7J-02

エッジコンピューティングによる畳み込みニューラルネットワークを用いた物体検出

Object Detection Using Convolutional Neural Network by Edge Computing

大内 佑一朗† 吉田 明正†
Yuichiro Ouchi Akimasa Yoshida

1 はじめに

ディープラーニングを用いた物体検出は，従来，エッジデバイスからサーバを利用して処理を行っていたが，近年，物体検出においてリアルタイム性がより重視されており，サーバへのデータ転送によるレイテンシが問題とされている．そのような問題点を解決するために，デバイス上で処理を行うエッジコンピューティングが期待されている．小規模な計算資源を有するエッジデバイスにおいて，推論を高速化する方法としては，NVIDIA TensorRT ライブラリによるモデル最適化が注目されており，このライブラリにより最適化された推論モデルを使用することで推論時間の大幅な短縮が可能となる．本稿では，NVIDIA Jetson AGX Xavier 上で TensorRT[1] によるモデル最適化の性能評価を行い，その有効性を確認する．

2 畳み込みニューラルネットワークを用いた物体検出

物体検出とは，画像から物体の位置と大きさ，カテゴリを抽出するものである．近年，畳み込みニューラルネットワークを用いたディープラーニング技術の進展により精度が向上している．本稿では，物体検出のフレームワークとして，YOLOv4[2] を用いる．

2.1 エッジデバイスにおける物体検出

従来の物体検出では，カメラなどのデバイスからクラウド上のサーバにデータを送信し，サーバ上で処理を行い，デバイス側にフィードバックすることが多かった．しかし近年では，自動運転などに用いられる物体検出において，よりリアルタイム性が重視されるようになり，特に，データ送受信におけるレイテンシの低下が求められるようになった．そこで，サーバに比べると，計算資源は乏しいが，データ送受信におけるレイテンシを減らすことができるエッジデバイスの利用が注目されている．本稿では，100 × 87mm ほどの大きさながら，最大 32TOPS の計算処理能力を持つ NVIDIA Jetson AGX Xavier を用いて性能評価を行う．

2.2 YOLOv4 フレームワーク

YOLO は物体検出のフレームワークのひとつであり，C 言語で作成された機械学習用のフレームワーク Darknet[3] を用いて作成されている．2020 年に発表された YOLOv4 では，YOLOv3[4] と同程度の速度ながら，高い精度を実現しており，その他の従来手法と比較しても，高いパフォーマンスが確認されている．また，層数が少なく，精度が落ちるが，高速に学習・推論を行うことができる YOLOv4-tiny も開発されており，よりリアルタイム性が求められる場面では，YOLOv4-tiny が使用されることも多い．また，現在 YOLOv5[5] が発表されているが， Darknet を使用しておらず，PyTorch を用いて作成されたフレームワークであるため，本稿では扱わない．

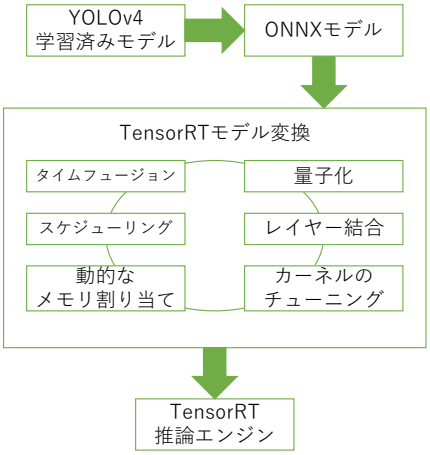

図 1 TensorRT フレームワークを用いたモデル変換．

3 学習済みモデル変換による推論の高速化

YOLOv4 などの物体検出フレームワークにおける畳み込みニューラルネットワークのモデルは，学習用にチューニングされていることが多く，学習済みのモデルを使用して推論を行う場合，思うような性能が出ないことがある．その対策として，推論用のモデルに変換して推論を行うことで，高速化が期待できる．本稿では， YOLOv4 で学習したモデルに対して，図 1 のように，YOLOv4 を ONNX[6] と TensorRT を使用してモデル変換を行うことで，推論の高速化を目指す．

3.1 ONNX によるモデル表現

ONNX は Open Neural Network Exchange の略であり，ニューラルネットワークのモデルを様々なフレームワーク間で交換するためのフォーマットである．従来までは学習から推論まで，同一のフレームワークを用いて開発を行う必要があったが，ONNX を用いることで，各プロジェクトに適したフレームワークを部分ごとに使用できるようになった．本稿では，YOLOv4 で学習した学習済みモデルに対し，TensorRT フレームワークを使用してモデル変換を行うための橋渡しとして使用する．

†明治大学大学院先端数理科学研究科ネットワークデザイン専攻
Department of Network Design, Graduate School of Advanced Mathematical Sciences, Meiji University

3.2 TensorRT フレームワークによるモデル変換

TensorRT フレームワークは機械学習の推論用の SDK であり，学習済みのモデルを TensorRT での推論用に変換し，高速かつ高精度に推論を行うことができるものである．NVIDIA の並列プログラミングモデルである CUDA 環境での実行を対象としており，図 1 のように，量子化やレイヤーの結合などの変換を行い，最適化された推論エンジンを作成する．以下でそれぞれについて詳しく述べる．

