```
nbdiff notebooks/001-hello-world/001-hello-world.ipynb (main) notebooks/001-hello-world/001-hello-world.ipynb
--- notebooks/001-hello-world/001-hello-world.ipynb (main)  (no timestamp)
+++ notebooks/001-hello-world/001-hello-world.ipynb  2021-09-11 13:51:39.697519
## modified /cells/3/source:
-  ## Load the network
+  ## Load the Model

## modified /cells/4/source:
@@ -1,6 +1,8 @@
 ie = IECore()
-net = ie.read_network(model="model/v3-small_224_1.0_float.xml")
-exec_net = ie.load_network(net, "CPU")
+net = ie.read_network(
+    model="model/v3-small_224_1.0_float.xml", weights="model/v3-small_224_1.0_float.bin"
+)
+exec_net = ie.load_network(network=net, device_name="CPU")

 input_key = next(iter(exec_net.input_info))
 output_key = next(iter(exec_net.outputs.keys()))


## modified /cells/6/source:
@@ -1,7 +1,7 @@
 # The MobileNet network expects images in RGB format
-image = cv2.cvtColor(cv2.imread("data/coco.jpg"), cv2.COLOR_BGR2RGB)
+image = cv2.cvtColor(cv2.imread(filename="data/coco.jpg"), code=cv2.COLOR_BGR2RGB)
 # resize to MobileNet image shape
-input_image = cv2.resize(image, (224, 224))
+input_image = cv2.resize(src=image, dsize=(224, 224))
 # reshape to network input shape
 input_image = np.expand_dims(input_image.transpose(2, 0, 1), 0)
 plt.imshow(image);


nbdiff notebooks/002-openvino-api/002-openvino-api.ipynb (main) notebooks/002-openvino-api/002-openvino-api.ipynb
--- notebooks/002-openvino-api/002-openvino-api.ipynb (main)  (no timestamp)
+++ notebooks/002-openvino-api/002-openvino-api.ipynb  2021-09-11 13:51:39.697519
## modified /cells/5/source:
@@ -1,4 +1,4 @@
 devices = ie.available_devices
 for device in devices:
-    device_name = ie.get_metric(device, "FULL_DEVICE_NAME")
+    device_name = ie.get_metric(device_name=device, metric_name="FULL_DEVICE_NAME")
     print(f"{device}: {device_name}")


## modified /cells/33/source:
@@ -1,4 +1,6 @@
 import numpy as np

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


## modified /cells/41/source:
@@ -8,7 +8,9 @@ segmentation_output_layer = next(iter(segmentation_net.outputs))

 print("~~~~ ORIGINAL MODEL ~~~~")
 print(f"input layout: {segmentation_net.input_info[segmentation_input_layer].layout}")
-print(f"input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}")
+print(
+    f"input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}"
+)
 print(f"output shape: {segmentation_net.outputs[segmentation_output_layer].shape}")

 new_shape = (1, 3, 544, 544)
@@ -16,7 +18,9 @@ segmentation_net.reshape({segmentation_input_layer: new_shape})
 segmentation_exec_net = ie.load_network(network=segmentation_net, device_name="CPU")

 print("~~~~ RESHAPED MODEL ~~~~")
-print(f"net input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}")
+print(
+    f"net input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}"
+)
 print(
     f"exec_net input shape: "
     f"{segmentation_exec_net.input_info[segmentation_input_layer].tensor_desc.dims}"

## modified /cells/42/source:
-  The input shape for the segmentation network is [1,3,512,512], with an NHCW layout: the network expects 3-channel
images with a width and height of 512 and a batch size of 1. We reshape the network to make it accept input images
with a width and height of 544 with the `.reshape()` method of `IENetwork`. This segmentation network always returns
arrays with the same width and height as the input width and height, so setting the input dimensions to 544x544 also
```

```
modifies the output dimensions. After reshaping, load the network to the device again.
+  The input shape for the segmentation network is [1,3,512,512], with an NCHW layout: the network expects 3-channel
images with a width and height of 512 and a batch size of 1. We reshape the network to make it accept input images
with a width and height of 544 with the `.reshape()` method of `IENetwork`. This segmentation network always returns
arrays with the same width and height as the input width and height, so setting the input dimensions to 544x544 also
modifies the output dimensions. After reshaping, load the network to the device again.

