WP521 (v1.0.1) 2020 年 6 月 24 日

ザイリンクス デバイスで INT4 最適化を使用した、たたみ込みニューラル ネットワーク

INT8 は、AI 推論において浮動小数点と比較しても互角の精度で優れたパフォーマンスを提供します。ただし、リソースが限られ、INT8 では目標とするパフォーマンスを達成できない場合は、INT4 の最適化が答えとなります。INT4 最適化を使用すると、現行の INT8 ソリューションと比べて、実際のハードウェア上でパフォーマンスを最大 77% 高めることができます。

概要

ザイリンクスは、ザイリンクス ハードウェア プラットフォーム上で動作する INT8 AI 推論アクセラレータ、深層学習プロセッサ ユニット (XDPU) を提供しています。ただし、リソースが限られた高性能かつ低レイテンシのシナリオ (リソースの消費電力の影響を受けやすいエッジ側や低レイテンシ ADAS など) では、INT8 よりも少ない消費電力で高い性能を実現するために、ニューラル ネットワークの低ビット量子化が必要になります。一方で、極端な低ビット量子化 (バイナリやターナリなど) では、精度が低下してしまいます。

そこで、4 ビットの活性化と 4 ビットの重み (4A4W) からなるフルプロセスのハードウェア処理しやすい量子化ソリューションを使用すると、精度とリソースのバランスを取ることができます。このホワイト ペーパーでは、Zynq® UltraScale+™ MPSoC および Zynq-7000 SoC ファミリ (16nm、28nm) に CNN 4 ビット XDPU の低精度アクセラレータを実装して、たたみ込み演算の効率的なマッピングによって DSP 機能を存分に活用する方法について説明します。このソリューションでは、ソリューション レベルの性能が XDPU の 2 倍になります。ADAS システムの 2D 検出タスクでは、この実装により Zynq UltraScale+ MPSoC ZCU102 ボードで 230fps の推論速度を達成しており、その際の性能向上は 8 ビット XDPU の 1.52 倍になっています。さらに、このソリューションによって、ADAS システムの各種タスクで全精度モデルと互角の結果を出すことができます。

はじめに

データセンター、オートモーティブ、産業、医療などの分野では、AI ベースのシステムの製品化に注力する企業が増えています。これには主に次の 2 つの課題があります。

- AI 推論を実現するには、価格、消費電力、レイテンシ、フォーム ファクターはそのままで、桁違いに多くの演算が必要です。
- AI サイエンティストは、さまざまなハードウェア アーキテクチャの最適化が必要となるアルゴリズムやモデルの革新を日々続けています。

絶え間ない技術革新に対する高い需要に応えるには、適応型の特定分野向けアーキテクチャ (DSA) が必要です。AI 推論のパフォーマンス最適化と消費電力削減における主なトレンドの 1 つは、低精度や混合精度を使用することです。ハードウェア デザインの複雑性を減らすため、さまざまなハードウェア プラットフォームにはモデル量子化が主要技術として応用されています。CNN の演算コストとストレージ コストを最小限に抑えるため、多くの取り組みがなされてきました。この取り組みにより、コンピューター ビジョンの大半のタスクでは、精度を大きく低下させることなく、INT8 によって重みと活性化を表現できることが広く実証されています。ただし、エッジ アプリケーションによっては、ハードウェア リソースが依然として不十分な場合があります。比較的小さいビット幅 (1 ビット、2 ビットなど) をエッジ アプリケーションに使用する場合、一般的なハードウェア デザイン ソリューションでは簡素化された乗算器を使用することがあります。このようなソリューションでは低レイテンシと高スループットが得られますが、それでも全精度モデルでは確度に大きなギャップが生じます。そのため、モデルの精度とハードウェア パフォーマンスのバランスを取ることが必要です。

ザイリンクスでは、ImageNet 分類 [参照 1] タスクに各種の量子化メソッドを使用した実験を、いくつかの一般的なネットワーク構造 (ResNet50V1 [参照 2]、ResNet50V2 [参照 3]、MobilenetV1 [参照 4]、MobilenetV2 [参照 5]) で実施しました。その結果、ビット幅を減らすと精度が低下することがわかりました。特に、ビット幅が 4 未満の場合は、精度が大幅に低下します。ザイリンクスでは、Williams 氏などが紹介しているルーフライン モデル [参照 6] も使用して、異なるビット幅でハードウェア パフォーマンスを分析しました (図 1 参照)。ザイリンクス ZCU102 評価ボードをサンプルとして使用したところ、MAC の精度を下げると、ハードウェア コストが低下し、パフォーマンスが向上しました。この実験では、メモリ要件を低減すると、低ビットの量子化でパフォーマンスを向上させることができることもわかりました。このことは、ResNet-50 ニューラル ネットワークにおけるたたみ込みの演算強度の図 (8 ビット精度演算と 4 ビット精度演算の 2 つのケース) で示されています。したがって INT4 が、モデルの精度とハードウェア パフォーマンスのバランスが最も良いと言えます。

