OpenVINO™ と NNCF を使用した量子化で Big Transfer (BiT) モデルをさらに高速化

この記事は、Medium に公開されている「Accelerate Big Transfer (BiT) Model Even More with Quantization using OpenVINO and Neural Network Compression Framework (NNCF)」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

1. はじめに

このシリーズのパート 1 (英語) では、OpenVINO™ ツールキットを使用してコンピューター・ビジョン・タスクの Big Transfer (BiT) モデルの推論を高速化する方法を説明しました。具体的には、BiT モデルを OpenVINO™ 環境にインポートし、ハードウェアの最適化を活用して、パフォーマンスのベンチマークを測定するプロセスを説明しました。OpenVINO™ を使用することにより、オリジナルの TensorFlow* 実装と比較して、BiT のパフォーマンスが大幅に向上し、推論のレイテンシーが減少しました。非常に優れた結果が得られましたが、最適化の余地はまだ残されています。このパート 2 では、OpenVINO™ と Neural Network Compression Framework (ニューラル・ネットワーク圧縮フレームワーク、NNCF) および低精度 (INT8) の推論を利用して BiT モデルの推論をさらに強化します。NNCF は、ディープラーニングの推論向けに調整された量子化、プルーニング、スパース性などの、ニューラル・ネットワーク圧縮のための高度なツールを提供します。これらを組み合わせることで、元のサイズでは容易に実行できない BiT モデルを、電力とメモリーに制約がある環境でも実行できるようになります。この記事で説明する手法は、BiT 以外のディープラーニング・モデルにも適用できます。

2. モデルの量子化

モデルの量子化は、ニューラル・ネットワークの重みと活性化の精度を下げる最適化手法です。32 ビット浮動小数点表現 (FP32) を、16 ビット浮動小数点 (FP16)、8 ビット整数 (INT8)、4 ビット整数 (INT4) などのより小さなビット幅に変換します。主な利点は、モデルのサイズが小さくなり、推論時間が短縮されて、効率が向上することです。これらの改善により、サーバー・プラットフォーム上での効率が向上するだけでなく、リソースに制約のあるエッジデバイスへのデプロイも可能になります。サーバー・プラットフォームのパフォーマンスの向上も重要ですが、特筆すべき効果は、新しいデプロイの機会がもたらされることです。量子化により、データセンターに限定されていたモデルが計算やメモリーに制限がある低電力デバイスにもデプロイできるモデルに変わることにより、AI の範囲が真のエッジまで大きく拡大されます。

主要なモデルの量子化の概念を次に示します。

- **精度の削減** — 重みと活性化を表すために使用されるビット幅を減らします。INT8 や FP16 などのビット幅の、より小さなモデルを利用できます
- **効率** — 圧縮モデルは小さく高速であるため、システムリソースを効率良く利用できます。
- **トレードオフ** — ターゲット・ハードウェアに合わせてモデルの圧縮、速度、精度のバランスを取ります。目標は、あらゆる面にわたって最適化することです。
- **手法** — トレーニング後の量子化と量子化を考慮したトレーニング。復元力を考慮に入れて精度を下げます。
- **スキーム** — 重み、活性化、組み合わせメソッドなどの量子化手法は、モデルの圧縮と精度の維持を両立させます。
- **精度の維持** — 微調整、キャリブレーション、再トレーニングにより、実環境データの品質を維持します。

3. Neural Network Compression Framework (ニューラル・ネットワーク圧縮フレームワーク、NNCF)

NNCF は、Big Transfer (BiT) モデルなどのディープラーニング・モデルを最適化し、エッジからデータセンターまで、さまざまなハードウェアでパフォーマンスの向上を実現するための強力なツールです。モデル最適化のための包括的な機能のセットを提供し、開発者が低い精度の推論向けにモデルを容易に最適化できるようにします。主な機能を次に示します。

- 精度の低下を最小限に抑えつつ、さまざまなトレーニング後およびトレーニング時のアルゴリズムをサポートします。
- プルーニング、スパース性、量子化アルゴリズムのシームレスな組み合わせ。
- さまざまなモデルのサポート: NNCF を使用して、TensorFlow*、PyTorch*、ONNX*、OpenVINO™ などのさまざまなフレームワークからモデルを最適化できます。

異なるユースケースやモデルの圧縮アルゴリズムの使用法を示す NNCF のサンプルは、こちら (英語) を参照してください。NNCF 圧縮のサンプルの結果は、Model Zoo ページ (英語) を参照してください。

4. OpenVINO™ を使用した BiT 分類モデルの最適化

注: 以下のステップに進む前に、conda 環境が設定されていることを確認してください。conda 環境の設定の詳しい手順は、このシリーズのパート 1 (英語) を参照してください。

