TensorFlow Lite for Microcontrollers を使用した組込み AI の実現性検討

小河原 徹，渡辺 泰久

近年，深層学習アルゴリズムがセンシングに適用され，画像センサなどの高精度化に貢献している。

本研究では，単純な一次元信号を扱うシンプルなセンサに深層学習アルゴリズムを適用し，センサの精度向上を目的とする。シンプルなセンサは製品価格を廉価にしなければならず，コスト面の制約がある。このため，大規模なアルゴリズムの搭載は難しいが，組込み用深層学習プラットフォームの活用とモデルの軽量化により実現を目指す。

本稿では，シンプルなセンサの実例として血圧計を用いる。まず深層学習プラットフォームの選定を行い，次にアルゴリズム開発環境で学習済みのモデルを血圧計に組込む手法と，モデルの軽量化手法について説明する。最後に ROM・RAM 容量や精度・実行時間を評価し，提案手法の有効性を示す。

Feasibility Study for Embedded AI Using TensorFlow Lite for Microcontrollers

KOGAWARA Toru and WATANABE Yasuhisa

In recent years, deep learning algorithms have been applied to sensing, and are contributing to the enhancement of high-precision devices such as image sensors.

This study aims to apply deep learning algorithms to simple sensors that handle one-dimensional signals, with the goal of improving their accuracy. Due to cost constraints to realize lower product prices, it is challenging to implement large-scale algorithms into simple sensors; however, we aim to achieve this by utilizing embedded deep learning platforms and applying model optimization techniques.

In this paper, we use a blood pressure monitor as a practical example of a simple sensor. First, we choose a deep learning platform, and then we describe the method for integrating a pre-trained model into the blood pressure monitor, as well as the techniques for model optimization. Finally, we evaluate the ROM and RAM capacity, accuracy, and execution time, demonstrating the feasibility of the proposed method.

1. まえがき

近年，深層学習技術がセンシングのアルゴリズムに適用され始めている。深層学習の長所は，高精度化とノイズに対するロバスト性である。深層学習を利用することで，複雑なデータパターンの自動抽出による精度の高い認識や，ノイズの多い環境や多様なデータにも適応してノイズに惑わされない認識を実現できる。深層学習センシングは，自動運転車両における画像認識や，スマートスピーカーにおける音声認識などに適用されている。

オムロンはセンシングをコア技術の一つに掲げており，これまで多数のセンサ商品を創出してきた。画像センサのような大容量の信号を扱う複雑な商品もあるが，単純な一次元の信号を扱うシンプルなセンサ商品も多く存在している。シンプルなセンサにおいては，製品価格を高額にできないというコストの観点からハードウェアに制約があり，深層学習のような大規模なアルゴリズムを搭載することは難しい。

一方で，シンプルなセンサ商品においても，認識精度の向上やノイズ適応性を高めることは常に望まれている。そこで我々は，この深層学習を単純な一次元の信号を扱うセンサに搭載することで，低コストでセンシング精度をさらに改善することを目的として検討を行うこととした。

深層学習には「学習」と「推論」の 2 つのフェーズがあるが，一般に学習機能は推論機能に比べてアルゴリズムが

Contact : *KOGAWARA Toru* toru.kogawara@omron.com

複雑であり，必要とするメモリ容量や計算量が大きい。ハードウェア制約が厳しいことを考慮し，まず推論機能のみをセンサに搭載することから検討を開始する。

従って本稿での実現性検討の流れは，まずアルゴリズム開発環境で深層学習モデルを開発して学習させた後，そのモデルをセンサに搭載して計測・推論を実行させる，とする。図 1 に本稿で検討する深層学習センシング開発の構成を示す。

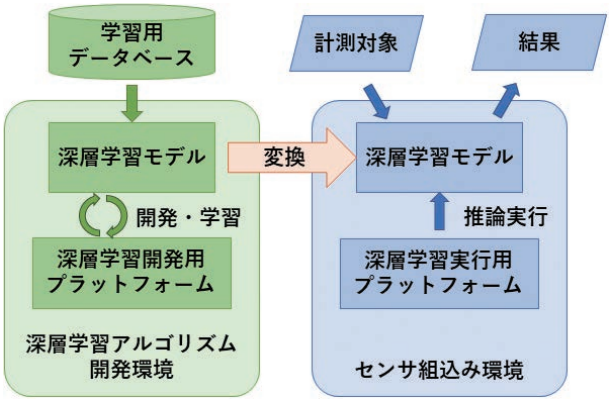


図 1　本稿で検討する深層学習センシング開発

本稿では，まず深層学習の開発用プラットフォームと実行用プラットフォームについて説明し，選定を行う。その後，開発環境での深層学習モデルをセンサ組込み環境用に変換する手法と，モデルを軽量化する手法について説明する。最後に，提案手法の評価のため，実装例として血圧計を使用してハードウェアリソース使用量や実行時間，精度などの結果を示す。