## modified /cells/45/source:
@@ -9,5 +9,7 @@ segmentation_net.batch_size = 2
 segmentation_exec_net = ie.load_network(network=segmentation_net, device_name="CPU")

 print(f"input layout: {segmentation_net.input_info[segmentation_input_layer].layout}")
-print(f"input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}")
+print(
+    f"input shape: {segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims}"
+)
 print(f"output shape: {segmentation_net.outputs[segmentation_output_layer].shape}")


## modified /cells/47/source:
@@ -6,7 +6,9 @@ segmentation_model_xml = "model/segmentation.xml"
 segmentation_net = ie.read_network(model=segmentation_model_xml)
 segmentation_input_layer = next(iter(segmentation_net.input_info))
 segmentation_output_layer = next(iter(segmentation_net.outputs))
-input_data = np.random.rand(*segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims)
+input_data = np.random.rand(
+    *segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims
+)
 segmentation_net.batch_size = 2
 segmentation_exec_net = ie.load_network(network=segmentation_net, device_name="CPU")
 result_batch = segmentation_exec_net.infer({segmentation_input_layer: input_data})

## modified /cells/49/source:
@@ -7,7 +7,9 @@ segmentation_net = ie.read_network(model=segmentation_model_xml)
 segmentation_input_layer = next(iter(segmentation_net.input_info))
 segmentation_output_layer = next(iter(segmentation_net.outputs))
 segmentation_net.batch_size = 2
-input_data = np.random.rand(*segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims)
+input_data = np.random.rand(
+    *segmentation_net.input_info[segmentation_input_layer].tensor_desc.dims
+)
 segmentation_exec_net = ie.load_network(network=segmentation_net, device_name="CPU")
 result_batch = segmentation_exec_net.infer({segmentation_input_layer: input_data})


nbdiff notebooks/004-hello-detection/004-hello-detection.ipynb (main) notebooks/004-hello-detection/004-hello-
detection.ipynb
--- notebooks/004-hello-detection/004-hello-detection.ipynb (main)  (no timestamp)
+++ notebooks/004-hello-detection/004-hello-detection.ipynb  2021-09-11 13:51:39.701519
## modified /cells/3/source:
-  ## Load the network
+  ## Load the Model

## modified /cells/6/source:
@@ -10,4 +10,4 @@ resized_image = cv2.resize(image, (W, H))
 # Reshape to network input shape
 input_image = np.expand_dims(resized_image.transpose(2, 0, 1), 0)

-plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

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


## modified /cells/9/source:
-  ## Visualize data
+  ## Visualize Results

## modified /cells/11/source:
@@ -1,3 +1,3 @@
 plt.figure(figsize=(10, 6))
 plt.axis("off")
-plt.imshow(convert_result_to_image(image, resized_image, boxes, conf_labels=False))

+plt.imshow(convert_result_to_image(image, resized_image, boxes, conf_labels=False));


nbdiff notebooks/101-tensorflow-to-openvino/101-tensorflow-to-openvino.ipynb (main) notebooks/101-tensorflow-to-
openvino/101-tensorflow-to-openvino.ipynb
--- notebooks/101-tensorflow-to-openvino/101-tensorflow-to-openvino.ipynb (main)  (no timestamp)
+++ notebooks/101-tensorflow-to-openvino/101-tensorflow-to-openvino.ipynb  2021-09-11 13:51:39.701519
## modified /cells/2/source:
@@ -7,4 +7,5 @@ import cv2
 import matplotlib.pyplot as plt
 import mo_tf
 import numpy as np
+from IPython.display import Markdown
 from openvino.inference_engine import IECore
```

```
## modified /cells/6/source:
@@ -1,9 +1,5 @@
-# Get the path to the Model Optimizer script
-mo_path = str(Path(mo_tf.__file__))
-
 # Construct the command for Model Optimizer
-mo_command = f""""{sys.executable}"
-                  "{mo_path}"
+mo_command = f"""mo
                  --input_model "{model_path}"
                  --input_shape "[1,224,224,3]"
                  --mean_values="[127.5,127.5,127.5]"
@@ -13,4 +9,4 @@ mo_command = f""""{sys.executable}"
                  """
 mo_command = " ".join(mo_command.split())
 print("Model Optimizer command to convert TensorFlow to OpenVINO:")
-print(mo_command)

+display(Markdown(f"`{mo_command}`"))


## modified /cells/10/source:
@@ -1,3 +1,3 @@
 ie = IECore()
-net = ie.read_network(str(ir_path))
-exec_net = ie.load_network(net, "CPU")

+net = ie.read_network(model=ir_path, weights=ir_path.with_suffix(".bin"))
+exec_net = ie.load_network(network=net, device_name="CPU")


## modified /cells/14/source:
@@ -1,7 +1,7 @@
-image = cv2.cvtColor(cv2.imread("data/coco.jpg"), cv2.COLOR_BGR2RGB)
+image = cv2.cvtColor(cv2.imread(filename="data/coco.jpg"), code=cv2.COLOR_BGR2RGB)
 # Resize image to network input image shape
-resized_image = cv2.resize(image, (224, 224))
+resized_image = cv2.resize(src=image, dsize=(224, 224))
 # Transpose image to network input shape
 input_image = np.reshape(resized_image, network_input_shape) / 255
 input_image = np.expand_dims(np.transpose(resized_image, (2, 0, 1)), 0)
-plt.imshow(image)

+plt.imshow(image);


## modified /cells/17/source:
@@ -1,5 +1,3 @@
 imagenet_classes = json.loads(open("utils/imagenet_class_index.json").read())
-imagenet_classes = {
-    int(key) + 1: value for key, value in imagenet_classes.items()
-}
+imagenet_classes = {int(key) + 1: value for key, value in imagenet_classes.items()}
 imagenet_classes[result_index]


nbdiff notebooks/102-pytorch-onnx-to-openvino/102-pytorch-onnx-to-openvino.ipynb (main) notebooks/102-pytorch-onnx-to-
openvino/102-pytorch-onnx-to-openvino.ipynb
--- notebooks/102-pytorch-onnx-to-openvino/102-pytorch-onnx-to-openvino.ipynb (main)  (no timestamp)
+++ notebooks/102-pytorch-onnx-to-openvino/102-pytorch-onnx-to-openvino.ipynb  2021-09-11 13:51:39.701519
## modified /cells/3/source:
@@ -1,6 +1,6 @@
+import os
 import sys
 import time
-import os
 from pathlib import Path

 import cv2
@@ -9,8 +9,12 @@ import mo_onnx
 import numpy as np
 import torch
 from fastseg import MobileV3Large
+from IPython.display import Markdown, display
 from openvino.inference_engine import IECore

-
 sys.path.append("../utils")
-from notebook_utils import CityScapesSegmentation, segmentation_map_to_image, viz_result_image

+from notebook_utils import (
+    CityScapesSegmentation,
+    segmentation_map_to_image,
+    viz_result_image,
+)
```

```
## modified /cells/5/source:
@@ -1,7 +1,7 @@
 IMAGE_WIDTH = 1024  # Suggested values: 2048, 1024 or 512. The minimum width is 512.