4.1. BiT_M_R50x1_1 tf 分類モデルのダウンロード

```
wget https://tfhub.dev/google/bit/m-r50x1/1?tf-hub-format=compressed
-O bit_m_r50x1_1.tar.gz

mkdir -p bit_m_r50x1_1 && tar -xvf bit_m_r50x1_1.tar.gz -C bit_m_r50x1_1
```

4.2. OpenVINO™ モデルの最適化

conda 環境内で次のコマンドを実行して、`bit_m_r50x1_1` モデルの OpenVINO™ IR モデルファイル (.xml および .bin) を生成します。これらのモデルファイルは、後のセクションでさらなる最適化と推論精度の検証に使用します。

```
ovc ./bit_m_r50x1_1 --output_model ./bit_m_r50x1_1/ov/fp32/bit_m_r50x1_1
--compress_to_fp16 False
```

5. データの準備

BiT モデルの量子化の精度に対する影響を評価するには、適切なデータセットが必要です。ここでは、1,000 クラスで 50,000 枚のイメージを含む ImageNet 2012 検証セット (英語) を利用しました。精度測定中のモデル予測の相互参照には、ILSVRC2012 検証グラウンドトゥルース (英語) を使用しました。

ImageNet 検証データなどの確立されたデータで圧縮モデルをテストすることにより、最適化の実際の有用性を正しく理解できます。リソース使用率を最小限に抑えつつ最大の精度を維持することは、エッジでのデプロイで非常に重要です。このデータセットにより、これらのトレードオフを効果的に検証するための厳密で公平な手段が提供されます。

注: ImageNet データセットのアクセスおよびダウンロードには、登録 (英語) が必要です。

6. NNCF を使用した量子化

このセクションでは、NNCF を使用して BiT モデルを量子化する際の具体的な手順を説明します。量子化プロセスには、キャリブレーション・データセットの準備、モデルへの 8 ビット量子化の適用、精度の評価が含まれます。

6.1. キャリブレーション・データセットの準備

このステップでは、キャリブレーション・データセットを表す `nncf.Dataset` クラスのインスタンスを作成します。`nncf.Dataset` クラスは、モデルのトレーニングまたは検証に使用するフレームワーク・データセット・オブジェクトのラッパーにすることができます。変換されたデータサンプルを使用した `nncf.Dataset()` 呼び出しのサンプルコードを次に示します。

```
# TF Dataset split for nncf calibration
img2012_val_split = get_val_data_split(tf_dataset_, \
                                       train_split=0.7, \
                                       val_split=0.3, \
                                       shuffle=True, \
                                       shuffle_size=50000)

img2012_val_split = img2012_val_split.map(nncf_transform).batch(BATCH_SIZE)

calibration_dataset = nncf.Dataset(img2012_val_split)
```

変換関数は、データセットからサンプルを取得し、推論のためにモデルに渡すデータを返す関数です。データ変換のサンプルコードを次に示します。

```
# Data transform function for NNCF calibration
def nncf_transform(image, label):
  image = tf.io.decode_jpeg(tf.io.read_file(image), channels=3)
  image = tf.image.resize(image, IMG_SIZE)
  return image
```

6.2. NNCF 量子化 (FP32 から INT8)

キャリブレーション・データセットの準備が整い、モデル・オブジェクトがインスタンス化されたら、モデルに 8 ビット量子化を適用します。この処理には、`nncf.quantize()` API を使用します。この API は、前のステップで生成された OpenVINO™ FP32 モデルとキャリブレーションされたデータセット値を受け取り、量子化プロセスを開始します。`nncf.quantize()` は多数の高度な設定を提供します。この例のように、多くの場合はそのまま、またはわずかな調整を行うだけで使用できます。`nncf.quantize()` API 呼び出しのサンプルコードを次に示します。

```
ov_quantized_model = nncf.quantize(ov_model, \
                                   calibration_dataset, \
                                   fast_bias_correction=False)
```

詳細は、公式ドキュメント (英語) で提供されている、環境の設定、キャリブレーション・データセットの準備、モデルの量子化など、基本的な量子化フローに関する包括的なガイドを参照してください。

6.3. 精度の評価

NNCF モデルの量子化プロセスにより、OpenVINO™ INT8 量子化モデルが生成されます。モデルの精度に対する量子化の影響を評価するため、オリジナルの FP32 モデルと量子化された INT8 モデルの間で包括的なベンチマークを実行して比較します。この比較には、ImageNet 2012 検証データセットでの BiT モデル (m-r50x1/1) の精度の測定が含まれます。精度の評価の結果を表 1 に示します。

