開発者の演習 | 何でもセグメント化して定量的に高速化

この記事は、Medium に公開されている「Developers' Hands-on | Segment Anything Quantitative Acceleration」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

「Segment Anything, We All Get Unemployed! (何でもセグメント化して、みんなで失業しよう!)」 — 最近、こんな言葉がソーシャルメディアで話題になりました。これは、何でもセグメント化モデル (Segment Anything Model、略称 SAM) のことを指したものです。SAM とはそもそも何なのでしょうか? どのような機能を備えているのでしょうか? 本当に強力なのでしょうか? この記事で調べてみましょう!

SAM は、Meta AI により開発された、強力な人工知能画像セグメンテーション・アプリケーションです。画像内のどのピクセルがオブジェクトに属しているか自動的に識別して画像内のさまざまなオブジェクトに自動文体処理を実行するもので、科学画像の分析、写真の編集などに幅広く使用できます。

SAM のアプリケーションは、画像エンコーダー・モデルとマスクデコーダー + プロンプト・エンコーダー・モデルで構成され、どちらも個別の静的モデルとして解釈できます。推論中は画像エンコーダーが主な計算ワークロードを処理します。つまり、画像エンコーダーの実行効率を向上させることが、SAM アプリケーションの主な最適化目標の 1 つになります。

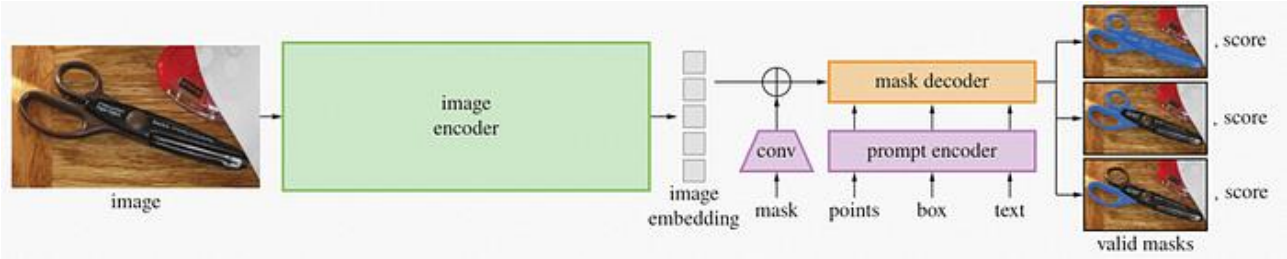

図 1: SAM モデルのタスク・パイプライン

この記事では、OpenVINO™ NNCF モデル圧縮ツールを使用して SAM エンコーダーの量子化を圧縮し、CPU での推論のパフォーマンスを向上させる方法を説明します。

2. 量子化の概要

実装を行う前に、量子化の概念について説明しておきましょう。量子化とは、モデルの構造を変更することなく、モデルのパラメーターの表現範囲を FP32 から INT8 または INT4 にマップすることを指します。同じ情報をより小さな値のビット幅で表現し、モデルのサイズを圧縮して、メモリーの消費量を減らします。モデル・ネットワークの実行プロセス中に、システムは、小さなビット数のデータ向けに最適化された特殊なハードウェア・プラットフォーム命令またはカーネル関数を自動的に呼び出してパフォーマンスを向上させます。