 # Set IMAGE_HEIGHT manually for custom input sizes. Minimum height is 512
 IMAGE_HEIGHT = 1024 if IMAGE_WIDTH == 2048 else 512
-DIRECTORY_NAME = 'model'
+DIRECTORY_NAME = "model"
 BASE_MODEL_NAME = DIRECTORY_NAME + f"/fastseg{IMAGE_WIDTH}"

 # Paths where PyTorch, ONNX and OpenVINO IR models will be stored

## modified /cells/6/source:
@@ -1,3 +1,3 @@
 ### Download the Fastseg Model

-This downloads and loads the model and pretrained weights. It may take some time.

+Download, load and save the model with pretrained weights. This may take some time if you have not downloaded the
model before.


## modified /cells/7/source:
@@ -3,8 +3,6 @@ model = MobileV3Large.from_pretrained().cpu().eval()
 print("Loaded PyTorch Fastseg model")

 # Save the model
-path_to_dir = f"{os.getcwd()}/{DIRECTORY_NAME}"
-os.makedirs(path_to_dir, exist_ok=True)
-print("\nSaving the model")
+model_path.parent.mkdir(exist_ok=True)
 torch.save(model.state_dict(), str(model_path))
 print(f"Model saved at {model_path}")


## modified /cells/12/source:
@@ -1,9 +1,5 @@
-# Get the path to the Model Optimizer script
-mo_path = str(Path(mo_onnx.__file__))
-
 # Construct the command for Model Optimizer
-mo_command = f""""{sys.executable}"
-                  "{mo_path}"
+mo_command = f"""mo
                   --input_model "{onnx_path}"
                   --input_shape "[1,3, {IMAGE_HEIGHT}, {IMAGE_WIDTH}]"
                   --mean_values="[123.675, 116.28 , 103.53]"
@@ -13,4 +9,4 @@ mo_command = f""""{sys.executable}"
                   """
 mo_command = " ".join(mo_command.split())
 print("Model Optimizer command to convert the ONNX model to OpenVINO:")
-print(mo_command)

+display(Markdown(f"`{mo_command}`"))


## modified /cells/20/source:
@@ -1,5 +1,7 @@
 # Convert network result to segmentation map and display the result
 result_mask_onnx = np.squeeze(np.argmax(res_onnx, axis=1)).astype(np.uint8)
 viz_result_image(
-    image, segmentation_map_to_image(result_mask_onnx, CityScapesSegmentation.get_colormap()), resize=True
+    image,
+    segmentation_map_to_image(result_mask_onnx, CityScapesSegmentation.get_colormap()),
+    resize=True,
 )


## modified /cells/23/source:
@@ -1,4 +1,6 @@
 result_mask_ir = np.squeeze(np.argmax(res_ir, axis=1)).astype(np.uint8)
 viz_result_image(
-    image, segmentation_map_to_image(result_mask_ir, CityScapesSegmentation.get_colormap()), resize=True
+    image,
+    segmentation_map_to_image(result_mask_ir, CityScapesSegmentation.get_colormap()),
+    resize=True,
 )


## modified /cells/25/source:
@@ -3,5 +3,9 @@ with torch.no_grad():

 result_mask_torch = torch.argmax(result_torch, dim=1).squeeze(0).numpy().astype(np.uint8)
 viz_result_image(
-    image, segmentation_map_to_image(result_mask_torch, CityScapesSegmentation.get_colormap()), resize=True
+    image,
+    segmentation_map_to_image(
+        result=result_mask_torch, colormap=CityScapesSegmentation.get_colormap()
+    ),
+    resize=True,
 )


## modified /cells/27/source:
@@ -31,26 +31,25 @@ print(
     f"FPS: {num_images/time_torch:.2f}"
 )

-## Uncomment the following lines for iGPU performance stats
-
-# exec_net_onnx_gpu = ie.load_network(network=net_ir, device_name="GPU")
-# start = time.perf_counter()
-# for _ in range(num_images):
-#     exec_net_onnx_gpu.infer(inputs={input_layer_onnx: input_image})
-# end = time.perf_counter()
-# time_onnx_gpu = end - start
-# print(
-#     f"ONNX model in Inference Engine/GPU: {time_onnx_gpu/num_images:.3f} "
-#     f"seconds per image, FPS: {num_images/time_onnx_gpu:.2f}"
-# )
+if "GPU" in ie.available_devices:
+    exec_net_onnx_gpu = ie.load_network(network=net_ir, device_name="GPU")
