C-010

エッジ端末を用いた「YOLOv5」の畳み込み演算の高速化の検討
Consideration of speeding up convolutional operations in “YOLOv5” using edge devices

三枝 渉† 中西 知嘉子‡
Ayumu Saegusa Chikako Nakanishi

1. はじめに

近年，セキュリティや通信遅延・通信コストの観点から，ネットワークを使用せず，手元のエッジ端末上で推論処理を行うエッジ AI が注目されている．

エッジ端末上でディープラーニングによる推論処理をリアルタイムに行わせるためには，大量の計算を高速に処理させるという課題を解決しなければならない．大量の計算を高速に処理するには GPU を用いると可能になるが，低コスト・低リソース環境下では，消費電力やコストの問題により，GPU なしでリアルタイムに推論処理を行わせることが難しい．また，専用回路の開発においても開発期間が非常に長く，仕様変更に対して柔軟に対応できない問題がある．そこで，畳み込み演算処理と活性化関数処理を FPGA で，その他の処理を CPU で処理させる方法で精度を維持しつつ推論処理の高速化を検討している．物体検出 AI モデルのひとつである YOLOv5s モデル[1]を高速化の対象とし，高負荷な処理を回路で実行し，その他の処理を CPU で処理させる方法で高速化を試みた[2][3]．その結果，CPU 側のメモリから回路へのデータ転送の方法に改善の余地があることがわかった．そこで提案手法に最適なデータ転送の手法について検討する．

2. 開発環境・使用モデル

本研究では，特に画像処理などで広く扱われている畳み込みニューラルネットワークについて着目し，その中で YOLOv5 の比較的高速なモデルである，YOLOv5s モデルを用いて設計を行う．

2.1 Ultra96v2

本研究で使用するエッジ端末は，Xilinx 社の SoC FPGA を搭載した Avnet 社の Ultra96v2[4]を用いる．SoC FPGA は CPU と FPGA が同一チップ上にあるため，高速なデータ転送が可能で，FPGA と CPU で細かく処理を分割することが可能である．

2.2 YOLOv5

YOLOv5 は 2020 年 6 月に公開された物体検出 AI のひとつで，本研究では比較的高速な YOLOv5s モデルを用いる．

はじめに，本研究で用いる SoC FPGA チップを搭載したボードである Ultra96v2[4]の CPU のみで処理させた場合の高負荷な処理を調べた．表 2.2.1 に YOLOv5s の畳み込み層の種類を，図 2.2.1 に各層の処理時間の割合を示す．

表 2.2.1　YOLOv5s の最大データサイズ

フィルタサイズ	最大入力サイズ	最大チャネル数	最大カーネル数	ストライド
1×1	80×80×256 160×160×64	1024	256	1
3×3	320×320×32	256	512	1,2
6×6	640×640×3	3	32	2

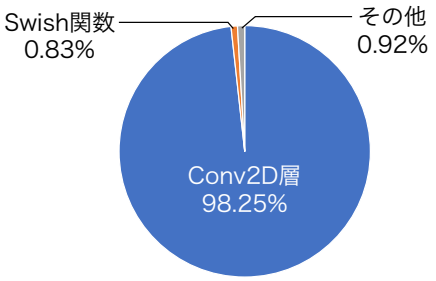

図 2.2.1　YOLOv5s の各層の処理時間

図 2.2.1 から，畳み込み演算を行う Conv2D 層が全体の処理の約 98%を占めていることがわかる．また，ほぼ全ての Conv2D 層の後に Swish 関数処理を行う Activation 層があることから，回路化の対象を Conv2D 層と Activation 層とし，それ以外の層を CPU で処理させる．

3. 開発手法

3.1 Ceras

CPU で行う処理については，Ceras[5]と呼ばれる先行研究で開発された，C++言語で学習済みモデルを推論させることができるツールを用いる．このツールは C++言語の標準ライブラリのみを用いて作成されているため，エッジ端末などで動作させるための環境構築の難易度が低下する．また C++言語を使用することにより，CPU と FPGA の処理を，層の内部構成まで考えて分担する範囲を考えることができる．

3.2 回路

回路設計は，C++言語を用いてアルゴリズムを設計し，RTL 言語へ高位合成を行うことで回路化を行なった．この方法を用いることで，演算のアルゴリズムを明確に定義することができ，必要に応じて柔軟に回路を変更できることから，設計の効率を上げることが可能となる[7]．さらに，回路のアルゴリズムが C++言語で設計されているため，CPU との協調動作の検証難易度が低下する．

回路の作成については，回路作成ツールである Xilinx 社の Vivado Design Suite 2020.2[7]と高位合成ツールである Vitis HLS 2020.2[7]の 2 つのツールを用いて，フィルタサイズ 3×3 のストライド 1 とストライド 2，フィルタサイズ 1×1 のストライド 1 の 3 種類の畳み込み演算と活性化関数である Swish 関数の回路化を行う．Swish 関数処理については，Swish 関数から Hard Swish 関数[8]に変更して回路化を行う．

フィルタサイズ 6×6 については，6×6 の Conv2D 層が，1 層しかなく，6×6 の畳み込み演算回路を作成することは有効でないため，カーネルと入力データを 4 分割して回路に転送し，3×3 の畳み込み演算として処理している．