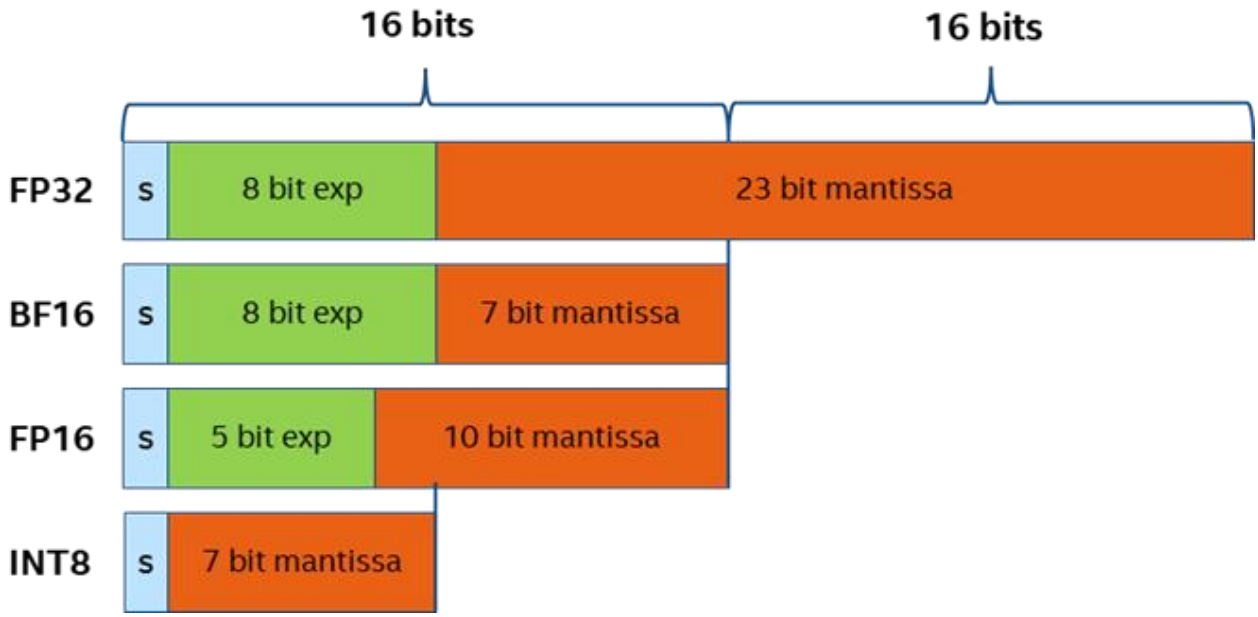

図 2: 異なる精度データの表現のビット幅

インテル® AVX-512 VNNI 拡張命令は、3 クロックサイクルの INT8 行列乗算と加算演算を 1 クロックサイクルに圧縮します。最新のインテル® AMX 命令セットでは、複数の VNNI モジュールがスタックされ、1 サイクル内で数倍のパフォーマンス向上を実現します。