+    start = time.perf_counter()
+    for _ in range(num_images):
+        exec_net_onnx_gpu.infer(inputs={input_layer_onnx: input_image})
+    end = time.perf_counter()
+    time_onnx_gpu = end - start
+    print(
+        f"ONNX model in Inference Engine/GPU: {time_onnx_gpu/num_images:.3f} "
+        f"seconds per image, FPS: {num_images/time_onnx_gpu:.2f}"
+    )

-# exec_net_ir_gpu = ie.load_network(network=net_ir, device_name="GPU")
-# start = time.perf_counter()
-# for _ in range(num_images):
-#     exec_net_ir_gpu.infer(inputs={input_layer_ir: input_image})
-# end = time.perf_counter()
-# time_ir_gpu = end - start
-# print(
-#     f"IR model in Inference Engine/GPU: {time_ir_gpu/num_images:.3f} "
-#     f"seconds per image, FPS: {num_images/time_ir_gpu:.2f}"
-# )

+    exec_net_ir_gpu = ie.load_network(network=net_ir, device_name="GPU")
+    start = time.perf_counter()
+    for _ in range(num_images):
+        exec_net_ir_gpu.infer(inputs={input_layer_ir: input_image})
+    end = time.perf_counter()
+    time_ir_gpu = end - start
+    print(
+        f"IR model in Inference Engine/GPU: {time_ir_gpu/num_images:.3f} "
+        f"seconds per image, FPS: {num_images/time_ir_gpu:.2f}"
+    )


## inserted before /cells/28:
+  markdown cell:
+    source:
+      **Show Device Information**
+  code cell:
+    source:
+      devices = ie.available_devices
+      for device in devices:
+          device_name = ie.get_metric(device_name=device, metric_name="FULL_DEVICE_NAME")
+          print(f"{device}: {device_name}")

## deleted /cells/28-29:
-  markdown cell:
-    source:
-      **Show CPU Information for reference**
-  code cell:
-    source:
-      try:
-          import cpuinfo
-
-          print(cpuinfo.get_cpu_info()["brand_raw"])
-      except Exception:
-          # OpenVINO installs cpuinfo, but if a different version is installed
-          # the command above may not work
-          import platform
-
-          print(platform.processor())
```

```
nbdiff notebooks/103-paddle-onnx-to-openvino/103-paddle-onnx-to-openvino-classification.ipynb (main) notebooks/103-
paddle-onnx-to-openvino/103-paddle-onnx-to-openvino-classification.ipynb
--- notebooks/103-paddle-onnx-to-openvino/103-paddle-onnx-to-openvino-classification.ipynb (main)  (no timestamp)
+++ notebooks/103-paddle-onnx-to-openvino/103-paddle-onnx-to-openvino-classification.ipynb  2021-09-11 13:51:39.701519
## inserted before /cells/1:
+  markdown cell:
+    source:
+      ## Preparation

## deleted /cells/1-2:
-  markdown cell:
-    source:
-      ## Preparation
-
-      ### Install PaddlePaddle and Upgrade OpenVINO
-
-      The Model Optimizer command in this notebook, `mo`, was introduced in OpenVINO 2021.4. If your OpenVINO version
is lower, running this cell will upgrade openvino-dev. You can also upgrade all the packages in openvino_env by
running `pip install --upgrade -r requirements.txt` in a terminal where you activated the openvino_env environment.
-  code cell:
-    source:
-      import openvino.inference_engine
-
-      if "2021.4" not in openvino.inference_engine.get_version():
-          print("Installing OpenVINO 2021.4. This may take a while...")
-          print("It is recommended to restart the Jupyter Kernel after installation, "
-                "with Kernel->Restart Kernel")
-          install_result = %sx python -m pip install --upgrade openvino-dev==2021.4.*
-          install_result = [line for line in install_result if "Requirement already " not in line]
-          print("\n".join(install_result))
-      else:
-          print("OpenVINO 2021.4 is installed.")