以上で述べた一連の流れを図 2 に示す。図中の吹き出しに付された「2.1」等の番号は，本稿の項目番号と対応している。

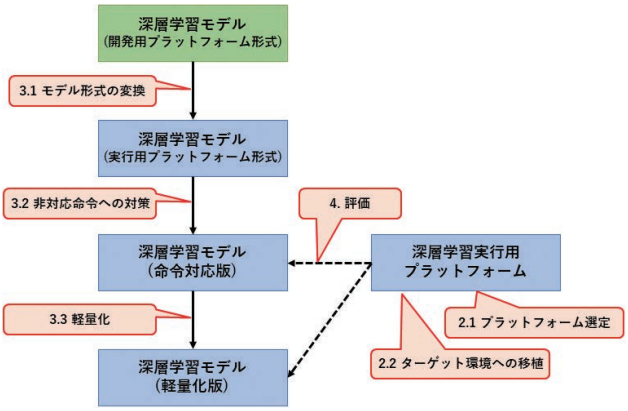


図 2　本稿での検討の流れ

2. プラットフォーム

深層学習アルゴリズムを開発する際には，まず深層学習プラットフォームを選定し，そのプラットフォーム上で動作するようにモデルを記述して学習させることでアルゴリズムを開発する。

開発したモデルを組込み環境で動作させるためには，組込み環境で動作するプラットフォームが別途必要になる。

2.1 深層学習開発用プラットフォーム

深層学習アルゴリズムを開発するためのプラットフォームの実例を表 1 にて説明する。

表 1　代表的な深層学習開発用プラットフォーム

プラットフォーム名	特徴
PyTorch	Facebook が開発したオープンソースライブラリ。代表的な深層学習開発プラットフォームの一つ。直感的にモデルを構築できる特徴があり，学術研究で多く用いられている。
TensorFlow (Keras)	Google が開発したオープンソースライブラリ。代表的な深層学習開発プラットフォームの一つ。TensorFlow に高レベル API を追加した Keras というプラットフォームもある。モデルのデプロイや管理等に関する周辺ツールが豊富。
TensorFlow Lite（以下 TFLite と略記）	TensorFlow を軽量化し，モバイル・エッジデバイス向けに最適化したもの。TensorFlow の命令のサブセットに対応。2024 年 9 月に LiteRT に改称した。
TVM	Apache が開発しているオープンソースの深層学習プラットフォーム。異なるハードウェアプラットフォームに対して最適化されたコードを生成できる。

本稿では，モデル構築や学習の容易さを重視し，PyTorch を採用した。

2.2 組込み環境プラットフォーム

深層学習アルゴリズム・モデルを組込み環境で動作させるためのプラットフォームにどのようなものがあるか調査し，選定を行う。本稿では深層学習をシンプルなセンサに搭載することを目的としている。実装例として血圧計を想定していることも考慮し，以下の 2 点を選定基準とする。

- シンプルなセンサにおいては設計におけるコスト制約が厳しいため，ハードウェアの選定において制約が少ないこと。
- Linux 等の高機能な OS を搭載できないような小規模なマイコンでも動作すること。

以上の基準でプラットフォームを調査した結果を表 2 に示す。

表 2　組込み用プラットフォーム調査結果

プラットフォーム名	特徴
TensorFlow Lite for Microcontrollers（以下，TFLM と略記）	TFLite をさらに組込みマイコン上で動作できるまで軽量化したオープンソースライブラリ。利用できるメモリサイズが数 kB のマイコンまで想定しており，対応命令も TFLite のさらにサブセットとなっている。モデル実行機能だけを持っており，再学習機能は持たない。
microTVM	TVM モデルを組込みマイコン上で動作させるオープンソースライブラリ。OS を搭載しないマイコンまで想定している。まだ開発中であり，今後大きな改変が予想される。
Edge Impulse	Edge Impulse 社の商用ソリューション。モデルの開発・学習機能から組込みデバイスへのデプロイまで総合的な機能を持つ。C 言語ソースコードとして出力し，他のソフトウェアに組込むことも可能。
STM32 X-CUBE-AI	STMicro 社のマイコン STM32 シリーズ統合開発環境の無償プラグイン。他のプラットフォームで開発されたモデルを取り込み，高効率な C 言語コードに変換する。モデルの実行機能だけを持っている。

microTVM と TFLM の 2 つが，先に挙げた選定条件を満たしている。ただし microTVM は大きな改変が予想されるため，検討結果が将来に活かせない可能性がある。よって本稿では TFLM を選定する。