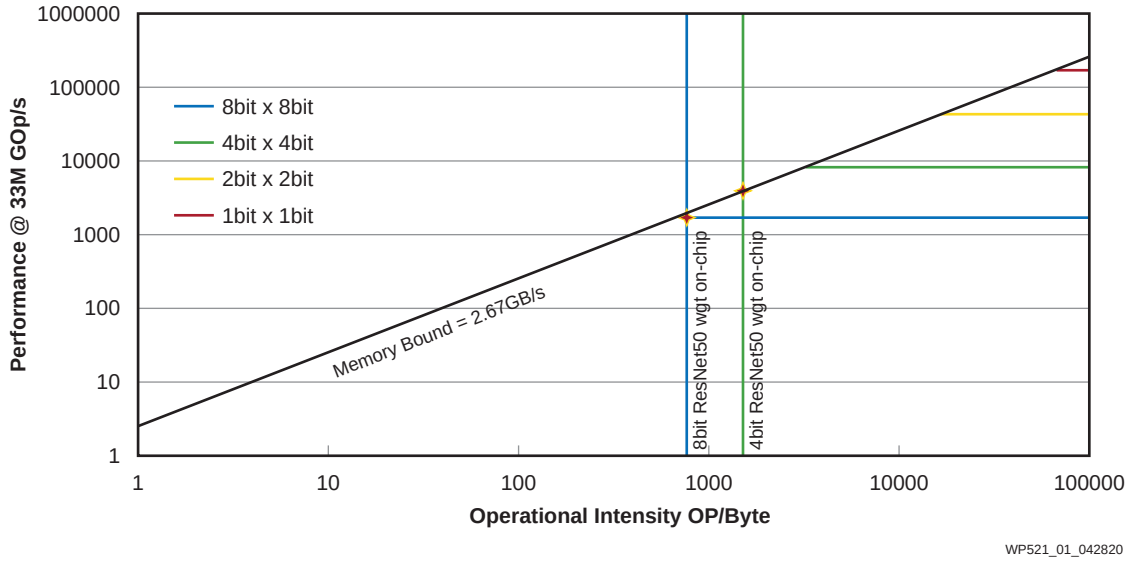

図 1: ビット幅別の ZCU102 のルーフライン モデル

フルプロセスのハードウェア処理しやすい CNN を量子化する方法

量子化プロセス全体をハードウェア処理しやすくするには、INT4 量子化メソッドを、量子化メカニズム、ハードウェア処理しやすい量子化デザイン、量子化認識トレーニングの 3 つのカテゴリに分けます。

量子化メカニズム

ザイリンクスでは、トレーニング済み量子化しきい値 (TQT) [参照 7] を使用して、DNN を単精度浮動小数点 (FP32) から INT4 に変換しました。重みと活性化については、量子化関数を形式的に次のように表すことができます。

$$q(x;s) := clip\left(\left\lfloor\frac{x}{s}\right\rfloor;n,p\right)\cdot s$$

式 1

where $n=-2^{b-1}, p=2^{b-1}-1$ and $s=\frac{2^{\lceil \log_2 t\rceil}}{2^{b-1}}$ for signed data; $n=0, p=2^{b}-1$ and $s=\frac{2^{\lceil \log_2 t\rceil}}{2^{b}}$ for unsigned data.

式 1 は、入力 x の量子化値が、しきい値 t、ビット幅 b、量子化スケール係数 s によって決まることを示しています。しきい値 t は通常、量子化されるテンソルの絶対値の最大値に初期設定されます。この値は、その後トレーニング中に $\log_2 t$ の形式で最適化されます。量子化スケール係数 s は、ハードウェア処理しやすい 2 のべき数です。クリップ演算は一部の外れデータを除外して、重みと活性化の緊密な分布を促すため、量子化に適しています。

前述したように、$\log_2 t$ はトレーニング中に学習可能なパラメーターであり、適切な量子化範囲を見つけるよう最適化されます。逆に、$\log_2 t$ の勾配はチェーン ルールで作成できます。入力 x の勾配は次のように計算することもできます。

$$\nabla_{(\log_2 t)} q(x;s) \coloneqq s \ln 2 \cdot \begin{cases} \left\lfloor \frac{x}{s} \right\rceil - \frac{x}{s} & if\ n \le \left\lfloor \frac{x}{s} \right\rceil \le p \\ n & if \left\lfloor \frac{x}{s} \right\rceil < n \\ p & if \left\lfloor \frac{x}{s} \right\rceil > p \end{cases}$$

式 2

$$\nabla_x q(x;s) \coloneqq \begin{cases} 1 & if\ n \le \left\lfloor \frac{x}{s} \right\rceil \le p \\ 0 & otherwise \end{cases}$$

式 3

[x] (round) と [x] (ceil) の場合、非微分可能関数 STE を使用して、式 4 で定義された勾配が得られます。

$$\frac{\partial}{\partial x} \lfloor x \rceil = \frac{\partial}{\partial x} \lceil x \rceil = 1$$

式 4

TQT は、ログ表記法によってしきい値と入力のスケール不変性を実現できることを証明しています。トレーニングしきい値は、ログ ドメインではさらに管理しやすく、非常に効率的であることが証明されています。

ハードウェア処理しやすい量子化デザイン

低ビット ネットワークは、量子化トレーニング用の全精度ネットワークから構築する必要があります。次に、フルプロセスのハードウェア処理しやすい量子化をベースにした一般的なネットワーク構造と、いくつかの粗粒型モジュールの量子化ソリューションをまとめたものを示します。これらの量子化モジュールによって、INT4 量子化メソッドをさまざまなネットワーク構造に使用できます。図 2 に、一般的なモジュールの量子化ソリューションを示します。図 2 に示す点線は、実際のネットワーク構造に従って追加または削除可能であることを意味します。