フレームワーク (精度)	Top-1 精度
TensorFlow* (FP32)	0.70154
OpenVINO™ (FP32)	0.70154
OpenVINO™ (INT8)	0.69998

表 1: ImageNet 2012 検証データセットでの BiT_m_r50x1_1 モデルの分類精度。

TensorFlow* (FP32) から OpenVINO™ (FP32) モデルへの最適化では、分類精度は 0.70154 のままであり、OpenVINO™ モデル表現への変換は精度に影響を与えていません。8 ビット整数モデルへの NNCF 量子化でも、精度への影響は 0.03% 未満とわずかであり、量子化プロセスによるモデルの分類能力の低下はほとんどありません。

データの準備、モデルの最適化、NNCF 量子化タスク、精度の評価を含む、Python* スクリプト `bit_ov_model_quantization.py` は、付録 A を参照してください。

`bit_ov_model_quantization.py` スクリプトの使用法は次のとおりです。

```
$python bit_ov_model_quantization.py --help
usage: bit_ov_model_quantization.py [-h] [--inp_shape INP_SHAPE] --
dataset_dir DATASET_DIR --gt_labels GT_LABELS --bit_m_tf BIT_M_TF --
bit_ov_fp32 BIT_OV_FP32
                                    [--bit_ov_int8 BIT_OV_INT8]

BiT Classification model quantization and accuracy measurement

required arguments:
  --dataset_dir DATASET_DIR
                        Directory path to ImageNet2012 validation dataset
  --gt_labels GT_LABELS
                        Path to ImageNet2012 validation ds gt labels file
  --bit_m_tf BIT_M_TF   Path to BiT TF fp32 model file
  --bit_ov_fp32 BIT_OV_FP32
                        Path to BiT OpenVINO fp32 model file

optional arguments:
  -h, --help            show this help message and exit
  --inp_shape INP_SHAPE
                        N,W,H,C
  --bit_ov_int8 BIT_OV_INT8
                        Path to save BiT OpenVINO INT8 model file
```

7. まとめ

OpenVINO™ と NNCF を使用すると、計算要件を最小限に抑えつつ、モデルの効率を最適化できることが分かりました。特に、モデルを INT8 精度に圧縮してもパフォーマンスと精度を保持できることは、リソースに制約のある環境を含むさまざまな環境でのデプロイに OpenVINO™ を活用することの実用性を示すものです。NNCF は、分類の精度を大幅に低下させることなくモデルのサイズと計算効率のバランスを取ろうとする専門家にとって優れたツールであることが証明されており、さまざまなハードウェア構成でモデルのデプロイを強化する道を切り開く助けとなるでしょう。

関連情報 (英語)

1. OpenVINO™ 入門
2. OpenVINO™ モデルの最適化
3. OpenVINO™ ノートブック
4. NNCF
5. パート 1: OpenVINO™ ツールキットを使用して Big Transfer (BiT) モデルの推論を高速化

付録 A

ソフトウェア構成:

テスト日	11/22/2023	11/22/2023
フレームワーク/ツールキット	TensorFlow* 2.12.1	OpenVINO™ 2023.1.0
トポロジー/ML アルゴリズム	Big Transfer (BiT) https://github.com/google-research/big_transfer https://tfhub.dev/google/bit/m-r50x1/1	
入力形状	[1,224,224,3]	[1,224,224,3]
精度	FP32	FP32 および INT8

ILSVRC2012 グラウンドトゥルース: ground_truth_ilsvrc2012_val.txt (英語)