2.3 ターゲットの選定

最終目的である小規模なハードウェアに，最初から実装するには技術的な困難が予想される。そこで，小規模なものから大規模なものまでバリエーションのあるマイコンファミリーの中で，比較的ハードウェアが大規模なものを使って実装検討を行うものとした。まずは大規模なマイコンで実装に成功した後で，小規模なマイコンに搭載できるような軽量化を検討することで，実現性の検討を容易にするためである。

本稿では，バリエーションの幅が広い STM32 ファミリーの中から，評価ボード STM32F769I-DISCO を採用する。このボードは STM32 ファミリーでは上位のマイコンを搭載しており，マイコン内蔵メモリに加えて外部 Flash メモリと SDRAM を搭載しているため，組込み環境に最適化する前でも動作することが期待できる。選定したターゲットの詳細仕様を表 3 に示す。

表 3　実装ターゲットの仕様

項目	仕様	
ボード名	STM32F769I-DISCO	
マイコン	STM32F769NIH6	
CPU コア	Arm Cortex-M7　216 MHz	
内蔵 Flash メモリ	2 MB	ROM 合計 66 MB
外部 Flash メモリ	64 MB	
内蔵 RAM	512 kB	RAM 合計 16.5 MB
外部 SDRAM	16 MB	

2.4 移植実行

本稿で採用した TFLM はソースコードで提供されているため，ユーザが使用する環境に合わせてコンパイル・ビルドする必要がある。本稿の検討では，Cortex-M7 用の静的ライブラリとしてビルドし，アプリケーションに静的リンクして用いる。TFLM を利用する場合のソフトウェア構成を図 3 に示す。

TFLM ライブラリは，TFLite 形式モデルのインタープリタとして働き，モデルの記述を逐次翻訳して一階層ずつ計算を進める。

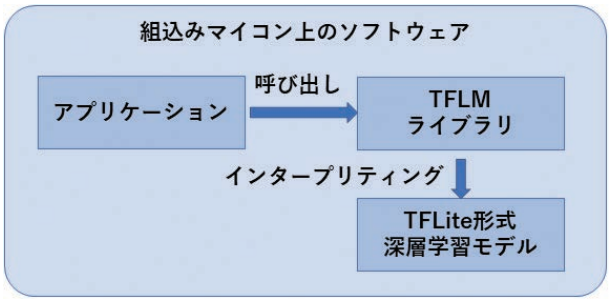


図 3　TFLM ライブラリ使用時のソフトウェア構成

3. モデル

深層学習アルゴリズムのモデルを表現する形式は，複数存在している。一つのプラットフォームが複数のモデル形式に対応していることが多く，またモデル形式を相互に変換するツールも存在している。

3.1 使用するモデル形式と変換ツールの選定

本稿で用いる深層学習モデルは，PyTorch 上で開発されたモデルである。そのモデルを TFLM 上で実行するため，モデル形式の変換が必要となる。

3.1.1 モデル形式

代表的な深層学習モデルを表 4 に示す。

表 4　代表的な深層学習モデル

プラットフォーム名	特徴
PyTorch Saved Model	PyTorch で開発したモデルの情報。各レイヤーの形式や重み情報は保存されているが，レイヤー間の接続は Python プログラム側に記述されるため，PyTorch 上でしか利用できない。
ONNX	Open Neural Network eXchange の略で，様々なプラットフォームで利用可能になっているオープンな形式。モデル構造と重み情報の両方が保存される。
TensorFlow	TensorFlow で開発したモデルの情報。ONNX と同様，モデル構造と重み情報の両方が保存される。
TFLite	TensorFlow のモデル情報を TFLite 命令に置き換え，Flatbuffers（Google が開発したシリアル化ライブラリ）を利用してサイズを圧縮した形式。

上記のモデル形式と，2.1，2.2 に示したプラットフォームとの対応を表 5 に示す。

表 5　プラットフォームとモデルの対応表

プラットフォーム ＼ モデル形式	PyTorch Saved Model	ONNX	TensorFlow	TFLite
PyTorch	○	○		
TensorFlow			○	
TFLite, TFLM				○
TVM		○	○	
Edge Impulse		○	○	○
STM32 X-CUBE-AI		○		○

3.1.2 変換ツールの選定

現在使用されている深層学習モデルのモデル形式を変換するツールは，複数存在する。モデル形式間の変換可否とそのツールを表 6 に示す。

表 6　モデル形式の相互変換表

変換元形式 ＼ 変換先形式	PyTorch Saved Model	ONNX	TensorFlow	TFLite
PyTorch Saved Model		(1)		
ONNX	(1)		(2) (4)	(2)
TensorFlow				(3)
TFLite				

