(task='segment',
          mode='train',
          model=None,
          data='datasets\\coco.yaml',
          epochs=100,
          patience=50,
          batch=16,
          imgsz=640,
          save=True,
          save_period=-1,
          cache=False,
          device=None,
          workers=8,
          project=None,
          name=None,
          exist_ok=False,
          pretrained=False,
          optimizer='SGD',
          verbose=True,
          seed=0,
          deterministic=True,
          single_cls=False,
          image_weights=False,
          rect=False,
          cos_lr=False,
          close_mosaic=10,
          resume=False,
          min_memory=False,
          overlap_mask=True,
          mask_ratio=4,
          dropout=0.0,
          val=True,
          split='val',
          save_json=False,
          save_hybrid=False,
          conf=0.001,
          iou=0.7,
          max_det=300,
          half=False,
          dnn=False,
          plots=True,
          source=None,
          show=False,
          save_txt=False,
          save_conf=False,
          save_crop=False,
          hide_labels=False,
          hide_conf=False,
          vid_stride=1,
          line_thickness=3,
          visualize=False,
          augment=False,
          agnostic_nms=False,
          classes=None,
          retina_masks=False,
          boxes=True,
          format='torchscript',
          keras=False,
          optimize=False,
          int8=False,
          dynamic=False,
          simplify=False,
          opset=None,
          workspace=4,
          nms=False,
          lr0=0.01,
          lrf=0.01,
          momentum=0.937,
          weight_decay=0.0005,
          warmup_epochs=3.0,
          warmup_momentum=0.8,
          warmup_bias_lr=0.1,
          box=7.5,
          cls=0.5,
          dfl=1.5,
          fl_gamma=0.0,
          label_smoothing=0.0,
          nbs=64,
          hsv_h=0.015,
          hsv_s=0.7,
          hsv_v=0.4,
          degrees=0.0,
          translate=0.1,
          scale=0.5,
          shear=0.0,
          perspective=0.0,
          flipud=0.0,
          fliplr=0.5,
          mosaic=1.0,
          mixup=0.0,
          copy_paste=0.0,
          cfg=None,
          v5loader=False,
          tracker='botsort.yaml')
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
det_validator.is_coco = True
det_validator.class_map = ops.coco80_to_coco91_class()
det_validator.names = det_model.model.names
det_validator.metrics.names = det_validator.names
det_validator.nc = det_model.model.model[-1].nc
```

```
seg_validator = seg_model.ValidatorClass(args=args)
seg_validator.data = check_det_dataset(args.data)
seg_data_loader = seg_validator.get_dataloader("datasets/coco/", 1)

seg_validator.is_coco = True
seg_validator.class_map = ops.coco80_to_coco91_class()
seg_validator.names = seg_model.model.names
seg_validator.metrics.names = seg_validator.names
seg_validator.nc = seg_model.model.model[-1].nc
seg_validator.nm = 32
seg_validator.process = ops.process_mask
seg_validator.plot_masks = []
```

```
val: Scanning datasets\coco\labels\val2017.cache... 4952 images, 48 backgrounds, 0 c
orrupt: 100%|██████████| 5000/5000
```

After definition test function and validator creation, we are ready for getting accuracy metrics

> **Note**: Model evaluation is time consuming process and can take several minutes, depending on the hardware. For reducing calculation time, we define `num_samples` parameter with evaluation subset size, but in this case, accuracy can be noncomparable with originally reported by the authors of the model, due to validation subset difference. *To validate the models on the full dataset set* `NUM_TEST_SAMPLES = None`.

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
NUM_TEST_SAMPLES = 300
```

```
fp_det_stats = test(det_ov_model, core, det_data_loader, det_validator, num_samples=NUM_TEST_SAMPLES)
```

```
100% ████████████████████████████████████████ 300/300 [00:16<00:00, 18.41it/s]
```

```
print_stats(fp_det_stats, det_validator.seen, det_validator.nt_per_class.sum())
```

print_stats(stats:np.ndarray, total_images:int, total_objects:int)

```
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.594       0.543       0.579       0.417
```

```
fp_seg_stats = test(seg_ov_model, core, seg_data_loader, seg_validator, num_samples=NUM_TEST_SAMPLES)
```

```
100% ████████████████████████████████████████ 300/300 [00:33<00:00, 8.66it/s]
```

```
print_stats(fp_seg_stats, seg_validator.seen, seg_validator.nt_per_class.sum())
```

```
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.609       0.524       0.579       0.416
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.602       0.501       0.557       0.354
```

`print_stats` reports the following list of accuracy metrics:

- `Precision` is the degree of exactness of the model in identifying only relevant objects.
- `Recall` measures the ability of the model to detect all ground truths objects.
- `mAP@t` - mean average precision, represented as area under the Precision-Recall curve aggregated over all classes in the dataset, where `t` is the Intersection Over Union (IOU) threshold, degree of overlapping between ground truth and predicted objects. Therefore, `mAP@.5` indicates that mean average precision is calculated at 0.5 IOU threshold, `mAP@.5:.95` - is calculated on range IOU thresholds from 0.5 to 0.95 with step 0.05.

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Optimize model using NNCF Post-training Quantization API 8)

NNCF provides a suite of advanced algorithms for Neural Networks inference optimization in OpenVINO with minimal accuracy drop. We will use 8-bit quantization in post-training mode (without the fine-tuning pipeline) to optimize YOLOv8.

The optimization process contains the following steps:

1. Create a Dataset for quantization.
2. Run `nncf.quantize` for getting an optimized model.
3. Serialize OpenVINO IR model, using the `openvino.runtime.serialize` function.