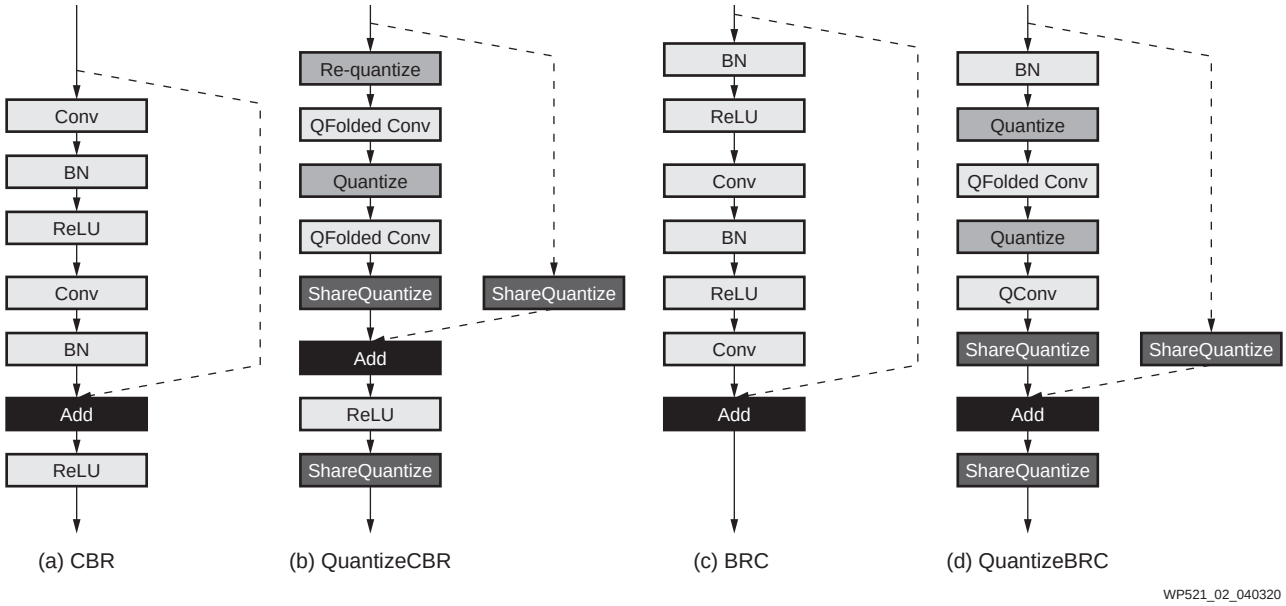

図 2: モジュールの量子化

モジュール 1: CBR(Conv+BN+ReLU)

CNN の一般的な構造として、トレーニングおよび推論中のフロップを減らすために BN レイヤーがマージされます。ただし、BN レイヤーには矛盾があります。バッチ処理では、トレーニング時に現行バッチの平均値と分散を使用し、推論時にその平均値と分散を動かします。現行バッチの平均値と分散から得られたマージ パラメーターが量子化されると、推論時に偏差が生じます。この不一致を解消するには、次のベスト プラクティス [参照 8]、[参照 9] を使用して、この構造を量子化する必要があります。BN を Conv にフォールドした後、INT4 にフォールドされたパラメーターが量子化されます。このモジュールの量子化を図 2 (b) に示します。

モジュール 2: BRC(BN+ReLU+Conv)

図 2 (c) に示すように、たたみ込みレイヤーに続いて BN レイヤーがマージされても、まだ独立した BN レイヤーが存在します。ただし、既存の INT4 量子化メソッドでは、BN レイヤーはほとんど注目されていません。この独立した BN レイヤーを効果的に展開するには、量子化ニューラル ネットワーク用に合理化された運用環境 [参照 10] を使用して、トレーニング時に全精度を維持し、推論時にしきい値までの浮動スケールとバイアスを吸収します。このメソッドを、精度を維持しながら、推論時のたたみ込みを含むすべての線形演算に拡張できます。このモジュールの量子化を図 2 (d) に示します。

モジュール 3: Add

Add モジュールでは、共有型の量子化レイヤーを使用して、すべての入力と出力を量子化します。経験則では、Add 演算は精度の影響を受けやすいものの、必要なハードウェア リソースは少なくて済みます。そのため、このレイヤーは、通常は 8 ビットに量子化されます。さらに、すべての入力と出力を量子化するため、スケール共有ポリシーを使用します。共有ポリシーを使用すると、ハードウェアがスケールの計算をバイパスでき、浮動小数点乗算が不要になります。図 2 (b) に示す「ShareQuantize」は、これらの量子化レイヤーが同じスケールを共有することを意味します。

その他:

Conv 演算の入力を必ず 4 ビットにするには、図 2 の「Re-quantize」に示すように、Add 演算の 8 ビットの出力を 4 ビットに再び量子化する必要があります。最初と最後のレイヤーでは、それでも INT4 量子化が実行されます。ネットワーク全体の出力は 8 ビットに量子化されます。内積レイヤーは、たたみ込みレイヤーと一致します。

量子化認識トレーニング

量子化認識トレーニングは一般に、低ビットと全精度の間に生じる精度のギャップを減らす主要技術として使用されます。このホワイトペーパーで説明する INT4 量子化メソッドでも、非常に重要な役割を担います。量子化認識トレーニング プロセスには、次に示すアルゴリズム 1 を使用します。

アルゴリズム 1: Layer-Wise の量子化認識トレーニング

入力:

全精度入力、重み、バイアス: X, W, Bias

入力と重みに対する学習可能なログ ドメインしきい値a_x, a_w, a_{bias}

ビット幅: 入力と重みは b=4、バイアスは b=8

出力:

出力Y

1.初期設定: $a_x = \log_2 \max(|x|)$, $a_w = \log_2 \max(|w|)$, $a_{bias} = \log_2 \max(|bias|)$

2.式 1 に従って q(x)、q(w)、q(bias) を計算

3.Y = Forward(q(x), q(w), q(bias))

4.分類損失 Loss を計算。学習可能なすべてのパラメーターに正則化法が使用される。

5. 式 3を参照

$$\nabla_X L = \frac{\partial L}{\partial q(x)} * \frac{\partial q(x)}{\partial X}, \frac{\partial q(x)}{\partial X}$$