（1）PyTorch で ONNX インポート／エクスポート可能
（2）オープンソースソフトウェア onnx2tf[1)]
（3）TensorFlow 公式配布の TensorFlow Lite Converter[2)]
（4）ONNX コミュニティ配布の onnx-tensorflow[3)]

本稿では，PyTorch で開発した深層学習モデルを TFLite 形式に変換する。この場合，以下の 2 つの経路が考えられる。

A）(1)→(2)：PyTorch にて ONNX 形式でエクスポートした後，onnx2tf で TFLite 形式に変換
B）(1)→(4)→(3)：PyTorch にて ONNX 形式でエクスポートした後，onnx-tensorflow で TensorFlow 形式に変換し，さらに TensorFlow Lite Converter で TFLite 形式に変換

このうち，B)の経路で使用する onnx-tensorflow は最終更新が 2022 年 11 月であり，本稿執筆時点（2024 年 11 月）でも 2 年近く更新が無い。つまり ONNX や TensorFlow の最新の機能を利用することができないという短所がある。

よって本稿では，A)の経路でモデルの変換を行うこととする。ただし onnx2tf は全ての ONNX 命令に対応しているわけではない。ONNX モデルに onnx2tf が未対応の命令が使用されている場合，エラーとなり変換できないため，何らかの対策が必要となる。この詳細は，次の 3.2 で説明する。

3.2 非対応命令への対策

3.1.2 で触れたように，ONNX の命令によっては onnx2tf が未対応であり，TFLite 形式に変換できない場合がある。

また，TFLite 形式に変換できたとしても，TFLM は全ての TFLite 命令に対応しているわけではない。変換した TFLite モデルに TFLM 未対応命令が含まれる場合，プログラム実行時にエラーとなる。

このような障害と，その対策案について概要を図 4 に示

し，詳細を説明する。図中の吹き出しに付された「3.2.1」等の番号は，本稿の項目番号と対応している。

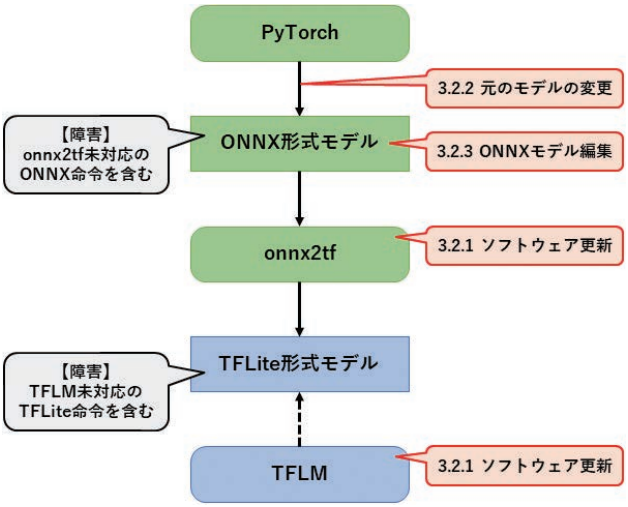


図 4　非対応命令による障害と対策

3.2.1 ソフトウェアの更新

onnx2tf も TFLM も，活発に開発が継続されているオープンソースソフトウェアである。あるバージョンでは未対応だった命令が，数か月後には動作可能になっていたり，別の単純な命令に変換されるようになっていたりすることがある。定期的にソフトウェアのアップデート情報を参照し，状況が変化していないかを確認する必要がある。

ソフトウェアアップデートによってモデル変換や動作が可能になっていれば，それが最善の対策となる。

3.2.2 変換前モデルの使用命令を変更

変換後のモデル形式では非対応な命令を使用している場合，変換前のモデルで使用する命令を別のものに変更することで，変換後のモデル形式でも対応している命令になるように改善することができる。

ニューラルネットワークの活性化関数 SELU を ReLU に置き換えた事例を示す。

ReLU（Rectified Linear Unit）は，深層学習の中間層の活性化関数として一般的に用いられている。深層学習の精度を向上させるための工夫として，ReLU の代わりに SELU（Scaled Exponential Linear Unit）が考案された。本稿で対象とするアルゴリズムも，SELU を用いて開発された。図 5 に各活性化関数の入出力を示す。

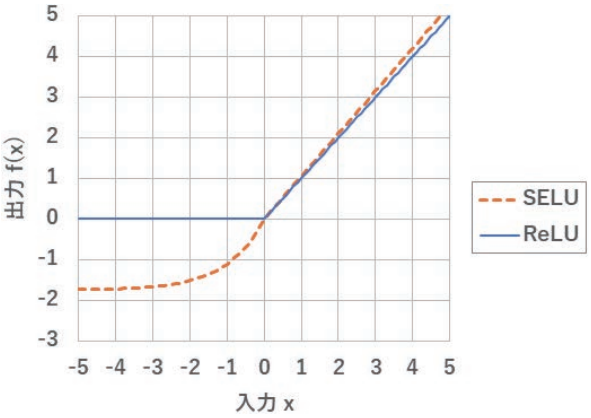


図 5　活性化関数 SELU と ReLU

図 5 に示した通り，SELU と ReLU の出力が大きく異なるのは入力 x が負の場合のみである。この差異により，深層学習モデルとしては SELU の方がわずかに高い精度を示す場合がある[4]。