Reuse validation dataloader in accuracy testing for quantization. For that, it should be wrapped into the `nncf.Dataset` object and define a transformation function for getting only input tensors. As preprocessing for both models is the same, we can reuse one dataset for both models.

https://github.com/openvinotoolkit/nncf

# Neural Network Compression Framework (NNCF)

Key Features • Installation • Documentation • Usage • Tutorials and Samples • Third-party integration • Model Zoo

release v2.5.0 | website docs | license Apache 2.0 | downloads 217k

Neural Network Compression Framework (NNCF) provides a suite of post-training and training-time algorithms for neural networks inference optimization in OpenVINO™ with minimal accuracy drop.

NNCF is designed to work with models from PyTorch, TensorFlow, ONNX and OpenVINO™.

NNCF provides samples that demonstrate the usage of compression algorithms for different use cases and models. Compression results achievable with the NNCF-powered samples can be found in a table at the end of this document.

The framework is organized as a Python* package that can be built and used in a standalone mode. The framework architecture is unified to make it easy to add different compression algorithms for both PyTorch and TensorFlow deep learning frameworks.

- Automatic, configurable model graph transformation to obtain the compressed model.
  > NOTE: Limited support for TensorFlow models. The models created using Sequential or Keras Functional API are only supported.
- Common interface for compression methods.
- GPU-accelerated layers for faster compressed model fine-tuning.
- Distributed training support.
- Git patch for prominent third-party repository (huggingface-transformers) demonstrating the process of integrating NNCF into custom training pipelines

## Post-Training Compression Algorithms

| Compression algorithm | OpenVINO | PyTorch | TensorFlow | ONNX |
|---|---|---|---|---|
| Post-Training Quantization | Supported | Supported | Supported | Supported |

## Training-Time Compression Algorithms

| Compression algorithm | PyTorch | TensorFlow |
|---|---|---|
| Quantization Aware Training | Supported | Supported |
| Mixed-Precision Quantization | Supported | Not supported |
| Binarization | Supported | Not supported |
| Sparsity | Supported | Supported |
| Filter pruning | Supported | Supported |
| Movement pruning | Experimental | Not supported |

- Seamless combination of pruning, sparsity and quantization algorithms. Please refer to optimum-intel for examples of joint (movement) pruning, quantization and distillation (JPQD), end-to-end from NNCF optimization to compressed OpenVINO IR.
- Exporting PyTorch compressed models to ONNX* checkpoints and TensorFlow compressed models to SavedModel or Frozen Graph format, ready to use with OpenVINO™ toolkit.
- Support for Accuracy-Aware model training pipelines via the Adaptive Compression Level Training and Early Exit Training.

# Model Optimization & NNCF

https://github.com/openvinotoolkit/nncf

## Post-Training Quantization

- The NNCF PTQ is the simplest way to apply 8-bit quantization. To run the algorithm you only need your model and a small (~300 samples) calibration dataset.
- OpenVINO is the preferred backend to run PTQ with, and PyTorch, TensorFlow and ONNX are also supported.

▼ OpenVINO

```python
import nncf
import openvino.runtime as ov
import torch
from torchvision import datasets, transforms

# Instantiate your uncompressed model
model = ov.Core().read_model("/model_path")

# Provide validation part of the dataset to collect statistics needed for the compression algorithm
val_dataset = datasets.ImageFolder("/path", transform=transforms.Compose([transforms.ToTensor()]))
dataset_loader = torch.utils.data.DataLoader(val_dataset, batch_size=1)

# Step 1: Initialize transformation function
def transform_fn(data_item):
    images, _ = data_item
    return images

# Step 2: Initialize NNCF Dataset
calibration_dataset = nncf.Dataset(dataset_loader, transform_fn)
# Step 3: Run the quantization pipeline
quantized_model = nncf.quantize(model, calibration_dataset)
```

▼ PyTorch

```python
import nncf
import torch
from torchvision import datasets, models

# Instantiate your uncompressed model
model = models.mobilenet_v2()

# Provide validation part of the dataset to collect statisti
val_dataset = datasets.ImageFolder("/path", transform=transf
dataset_loader = torch.utils.data.DataLoader(val_dataset)

# Step 1: Initialize the transformation function
def transform_fn(data_item):
    images, _ = data_item
    return images

# Step 2: Initialize NNCF Dataset
calibration_dataset = nncf.Dataset(dataset_loader, transform
# Step 3: Run the quantization pipeline
quantized_model = nncf.quantize(model, calibration_dataset)
```

# Model Optimization & NNCF

https://github.com/openvinotoolkit/nncf

## Training-Time Compression

- Below is an example of Accuracy Aware Quantization pipeline where model weights and compression parameters may be fine-tuned to achieve a higher accuracy.