## modified /cells/8/source:
@@ -1,6 +1,6 @@
 classifier = hub.Module(name=MODEL_NAME)
 # Load image in BGR format, as specified in model documentation
-image = cv2.imread(IMAGE_FILENAME)
+image = cv2.imread(filename=IMAGE_FILENAME)
 plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
 result = classifier.classification(images=[image], top_k=3)
 for class_name, softmax_probability in result[0].items():

## modified /cells/24/source:
@@ -1,7 +1,7 @@
 # Load Inference Engine and IR model
 ie = IECore()
-net = ie.read_network(f"{MODEL_NAME}.xml")
-exec_net = ie.load_network(net, "CPU")
+net = ie.read_network(model=f"{MODEL_NAME}.xml", weights=f"{MODEL_NAME}.bin")
+exec_net = ie.load_network(network=net, device_name="CPU")

 # Get model input and outputs
 input_layer = next(iter(net.input_info))

## modified /cells/26/source:
@@ -1,5 +1,5 @@
 num_images = 50
 # PaddlePaddle's classification method expects a BGR numpy array
-image = cv2.imread(IMAGE_FILENAME)
+image = cv2.imread(filename=IMAGE_FILENAME)
 # The process_image function expects a PIL image
-pil_image = Image.open(IMAGE_FILENAME)

+pil_image = Image.open(fp=IMAGE_FILENAME)


## inserted before /cells/27:
+  code cell:
+    source:
+      # Show CPU information
+      ie = IECore()
+      print(f"CPU: {ie.get_metric('CPU', 'FULL_DEVICE_NAME')}")

## deleted /cells/27:
-  code cell:
-    source:
-      # Show devices available for OpenVINO Inference Engine
-      ie = IECore()
-      for device in ie.available_devices:
-          device_name = ie.get_metric(device, "FULL_DEVICE_NAME")
-          print(f"{device}: {device_name}")

## modified /cells/28/source:
@@ -1,3 +1,4 @@
+# Show inference speed on PaddlePaddle model
 start = time.perf_counter()
 for _ in range(num_images):
     result = classifier.classification(images=[image], top_k=3)
@@ -10,4 +11,4 @@ print(
 print("PaddlePaddle result:")
 for class_name, softmax_probability in result[0].items():
     print(f"{class_name}, {softmax_probability:.5f}")
-plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

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


## modified /cells/29/source:
@@ -1,12 +1,11 @@
-device = "CPU"
-exec_net = ie.load_network(net, device)
+# Show inference speed on OpenVINO IR model
+exec_net = ie.load_network(net, "CPU")

 start = time.perf_counter()
 input_image = process_image(pil_image)

 for _ in range(num_images):
     ie_result = exec_net.infer(inputs={input_layer: input_image})[output_layer][0]
-
     result_index = np.argmax(ie_result)
     class_name = classifier.label_list[np.argmax(ie_result)]
     softmax_result = softmax(ie_result)
@@ -16,12 +15,12 @@ for _ in range(num_images):
 end = time.perf_counter()
 time_ir = end - start

-
 print(
-    f"IR model in Inference Engine/{device}: {time_ir/num_images:.4f} "
-    f"seconds per image, FPS: {num_images/time_ir:.2f}\n"
+    f"IR model in Inference Engine (CPU): {time_ir/num_images:.4f} "
+    f"seconds per image, FPS: {num_images/time_ir:.2f}"
 )
+print()
 print("OpenVINO result:")
 for index, softmax_probability in zip(top_indices, top_softmax):
     print(f"{classifier.label_list[index]}, {softmax_probability:.5f}")
-plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

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


nbdiff notebooks/105-language-quantize-bert/105-language-quantize-bert.ipynb (main) notebooks/105-language-quantize-
bert/105-language-quantize-bert.ipynb
--- notebooks/105-language-quantize-bert/105-language-quantize-bert.ipynb (main)  (no timestamp)
+++ notebooks/105-language-quantize-bert/105-language-quantize-bert.ipynb  2021-09-11 13:51:39.701519
## modified /cells/1/source:
@@ -3,6 +3,7 @@ import sys
 import time
 import warnings
 from pathlib import Path
+from zipfile import ZipFile

 import numpy as np
 import torch
@@ -16,10 +17,11 @@ from compression.pipeline.initializer import create_pipeline
 from compression.utils.logger import get_logger, init_logger
 from torch.utils.data import TensorDataset
 from transformers import BertForSequenceClassification, BertTokenizer
-from transformers import glue_convert_examples_to_features as convert_examples_to_features
+from transformers import (
+    glue_convert_examples_to_features as convert_examples_to_features,
+)
 from transformers import glue_output_modes as output_modes
 from transformers import glue_processors as processors
-from zipfile import ZipFile

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


## modified /cells/2/source:
@@ -1,7 +1,7 @@
 DATA_DIR = "data"
 MODEL_DIR = "model"
 MODEL_LINK = "https://download.pytorch.org/tutorial/MRPC.zip"
-FILE_NAME = MODEL_LINK.split('/')[-1]
+FILE_NAME = MODEL_LINK.split("/")[-1]

 os.makedirs(DATA_DIR, exist_ok=True)
 os.makedirs(MODEL_DIR, exist_ok=True)
```

```
## modified /cells/4/source:
@@ -1,3 +1,3 @@
 download_file(MODEL_LINK, directory=MODEL_DIR, show_progress=False)
-with ZipFile(f'{MODEL_DIR}/{FILE_NAME}', 'r') as zip_ref:
+with ZipFile(f"{MODEL_DIR}/{FILE_NAME}", "r") as zip_ref:
     zip_ref.extractall(MODEL_DIR)


## modified /cells/10/source:
@@ -1 +1,4 @@
-download_file("https://raw.githubusercontent.com/huggingface/transformers/f98ef14d161d7bcdc9808b5ec399981481411cc1
/utils/download_glue_data.py", show_progress=False)

+download_file(
+    "https://raw.githubusercontent.com/huggingface/transformers/f98ef14d161d7bcdc9808b5ec399981481411cc1/utils
/download_glue_data.py",
+    show_progress=False,
+)


## modified /cells/11/source:
@@ -1,4 +1,3 @@
 from download_glue_data import format_mrpc

-os.makedirs(DATA_DIR, exist_ok=True)
 format_mrpc(DATA_DIR, "")


## modified /cells/15/source:
@@ -17,7 +17,9 @@ class Accuracy(Metric):
         return {self._name: np.ravel(self._matches).mean()}

     def update(self, output, target):
-        """Updates prediction matches.