この記事で説明した、NNCF を使用した BiT モデルの量子化パイプラインについては、以下の `bit_ov_model_quantization.py` を参照してください。

```
"""
 Copyright (c) 2022 Intel Corporation
 Licensed under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at
      http://www.apache.org/licenses/LICENSE-2.0
 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
"""

"""
This script is tested with TensorFlow v2.12.1 and OpenVINO v2023.1.0
Usage Example below (with required parameters):
python bit_ov_model_quantization.py
--gt_labels ./<path_to>/ground_truth_ilsvrc2012_val.txt
--dataset_dir ./<path-to-dataset>/ilsvrc2012_val_ds/
--bit_m_tf ./<path-to-tf>/model
--bit_ov_fp32 ./<path-to-ov>/fp32_ir_model
"""

import os, sys
from openvino.runtime import Core
import numpy as np
import argparse, os
import nncf
import openvino.runtime as ov
import pandas as pd
import re

import logging
logging.basicConfig(level=logging.ERROR)

import tensorflow as tf
```

```
import tensorflow_hub as hub
import tensorflow.compat.v2 as tf

from PIL import Image
from sklearn.metrics import accuracy_score

ie = Core()
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# For top 1 labels.
MAX_PREDS = 1
BATCH_SIZE = 1
IMG_SIZE = (224, 224)  # Default Imagenet image size
NUM_CLASSES = 1000  # For Imagenette dataset

# Data transform function for NNCF calibration
def nncf_transform(image, label):
  image = tf.io.decode_jpeg(tf.io.read_file(image), channels=3)
  image = tf.image.resize(image, IMG_SIZE)
  return image

# Data transform function for imagenet ds validation
def val_transform(image_path, label):
  image = tf.io.decode_jpeg(tf.io.read_file(image_path), channels=3)
  image = tf.image.resize(image, IMG_SIZE)
  img_reshaped = tf.reshape(image, [IMG_SIZE[0], IMG_SIZE[1], 3])
  image = tf.image.convert_image_dtype(img_reshaped, tf.float32)
  return image, label

# Validation dataset split
def get_val_data_split(tf_dataset_, train_split=0.7, val_split=0.3, \
                       shuffle=True, shuffle_size=50000):
    if shuffle:
        ds = tf_dataset_.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * shuffle_size)
    val_size = int(val_split * shuffle_size)
    val_ds = ds.skip(train_size).take(val_size)

    return val_ds

# OpenVINO IR model inference validation
def ov_infer_validate(model: ov.Model,
            val_loader: tf.data.Dataset) -> tf.Tensor:

    model.reshape([1,IMG_SIZE[0],IMG_SIZE[1],3]) # If MO ran with Dynamic
batching
    compiled_model = ov.compile_model(model)
    output = compiled_model.outputs[0]

    ov_predictions = []
    for img, label in val_loader:#.take(25000):#.take(5000):#.take(5):
        pred = compiled_model(img)[output]
        ov_result = tf.reshape(pred, [-1])
        top_label_idx = np.argsort(ov_result)[-MAX_PREDS::][::-1]
        ov_predictions.append(top_label_idx)
```

```
    return ov_predictions


# OpenVINO IR model NNCF Quantization
def quantize(ov_model, calibration_dataset): #, val_loader: tf.data.Dataset):
    print("Started NNCF qunatization process")
    ov_quantized_model = nncf.quantize(ov_model, calibration_dataset,
fast_bias_correction=False)
    return ov_quantized_model

# OpenVINO FP32 IR model inference
def ov_fp32_predictions(ov_fp32_model, validation_dataset):
    # Load and compile the OV model
    ov_model = ie.read_model(ov_fp32_model)
    print("Starting OV FP32 Model Inference...!!!")
    ov_fp32_pred = ov_infer_validate(ov_model, validation_dataset)
    return ov_fp32_pred

def nncf_quantize_int8_pred_results(ov_fp32_model, calibration_dataset, \
                          validation_dataset, ov_int8_model):

    # Load and compile the OV model
    ov_model = ie.read_model(ov_fp32_model)

    # NNCF Quantization of OpenVINO IR model
    int8_ov_model = quantize(ov_model, calibration_dataset)
    ov.serialize(int8_ov_model, ov_int8_model)
    print("NNCF Quantization Process completed..!!!")

    ov_int8_model = ie.read_model(ov_int8_model)
    print("Starting OV INT8 Model Inference...!!!")
    ov_int8_pred = ov_infer_validate(ov_int8_model, validation_dataset)

    return ov_int8_pred

def tf_inference(tf_saved_model_path, val_loader: tf.data.Dataset):

    tf_model = tf.keras.models.load_model(tf_saved_model_path)
    print("Starting TF FP32 Model Inference...!!!")
    tf_predictions = []
    for img, label in val_loader:
        tf_result = tf_model.predict(img, verbose=0)
        tf_result = tf.reshape(tf_result, [-1])
        top5_label_idx = np.argsort(tf_result)[-MAX_PREDS::][::-1]
        tf_predictions.append(top5_label_idx)

    return tf_predictions


"""
Module: bit_classificaiton
Description: API to run BiT classificaiton OpenVINO IR model INT8
Quantization on using NNCF and