▼ PyTorch

```
import torch
import nncf.torch  # Important - must be imported before any other external package that depends on torch

from nncf import NNCFConfig
from nncf.torch import create_compressed_model, register_default_init_args

# Instantiate your uncompressed model
from torchvision.models.resnet import resnet50
model = resnet50()

# Load a configuration file to specify compression
nncf_config = NNCFConfig.from_json("resnet50_int8.json")

# Provide data loaders for compression algorithm initialization, if necessary
import torchvision.datasets as datasets
representative_dataset = datasets.ImageFolder("/path", transform=transforms.Compose([transforms.ToTensor()]))
init_loader = torch.utils.data.DataLoader(representative_dataset)
nncf_config = register_default_init_args(nncf_config, init_loader)

# Apply the specified compression algorithms to the model
compression_ctrl, compressed_model = create_compressed_model(model, nncf_config)

# Now use compressed_model as a usual torch.nn.Module
# to fine-tune compression parameters along with the model weights

# ... the rest of the usual PyTorch-powered training pipeline

# Export to ONNX or .pth when done fine-tuning
compression_ctrl.export_model("compressed_model.onnx")
torch.save(compressed_model.state_dict(), "compressed_model.pth")
```

# Model Optimization & NNCF

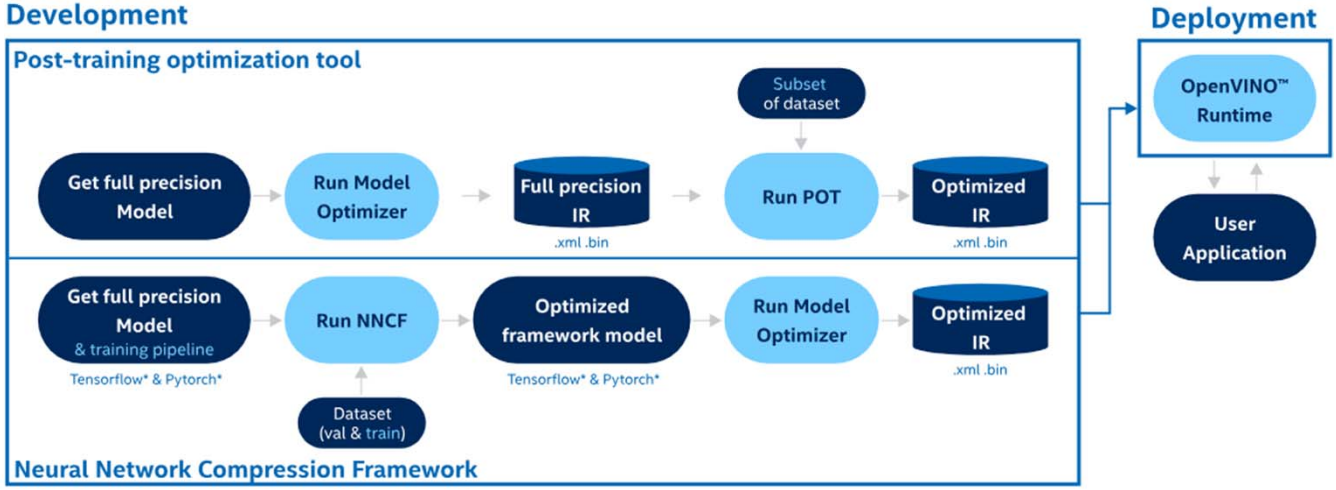
https://docs.openvino.ai/2022.3/openvino_docs_model_optimization_guide.html


## Model Optimization

Model optimization is an optional offline step of improving final model performance by applying special optimization methods, such as quantization, pruning, preprocessing optimization, etc. OpenVINO provides several tools to optimize models at different steps of model development:

- Model Optimizer implements most of the optimization parameters to a model by default. Yet, you are free to configure mean/scale values, batch size, RGB vs BGR input channels, and other parameters to speed up preprocess of a model (Embedding Preprocessing Computation).
- Post-training Quantization is designed to optimize inference of deep learning models by applying post-training methods that do not require model retraining or fine-tuning, for example, post-training 8-bit integer quantization.
- Training-time Optimization, a suite of advanced methods for training-time model optimization within the DL framework, such as PyTorch and TensorFlow 2.x. It supports methods, like Quantization-aware Training and Filter Pruning. NNCF-optimized models can be inferred with OpenVINO using all the available workflows.

## Workflow



# Model Optimization & NNCF

https://docs.openvino.ai/2022.3/openvino_docs_model_optimization_guide.html

- Post-training methods are limited in terms of achievable accuracy-performance trade-off for optimizing models. In this case, training-time optimization with NNCF is an option.
- Once the model is optimized using the aforementioned tools it can be used for inference using the regular OpenVINO inference workflow. No changes to the inference code are required.

**Read Tutorial !!!**

**https://docs.openvino.ai/2022.3/openvino_docs_model_optimization_guide.html**

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```python
import nncf  # noqa: F811
from typing import Dict

def transform_fn(data_item:Dict):
    """
    Quantization transform function. Extracts and preprocess input data
    from dataloader item for quantization.
    Parameters:
       data_item: Dict with data item produced by DataLoader during iteration
    Returns:
        input_tensor: Input data for quantization
    """
    input_tensor = det_validator.preprocess(data_item)['img'].numpy()
    return input_tensor

quantization_dataset = nncf.Dataset(det_data_loader, transform_fn)
```

```
INFO:nncf:NNCF initialized successfully. Supported frameworks detected: torch, tensorflow, onnx, openvino
```

The `nncf.quantize` function provides an interface for model quantization. It requires an instance of the OpenVINO Model and quantization dataset. Optionally, some additional parameters for the configuration quantization process (number of samples for quantization, preset, ignored scope, etc.) can be provided. YOLOv8 model contains non-ReLU activation functions, which require asymmetric quantization of activations. To achieve a better result, we will use a `mixed` quantization preset. It provides symmetric quantization of weights and asymmetric quantization of activations. For more accurate results, we should keep the operation in the postprocessing subgraph in floating point precision, using the `ignored_scope` parameter.

