Using Real-ESRGAN to Upscale Images on the Intel Platform

User Guide

February 2024

Document Number: 816445-0.8

Contents

Figures

Tables

Revision History

Date	Revision	Description
February 2024	0.8	Initial release.

1.0 Introduction

Real-ESRGAN is an open-source project for General Image/Video Restoration. It is an image enhancement model designed for upscaling low-resolution images while preserving and enhancing details. It is widely used in various image enhancements like video upscaling, medical imaging, digital art, design, etc.

This guide will take a ×4 model as an example, demonstrating how to deploy this mode efficiently on the Intel platform with static and dynamic input shapes.

OpenVINO is an Intel open-source toolkit designed for optimizing and deploying AI inference on Intel hardware. It provides a comprehensive set of tools and libraries to leverage the capabilities of Intel processors for efficient and high-performance AI processing tasks.

2.0 Prerequisites

The environment for this guide is Ubuntu 22.04. To take advantage of the integrated GPU (iGPU) and discrete GPU (dGPU) for acceleration, follow the official OpenVINO guide to install the iGPU driver and related dependencies.

The examples in this guide have also been validated on platforms with Intel® Core™ processors with integrated GPU and Intel® Arc™ graphics.

Table 1. Main Components

Components	Version
OS	Ubuntu 22.04
OpenVINO	OpenVINO 2023.2
Python	3.8.18
Real-ESRGAN	V0.3.0
Hardware platform	Intel® Core™ platform (validate with Tiger Lake / Alder Lake / Raptor Lake)

3.0 Model Conversion

This section will show how to convert the model from Real-ESRGAN projects.

Clone the repo and install dependent packages:

```
git clone https://github.com/xinntao/Real-ESRGAN.git
cd Real-ESRGAN
pip install -r requirements.txt
python setup.py develop
```

Download the weighted file and put it in the default pre-trained model location: experiments/pretrained_models.

```
Wget https://github.com/xinntao/Real-ESRGAN/releases/download/v0.1.0/RealESRGAN_x4plus.pth
```

3.1 Convert the Model From pytorch to onnx With Static Input Shapes

Real-ESRGAN provides one pytorch to onnx script under: scripts/pytorch2onnx.py. To export the onnx model as static input, modify "x = torch.rand(1, 3, 512, 512)" below, where the 3rd and 4th values represent the height and width of the model inputs. You can modify the height and width to any size you want.

The model inference speed is much faster with static input shapes. You can convert the image to a static input shape if the following applies to your application:

- The images to upscale are to same size.
- You can accept the same resized and upscaled original image.

```
x = torch.rand(1, 3, 512, 512)
torch.onnx.export(model,               # model being run
            x,                         # model input (or a tuple for multiple inputs)
            "RealESRGAN_x4plus.onnx",   # where to save the model (can be a file or file-like object)
            export_params=True,        # store the trained parameter weights inside the model file
            opset_version=12,          # the ONNX version to export the model to
            do_constant_folding=True,  # whether to execute constant folding for optimization
            input_names = ['input'],   # the model's input names
            output_names = ['output']) # the model's output names
)
```

3.2 Convert the Model From pytorch to onnx With Dynamic Input Shapes

The pytorch dynamic axes empower models to adjust to input data with diverse dimensions. In the example below, the height and width are set as dynamic. Refer to the provided code for exporting. If the upscaling model is required to handle images with varying shapes, you can employ this approach. However, it comes with a trade-off, which may increase the inference time.

```
  dynamic_axes = {
    'input': { 2: 'height', 3: 'width'}
}
```

```
torch.onnx.export(model,              # model being run
            input,                    # model input (or a tuple for multiple inputs)
            onnx_path,                #"RealESRGAN_x4plus_dynamic.onnx",   # where to save the model (can be
a file or file-like object)
            export_params=True,       # store the trained parameter weights inside the model file
            opset_version=12,         # the ONNX version to export the model to
            do_constant_folding=True,  # whether to execute constant folding for optimization
            input_names = ['input'],   # the model's input names
            output_names = ['output']   # the model's output names
            dynamic_axes = dynamic_axes  # save the model as dynamic shape
            )
```

3.3 Convert the Model From onnx to IR

The Intermediate Representation (IR) is the format the OpenVINO accepts for inference. It is recommended to convert the onnx model to the IR model for better performance.