6.Adam を使用して全精度パラメーターを更新

ザイリンクス DSP スライスでの INT4 最適化

累積乗算 (MAC) 演算は、DSP ハードウェア リソースを使用して実行できます。最適化により、DSP が 16nm または 28nm デバイスでできるだけ多くの MAC 演算を処理できるようになります。たとえば 16nm のザイリンクス プログラマブル デバイス、UltraScale™ アーキテクチャの DSP48E2 スライスは専用スライスです [参照 11]。DSP48E2 スライスは、27 x 18 の 2 の補数乗算器と 48 ビット アキュムレータで構成されます。図 3 に示すように、ザイリンクス DSP スライスで MAC を実行できます。

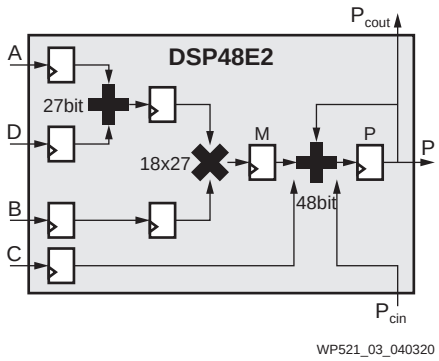

図 3: MAC モード内の DSP48E2 スライス

INT4 最適化

低精度の MAC 演算では、乗算は a*b で表されます (a は 4 ビットの符号なし活性化データ、b は 4 ビットの符号付き重みデータ)。図 4 に示すように、DSP48E2 スライスは 4 チャネルの乗算演算用に構成できます。

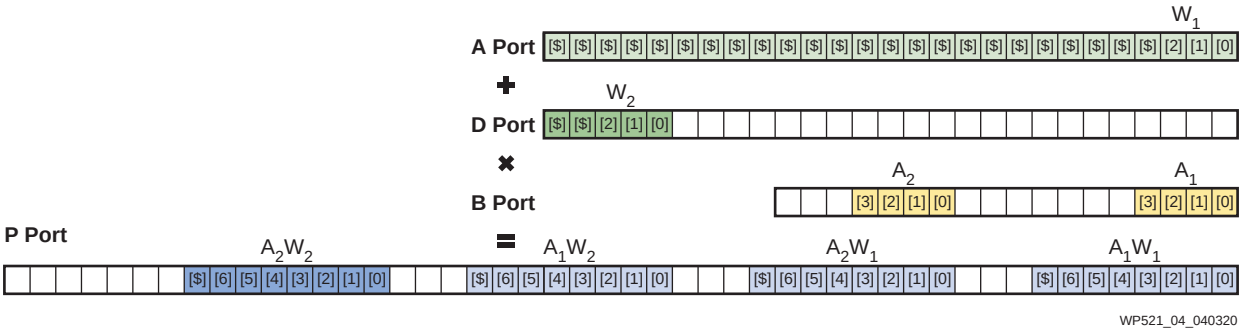

図 4: 4 チャネル パック用の DSP48E2 構成モード

DSP48E2 スライスのポート A は 27 ビット幅です。ポート B は 18 ビット幅です。int4 * uint4 の乗算により、少なくとも 8 ビット幅を要する結果が生成されます。複数の乗算をパッケージ化しても出力が正しいままであることが、DSP リソースを十分に活用する前提となります。これを実現するため、チャネルの間にガードが追加されます。MAC の 4 つのチャネルをパックする際には、2 つの入力の間に十分なガード ビットを置く必要があります。DSP48E2 スライスの設計に従って、ガード ビットは 3 ビットに設定されます。

$$(A_2 \cdot 2^{11} + A_1) \cdot (W_2 \cdot 2^{22} + W_1) = A_2W_2 \cdot 2^{33} + A_1W_2 \cdot 2^{22} + A_2W_1 \cdot 2^{11} + A_1W_1$$

式 5

最初のチャネル A_1*W_1 は、各ポートの 4 つの LSB に配置されます。次のチャネル A_2*W_1 は、正しく計算するために少なくとも 8 ビット シフトする必要があります。2 番目のチャネルは、重みデータ W_1 を最初のチャネルと共有します。A_2 はポート B で 11 ビット シフトされます。3 ビットのガードは、DSP リソースを最大限利用するためです。最後のエレメント W_2 はポート A に割り当てられます。残り 2 つのチャネルは A_1*W_2 と A_2*W_2 です。重みは符号付きデータです。乗算の前に、2 つの重みデータが 27 ビット前置加算器によってパックされます。W_1 には符号拡張が必要なため、W_2 を D ポートの 4 つの MSB には配置できません [参照 12]。W_2 が MSB にあると、$W_1 < 0$ および $W_2 = –8$ の場合に前置加算器がオーバーフローします。48 ビット後置加算器をアキュムレータとして使用すると、カスケード接続によって前のレベルの DSP 結果を加算できます。こうすると、1 つの DSP48E2 で、4 チャネルの MAC を 1 クロック サイクルで実行できます。

結果のビット幅は、累算後により大きくなります。ハードウェア処理しやすいクオンタイザーとは、一連のシフト レジスタであり、命令によってシフトされるビット数を制御できます。シフト演算は、ハードウェア処理しやすい演算です。低精度 CNN では、2 種類の量子化のいずれかをたたみ込みに使用できます。1 つは、element-wise 用に 8 ビットを出力します。もう 1 つは、次のたたみ込み用に 4 ビットを出力します。アルゴリズムの最適化によって、どちらの量子化メソッドも 2^k のステップとして量子化できます。違いは出力データのビット幅と、符号付きデータかどうかです。

DSP の拡張使用

DSP DDR (ダブル データ レート) 手法を使用すると、DSP48 スライスで達成されるパフォーマンスが向上します [参照 13]。つまり、DPU には汎用ロジック用と DSP スライス用に 2 つの入力クロックが必要です。図 5 に、DSP DDR 手法を使用しない DPU と、使用した場合の DPU アーキテクチャの違いを示します。