- 量子化： モデルを FP16 や INT8 に量子化し，メモリ使用及び演算量を削減する．またその際に，量子化による制度の低下を抑えるためのチューニングを行う．
- レイヤー結合： 推論実行時のレイヤー数やカーネルの起動回数を少なくし，GPU メモリと帯域の使用を最適化する．
- カーネルの自動チューニング： 使用するプラットフォームの GPU に基づいた，最適なデータ層とアルゴリズムを選択する．
- 動的なメモリ割り当て： 演算中のメモリ使用を最小化し，メモリ再利用性やメモリ割り当てオーバーヘッドの回避を行う．
- マルチストリーム処理： 複数の入力に対するスケジューリングを最適化する．
- タイムフュージョン： 動的に生成されるカーネルを用いてニューラルネットワークの処理時間を最適化する．

このような変換を行うことで，推論時のオーバーヘッドを減らし，高速に推論を行うことができる．

4 モデル最適化を伴う物体検出の性能評価

本稿では，YOLOv4 フレームワークに対し，TensorRT によるモデル変換を行い，変換後のモデルの評価を行う．

4.1 性能評価環境

本性能評価では表 1 のマシンを利用し，性能評価を行った．また，モデル変換には ONNX1.4.1，TensorRT7.1.3 を使用した．

表 1 性能評価に用いる NVIDIA Jetson AGX Xavier.

CPU	8Core ARM v8.2 64-Bit CPU
メモリ	32GB
GPU	512Core Volta GPU with Tensor Cores
OS	Ubuntu18.04LTS
処理系	CUDA10.2

表 2 TensorRT を用いた YOLOv4 のモデル変換の性能評価.

推論枚数	1 枚 [s]	10 枚 [s]	100 枚 [s]	200 枚 [s]
TensorRT なし	5.22	7.35	25.97	48.16
TensorRT あり	3.61	4.33	8.84	13.88

4.2 TensorRT を用いた推論の性能評価

本性能評価では，YOLOv4 において，学習済みモデルを ONNX と TensorRT を用いてモデル変換を行い，変換後のモデルの評価を行った．学習済みモデルは MS COCO データセットを使用して学習したモデルを使用した．推論画像枚数ごとの時間を測定したところ，性能評価結果は表 2 の通りとなり，200 枚の推論で YOLOv4 の 48.16[s] に対し，TensorRT は 13.88[s] と 3.47 倍の速度向上が得られた．また，YOLOv4-tiny においても同様に評価を行ったところ，評価結果は表 3 の通りとなり，200 枚の推論で YOLOv4-tiny の 15.22[s] に対し，TensorRT は 7.58[s] と 2.01 倍の速度向上が得られた．

また，1 秒間に推論可能な枚数をまとめたものが表 4 であり，YOLOv4 の 4.83[fps] に対し，TensorRT は 20.00[fps] で 4.14 倍，YOLOv4-tiny の 14.59[fps] に対し，TensorRT は 35.43[fps] で 2.43 倍の速度向上が得られた．

表 3 TensorRT を用いた YOLOv4-tiny のモデル変換の性能評価.

推論枚数	1 枚 [s]	10 枚 [s]	100 枚 [s]	200 枚 [s]
TensorRT なし	2.23	2.94	9.11	15.22
TensorRT あり	2.85	2.97	5.51	7.58

表 4 TensorRT を用いた YOLOv4/YOLOv4-tiny のモデル変換における 1 秒当たりの推論可能枚数.

推論モデル	画像 1 枚の推論時間 [ms]	1 秒当たりの推論枚数 [fps]
YOLOv4(TensorRT なし)	207	4.83
YOLOv4(TensorRT あり)	50	20.00
YOLOv4-tiny(TensorRT なし)	69	14.59
YOLOv4-tiny(TensorRT あり)	28	35.43

5 おわりに

本稿では，物体検出に用いられる YOLOv4 フレームワークに対して，ONNX と TensorRT を用いたモデル変換を行い，推論を行った．

性能評価では，YOLOv4 フレームワークに対して ONNX と TensorRT を用いてモデル変換を行うことで，3.47 倍の速度向上が，YOLOv4-tiny に対してモデル変換を行うことで 2.01 倍の速度向上が得られた．また fps においても YOLOv4 は 4.14 倍，YOLOv4-tiny は 2.43 倍の速度向上が得られた．

以上の結果から，YOLOv4 並びに YOLOv4-tiny における，ONNX と TensorRT によるモデル変換の有効性が確認された．

参考文献

[1] TensorRT. https://developer.nvidia.com/tensorrt 2021.

[2] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection, arXiv:2004.10934, 2020.

[3] Joseph Redmon. Darknet: Open Source Neural Networks in C, http://pjreddie.com/darknet/, 2013-2016.

[4] Joseph Redmon, Ali Farhadi. YOLOv3: An Incremental Improvement, arXiv:1804.02767, 2018.

[5] YOLOv5. https://github.com/ultralytics/yolov5 2021.

[6] ONNX. https://onnx.ai/ 2019.