Install the OpenVINO. Version OpenVINO release 2023.2 is used here.

```
pip install openvino==2023.2
```

Convert the model to IR format:

```
import openvino as ov

onnx_path = 'RealESRGAN_x4plus_512.onnx'
ov_model = ov.convert_model(onnx_path)

ov.save_model(ov_model, onnx_path.split(".")[0]+ '.xml')
```

After this step, the following two model IR files are generated:

- Static model IR files RealESRGAN_x4plus_512.xml/bin
- Dynamic model IR files RealESRGAN_x4plus_dynamc.xml/bin

4.0 OpenVINO Inference

Install Opencv for image processing.

```
pip install opencv-python
```

4.1 OpenVINO Inference Static Model

Define a class named 'Upscale_rESRGAN()'. Start by initializing the model path and output path. Then, check for available devices; if a GPU is available, use it for inference; otherwise, default to the CPU.

```
class Upscale_rESRGAN(object):
    def __init__(self):
    ##define model path
        model_all_local_path = "models/RealESRGAN_x4plus_512.xml"
        self.output_folder = "output/"

        os.makedirs(str(self.output_folder), exist_ok=True)
        core = ov.Core()
        device_list = core.available_devices
        print("Available device: ", device_list)
        gpu_devices = [device for device in device_list if 'GPU' in device]

        if gpu_devices:
            selected_device = gpu_devices[0]  # 选择第一个 GPU 设备
            print(f"选择的设备是：{selected_device}")
        else:
            selected_device = 'CPU'   # 选择 CPU 设备
            print("没有找到 GPU 设备，选择 CPU。")
```

Read and compile the model on the inference device, then create an inference request and prepare for inference.

```
model = core.read_model(model_all_local_path)

        self.input_tensor_name = model.inputs[0].get_any_name()

        input_shape = model.input(self.input_tensor_name).shape

        compiled_model = core.compile_model(model, device_name=selected_device)
        self.output_tensor = compiled_model.outputs[0]

        self.infer_request = compiled_model.create_infer_request()
            return
```

Define a run function under 'Upscale_rESRGAN()'; it takes the path to the image as input and returns the result of upscaling.

The function reads the image path and converts the image from BGR to RGB, then resizes the image to model input size shape; the image is also normalized and transposed before inferencing.

```
def run(self, image_path):

    t1 = time.time()
    #图像前处理
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None:
       raise OpenError("Can't open the image from {}".format(image_path))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (self.input_width, self.input_height))
    image = image / 255.0
    image = np.transpose(image, (2, 0, 1))
    image = image.reshape(1, *image.shape)

    print("image.shape: ", image.shape, self.input_width, self.input_height)
    input_data = {self.input_tensor_name: image}
    # Set input tensor for model with one input
    result = self.infer_request.infer(input_data)[self.output_tensor]
print("resrgan execution time: {:.2f} seconds".format(t2-t1) )
```

Then, perform the post-processing of the output data and return the output, which is the image after upscaling.

```
    output = result[0].squeeze()
    output = np.clip(output, 0, 1) * 255.0
    output = output.astype(np.uint8)
    output = cv2.cvtColor(output.transpose(1, 2, 0), cv2.COLOR_RGB2BGR)

    return output
```