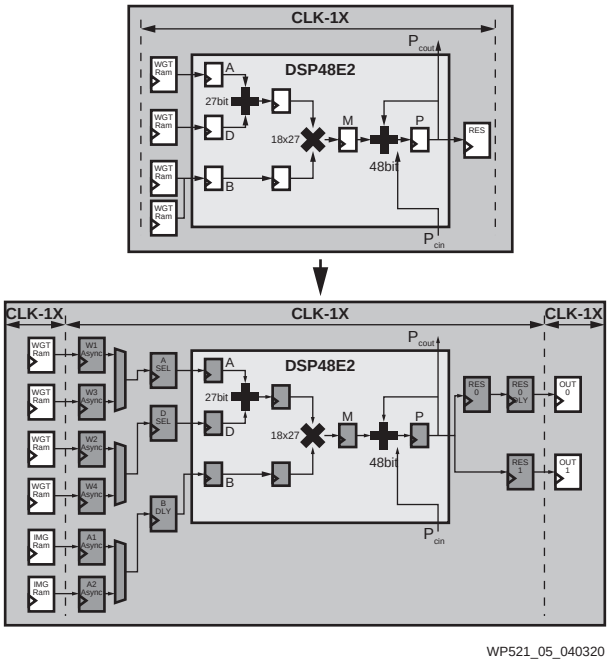

図 5: DDR を使用しない場合と使用した場合の DSP の違い

CNN 要件との演算マップ

たたみ込みは、CNN ネットワークの主要なコンピューティング要件です。たたみ込みの実際の計算タスクは次のとおりです。

$$X_f = \sum_{n=0}^{N} A_{nf} \cdot W_{nf} + Bias_f$$

式 6

ここで、A_{nf} は浮動小数点の機能マップ、W_{nf} は浮動小数点の重みを表します。重要なのは MAC 演算です。ザイリンクスの新しい量子化認識トレーニング ソリューションに従って、浮動小数点のたたみ込み計算は次のように量子化されます。

$$X_f = \sum_{n=0}^{N} \alpha_{xf}\, A_{int} \cdot \alpha_{wf} W_{int} + \alpha_{bf}\, Bias_{int}$$

式 7

$$X_{fixed} = 2^k \left(\sum_{n=0}^{N} A_{int} \cdot W_{int} + 2^{j-k} Bias_{int} \right)$$

式 8

ここで、a_{xf}、a_{wf}、a_{bf} はスケールを表します。これらの浮動小数点パラメーターは $2^k * 2^k$ に変換されます。これは、FPGA 上でシフト演算を使用して容易に実装可能な、ハードウェア処理しやすいスケールです。

DSP ブロックは、1 クロック サイクルで 2 つの重みと 2 つの機能を必要とします。図 6 に示すように、それぞれは共有可能です。

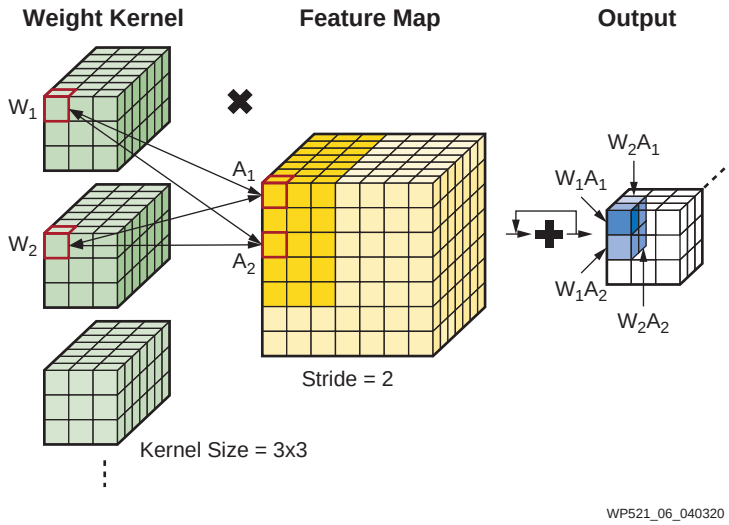

図 6: たたみ込み計算タスクと乗算器の共有方法

W_1 が置かれるカーネルでは、kernelwidth*kernelhigh*channel の全ピクセルをこの機能で乗算し、合計して 1 ピクセルの出力を得る必要があります。同じレイヤー内の各重みカーネルは、同じ機能マップを共有します。パックされる 2 つの重みは、2 つの異なる重みカーネルから来る必要があります。機能マップで重みカーネルがスライドする際は、各ステップの同じ重みカーネルで対応する機能データを乗算します。1 つの DSP48 ブロック内の 2 つの機能は、同じ機能マップ内の異なるスライディング ウィンドウから来る必要があります。

モデル量子化とパフォーマンス シミュレーション

以降のセクションでは、量子化認識トレーニングに使用する CV タスクについて説明します。タスクには、イメージ分類、姿勢推定、2D 検出、3D 検出、セマンティック セグメンテーション、マルチタスクが含まれます。

ベンチマーク分類モデル

ImageNet 分類データセットの実験は、次のような結果になりました。ネットワークには、ResNet50-V1、ResNet50-V2 が含まれます。どの実験でも、データセットはフロート モデルから微調整されます。バイアス パラメーターはすべて 8 ビットに量子化されます。表 1 に、実験結果を示します。

表 1: 異なるビット幅での ResNet50 型ネットワークの精度

モデル	A/W	入力/出力	最初のレイヤー / 最後のレイヤー	Element-Wise	トップ 1	トップ 5
ResNet50V1	フロート	フロート	フロート	フロート	76.15	92.87
	8/8	8/8	8/8	8	76.024	92.940
	4/4	4/8	4/4	8	74.588	91.998
ResNet50V2	フロート	フロート	フロート	フロート	77.596	93.53
	8/8	8/8	8/8	8	77.474	93.536
	4/4	4/8	4/4	8	74.124	91.422