```python
ignored_scope = nncf.IgnoredScope(
    types=["Multiply", "Subtract", "Sigmoid"],  # ignore operations
    names=[
        "/model.22/dfl/conv/Conv",  # in the post-processing subgraph
        "/model.22/Add",
        "/model.22/Add_1",
        "/model.22/Add_2",
        "/model.22/Add_3",
        "/model.22/Add_4",
        "/model.22/Add_5",
        "/model.22/Add_6",
        "/model.22/Add_7",
        "/model.22/Add_8",
        "/model.22/Add_9",
        "/model.22/Add_10" ]
)

# Detection model
quantized_det_model = nncf.quantize(
    det_ov_model,
    quantization_dataset,
    preset=nncf.QuantizationPreset.MIXED,
    ignored_scope=ignored_scope
)
```

```
quantized_det_model
```

```
<Model: 'torch_jit'
inputs[
<ConstOutput: names[images] shape[1,3,640,640] type: f32>
]
outputs[
<ConstOutput: names[output0] shape[1,84,8400] type: f32>
]>
```

```
Statistics collection: 100%|██████████| 300/300 [00:50<00:00,  5.92it/s]
Biases correction: 100%|██████████| 63/63 [00:05<00:00, 12.32it/s]
```

**Note**: Model post-training quantization is time-consuming process. Be patient, it can take several minutes depending on your hardware.

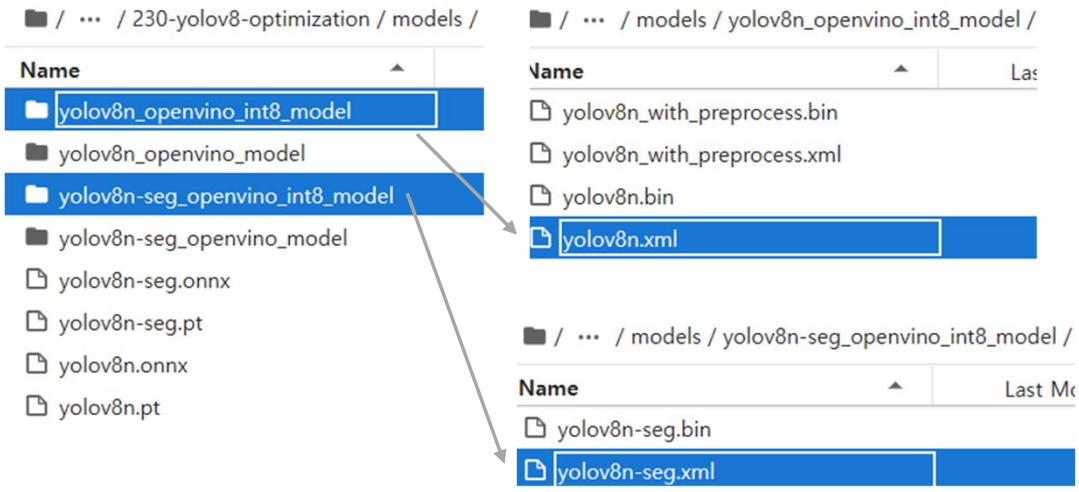
# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
from openvino.runtime import serialize
int8_model_det_path = models_dir / f'{DET_MODEL_NAME}_openvino_int8_model/{DET_
print(f"Quantized detection model will be saved to {int8_model_det_path}")
serialize(quantized_det_model, str(int8_model_det_path))
```

```
Quantized detection model will be saved to models\yolov8n_openvino_int8_model
\yolov8n.xml
```

int8_model_det_path = models_dir / f'{DET_MODEL_NAME}_openvino_int8_model/{DET_MODEL_NAME}.xml'




```
# Instance segmentation model

quantized_seg_model = nncf.quantize(
    seg_ov_model,
    quantization_dataset,
    preset=nncf.QuantizationPreset.MIXED,
    ignored_scope=ignored_scope
)
```

```
Statistics collection: 100%|████████████████████████████████████████
██████| 300/300 [00:54<00:00,  5.48it/s]
Biases correction: 100%|████████████████████████████████████████████
████████| 75/75 [00:06<00:00, 12.47it/s]
```

```
int8_model_seg_path = models_dir / f'{SEG_MODEL_NAME}_openvino_int8_model/{SEG_
print(f"Quantized segmentation model will be saved to {int8_model_seg_path}")
serialize(quantized_seg_model, str(int8_model_seg_path))
```

```
Quantized segmentation model will be saved to models\yolov8n-seg_openvino_int8
_model\yolov8n-seg.xml
```

int8_model_seg_path = models_dir / f'{SEG_MODEL_NAME}_openvino_int8_model/{SEG_MODEL_NAME}.xml'

```
quantized_seg_model
```

```
<Model: 'torch_jit'
inputs[
<ConstOutput: names[images] shape[1,3,640,640] type: f32>
]
outputs[
<ConstOutput: names[output0] shape[1,116,8400] type: f32>,
<ConstOutput: names[output1] shape[1,32,160,160] type: f32>
]>
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Validate Quantized model inference 9)

`nncf.quantize` returns the OpenVINO Model class instance, which is suitable for loading on a device for making predictions. `INT8` model input data and output result formats have no difference from the floating point model representation. Therefore, we can reuse the same `detect` function defined above for getting the `INT8` model result on the image.

### Object detection:

```
if device != "CPU":
    quantized_det_model.reshape({0, [1, 3, 640, 640]})
quantized_det_compiled_model = core.compile_model(quantized_det_model, device)
input_image = np.array(Image.open(IMAGE_PATH))
detections = detect(input_image, quantized_det_compiled_model)[0]
image_with_boxes = draw_results(detections, input_image, label_map)

Image.fromarray(image_with_boxes)
```

### Instance segmentation:

```
if device != "CPU":
    quantized_seg_model.reshape({0, [1, 3, 640, 640]})
quantized_seg_compiled_model = core.compile_model(quantized_seg_model, device)
input_image = np.array(Image.open(IMAGE_PATH))
detections = detect(input_image, quantized_seg_compiled_model)[0]
image_with_masks = draw_results(detections, input_image, label_map)

Image.fromarray(image_with_masks)
```


# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Compare Performance of the Original and Quantized Models 10)

Finally, use the OpenVINO Benchmark Tool to measure the inference performance of the `FP32` and `INT8` models.

> **Note**: For more accurate performance, it is recommended to run `benchmark_app` in a terminal/command prompt after closing other applications. Run `benchmark_app -m <model_path> -d CPU -shape "<input_shape>"` to benchmark async inference on CPU on specific input data shape for one minute. Change `CPU` to `GPU` to benchmark on GPU. Run `benchmark_app --help` to see an overview of all command-line options.

### Compare performance object detection models

```
# Inference FP32 model (OpenVINO IR)
!benchmark_app -m $det_model_path -d $device -api async -shape "[1,3,640,640]"
```

```
# Inference INT8 model (OpenVINO IR)
!benchmark_app -m $int8_model_det_path -d $device -api async -shape "[1,3,640,640]" -t 15
```

### Instance segmentation

```
!benchmark_app -m $seg_model_path -d $device -api async -shape "[1,3,640,640]" -t 15
```

```
!benchmark_app -m $int8_model_seg_path -d $device -api async -shape "[1,3,640,640]" -t 15
```

Object Detection

Inference FP32 model

```
[Step 11/11] Dumping statistics report
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            2288 iterations
[ INFO ] Duration:         60163.79 ms
[ INFO ] Latency:
[ INFO ]    Median:        104.59 ms
[ INFO ]    Average:       105.01 ms
[ INFO ]    Min:           69.95 ms
[ INFO ]    Max:           135.35 ms
[ INFO ] Throughput:   38.03 FPS
```

Inference INT8 model

```
[Step 11/11] Dumping statistics report
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            988 iterations
[ INFO ] Duration:         15100.55 ms
[ INFO ] Latency:
[ INFO ]    Median:        60.61 ms
[ INFO ]    Average:       61.00 ms
[ INFO ]    Min:           50.56 ms
[ INFO ]    Max:           70.08 ms
[ INFO ] Throughput:   65.43 FPS
```

Instance Segmentation

Inference FP32 model

```
[Step 11/11] Dumping statistics report
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            436 iterations
[ INFO ] Duration:         15172.09 ms
[ INFO ] Latency:
[ INFO ]    Median:        138.05 ms
[ INFO ]    Average:       138.76 ms
[ INFO ]    Min:           106.96 ms
[ INFO ]    Max:           175.45 ms
[ INFO ] Throughput:   28.74 FPS
```

Inference INT8 model