Below is one simple main function to define one upscaler, infer the result, and write the result as an image in the output path.

```
if __name__ == "__main__":

    t0 = time.time()
    image_path = "./7.png"
    output_folder = "output/"
    upscaler = Upscale_rESRGAN()
    image = upscaler.run(image_path)

    filename = os.path.basename(image_path)  # 获取文件名（例如：'0014.jpg'）
    image_name = os.path.splitext(filename)[0]  # 移除文件扩展名（例如：'0014'）

    output_path = str(Path(output_folder) / (image_name + "x4_s.jpg"))
    cv2.imwrite(output_path, image)
```

4.2 OpenVINO Inference Dynamic Model

The initialization first part is the same as the static part:

```
class Upscale_rESRGAN(object):
    def __init__(self):
    ##define model path
        model_all_local_path = "models/RealESRGAN_x4plus_dynamic.xml"
        self.output_folder = "output/"

        os.makedirs(str(self.output_folder), exist_ok=True)
        core = ov.Core()
        device_list = core.available_devices
        print("Available device: ", device_list)
        gpu_devices = [device for device in device_list if 'GPU' in device]

        if gpu_devices:
            selected_device = gpu_devices[0]  # 选择第一个 GPU 设备
            print(f"选择的设备是：{selected_device}")
        else:
            selected_device = 'CPU'  # 选择 CPU 设备
            print("没有找到 GPU 设备，选择 CPU。")
```

Read and compile the model on the inference device; the model needs to be reshaped to reflect what was set for the 3rd and 4th parameters as a dynamic shape. It is recommended to specify the dimension bounds if we are sure that the height/width high and lower bounds are known, like in the range of 512 to 768, as in the case below. This gives additional optimizations during the model compilation.

```
  model = core.read_model(model_all_local_path)

# Set third and fourth dimensions as dynamic
      model.reshape([1, 3, -1, -1])
      #model.reshape ([1, 3, (512, 768), (512, 768)])

      self.input_tensor_name = model.inputs[0].get_any_name()

      input_shape = model.input(self.input_tensor_name).shape

      compiled_model = core.compile_model(model, device_name=selected_device)
      self.output_tensor = compiled_model.outputs[0]

      self.infer_request = compiled_model.create_infer_request()
        return
```

The other parts are similar, except resizing the model before inferencing is unnecessary.

With dynamic input, the image is enhanced with a different image size without scaling the ratio.

4.3 Validate Inference Results

Select an image from Real-ESRGAN and utilize the dynamic model for upscaling. Real-ESRGAN effectively enhances the quality of blurred images, offering notable improvements in clarity and detail.

Figure 1. Original Image vs. Upscaled Image

5.0 Summary

This guide illustrates an example of image upscaling using OpenVINO with static and dynamic inputs. This technique proves valuable in enhancing or restoring visual quality and details. The provided Python script is easily integrable into image processing pipelines or other applications.

6.0 Terminology

Table 2. Terminology

Term	Description
AI	Artificial Intelligence
BGR	Blue Green Red
CPU	Central Processing Unit
dGPU	Discrete Graphics Processing Unit
GPU	Graphics Processing Unit
iGPU	Integrate Graphics Processing Unit
Intel® Arc™	Intel® Arc™ Graphics
IR	Intermediate Representation
MO	Model Optimizer
onnx	Open Neural Network Exchange
OpenVINO	An Intel open-source toolkit designed for optimizing and deploying AI inference on Intel hardware.
Real-ESRGAN	An open-source project for General Image/Video Restoration.
RGB	Red Green Blue

7.0 Reference Documents

Log in to the Resource and Design Center (rdc.intel.com) to search for and download the document numbers listed in the following table. Contact your Intel field representative for access.

Note: Third-party links are provided as a reference only. Intel does not control or audit third-party benchmark data or the websites referenced in this document. You should visit the referenced website and confirm whether the data is accurate.

Table 3. Reference Documents

Document	Document No./Location
OpenVINO	https://github.com/openvinotoolkit/openvino
Real-ESRGAN	https://github.com/xinntao/Real-ESRGAN/