表 1 のベンチマーク分類モデルの結果は、このメソッドの有効性を示しています。特に ResNet50V1 の場合、4 ビット XDPU ソリューションと 8 ビット XDPU ソリューションの間のギャップは、トップ 1 の精度で 1.4%、トップ 5 の精度で 0.9% しかありません。

実世界の ADAS モデル (姿勢推定、検出、セグメンテーション、マルチタスクなど)

量子化メソッドの汎用性をさらに検証するために、その他の CV タスクが実際のシナリオで実行されました。

姿勢推定

姿勢推定タスクには、より複雑な Stacked Hourglass ネットワークが使用されます [参照 14]。MPII [参照 15] データセットでの姿勢推定実験では、layer-wise モードを使用した 2 つのネットワーク構造の精度が評価されます。表 2 に、実験結果を示します。

表 2: 異なるビット幅での Hourglass ネットワークの精度

モデル	A/W	入力/出力	最初のレイヤー / 最後のレイヤー	Element-Wise	確度 (%)
hg-s2-b1	フロート	フロート	フロート	フロート	71.29
	8/8	8/8	8/8	8	72.04
	4/4	4/8	4/4	8	69.67
hg-s8-b1	フロート	フロート	フロート	フロート	**83.46**
	8/8	8/8	8/8	8	83.06
	4/4	4/8	4/4	8	82.51

表 2 の hg-s2-b1 は、スタック数 2、ブロック数 1 を意味します。hg-s8-b1 は、スタック数 8、ブロック数 1 を意味します。表 2 は、ザイリンクスの INT4 量子化ソリューションがフロート モデルと互角の確度を実現することを示しています。

2D 検出

ADAS システムでは、BDD100K [参照 16] データセットが 2D 検出に使用されます。さらに、FPN 構造が ResNet18-SSD に追加され、検出ネットワークとして使用されます。表 3 に、実験結果を示します。

表 3: 異なるビット幅での検出精度

タスク	A/W	入力/出力	最初のレイヤー / 最後のレイヤー	Element-Wise	mAP@0.5 (%)
2D 検出	フロート	フロート	フロート	フロート	39.0
	8/8	8/8	8/8	8	39.5
	4/4	4/8	4/4	8	37.8

表 3 は、微調整の後、8 ビットの量子化モデルがフロート モデルよりも高い mAP を達成したことを示しています。8 ビットから 4 ビットにプログレッシブ方式で微調整すると、最終的な 4 ビットの量子化モデルの mAP 損失は 2% 以内になります。図 7 に、2D 検出を視覚化したものを示します。

(a) GroundTruth (b) 8-Bit (c) 4-Bit

WP521_07_060920

図 7: 2D 検出の視覚化

3D 検出

ADAS システムの 3D 検出タスクでは、KITTI データセット [参照 17] が使用されました。3D 予測タスクの実行には、PointPillars [参照 18] が使用されます。表 4 に、実験結果を示します。

表 4: 異なるビット幅での 3D 検出結果

タスク	A/W	入力/出力	最初のレイヤー /最後のレイヤー	Element-Wise	中レベル車 AP@0.5(%)	
					BEV	3D
3D 検出	フロート	フロート	フロート	フロート	90.12	90.03
	8/8	8/8	8/8	8	90.00	89.84
	4/4	4/8	4/4	8	90.07	89.87

表 4 に示すように、微調整をすることで、4 ビットの量子化モデルの結果的なギャップは、フロート モデルと比べてわずか 0.16% になります。図 8 に、8 ビットと 4 ビットの 3D 検出結果を示します。

(a)GroundTruth (b)8-bit (c)4-bit

図 8: カメラでの 3D 検出の視覚化と鳥瞰図

セマンティック セグメンテーション

ADAS システムのセマンティック セグメンテーション タスクでは、都市の視覚的シーンの理解に重点を置いた CityScape のデータセット [参照 19] が使用されます。この実験は、ResNet18 をバックボーンとした Feature Pyramid Network (FPN) で実施されました。表 5 に、実験結果を示します。

表 5: 異なるビット幅でのセマンティック セグメンテーションの精度

タスク	A/W	入力/出力	最初のレイヤー / 最後のレイヤー	Element-Wise	mIoU(%)
セグメンテーション	フロート	フロート	フロート	フロート	63.62
	8/8	8/8	8/8	8	63.97
	4/4	4/8	4/4	8	61.90

表 5 は、8 ビット モデルがフロート モデルよりも高い mIoU を達成し、4 ビット モデルでも mIoU が 1.7% しか劣らないことを示しています。図 9 に、セマンティック セグメンテーションを視覚化したものを示します。

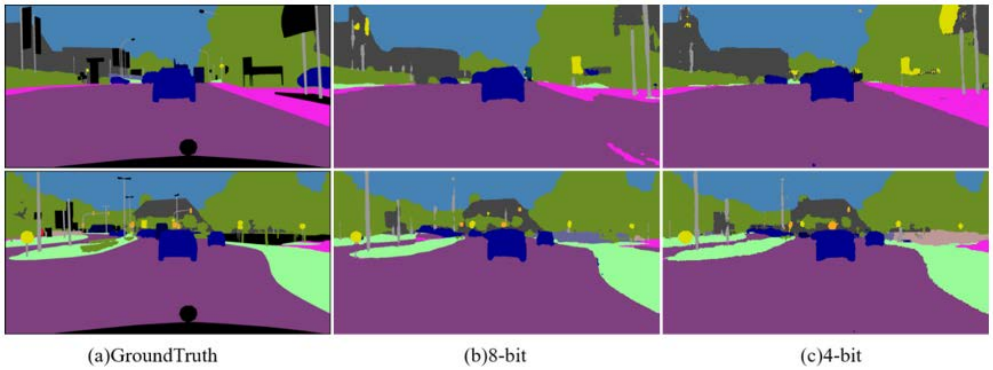

図 9: セマンティック セグメンテーションの視覚化

マルチタスク学習

モデルの汎化能力と精度を向上させるため、マルチタスク モデルでは複数のトレーニング データセット (検出用の Waymo と BDD100k、セグメンテーション用の BDD100k と Cityscapes [参照 19] など) が使用されます。これらの研究は、ResNet18 をバックボーンとした Feature Pyramid Network (FPN) で実施されました。表 6 に、実験結果を示します。