しかし，TFLite は SELU に相当する命令を持たない。そのため，SELU 命令を用いた ONNX モデルを onnx2tf で変換する際，onnx2tf は複数の命令を組み合わせて SELU と同等の計算をするモデルを生成する（図 6 参照）。計算結果は等価になるものの，計算時間もメモリ使用量も非常に大きくなってしまう。

対策として，PyTorch でのアルゴリズム開発時点で活性化関数に SELU ではなく，TFLite も対応している ReLU を使用するよう変更する。深層学習アルゴリズムの精度はわずかに低下する可能性があるが，ONNX 形式を経由して TFLite 形式に変換した際も ReLU 命令のままであり，計算時間もメモリ使用量も大きく節約できる（図 7 参照）。

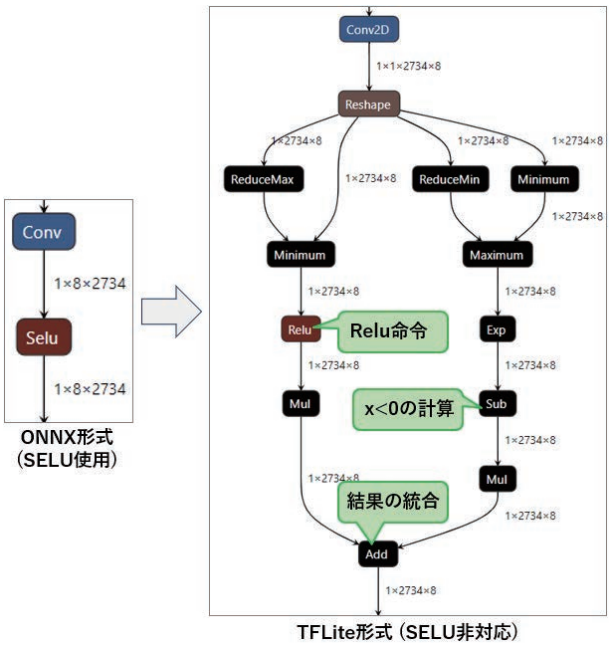


図 6　SELU 命令の TFLite 変換

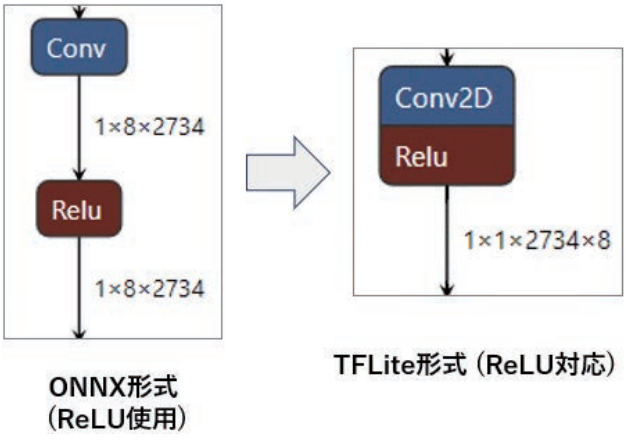


図 7　ReLU 命令の TFLite 変換

3.2.3 ONNX モデルの編集

PyTorch から出力された ONNX 形式モデルの中に，onnx2tf が未対応のため変換できない命令が含まれており，なおかつ PyTorch モデルの変更ではこの問題を解決できない場合がある。

この未対応命令を，動作に互換性があって onnx2tf が対応している別の ONNX 命令に置き換えることで，onnx2tf での変換が可能となる。

例として，PyTorch の MultiHeadAttention 命令を ONNX 形式で出力した際の冒頭部分のモデルを図 8 に示す。ONNX は MultiHeadAttention 命令に対応していないため，多数の命令を組み合わせたモデルが出力されるが，この中で用いられている ATen::unflatten 命令に onnx2tf は未対応であるため，TFLite 形式に変換できない。

ATen::unflatten 命令は，多次元配列に対して“dim”で指定された 1 つの次元を，“size”で指定された次元と大きさに変換する命令である。これは ONNX や TFLite で標準的な Reshape 命令でも同じ動作を実現できるため，図 9 のように Reshape 命令に置き換えれば，動作を保ったまま onnx2tf で変換できるようになる。

