401-nncf

March 29, 2024

0.1 Setting up the Working Directory

This cell is to ensure we change the directory to anomalib source code to have access to the datasets and config files. We assume that you already went through `001_getting_started.ipynb` and install the required packages.

[1]:
```
import os
from pathlib import Path

from git.repo import Repo

current_directory = Path.cwd()
if current_directory.name == "400_openvino":
    # On the assumption that, the notebook is located in
    #   ~/anomalib/notebooks/400_openvino/
    root_directory = current_directory.parent.parent
elif current_directory.name == "anomalib":
    # This means that the notebook is run from the main anomalib directory.
    root_directory = current_directory
else:
    # Otherwise, we'll need to clone the anomalib repo to the␣
 ↪`current_directory`
    repo = Repo.clone_from(url="https://github.com/openvinotoolkit/anomalib.
 ↪git", to_path=current_directory)
    root_directory = current_directory / "anomalib"

os.chdir(root_directory)
```

1 int-8 Model Quantization via NNCF

It is possible to use Neural Network Compression Framework (NNCF) with anomalib for inference optimization in OpenVINO with minimal accuracy drop.

This notebook demonstrates how NNCF is enabled in anomalib to optimize the model for inference. Before diving into the details, let's first train a model using the standard Torch training loop.

1.1 1. Standard Training without NNCF

To train model without NNCF, we use the standard training loop. We use the same training loop as in the Getting Started Notebook.

[ ]:
```
from anomalib.engine import Engine

from anomalib.config import get_configurable_parameters
from anomalib.data import get_datamodule
from anomalib.models import get_model
from anomalib.utils.callbacks import get_callbacks
```

1.1.1 Configuration

Similar to the Getting Started Notebook, we will start with the PADIM model. We follow the standard training loop, where we first import the config file, with which we import datamodule, model, callbacks and trainer, respectively.

[ ]:
```
MODEL = "padim"  # 'padim', 'stfpm'
CONFIG_PATH = f"src/anomalib/models/{MODEL}/config.yaml"

config = get_configurable_parameters(config_path=CONFIG_PATH)
```

Note that we will run OpenVINO's `benchmark_app` on the model, which requires the model to be in the ONNX format. Therefore, we set the `export_type` flag to `onnx` in the `optimization` config file. Let's check the current config:

[4]:
```
print(config["optimization"])
```

```
{'export_type': None}
```

As mentioned above, we need to explicitly state that we want onnx export mode to be able to run `benchmark_app` to compute the throughput and latency of the model.

[5]:
```
config["optimization"]["export_type"] = "onnx"
```

[ ]:
```
datamodule = get_datamodule(config)
model = get_model(config)
callbacks = get_callbacks(config)

# start training
engine = Engine(**config.trainer, callbacks=callbacks)
engine.fit(model=model, datamodule=datamodule)
fp32_results = engine.test(model=model, datamodule=datamodule)
```

1.2 2. Training with NNCF

The above results indicate the performance of the standard fp32 model. We will now use NNCF to optimize the model for inference using int8 quantization. Note, that NNCF and Anomalib integration is still in the experimental stage and currently only Padim and STFPM models are optimized. It is likely that other models would work with NNCF; however, we do not guarantee that the accuracy will not drop significantly.

1.2.1 2.1. Padim Model

To optimize the Padim model for inference, we need to add NNCF configurations to the `optimization` section of the config file. The following configurations are added to the config file:

```
optimization:
  export_type: null # options: torch, onnx, openvino
  nncf:
    apply: true
    input_info:
      sample_size: [1, 3, 256, 256]
    compression:
      algorithm: quantization
      preset: mixed
      initializer:
        range:
          num_init_samples: 250
        batchnorm_adaptation:
          num_bn_adaptation_samples: 250
```

After updating the `config.yaml` file, `config` could be reloaded and the model could be trained with NNCF enabled. Alternatively, we could manually add these NNCF settings to the `config` dictionary here. Since we already have the `config` dictionary, we will choose the latter option, and manually add the NNCF configs.

[7]:
```
config["optimization"]["nncf"] = {
    "apply": True,
    "input_info": {"sample_size": [1, 3, 256, 256]},
    "compression": {
        "algorithm": "quantization",
        "preset": "mixed",
        "initializer": {"range": {"num_init_samples": 250},␣
 ↪"batchnorm_adaptation": {"num_bn_adaptation_samples": 250}},
    },
}
```

Now that we have updated the config with the NNCF settings, we could train and tests the NNCF model that will be optimized via int8 quantization.

[ ]:
```
datamodule = get_datamodule(config)
model = get_model(config)
callbacks = get_callbacks(config)

# start training
```

```
engine = Engine(**config.trainer, callbacks=callbacks)
engine.fit(model=model, datamodule=datamodule)
int8_results = engine.test(model=model, datamodule=datamodule)
```

[9]:
```
print(fp32_results)
```

```
[{'pixel_F1Score': 0.7241847515106201, 'pixel_AUROC': 0.9831862449645996,
'image_F1Score': 0.9921259880065918, 'image_AUROC': 0.9960317611694336}]
```

[10]:
```
print(int8_results)
```

```
[{'pixel_F1Score': 0.7164928913116455, 'pixel_AUROC': 0.9731722474098206,
'image_F1Score': 0.9841269850730896, 'image_AUROC': 0.9904761910438538}]
```

As can be seen above, there is a slight performance drop in the accuracy of the model. However, the model is now ready to be optimized for inference. We could now use `benchmark_app` to compute the throughput and latency of the fp32 and int8 models and compare the results.

[ ]:
```
# Check the throughput and latency performance of the fp32 model.
!benchmark_app -m results/padim/mvtec/bottle/run/onnx/model.onnx -t 10
```

[ ]:
```
# Check the throughput and latency performance of the int8 model.
!benchmark_app -m results/padim/mvtec/bottle/run/compressed/model_nncf.onnx -t␣
↪10
```

We have observed approximately 30.1% performance improvement in the throughput and latency of the int8 model compared to the fp32 model. This is a significant performance improvement, which could be achieved with 6.94% drop pixel F1-Score.

1.2.2 2.2. STFPM Model

Same steps in 2.1 could be followed to optimize the STFPM model for inference. The only difference is that the `config.yaml` file for STFPM model, located here, should be updated with the following:

```
optimization:
  export_type: null # options: torch, onnx, openvino
  nncf:
    apply: true
    input_info:
      sample_size: [1, 3, 256, 256]
    compression:
      algorithm: quantization
      preset: mixed
      initializer:
        range:
          num_init_samples: 250
        batchnorm_adaptation:
          num_bn_adaptation_samples: 250
      ignored_scopes:
```

```yaml
      - "{re}.*__pow__.*"
    update_config:
      hyperparameter_search:
        parameters:
          lr:
            min: 1e-4
            max: 1e-2
```

This is to ensure that we achieve the best accuracy vs throughput trade-off.