```
[Step 11/11] Dumping statistics report
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            704 iterations
[ INFO ] Duration:         15090.00 ms
[ INFO ] Latency:
[ INFO ]    Median:        84.92 ms
[ INFO ]    Average:       85.53 ms
[ INFO ]    Min:           69.50 ms
[ INFO ]    Max:           108.89 ms
[ INFO ] Throughput:   46.65 FPS
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Inference FP32 model (OpenVINO IR)

```
[Step 1/11] Parsing and validating input arguments
[ INFO ] Parsing input parameters
[Step 2/11] Loading OpenVINO Runtime
[ INFO ] OpenVINO:
[ INFO ] Build ..... 2023.0.0-10926-b4452d56304-releases/2023/0
[ INFO ]
[ INFO ] Device info:
[ INFO ] CPU
[ INFO ] Build ..... 2023.0.0-10926-b4452d56304-releases/2023/0
[ INFO ]
[ INFO ]
[Step 3/11] Setting device configuration
[ WARNING ] Performance hint was not explicitly specified in command line.
Device(CPU) performance hint will be set to PerformanceMode.THROUGHPUT.
[Step 4/11] Reading model files
[ INFO ] Loading model files
[ INFO ] Read model took 39.00 ms
[ INFO ] Original model I/O parameters:
[ INFO ] Model inputs:
[ INFO ]     images (node: images) : f32 / [...] / [?,3,?,?]
[ INFO ] Model outputs:
[ INFO ]     output0 (node: output0) : f32 / [...] / [?,84,?]
[Step 5/11] Resizing model to match image sizes and given batch
[ INFO ] Model batch size: 1
[ INFO ] Reshaping model: 'images': [1,3,640,640]
[ INFO ] Reshape model took 22.00 ms
[Step 6/11] Configuring input of the model
[ INFO ] Model inputs:
[ INFO ]     images (node: images) : u8 / [N,C,H,W] / [1,3,640,640]
[ INFO ] Model outputs:
[ INFO ]     output0 (node: output0) : f32 / [...] / [1,84,8400]
[Step 7/11] Loading the model to the device
[ INFO ] Compile model took 204.00 ms
[Step 8/11] Querying optimal runtime parameters
[ INFO ] Model:
[ INFO ]   NETWORK_NAME: torch_jit
[ INFO ]   OPTIMAL_NUMBER_OF_INFER_REQUESTS: 4
[ INFO ]   NUM_STREAMS: 4
[ INFO ]   AFFINITY: Affinity.NONE
[ INFO ]   INFERENCE_NUM_THREADS: 12
[ INFO ]   PERF_COUNT: False
[ INFO ]   INFERENCE_PRECISION_HINT: <Type: 'float32'>
[ INFO ]   PERFORMANCE_HINT: PerformanceMode.THROUGHPUT
[ INFO ]   EXECUTION_MODE_HINT: ExecutionMode.PERFORMANCE
[ INFO ]   PERFORMANCE_HINT_NUM_REQUESTS: 0
[ INFO ]   ENABLE_CPU_PINNING: False
[ INFO ]   SCHEDULING_CORE_TYPE: SchedulingCoreType.ANY_CORE
[ INFO ]   ENABLE_HYPER_THREADING: True
[ INFO ]   EXECUTION_DEVICES: ['CPU']
[Step 9/11] Creating infer requests and preparing input tensors
[ WARNING ] No input files were given for input 'images'!. This input will be filled with
random values!
[ INFO ] Fill input 'images' with random values
[Step 10/11] Measuring performance (Start inference asynchronously, 4 inference
requests, limits: 60000 ms duration)
[ INFO ] Benchmarking in inference only mode (inputs filling are not included in
measurement loop).
[ INFO ] First inference took 38.46 ms
[Step 11/11] Dumping statistics report
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:          2288 iterations
[ INFO ] Duration:       60163.79 ms
[ INFO ] Latency:
[ INFO ]    Median:      104.59 ms
[ INFO ]    Average:     105.01 ms
[ INFO ]    Min:         69.95 ms
[ INFO ]    Max:         135.35 ms
[ INFO ] Throughput:   38.03 FPS
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Inference INT8 model (OpenVINO IR)

**[Step 1/11] Parsing and validating input arguments**
[ INFO ] Parsing input parameters
**[Step 2/11] Loading OpenVINO Runtime**
[ INFO ] OpenVINO:
[ INFO ] Build ..... 2023.0.0-10926-b4452d56304-releases/2023/0
[ INFO ]
[ INFO ] Device info:
[ INFO ] CPU
[ INFO ] Build ..... 2023.0.0-10926-b4452d56304-releases/2023/0
[ INFO ]
[ INFO ]
**[Step 3/11] Setting device configuration**
[ WARNING ] Performance hint was not explicitly specified in command line. Device(CPU) performance hint will be set to PerformanceMode.THROUGHPUT.
**[Step 4/11] Reading model files**
[ INFO ] Loading model files
[ INFO ] Read model took 53.00 ms
[ INFO ] Original model I/O parameters:
[ INFO ] Model inputs:
[ INFO ]  images (node: images) : f32 / [...] / [1,3,?,?]
[ INFO ] Model outputs:
[ INFO ]  output0 (node: output0) : f32 / [...] / [1,84,21..]
**[Step 5/11] Resizing model to match image sizes and given batch**
[ INFO ] Model batch size: 1
[ INFO ] Reshaping model: 'images': [1,3,640,640]
[ INFO ] Reshape model took 23.03 ms
**[Step 6/11] Configuring input of the model**
[ INFO ] Model inputs:
[ INFO ]  images (node: images) : u8 / [N,C,H,W] / [1,3,640,640]
[ INFO ] Model outputs:
[ INFO ]  output0 (node: output0) : f32 / [...] / [1,84,8400]
**[Step 7/11] Loading the model to the device**
[ INFO ] Compile model took 548.97 ms
**[Step 8/11] Querying optimal runtime parameters**
[ INFO ] Model:
[ INFO ]  NETWORK_NAME: torch_jit
[ INFO ]  OPTIMAL_NUMBER_OF_INFER_REQUESTS: 4
[ INFO ]  NUM_STREAMS: 4
[ INFO ]  AFFINITY: Affinity.NONE
[ INFO ]  INFERENCE_NUM_THREADS: 12
[ INFO ]  PERF_COUNT: False
[ INFO ]  INFERENCE_PRECISION_HINT: <Type: 'float32'>
[ INFO ]  PERFORMANCE_HINT: PerformanceMode.THROUGHPUT
[ INFO ]  EXECUTION_MODE_HINT: ExecutionMode.PERFORMANCE
[ INFO ]  PERFORMANCE_HINT_NUM_REQUESTS: 0
[ INFO ]  ENABLE_CPU_PINNING: False
[ INFO ]  SCHEDULING_CORE_TYPE: SchedulingCoreType.ANY_CORE
[ INFO ]  ENABLE_HYPER_THREADING: True
[ INFO ]  EXECUTION_DEVICES: ['CPU']
**[Step 9/11] Creating infer requests and preparing input tensors**
[ WARNING ] No input files were given for input 'images'!. This input will be filled with random values!
[ INFO ] Fill input 'images' with random values
**[Step 10/11] Measuring performance (Start inference asynchronously, 4 inference requests, limits: 15000 ms duration)**
[ INFO ] Benchmarking in inference only mode (inputs filling are not included in measurement loop).
[ INFO ] First inference took 25.07 ms
**[Step 11/11] Dumping statistics report**
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:  988 iterations
[ INFO ] Duration:  15100.55 ms
[ INFO ] Latency:
[ INFO ]  Median:  60.61 ms
[ INFO ]  Average:  61.00 ms
[ INFO ]  Min:  50.56 ms
[ INFO ]  Max:  70.08 ms
[ INFO ] Throughput:  65.43 FPS

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Validate quantized model accuracy 11)

As we can see, there is no significant difference between `INT8` and float model result in a single image test. To understand how quantization influences model prediction precision, we can compare model accuracy on a dataset.

### Object detection

int8_det_stats = test(quantized_det_model, core, det_data_loader, det_validator, num_samples=NUM_TEST_SAMPLES)