表 6: 異なるビット幅でのマルチタスクの精度

タスク	A/W	入力/出力	最初のレイヤー/最後のレイヤー	Element-Wise	確度 (%)	
					Det(mAP)	Seg(mIoU)
マルチタスク	フロート	フロート	フロート	フロート	39.06	42.12
	8/8	8/8	8/8	8	39.94	45.3
	4/4	4/8	4/4	8	37.4	43.91

表 6 は、8 ビットの量子化モデルがフロート モデルよりも高い mAP と、フロート モデルと同じ mIoU を達成したことを示しています。プログレッシブ方式で微調整すると、最終的な 4 ビットの量子化モデルは、フロートと比べて mAP が 1.66% 減りましたが、mIoU は (8 ビットよりは低いものの) 1.79% 増えました。図 10 に、マルチタスクの視覚化結果を示します。

図 10: マルチタスク学習の視覚化

競合分析: 8 ビット vs. 4 ビット

4 ビット XDPU は、Ultra96、Zynq UltraScale+ MPSoC ZCU104 および ZCU102 の 3 つの評価ボード上で周波数 300MHz で動作します。表 7 に、4 ビット XDPU と 8 ビット XDPU のパフォーマンス比較を示します。異なる FPGA 上で、4 ビット XDPU は 1.5 ～ 2.0 倍のパフォーマンス向上を達成しました。たとえば、ZCU102 ボードで使用したハードウェア リソースは増やさずに、パフォーマンスが 2 倍に向上しました。

表 7: 4 ビット XDPU と 8 ビット XDPU のパフォーマンス比較

	Ultra96	ZCU104	ZCU102
8 ビット XDPU	691GOPs	2.45TOPs	3.69TOPs
4 ビット XDPU	1228GOPs	3.69TOPs	7.37TOPs

精度の異なる 2 つのアクセラレータで、すべての機能 (プーリング、element-wise、depth-wise のたたみ込み、平均プーリングなど) を有効化した後でリソースが比較されました。表 8 に示すように、同じパフォーマンス アーキテクチャで DSP と RAM の使用量が大幅に減りました。リソース消費量が減ったため、4 ビット XDPU アーキテクチャは最大サイズ B8192 まで拡張されました。B8192 アーキテクチャを使用すると、1 つのデバイスでパフォーマンスを高めることができます。

表 8: 4 ビット XDPU と 8 ビット XDPU のリソース比較

4 ビット XDPU					8 ビット XDPU				
アーキテクチャ	LUT	Regs	ブロック RAM	DSP	アーキテクチャ	LUT	Regs	ブロック RAM	DSP
B512 (4x 8x 8)	25322	32211	41.5	62	B512 (4x 8x 8)	26482	33530	73.5	110
B800 (4x10x10)	29137	38398	56	97	B800 (4x10x10)	29711	40184	91.5	157
B1024 (8x 8x 8)	31378	42699	57.5	122	B1024 (8x 8x 8)	32598	47282	105.5	218
B1152 (4x12x12)	32928	43337	73	116	B1152 (4x12x12)	31769	46462	123	212
B1600 (8x10x10)	36504	52101	76	192	B1600 (8x10x10)	36838	58204	127.5	312
B2304 (8x12x12)	38389	58090	97	230	B2304 (8x12x12)	40039	68469	167	422
B3136 (8x14x14)	43316	67901	120	324	B3136 (8x14x14)	43972	79141	210	548

表 8: 4 ビット XDPU と 8 ビット XDPU のリソース比較 (続き)

4 ビット XDPU					8 ビット XDPU				
B4096 (8x16x16)	48232	75896	145.5	370	B4096 (8x16x16)	49754	97882	257	690
B8192 (8x32x16)	55890	91426	201.5	690	B8192 (8x32x16)	サポート対象外			

たとえば、表 3 に示す 13.6 FLOPs の 2D 検出モデルでは、4/4 と 8/8 の 2 つの精度モデルが、それぞれ 4 ビット XDPU と 8 ビット XDPU を使用してテストされました。このネットワークのコンピューティング要件は 13.6 GOP です。表 9 に、2D 検出ネットワークのフレーム レートを示します。このテストには、前処理と後処理は含まれません。効率性とネットワーク タイプに違いがあるため、パフォーマンスとフレーム レートは線形ではありません。表 9 に示すように、4 ビット XDPU のフレーム レートは、すべてのプラットフォームで 8 ビット XDPU よりも優れています。

表 9: 4 ビット DPU と 8 ビット DPU のフレーム レート

	Ultra96	ZCU104	ZCU102
2D 検出 (8/8)	30fps	101fps	151fps
2D 検出 (4/4)	53fps	145fps	230fps