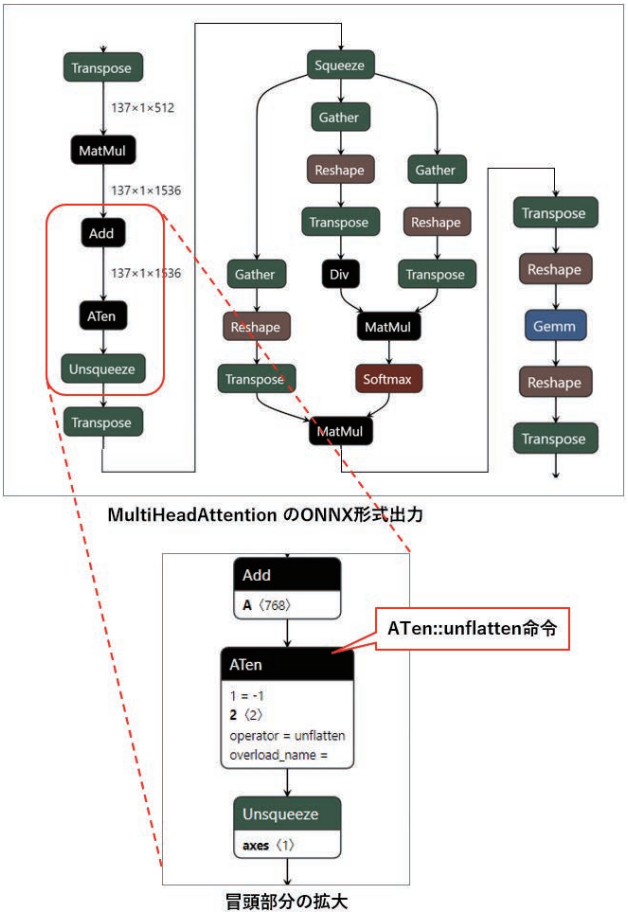


図 8　MultiHeadAttention の ONNX 形式出力

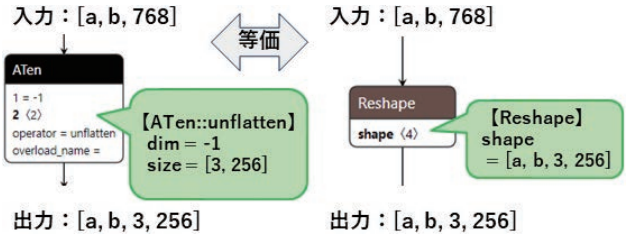


図 9　Aten::unflatten 命令と等価な Reshape 命令

ONNX モデルを編集するには，protocol buffer compiler[5]（以下 protoc）というソフトウェアを利用する。protoc を使って ONNX モデルをテキストファイルに変換し，テキストファイル上で命令を書き換えたうえで，再度 protoc を使って ONNX モデルに変換しなおす。手順を図 10 に示す。

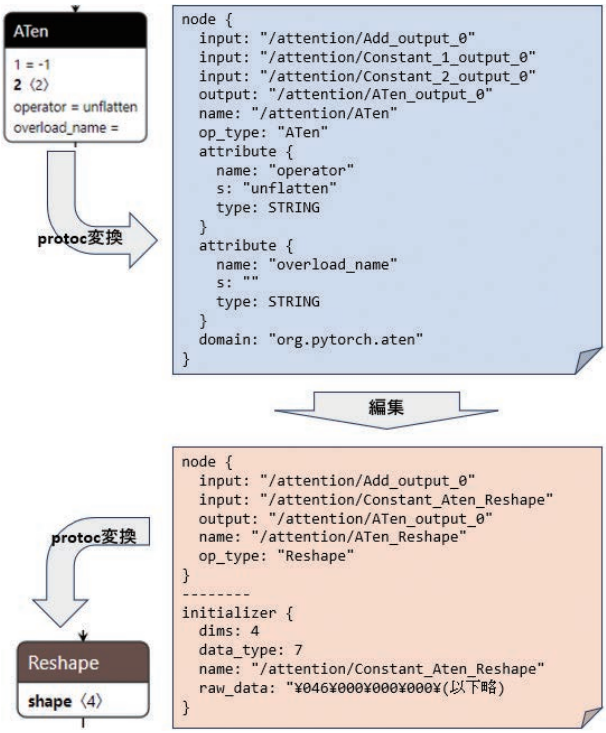


図 10　ONNX モデル編集フロー

3.3 軽量化

3.1，3.2 の工程を経て組込み環境に合わせたモデルが完成したが，組込み環境では ROM・RAM 容量に制約があるため，そのままでは動作しない場合がある。

ROM 容量とはプログラム実行時には値が固定された領域のことであり，プログラムコードやモデルのデータを格納する。特にモデルのデータ，その中でもニューラルネットワークのニューロンごとの重みデータが，ROM 容量の大部分を占めることになる。例えば全結合層（Linear, FullyConnected 等）では，「入力ノード数×出力ノード数」分の重みデータの容量が必要となる。