```
int8_det_stats = test(quantized_det_model, core, det_data_loader, det_validator, nur
```

Error displaying widget: model not found

```
print("FP32 model accuracy")
print_stats(fp_det_stats, det_validator.seen, det_validator.nt_per_class.sum())

print("INT8 model accuracy")
print_stats(int8_det_stats, det_validator.seen, det_validator.nt_per_class.sum())
```

```
FP32 model accuracy
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.594       0.543       0.579       0.417
INT8 model accuracy
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.617       0.523       0.574       0.405
```

### Instance segmentation

int8_seg_stats = test(quantized_seg_model, core, seg_data_loader, seg_validator, num_samples=NUM_TEST_SAMPLES)

```
int8_seg_stats = test(quantized_seg_model, core, seg_data_loader, seg_validator, n
```

Error displaying widget: model not found

```
print("FP32 model accuracy")
print_stats(fp_seg_stats, seg_validator.seen, seg_validator.nt_per_class.sum())

print("INT8 model accuracy")
print_stats(int8_seg_stats, seg_validator.seen, seg_validator.nt_per_class.sum())
```

```
FP32 model accuracy
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.609       0.524       0.579       0.416
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.602       0.501       0.557       0.354
INT8 model accuracy
Boxes:
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.607       0.511       0.575       0.407
               Class      Images      Labels   Precision      Recall      mAP@.5  mAP@.5:.95
                 all         300        2145       0.655       0.462       0.554        0.35
```

Looks like accuracy was changed, but not significantly and it meets passing criteria.

## Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

# Next steps

This section contains suggestions on how to additionally improve the performance of your application using OpenVINO.

## Async inference pipeline

The key advantage of the Async API is that when a device is busy with inference, the application can perform other tasks in parallel (for example, populating inputs or scheduling other requests) rather than wait for the current inference to complete first. To understand how to perform async inference using openvino, refer to Async API tutorial

## Integration preprocessing to model 12)

Preprocessing API enables making preprocessing a part of the model reducing application code and dependency on additional image processing libraries. The main advantage of Preprocessing API is that preprocessing steps will be integrated into the execution graph and will be performed on a selected device (CPU/GPU etc.) rather than always being executed on CPU as part of an application. This will improve selected device utilization.

For more information, refer to the overview of Preprocessing API.

For example, we can integrate converting input data layout and normalization defined in `image_to_tensor` function.

The integration process consists of the following steps:

1. Initialize a PrePostProcessing object.
2. Define the input data format.
3. Describe preprocessing steps.
4. Integrating Steps into a Model.

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Initialize PrePostProcessing API

The `openvino.preprocess.PrePostProcessor` class enables specifying preprocessing and postprocessing steps for a model.

```
from openvino.preprocess import PrePostProcessor

ppp = PrePostProcessor(quantized_det_model)
```

## Define input data format

To address particular input of a model/preprocessor, the `input(input_id)` method, where `input_id` is a positional index or input tensor name for input in `model.inputs`, if a model has a single input, `input_id` can be omitted. After reading the image from the disc, it contains U8 pixels in the `[0, 255]` range and is stored in the `NHWC` layout. To perform a preprocessing conversion, we should provide this to the tensor description.

```
from openvino.runtime import Type, Layout

ppp.input(0).tensor().set_shape([1, 640, 640, 3]).set_element_type(Type.u8).set_layo
pass
```

To perform layout conversion, we also should provide information about layout expected by model

ppp.input(0).tensor().set_shape([1, 640, 640, 3]).
set_element_type(Type.u8).set_layout(Layout('NHWC'))

## Describe preprocessing steps

Our preprocessing function contains the following steps:

- Convert the data type from `U8` to `FP32`.
- Convert the data layout from `NHWC` to `NCHW` format.
- Normalize each pixel by dividing on scale factor 255.

`ppp.input(input_id).preprocess()` is used for defining a sequence of preprocessing steps:

```
ppp.input(0).preprocess().convert_element_type(Type.f32).convert_layout(Layout('NCHW

print(ppp)
```

ppp.input(0).preprocess().convert_element_type(Type.f32).
convert_layout(Layout('NCHW')).scale([255., 255., 255.])

```
Input "images":
    User's input tensor: [1,640,640,3], [N,H,W,C], u8
    Model's expected tensor: [1,3,?,?], [N,C,H,W], f32
    Pre-processing steps (3):
      convert type (f32): ([1,640,640,3], [N,H,W,C], u8) -> ([1,640,640,3], [N,H,W,C], f32)
      convert layout [N,C,H,W]: ([1,640,640,3], [N,H,W,C], f32) -> ([1,3,640,640], [N,C,H,W], f32)
      scale (255,255,255): ([1,3,640,640], [N,C,H,W], f32) -> ([1,3,640,640], [N,C,H,W], f32)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Integrating Steps into a Model

Once the preprocessing steps have been finished, the model can be finally built. Additionally, we can save a completed model to OpenVINO IR, using `openvino.runtime.serialize`.

```
quantized_model_with_preprocess = ppp.build()
serialize(quantized_model_with_preprocess, str(int8_model_det_path.with_name(f"{DET
```