+        """
+        Updates prediction matches.
+
         :param output: model output
         :param target: annotations
         """
@@ -30,7 +32,9 @@ class Accuracy(Metric):
         self._matches.append(match)

     def reset(self):
-        """Resets collected matches"""
+        """
+        Resets collected matches
+        """
         self._matches = []

     def get_attributes(self):

## modified /cells/17/source:
@@ -23,10 +23,10 @@ algorithms = [


 # Step 1: Load the model.
-model = load_model(model_config)
+model = load_model(model_config=model_config)

 # Step 2: Initialize the data loader.
-data_loader = MRPCDataLoader(dataset_config)
+data_loader = MRPCDataLoader(config=dataset_config)

 # Step 3 (Optional. Required for AccuracyAwareQuantization): Initialize the metric.
 metric = Accuracy()
@@ -35,10 +35,10 @@ metric = Accuracy()
 engine = IEEngine(config=engine_config, data_loader=data_loader, metric=metric)

 # Step 5: Create a pipeline of compression algorithms.
-pipeline = create_pipeline(algorithms, engine)
+pipeline = create_pipeline(algo_config=algorithms, engine=engine)

 # Step 6 (Optional): Evaluate the original model. Print the results.
-fp_results = pipeline.evaluate(model)
+fp_results = pipeline.evaluate(model=model)
 if fp_results:
     print("FP16 model results:")
     for name, value in fp_results.items():

## modified /cells/18/source:
@@ -4,16 +4,15 @@ print(
     f"Quantizing model with {algorithms[0]['params']['preset']} preset and {algorithms[0]['name']}"
 )
 start_time = time.perf_counter()
-compressed_model = pipeline.run(model)
```

```
+compressed_model = pipeline.run(model=model)
 end_time = time.perf_counter()
 print(f"Quantization finished in {end_time - start_time:.2f} seconds")

 # Step 8 (Optional): Compress model weights to quantized precision
 #                     in order to reduce the size of final .bin file.
-compress_model_weights(compressed_model)
+compress_model_weights(model=compressed_model)

 # Step 9: Save the compressed model to the desired path.
-compressed_model_paths = save_model(
-    compressed_model, save_path=MODEL_DIR, model_name="quantized_bert_mrpc"
+compressed_model_paths = save_model(model=compressed_model, save_path=MODEL_DIR, model_name="quantized_bert_mrpc"
 )
 compressed_model_xml = compressed_model_paths[0]["model"]


## modified /cells/19/source:
@@ -1,5 +1,5 @@
 # Step 10 (Optional): Evaluate the compressed model. Print the results.
-int_results = pipeline.evaluate(compressed_model)
+int_results = pipeline.evaluate(model=compressed_model)

 if int_results:
     print("INT8 model results:")

nbdiff notebooks/201-vision-monodepth/201-vision-monodepth.ipynb (main) notebooks/201-vision-monodepth/201-vision-
monodepth.ipynb
--- notebooks/201-vision-monodepth/201-vision-monodepth.ipynb (main)  (no timestamp)
+++ notebooks/201-vision-monodepth/201-vision-monodepth.ipynb  2021-09-11 13:51:39.701519
## modified /cells/0/source:
@@ -1,3 +1,3 @@
-# MONODEPTH on OpenVINO IR Model
+# Monodepth with OpenVINO

 This notebook demonstrates Monocular Depth Estimation with MidasNet in OpenVINO. Model information:
https://docs.openvinotoolkit.org/latest/omz_models_model_midasnet.html


## modified /cells/5/source:
@@ -1,6 +1,5 @@
-import os
+import sys
 import time
-import urllib
 from pathlib import Path

 import cv2
@@ -16,4 +15,7 @@ from IPython.display import (
     clear_output,
     display,
 )
-from openvino.inference_engine import IECore

+from openvino.inference_engine import IECore
+
+sys.path.append("../utils")
+from notebook_utils import load_image


## modified /cells/7/source:
@@ -1,5 +1,4 @@
 DEVICE = "CPU"
 MODEL_FILE = "model/MiDaS_small.xml"

-model_name = os.path.basename(MODEL_FILE)
-model_xml_path = Path(MODEL_FILE).with_suffix(".xml")

+model_xml_path = Path(MODEL_FILE)


## modified /cells/9/source:
@@ -3,25 +3,6 @@ def normalize_minmax(data):
     return (data - data.min()) / (data.max() - data.min())


-def load_image(path: str):
-    """
-    Loads an image from `path` and returns it as BGR numpy array. `path`
-    should point to an image file, either a local filename or an url.