また RAM 容量とは，プログラム実行中の途中計算に用いる領域のことである。例えば畳み込み計算（Convolution）を使用する場合，「入力データ数×フィルタサイズ×チャネル数」分のデータを途中計算として RAM 上に保持する必要がある。ニューラルネットワークの一階層分ごとにまとめて計算するため，一階層の計算量が大きいモデルほど大量の RAM を必要とすることになる。

これらの ROM 必要量・RAM 必要量が，組込み環境で利用できる ROM 容量・RAM 容量に収まらない場合，モデルを軽量化する必要がある。以下にその手法を示す。

3.3.1 ハイパーパラメータチューニング

モデルの各階層の大きさなどを指定するパラメータを，「ハイパーパラメータ」と呼ぶ。ハイパーパラメータの調整は，深層学習アルゴリズムの開発の中でも重要な位置を占める。ハイパーパラメータの設定によりモデルのサイズは変化し，同時に精度も変化する。モデルのサイズを大きくしても必ずしも精度が向上するとは限らず，サイズと精度がバランスする組み合わせを探索する必要がある。

ハイパーパラメータのチューニングは，深層学習アルゴリズム開発環境で実施する。自動探索ツール Optuna[6] を用いて，各パラメータの候補の組合せを変えて再学習・評価を繰り返し，より精度の高くなる組み合わせを探索する。

例として，深層学習アルゴリズムの 10 個のハイパーパラメータに対し，それぞれ 3 ～ 5 通り程度の選択肢を与えて探索を行った。その探索結果において「ある組み合わせについて，よりサイズが小さく，かつ精度の高い結果は存在しない」という解が複数得られる。チューニングをモデルサイズと精度の最適化問題と捉えると，これらはパレート解と呼ばれる。実測値とパレート解を図 11 に示す。パレート解の中から，モデルサイズや精度の要件を満たすチューニング結果を選定する。

こうして得られたチューニング結果に対し，再度 3.1, 3.2 の工程を繰り返し，組込み環境に搭載可能なモデルを得る。

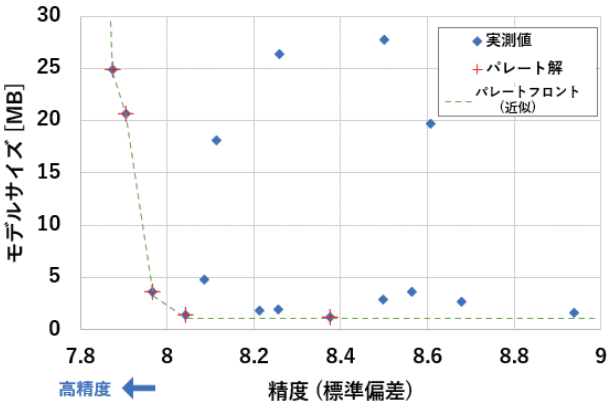


図 11　探索時のモデルサイズと精度

3.3.2 モデル量子化

深層学習モデルの軽量化において一般的な手法として，重みデータの量子化がある。量子化とは，浮動小数点数を固定小数点化し，低ビット数の整数型で表現することで計算量とメモリ使用量を削減することである。

深層学習モデルの量子化は，3.1 に挙げたモデルの変換時にオプションを指定することで実施する。入力データ群を与えることでニューラルネットワークの各層の数値範囲を自動的に計算し，その範囲にデータを収めて整数化するような命令が付与される。図 12 に概念図を示す。

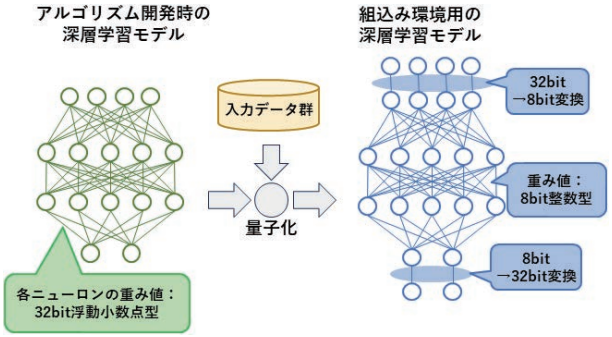


図 12　モデル変換時の量子化の概念図

32 bit→8 bit への量子化の場合，モデルサイズはおよそ 1/4 にまで縮小できる。この方法はアルゴリズムに影響を与えない一方で，アルゴリズムが量子化に最適化されないため，精度面に課題がある。