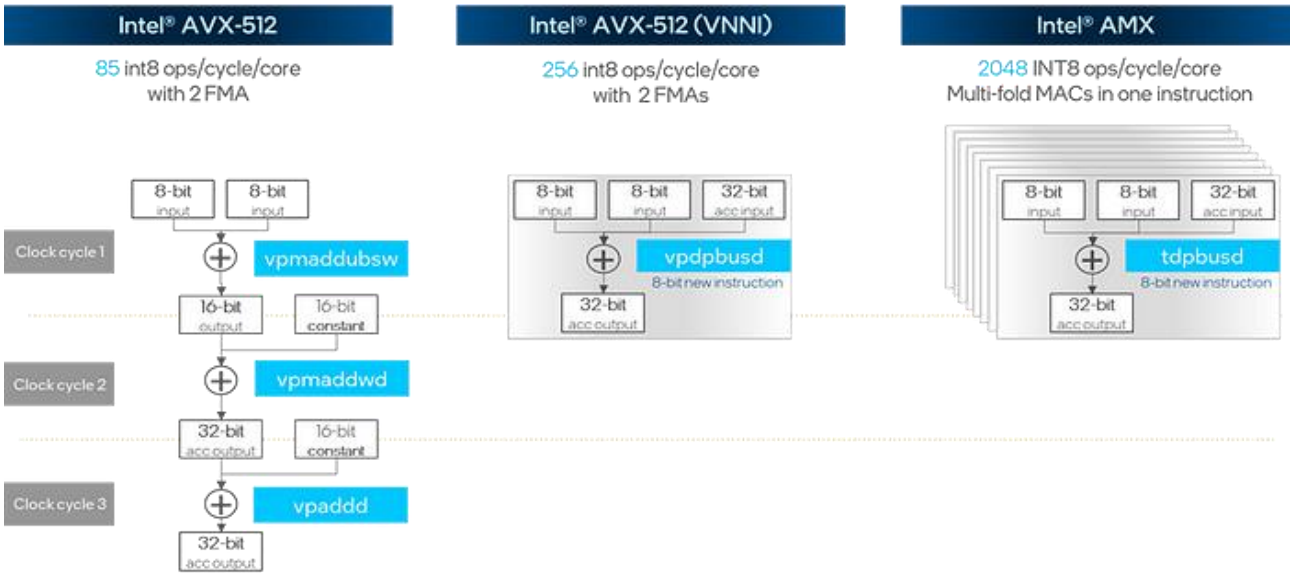

図 3: INT8 行列乗算と加算演算の命令セットの最適化

3. NNCF のトレーニング後の量子化モード

NNCF (Neural Network Compression Framework の略) は、モデルの圧縮と高速化のために特別に設計された OpenVINO™ ツールキット内のソリューション実装です。量子化、プルーニング、2 値化などのさまざまなモデル圧縮アルゴリズムが含まれています。NNCF の使用法は、トレーニング後の量子化 (PTQ) と量子化を考慮したトレーニング (QAT) の 2 つのモードに分類できます。QAT ではオリジナルのトレーニング・スクリプトとデータセットが必要ですが、PTQ では追加のトレーニング・スクリプトやラベル付きデータセットなしで、トレーニング済みモデルファイルを直接圧縮できます。これは、OpenVINO™ 2023.0 リリースの NNCF で導入された新機能です。PTQ は次の 2 つのステップで達成できます。

1. キャリブレーション・データセットを準備します。量子化プロセスでは、キャリブレーション・データはデータ範囲と分布の計算にのみ使用され、追加のラベル付きデータは必要ありません。例えば、画像認識タスクでは、約 200～300 の画像ファイルを使用します。さらに、DataLoader オブジェクトと transform_fn データ変換関数を定義する必要があります。DataLoader はキャリブレーション・データセットの各要素の読み取りを行い、transform_fn は読み取った要素を OpenVINO™ モデル推論向けの直接入力データに変換します。

```
import nncf
calibration_loader = torch.utils.data.DataLoader(...)
def transform_fn
(data_item):
    images, _ = data_item
    return images
calibration_dataset = nncf.Dataset(calibration_loader, transform_fn)
```

2. モデルの量子化を実行します。まず、モデル・オブジェクトをインポートし、nncf.quantize() インターフェイスを使用してモデル・オブジェクトをキャリブレーション・データセットにバインドして、量子化タスクを開始します。NNCF は、openvino.runtime.Model、torch.nn.Module、onnx.ModelProto、tensorflow.Module を含む、さまざまなモデル・オブジェクト型をサポートします。

```
model = ... #OpenVINO/ONNX/PyTorch/TF object
quantized_model = nncf.quantize(model, calibration_dataset)
```

3. (オプション) 精度制御モード。デフォルトのモードで NNCF でエクスポートしたモデルの精度が予想よりも低下した場合は、トレーニング後の量子化に精度制御モードを使用します。この場合、量子化プロセス中のモデル精度の損失に対する各層の影響の感度を評価するには、ラベル付きテスト・データセットが必要です。その後、モデルが目的の精度に達するまで、評価に基づいて層の精度が徐々に元の精度に戻されます。このモードでは、モデルの精度を維持しながらモデルのサイズを圧縮できます。具体的な方法は、こちら (英語) を参照してください。

4. OpenVINO™ Python* API を使用して YOLOv8 物体検出モデル推論プログラムを作成する

次に、NNCF の PTQ モードを使用して SAM エンコーダーの量子化を完了する方法を見てみましょう。

プロジェクトはここ (英語) から入手できます。

1. データローダーの定義

この例では、128 個の .jpg 形式の画像を含む、coco128 データセットをキャリブレーション・データセットとして使用しています。ONNX または IR 静的モデルを量子化する場合、データローダーは torch DataLoader クラスである必要があるため、torch.utils.data.Dataset を継承して、データセットの各オブジェクトを繰り返し処理するために getitem メソッドを含むデータセット・クラスを再構築し、len メソッドを使用してデータセットのオブジェクトの数を取得します。最後に、torch.utils.data.DataLoader メソッドを使用して DataLoader を生成します。

```
class COCOLoader(data.Dataset):
    def __init__(self, images_path):
        self.images = list(Path(images_path).iterdir())

    def __getitem__(self, index):
        image_path = self.images[index]
        image = cv2.imread(str(image_path))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        return image

    def __len__(self):
        return len(self.images)
coco_dataset = COCOLoader(OUT_DIR / 'coco128/images/train2017')
calibration_loader = torch.utils.data.DataLoader(coco_dataset)
```