The model with integrated preprocessing is ready for loading to a device. Now, we can skip these preprocessing steps in detect function:

serialize(quantized_model_with_preprocess, str(int8_model_det_path.with_name(f"{DET_MODEL_NAME}_with_preprocess.xml")))

```
def detect_without_preprocess(image:np.ndarray, model:Model):
    """
    OpenVINO YOLOv8 model with integrated preprocessing inference function.
    Preprocess image, runs model inference and postprocess results using NMS.
    Parameters:
        image (np.ndarray): input image.
        model (Model): OpenVINO compiled model.
    Returns:
        detections (np.ndarray): detected boxes in format
                                        [x1, y1, x2, y2, score, label]
    """
    output_layer = model.output(0)
    img = letterbox(image)[0]
    input_tensor = np.expand_dims(img, 0)
    input_hw = img.shape[:2]
    result = model(input_tensor)[output_layer]
    detections = postprocess(result, input_hw, image)
    return detections

compiled_model = core.compile_model(quantized_model_with_preprocess, device)
input_image = np.array(Image.open(IMAGE_PATH))
detections = detect_without_preprocess(input_image, compiled_model)[0]
image_with_boxes = draw_results(detections, input_image, label_map)

Image.fromarray(image_with_boxes)
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Live demo 13)

The following code runs model inference on a video:

```
import collections
import time
from IPython import display

# Main processing function to run object detection.
def run_object_detection(source=0, flip=False, use_popup=False, skip_first_frames=0,
    player = None
    if device != "CPU":
        model.reshape({0: [1, 3, 640, 640]})
    compiled_model = core.compile_model(model, device)
    try:
        # Create a video player to play with target fps.
        player = VideoPlayer(
            source=source, flip=flip, fps=30, skip_first_frames=skip_first_frames
        )
        # Start capturing.
        player.start()
        if use_popup:
            title = "Press ESC to Exit"
            cv2.namedWindow(
                winname=title, flags=cv2.WINDOW_GUI_NORMAL | cv2.WINDOW_AUTOSIZE
            )
```

def run_object_detection(source=0, flip=False, use_popup=False, skip_first_frames=0, model=det_model, device=device):

```
        processing_times = collections.deque()
        while True:
            # Grab the frame.
            frame = player.next()
            if frame is None:
                print("Source ended")
                break
            # If the frame is larger than full HD, reduce size to improve the perfor
            scale = 1280 / max(frame.shape)
            if scale < 1:
                frame = cv2.resize(
                    src=frame,
                    dsize=None,
                    fx=scale,
                    fy=scale,
                    interpolation=cv2.INTER_AREA,
                )
            # Get the results.
            input_image = np.array(frame)

            start_time = time.time()
            # model expects RGB image, while video capturing in BGR
            detections = detect(input_image[:, :, ::-1], compiled_model)[0]
            stop_time = time.time()

            image_with_boxes = draw_results(detections, input_image, label_map)
            frame = image_with_boxes
            processing_times.append(stop_time - start_time)
            # Use processing times from last 200 frames.
            if len(processing_times) > 200:
                processing_times.popleft()
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

```
    _, f_width = frame.shape[:2]
    # Mean processing time [ms].
    processing_time = np.mean(processing_times) * 1000
    fps = 1000 / processing_time
    cv2.putText(
        img=frame,
        text=f"Inference time: {processing_time:.1f}ms ({fps:.1f} FPS)",
        org=(20, 40),
        fontFace=cv2.FONT_HERSHEY_COMPLEX,
        fontScale=f_width / 1000,
        color=(0, 0, 255),
        thickness=1,
        lineType=cv2.LINE_AA,
    )
    # Use this workaround if there is flickering.
    if use_popup:
        cv2.imshow(winname=title, mat=frame)
        key = cv2.waitKey(1)
        # escape = 27
        if key == 27:
            break
    else:
        # Encode numpy array to jpg.
        _, encoded_img = cv2.imencode(
            ext=".jpg", img=frame, params=[cv2.IMWRITE_JPEG_QUALITY, 100]
        )
        # Create an IPython image.
        i = display.Image(data=encoded_img)
        # Display the image in this notebook.
        display.clear_output(wait=True)
        display.display(i)
```

```
    # ctrl-c
    except KeyboardInterrupt:
        print("Interrupted")
    # any different error
    except RuntimeError as e:
        print(e)
    finally:
        if player is not None:
            # Stop capturing.
            player.stop()
        if use_popup:
            cv2.destroyAllWindows()
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

## Run

### Run Live Object Detection and Segmentation

Use a webcam as the video input. By default, the primary webcam is set with `source=0`. If you have multiple webcams, each one will be assigned a consecutive number starting at 0. Set `flip=True` when using a front-facing camera. Some web browsers, especially Mozilla Firefox, may cause flickering. If you experience flickering, set `use_popup=True`.

> **NOTE**: To use this notebook with a webcam, you need to run the notebook on a computer with a webcam. If you run the notebook on a remote server (for example, in Binder or Google Colab service), the webcam will not work. By default, the lower cell will run model inferece on a video file. If you want to try live inference on your webcam set `WEBCAM_INFERENCE = True`

Run the object detection:

```
WEBCAM_INFERENCE = False

if WEBCAM_INFERENCE:
    VIDEO_SOURCE = 0  # Webcam
else:
    VIDEO_SOURCE = 'https://storage.openvinotoolkit.org/repositories/openvino_notebc
```

VIDEO_SOURCE = 'https://storage.openvinotoolkit.org/repositories/openvino_notebooks/data/data/video/people.mp4'

```
run_object_detection(source=VIDEO_SOURCE, flip=True, use_popup=False,
                     model=det_ov_model, device="AUTO")
```

Run instance segmentation:

```
run_object_detection(source=VIDEO_SOURCE, flip=True, use_popup=False, model=seg_ov_model, device="AUTO")
```

# Convert and Optimize YOLOv8 with OpenVINO™

https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1


Object Detection


Instance Segmentation