-    """
-    if path.startswith("http"):
-        # Set User-Agent to Mozilla because some websites block
-        # requests with User-Agent Python
-        request = urllib.request.Request(
-            path, headers={"User-Agent": "Mozilla/5.0"}
-        )
-        response = urllib.request.urlopen(request)
-        array = np.asarray(bytearray(response.read()), dtype="uint8")
-        image = cv2.imdecode(array, -1)  # Loads the image as BGR
-    else:
-        image = cv2.imread(path)
-    return image
-
-
 def convert_result_to_image(result, colormap="viridis"):
     """
     Convert network result of floating point numbers to an RGB image with

## modified /cells/10/source:
@@ -1,3 +1,3 @@
-## Load model and get model information
+## Load the Model

-Load the model in Inference Engine with `ie.read_network` and load it to the specified device with `ie.load_network`

+Load the model in Inference Engine with `ie.read_network` and load it to the specified device with `ie.load_network`.
Get input and output keys and the expected input shape for the model.


## modified /cells/11/source:
@@ -1,7 +1,5 @@
 ie = IECore()
-net = ie.read_network(
-    str(model_xml_path), str(model_xml_path.with_suffix(".bin"))
-)
+net = ie.read_network(model=model_xml_path, weights=model_xml_path.with_suffix(".bin"))
 exec_net = ie.load_network(network=net, device_name=DEVICE)

 input_key = list(exec_net.input_info)[0]

## modified /cells/13/source:
@@ -1,6 +1,6 @@
 IMAGE_FILE = "data/coco_bike.jpg"
-image = load_image(IMAGE_FILE)
+image = load_image(path=IMAGE_FILE)
 # resize to input shape for network
-resized_image = cv2.resize(image, (network_image_height, network_image_width))
+resized_image = cv2.resize(src=image, dsize=(network_image_height, network_image_width))
 # reshape image to network input shape NCHW
 input_image = np.expand_dims(np.transpose(resized_image, (2, 0, 1)), 0)


## modified /cells/15/source:
@@ -1,7 +1,7 @@
 result = exec_net.infer(inputs={input_key: input_image})[output_key]
 # convert network result of disparity map to an image that shows
 # distance as colors
-result_image = convert_result_to_image(result)
+result_image = convert_result_to_image(result=result)
 # resize back to original image shape. cv2.resize expects shape
 # in (width, height), [::-1] reverses the (height, width) shape to match this.
 result_image = cv2.resize(result_image, image.shape[:2][::-1])


## modified /cells/20/source:
@@ -18,7 +18,6 @@ SCALE_OUTPUT = 0.5
 FOURCC = cv2.VideoWriter_fourcc(*"vp09")

 # Create Path objects for the input video and the resulting video
-video_path = Path(f"{os.getcwd()}/output")
-
-os.makedirs(str(video_path), exist_ok=True)
-result_video_path = Path(f"{str(video_path.stem)}/coco_monodepth.mp4")
+output_directory = Path(f"output")
+output_directory.mkdir(exist_ok=True)
+result_video_path = output_directory / f"{Path(VIDEO_FILE).stem}_monodepth.mp4"


## modified /cells/22/source:
@@ -1,7 +1,7 @@
 cap = cv2.VideoCapture(str(VIDEO_FILE))
 ret, image = cap.read()
 if not ret:
-    raise ValueError(f"The video at {video_path} cannot be read.")
+    raise ValueError(f"The video at {VIDEO_FILE} cannot be read.")
 input_fps = cap.get(cv2.CAP_PROP_FPS)
 input_video_frame_height, input_video_frame_width = image.shape[:2]


## modified /cells/23/source:
-  ### Do Inference on video and create monodepth video
+  ### Do inference on video and create monodepth video
```

```
## modified /cells/24/source:
@@ -1,5 +1,5 @@
 # Initialize variables
-input_video_frame_nr = 0
+input_video_frame_nr = 0
 start_time = time.perf_counter()
 total_inference_duration = 0

@@ -15,9 +15,7 @@ out_video = cv2.VideoWriter(
 )

 num_frames = int(NUM_SECONDS * input_fps)
-total_frames = (
-    cap.get(cv2.CAP_PROP_FRAME_COUNT) if num_frames == 0 else num_frames
-)
+total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT) if num_frames == 0 else num_frames
 progress_bar = ProgressBar(total=total_frames)
 progress_bar.display()

@@ -34,9 +32,7 @@ try:
         # Only process every second frame
         # Prepare frame for inference
         # resize to input shape for network
-        resized_image = cv2.resize(
-            image, (network_image_height, network_image_width)
-        )
+        resized_image = cv2.resize(src=image, dsize=(network_image_height, network_image_width))
         # reshape image to network input shape NCHW
         input_image = np.expand_dims(np.transpose(resized_image, (2, 0, 1)), 0)

@@ -50,7 +46,7 @@ try:
         if input_video_frame_nr % (10 * ADVANCE_FRAMES) == 0:
             clear_output(wait=True)
             progress_bar.display()
-            # input_video_frame_nr // ADVANCE_FRAMES gives the number of
+            # input_video_frame_nr // ADVANCE_FRAMES gives the number of
             # frames that have been processed by the network
             display(
                 Pretty(
@@ -64,9 +60,7 @@ try:
         # Transform network result to RGB image
         result_frame = to_rgb(convert_result_to_image(result))
         # Resize image and result to target frame shape
-        result_frame = cv2.resize(
-            result_frame, (target_frame_width, target_frame_height)
-        )
+        result_frame = cv2.resize(result_frame, (target_frame_width, target_frame_height))
         image = cv2.resize(image, (target_frame_width, target_frame_height))
         # Put image and result side by side
         stacked_frame = np.hstack((image, result_frame))

## modified /cells/26/source:
@@ -1,9 +1,7 @@
 video = Video(result_video_path, width=800, embed=True)
 if not result_video_path.exists():
     plt.imshow(stacked_frame)
-    raise ValueError(
-        "OpenCV was unable to write the video file. Showing one video frame."