2. データ形式変換モジュールの定義

次のステップは、データ形式変換モジュールを定義することです。以前定義した preprocess_image 関数を使用してデータを前処理します。calibration_loader モジュールは単一のデータ・オブジェクトを torch tensor 形式で返しますが、OpenVINO™ Python* インターフェイスはこのデータ型をサポートしていないため、最初に NumPy* 形式に変換する必要があることに注意してください。

```
def transform_fn(image_data):
    image = image_data.numpy()
    processed_image = preprocess_image(np.squeeze(image))
    return processed_image

calibration_dataset = nncf.Dataset(calibration_loader, transform_fn)
```

3. NNCF 量子化の実行

量子化モデルの精度を確保するため、FP16 IR 形式モデルの代わりに、オリジナルの FP32 ONNX 形式モデルを入力オブジェクトとして使用します。次に、量子化のため、モデルを nncf.quantize インターフェイスに渡します。このインターフェイスには、いくつかの重要な追加パラメーターがあります。

● model_type: 特別な量子化手法を有効にするために使用するモデルタイプ。例えば、トランスフォーマー・モデルでは、モデルの精度を優先する必要があります。

● preset: 量子化モード。デフォルトのモードは PERFORMANCE で、畳み込みの重みとバイアスの両方に対称量子化を使用してモデルのパフォーマンスを向上させます。ここでは、モデルの精度とパフォーマンスのバランスを取るために MIXED モードを使用します。重みには対称量子化を使用し、バイアスには非対称量子化を使用します。非 ReLU または非対称活性化層を含むモデルに適しています。

```
# Load FP32 ONNX model
model = core.read_model(onnx_encoder_path)
quantized_model = nncf.quantize(model,
                                calibration_dataset,

model_type=nncf.parameters.ModelType.TRANSFORMER,

preset=nncf.common.quantization.structs.QuantizationPreset.MIXED)
ov_encoder_path_int8 = "sam_image_encoder_int8.xml"
serialize(quantized_model, ov_encoder_path_int8)
```

SAM エンコーダー・モデルのネットワーク構造は複雑で、量子化プロセスでは各層のパラメーターを複数回トラバースする必要があるため、量子化プロセスに時間がかかる場合があります。32GB 以上のメモリーを搭載したハードウェア・デバイスを使用することを推奨します。メモリーが少ない場合は、subset_size パラメーターを 100 に設定してキャリブレーション・データの数を減らします。

4. モデルの精度の比較

次に、INT8 モデルと FP16 モデルの推論結果を比較します。

図 4: プロンプトモードの FP16 と INT8 の結果の比較

図 5: 自動モードの FP16 と INT8 の結果の比較

プロンプトモードと自動モードの両方で、INT8 モデルは FP16 モデルと比較して精度がほとんど変化していないことがわかります。

注: 自動モードでは、マスクはランダムに生成された色で表示されます。

5. パフォーマンスの比較

最後に、OpenVINO™ で提供されている benchmark_app ツールを使用してパフォーマンスの指標を比較します。

```
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            60 iterations
[ INFO ] Duration:         75716.93 ms
[ INFO ] Latency:
[ INFO ]    Median:        14832.33 ms
[ INFO ]    Average:       14780.77 ms
[ INFO ]    Min:           10398.47 ms
[ INFO ]    Max:           16725.65 ms
[ INFO ] Throughput:   0.79 FPS
```

図 6: ベンチマークの結果 (FP16)

```
[ INFO ] Execution Devices:['CPU']
[ INFO ] Count:            72 iterations
[ INFO ] Duration:         68936.14 ms
[ INFO ] Latency:
[ INFO ]    Median:        11281.87 ms
[ INFO ]    Average:       11162.87 ms
[ INFO ]    Min:           6736.09 ms
[ INFO ]    Max:           12547.48 ms
[ INFO ] Throughput:   1.04 FPS
```

図 7: ベンチマークの結果 (INT8)

CPU では、INT8 モデルは FP16 モデルと比較してパフォーマンスが約 30% 向上し、モデルサイズが約 350MB から 100MB 未満に減っています。

5. まとめ

SAM の優れた自動セグメンテーション機能を考慮すると、このテクノロジーが導入されるアプリケーションのシナリオはますます増えることが予想されます。工業化プロセス中、開発者は多くの場合、コスト効率に優れたソリューションを得るため、パフォーマンスと精度のバランスを取ることに注目します。OpenVINO™ NNCF ツールは、モデルの精度に大きな影響を与えることなく、何でもセグメント化エンコーダーの一部を量子化および圧縮することにより、モデルの実行時間効率を大幅に向上させて、モデルが占有する空間を削らします。

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