perfom accuracy metrics for TF FP32, OV FP32 and OV INT8 on  ImageNet2012
Validation dataset
"""
def bit_classification(args):
```

```
    ip_shape = args.inp_shape
    if isinstance(ip_shape, str):
        ip_shape = [int(i) for i in ip_shape.split(",")]
        if len(ip_shape) != 4:
            sys.exit( "Input shape error. Set shape 'N,W,H,C'. For example:
'1,224,224,3' " )

    # Imagenet2012 validataion dataset used for TF and OV FP32 accuracy
testing.
    #dataset_dir = ../dataset/ilsvrc2012_val/1.0/ + "*.JPEG"
    dataset_dir = args.dataset_dir + "*.JPEG"
    tf_dataset = tf.data.Dataset.list_files(dataset_dir)

    gt_lables = open(args.gt_labels)

    val_labels = []
    for l in gt_lables:
        val_labels.append(str(l))

    # Generating ImageNet 2012 validation dataset dictionary (img, label)
    val_images = []
    val_labels_in_img_order = []
    for i, v in enumerate(tf_dataset):
        img_path =  str(v.numpy())
        id = int(img_path.split('/')[-1].split('_')[-1].split('.')[0])
        val_images.append(img_path[2:-1])
        val_labels_in_img_order.append(int(re.sub(r'\n','',val_labels[id-
1])))

    val_df = pd.DataFrame(data={'images': val_images, 'label':
val_labels_in_img_order})

    # Converting imagenet2012 val dictionary into tf.data.Dataset
    tf_dataset_ =
tf.data.Dataset.from_tensor_slices((list(val_df['images'].values),
val_df['label'].values))
    imgnet2012_val_dataset = tf_dataset_.map(val_transform).batch(BATCH_SIZE)

    # TF Dataset split for nncf calibration
    img2012_val_split_for_calib = get_val_data_split(tf_dataset_,
train_split=0.7, \
                                                     val_split=0.3,
shuffle=True,  \
                                                     shuffle_size=50000)

    img2012_val_split_for_calib =
img2012_val_split_for_calib.map(nncf_transform).batch(BATCH_SIZE)

    # TF Model Inference
    tf_model_path = args.bit_m_tf
    print(f"Tensorflow FP32 Model {args.bit_m_tf}")
    tf_p = tf_inference(tf_model_path, imgnet2012_val_dataset)

    #acc_score = accuracy_score(tf_pred, val_labels_in_img_order[0:25000])
    acc_score = accuracy_score(tf_p, val_labels_in_img_order)
    print(f"Accuracy of FP32 TF model = {acc_score}\n")

    # OpenVINO Model Inference
```

```
    print(f"OpenVINO FP32 IR Model {args.bit_ov_fp32}")
    ov_fp32_p = ov_fp32_predictions(args.bit_ov_fp32, imgnet2012_val_dataset)

    acc_score = accuracy_score(ov_fp32_p, val_labels_in_img_order)
    print(f"Accuracy of FP32 IR model = {acc_score}\n")

    print("Starting NNCF dataset Calibration....!!!")
    calibration_dataset = nncf.Dataset(img2012_val_split_for_calib)

    # OpenVINO IR FP32 to INT8 Model Quantization with NNCF and
    # INT8 predictions results on validation dataset
    ov_int8_p = nncf_quantize_int8_pred_results(args.bit_ov_fp32,
calibration_dataset, \
                                        imgnet2012_val_dataset,
args.bit_ov_int8)

    print(f"OpenVINO NNCF Quantized INT8 IR Model {args.bit_ov_int8}")
    acc_score = accuracy_score(ov_int8_p, val_labels_in_img_order)
    print(f"Accuracy of INT8 IR model = {acc_score}\n")

    #acc_score = accuracy_score(tf_p, ov_fp32_p)
    #print(f"TF Vs OV FP32 Accuracy Score = {acc_score}")

    #acc_score = accuracy_score(ov_fp32_p, ov_int8_p)
    #print(f"OV FP32 Vs OV INT8 Accuracy Score = {acc_score}")

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description="BiT Classification model
quantization and accuracy measurement")
    optional = parser._action_groups.pop()
    required=parser.add_argument_group("required arguments")
    optional.add_argument("--inp_shape", type=str, help="N,W,H,C",
default="1,224,224,3", required=False)
    required.add_argument("--dataset_dir", type=str, help="Directory path to
ImageNet2012 validation dataset", required=True)
    required.add_argument("--gt_labels", type=str, help="Path to ImageNet2012
validation ds gt labels file", required=True)
    required.add_argument("--bit_m_tf", type=str, help="Path to BiT TF fp32
model file", required=True)
    required.add_argument("--bit_ov_fp32", type=str, help="Path to BiT
OpenVINO fp32 model file", required=True)
    optional.add_argument("--bit_ov_int8", type=str, help="Path to save BiT
OpenVINO INT8 model file",
                          default="./bit_m_r50x1_1/ov/int8/saved_model.xml",
required=False)
    parser._action_groups.append(optional)

    args = parser.parse_args()
    bit_classification(args)
```

OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピューター・ビジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン / ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html