まとめ

このホワイト ペーパーでは、Zynq UltraScale+ MPSoC と Zynq-7000 SoC ファミリ (16nm、28nm) でフルプロセスのハードウェア処理しやすい量子化ソリューションを使用した、CNN 用の低精度アクセラレータについて説明しました。また、ザイリンクス DSP スライス上で最適化された INT4 を使用して、4 チャネルの INT4 乗算を 1 クロック サイクルで完了する方法についても説明しました。DSP のパッキングは、たたみ込みの演算要件を満たしています。DSP を使用した INT4 最適化では、INT8 XDPU ソリューションと比較して実際のハードウェアでのピーク GOPS を 2 倍、パフォーマンスを 1.77 倍にできます。ザイリンクスのソリューションは、さまざまな CV タスクで浮動小数点モデルと互角の結果を出すことができます。ザイリンクスは、リソースが限られたユース ケースや消費電力に制約のあるユース ケース向けに、深層学習アプリケーションの高速化を目的としてハードウェアとソフトウェアの両方を最適化する設計手法を開発し続けています。

謝辞

このホワイト ペーパーは、次のザイリンクス従業員が執筆または協力しました。Tiantian Han (ソフトウェア エンジニア、AI アルゴリズム担当)、Tianyu Zhang (デザイン エンジニア 2、AI エッジ コンピューティング担当)、Dong Li (シニア ソフトウェア開発マネージャー、AI アルゴリズム担当)、Guangdong Liu (デザイン エンジニア、AI エッジ コンピューティング担当)、Lu Tian (ソフトウェア開発ディレクター、AI アルゴリズム担当)、Dongliang Xie (デザイン エンジニア ディレクター、AI エッジ コンピューティング担当)、Yi Shan (シニア ディレクター、AI 担当)。

参考資料

1. Olga Russakovsky、Jia Deng、Hao Su、Jonathan Krause、Sanjeev Satheesh、Sean Ma、Zhiheng Huang 他著。2015 年。『Imagenet Large Scale Visual Recognition Challenge』International Journal of Computer Vision 115 (3): 211-252。
2. Kaiming He、Xiangyu Zhang、Shaoqing Ren、Jian Sun 著。2016a 年。『Deep Residual Learning for Image Recognition』In Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, 770-778。
3. Kaiming He、Xiangyu Zhang、Shaoqing Ren、Jian Sun 著。2016b 年。『Identity Mappings in Deep Residual Networks』In European Conference on Computer Vision, 630-645。Springer。
4. Andrew G Howard、Menglong Zhu、Bo Chen、Dmitry Kalenichenko、Weijun Wang、Tobias Weyand、Marco Andreetto、Hartwig Adam 著。2017 年。『Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications』arXiv Preprint arXiv:1704.04861。
5. Mark Sandler、Andrew Howard、Menglong Zhu、Andrey Zhmoginov、Liang-Chieh Chen 著。2018 年。『Mobilenetv2: Inverted Residuals and Linear Bottlenecks』In Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, 4510-4520。
6. Williams, S 著。2009 年。『Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore』
7. Sambhav R Jain、Albert Gural、Michael Wu、Chris Dick 著。2019 年。『Trained Quantization Thresholds For Accurate And Efficient Fixed-Point Inference Of Deep Neural Networks』arXiv Preprint arXiv:1903.08066。
8. Benoit Jacob、Skirmantas Kligys、Bo Chen、Menglong Zhu、Matthew Tang、Andrew Howard、Hartwig Adam、Dmitry Kalenichenko 著。2018 年。『Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference』In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2704-2713。
9. Raghuraman Krishnamoorthi 著。2018 年。『Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper』http://arxiv.org/abs/1806.08342。
10. Yaman Umuroglu、Magnus Jahre 著。2017 年。『Streamlined Deployment for Quantized Neural Networks』arXiv Preprint arXiv:1709.04060。
11. ザイリンクス『UltraScale アーキテクチャ DSP スライス ユーザー ガイド』、2019 年 5 月。https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf。
12. Yao Fu、Ephrem Wu、Ashish Sirasao、Sedny Attia、Kamran Khan、Ralph Wittig 著。2016 年。『ザイリンクス デバイスでの INT8 に最適化した深層学習の実装』ホワイト ペーパー。
13. ザイリンクス『DPU for Convolutional Neural Network v3.0 IP Product Guide』、2019 年 8 月。https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf。
14. Alejandro Newell、Kaiyu Yang、Jia Deng 著。2016 年。『Stacked Hourglass Networks for Human Pose Estimation』In European Conference on Computer Vision, 483-499。Springer。
15. Mykhaylo Andriluka、Leonid Pishchulin、Peter Gehler、Bernt Schiele 著。2014 年。『2d Human Pose Estimation: New Benchmark and State of the Art Analysis』In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3686-3693。
16. Fisher Yu、Wenqi Xian、Yingying Chen、Fangchen Liu、Mike Liao、Vashisht Madhavan、Trevor Darrell 著。2018 年。『Bdd100k: A Diverse Driving Video Database with Scalable Annotation Tooling』arXiv Preprint arXiv:1805.04687。
17. Andreas Geiger、Philip Lenz、Christoph Stiller、Raquel Urtasun 著。2013 年。『Vision Meets Robotics: The Kitti Dataset』The International Journal of Robotics Research 32 (11): 1231-1237。

18. Alex H. Lang、Sourabh Vora、Holger Caesar、Lubing Zhou、Jiong Yang、Oscar Beijbom 著。2019 年。『PointPillars: Fast Encoders for Object Detection from Point Clouds』In the IEEE Conference on Computer Vision and Pattern Recognition。
19. Marius Cordts、Mohamed Omran、Sebastian Ramos、Timo Rehfeld、Markus Enzweiler、Rodrigo Benenson、Uwe Franke、Stefan Roth、Bernt Schiele 著。2016 年。『The Cityscapes Dataset for Semantic Urban Scene Understanding』In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3213-3223。

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	内容
2020 年 6 月 24 日	1.0.1	誤植の修正
2020 年 6 月 19 日	1.0	初版

免責事項

自動車用のアプリケーションの免責条項