-    )
+    raise ValueError("OpenCV was unable to write the video file. Showing one video frame.")
 else:
     print(f"Showing monodepth video saved at\n{result_video_path.resolve()}")
     print(

nbdiff notebooks/301-tensorflow-training-openvino/301-tensorflow-training-openvino-pot.ipynb (main) notebooks/301-
tensorflow-training-openvino/301-tensorflow-training-openvino-pot.ipynb
--- notebooks/301-tensorflow-training-openvino/301-tensorflow-training-openvino-pot.ipynb (main)  (no timestamp)
+++ notebooks/301-tensorflow-training-openvino/301-tensorflow-training-openvino-pot.ipynb  2021-09-11 13:51:39.701519
## deleted /cells/2-3:
-  markdown cell:
-    source:
-      This notebook requires OpenVINO 2021.4. Running the next cell will check if OpenVINO 2021.4 is installed. If a
previous version is installed, running this cell will upgrade OpenVINO. That may take some time.
-  code cell:
-    source:
-      import openvino.inference_engine
-
-      if not "2021.4" in openvino.inference_engine.get_version():
-          print("Installing OpenVINO 2021.4. This may take a while...")
-          print("It is recommended to restart the Jupyter Kernel after installation, with Kernel->Restart Kernel")
-          install_result = %sx python -m pip install --upgrade openvino-dev==2021.4
-          install_result = [line for line in install_result if not "Requirement already satisfied" in line]
-          print("\n".join(install_result))
-      else:
-          print("OpenVINO 2021.4 is installed.")

## modified /cells/7/source:
```

```
@@ -1,7 +1,6 @@
 import copy
 import os
 import urllib
-from typing import List

 import cv2
 import matplotlib.pyplot as plt

## modified /cells/9/source:
@@ -1,5 +1,9 @@
 model_config = Dict(
-    {"model_name": "flower", "model": "model/flower/flower_ir.xml", "weights": "model/flower/flower_ir.bin"}
+    {
+        "model_name": "flower",
+        "model": "model/flower/flower_ir.xml",
+        "weights": "model/flower/flower_ir.bin",
+    }
 )

 engine_config = Dict({"device": "CPU", "stat_requests_number": 2, "eval_requests_number": 2})

## modified /cells/15/source:
@@ -1,28 +1,30 @@
 # Step 1: Load the model.
-model = load_model(model_config)
+model = load_model(model_config=model_config)
 original_model = copy.deepcopy(model)

 # Step 2: Initialize the data loader.
-data_loader = ClassificationDataLoader(data_dir)
+data_loader = ClassificationDataLoader(data_source=data_dir)

 # Step 3 (Optional. Required for AccuracyAwareQuantization): Initialize the metric.
 #        Compute metric results on original model
 metric = Accuracy()

 # Step 4: Initialize the engine for metric calculation and statistics collection.
-engine = IEEngine(engine_config, data_loader, metric)
+engine = IEEngine(config=engine_config, data_loader=data_loader, metric=metric)

 # Step 5: Create a pipeline of compression algorithms.
-pipeline = create_pipeline(algorithms, engine)
+pipeline = create_pipeline(algo_config=algorithms, engine=engine)

 # Step 6: Execute the pipeline.
-compressed_model = pipeline.run(model)
+compressed_model = pipeline.run(model=model)

 # Step 7 (Optional): Compress model weights quantized precision
 #                    in order to reduce the size of final .bin file.
-compress_model_weights(compressed_model)
+compress_model_weights(model=compressed_model)

 # Step 8: Save the compressed model and get the path to the model
-compressed_model_paths = save_model(compressed_model, os.path.join(os.path.curdir, "model/optimized"))
+compressed_model_paths = save_model(
+    model=compressed_model, save_path=os.path.join(os.path.curdir, "model/optimized")
+)
 compressed_model_xml = Path(compressed_model_paths[0]["model"])
 print(f"The quantized model is stored in {compressed_model_xml}")


## modified /cells/19/source:
@@ -18,7 +18,7 @@ inp_file_name = "output/A_Close_Up_Photo_of_a_Dandelion.jpg"
 urllib.request.urlretrieve(inp_img_url, inp_file_name)

 # Pre-process the image and get it ready for inference.
-input_image = pre_process_image(inp_file_name)
+input_image = pre_process_image(imagePath=inp_file_name)

 res = exec_net_pot.infer(inputs={input_layer: input_image})
 res = res[output_layer]

## modified /cells/20/source:
@@ -1,4 +1,4 @@
-## Compare inference speed
+### Compare Inference Speed

 Measure inference speed with the [OpenVINO Benchmark App](https://github.com/openvinotoolkit/openvino/tree/master
/inference-engine/tools/benchmark_tool).
```