表 3.2.1 に作成した回路の最大保持サイズを示す．

表 3.2.1　回路の最大保持サイズ

フィルタサイズ	最大入力サイズ	最大チャネル数	最大カーネル数
1×1	82×82	576	128
3×3		64	
6×6 *1			

*1 フィルタサイズ 6×6 は，フィルタサイズ 3×3 として処理

表 3.2.1 より，入力サイズや最大チャネル数，最大カーネル数が最大保持サイズを超える場合は，CPU 側で分割処理を行い，複数回回路を使用して処理を行う．チャネル分割を行なった場合は，回路で Hard Swish 関数の処理を行わず，CPU 上で Hard Swish 関数の処理を行う．

YOLOv5s モデルにより適した回路にするため，FPGA のリソースを考慮した上で，先行研究で使用した回路の最大入力サイズの 66×66 から 82×82 に変更している．YOLOv5s モデルは縦横の入力サイズが 20，40，80，160，320，640 で構成されている．よって 82×82 に変更することにより，分割回数が減り，回路の実行回数が減る．最大チャネル数や最大カーネル数については変更していない．

精度への影響は Swish から Hard Swish へ変更したことのみであるため，精度の評価は行わず，推論処理時間で評価を行う．

4. 最適なデータ転送の検討

4.1 フィルタとバイアスの転送時間の短縮

先行研究は推論処理の前に重み情報を読み込み，CPU 側のメモリに格納する．その後，推論時に各層の必要なデータを，CPU 側のメモリから共有メモリに書き込み，転送するデータの先頭アドレスと転送サイズを指定する．そして DMA 転送を行う．重み情報読み込み時の動作を変更し，読み込み時に CPU 側のメモリに書き込まず，直接共有メモリに書き込んでおくように変更することで転送時間を短縮できる．DMA 転送で回路に転送する際は，CPU が管理している共有メモリの転送開始アドレスと転送サイズを指定する．以降，重み情報読み込み時にバイアスとフィルタデータを共有メモリに書き込むことをオフセット転送と呼ぶ．

4.2 Conv2D 層の出力データの共有メモリ活用

先行研究では，入力データを共有メモリに書き込み，DMA 転送を用いて回路に転送し，回路の実行完了を待つ．回路での演算が終了すると，DMA 転送を用いて演算結果を共有メモリに転送する．その後，共有メモリから CPU 側のメモリに演算結果を書き込むことで，畳み込み演算を行っていた．

CPU と回路間のデータ転送時間の短縮を図るため，Conv2D 層が連続していれば，CPU 側メモリへの書き込みを行わず次の Conv2D 層の処理に移行する．次の Conv2D 層を処理する際の入力するデータは，前の Conv2D 層の出力データとなるので，その出力データを DMA 転送で回路に転送するように変更した．この変更により，共有メモリから CPU 側のメモリに書き込む時間と，CPU 側から共有メモリへ書き込む時間を短縮している．

5. 処理時間の測定と評価

先行研究の処理時間と改良したソフトウェアの処理時間の比較を，回路を用いて推論処理時間の測定を行った．推論時間測定結果を図 5.1 に示す．

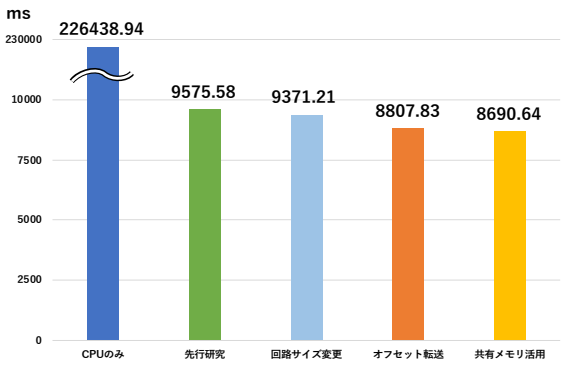

図 5.1　推論処理時間測定結果

図 5.1 より，フィルタとバイアスの書き込み時間移動によって，先行研究の推論処理時間である 9575.58ms から回路の最大サイズ変更によって，9371.21ms まで短縮，オフセット転送による転送方法によって 8807.83ms まで推論時間を短縮することができた．さらに，オフセット転送による転送方法の 8807.83ms から共有メモリを有効活用することで，8690.64ms まで推論時間を短縮することができた．

6. まとめ

回路の最大入力サイズを変更したことによる処理時間は先行研究と比較して 200ms ほど短縮することができた．また，サイズ変更に加え，オフセット転送による処理時間，CPU 側のメモリにデータを格納せず，共有メモリを活用する手法によって，推論処理中にかかる共有メモリへの書き込み時間の短縮を図った．結果として，先行研究と比較してオフセット転送による転送が 700ms 程度，オフセット転送と CPU 側のメモリを一部使用せずにデータを転送した場合，800ms 程度短縮することができた．

7. 参考文献

[1] YOLOv5,<https://github.com/ultralytics/yolov5>
[2] 大戸彰馬,中西知嘉子,"推論処理における畳み込み処理の回路化の検討",電子情報通信学会総合大会(2022).
[3] 三枝渉,中西知嘉子,”物体検出モデル「YOLOv5」のエッジデバイスへの実装の検討”,電子情報通信学会総合大会(2023).
[4] Ultra96v2,<https://www.avnet.com/wps/portal/japan/products/product-highlights/ultra96/>
[5] Ceras,西岡駿,中西知嘉子,"機械学習ライブラリの C 言語 化の実現",電子情報通信学会ソサイエティ大会(2021).
[6] keras2cpp,<keras2cpp,https://github.com/gosha20777/keras2cpp >
[7] Vivado ツール,< https://japan.xilinx.com/products/design-tools/vivado.html#webpack >
[8] Hard Swish,< https://arxiv.org/pdf/1905.02244v5.pdf>