4. 評価

2 章，3 章で説明した深層学習モデルの組込み手法を，実際のセンサ商品を想定したアルゴリズムとハードウェアに対して実施し，手法の是非を評価する。

4.1 評価対象

想定商品として血圧計を例とする。血圧計は腕帯（カフ）を膨張させて血管を圧迫することで得られる「カフ圧」と，心収縮の際に生じる波動である「脈波」という 2 組の一次元の時系列データを入力とし，「最高血圧値（収縮期血圧値）」と「最低血圧値（拡張期血圧値）」という 2 つの数値を出力とするセンサである。加圧式の血圧計の場合，カフ圧と脈波それぞれの入力は図 13 に示すグラフのようになる。

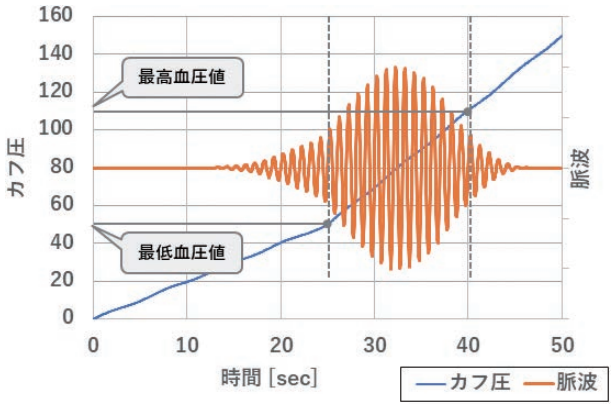


図 13　血圧計の入力と出力

血圧計は一般的に，PC 等に比べて製品価格を抑える必要があるため，PC の CPU より何桁も低容量・低速のマイコンで制御されている。そのため PC 上で開発した深層学習アルゴリズムをそのまま載せるのは困難であるというハードウェア的制約がある。2.3 で選定した実装ターゲットのハードウェア仕様を抜粋し，表 7 に再掲する。

表 7　評価環境のハードウェア仕様

項目	仕様	備考
CPU コア	Cortex-M7	FPU 内蔵
クロック周波数	216 MHz	
Flash ROM 容量	66 MB	内蔵 2 MB，外付 64 MB
RAM 容量	16.5 MB	内蔵 532 kB，外付 16 MB

評価対象のモデルは以下の 3 種を用意する。

ⅰ．サイズ無視モデル
　アルゴリズム開発環境で，精度を最重要視して開発したモデル。3.3.1 のハイパーパラメータチューニング結果にて，最も精度の高かったモデルを選定する。

ⅱ．サイズと精度のバランスモデル
　3.3.1 のチューニング結果から，サイズと精度のバランスを取って選定したモデル。
　4.2 に示した評価基準のうちの「誤差の標準偏差」が基準を満たす範囲で，最もサイズが小さいモデルを選定する。

ⅲ．量子化モデル
　上記のバランスモデルに対し，3.3.2 の手法にて量子化を行ったモデル。

上記の各モデルと，パラメータチューニング結果との対応を図 14 に示す。

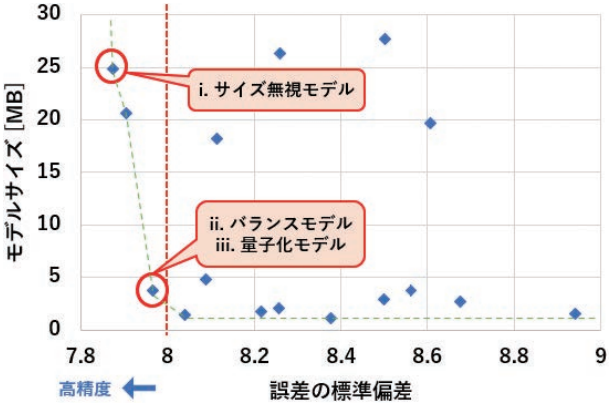


図 14　評価対象モデルとチューニング結果の対応

4.2 評価基準

4.1 で示したⅰ，ⅱ，ⅲ，の各モデルに対して，以下の指標に対して評価を行う。

・MAE（平均絶対誤差）
　精度の指標の一つ。血圧計の国際規格[7] によれば，5 未満が求められる。
・SDE（誤差の標準偏差）
　精度の指標の一つ。血圧計の国際規格[7] によれば，8 未満が求められる。
・実行時間
　測定データから血圧値を算出するまでの時間。市販の血圧計において，測定終了から血圧値表示までの時間が 3 秒程度であったので，3 秒未満で実行できることが望ましい。
・ROM 容量
・RAM 容量

4.3 評価方法と評価結果

テスト用の脈波とカフ圧のデータ約 500 組を入力して深層学習モデルにより推定を行い，最高血圧値において出力値と真値の誤差を算出した。その模式図を図 15 に示す。

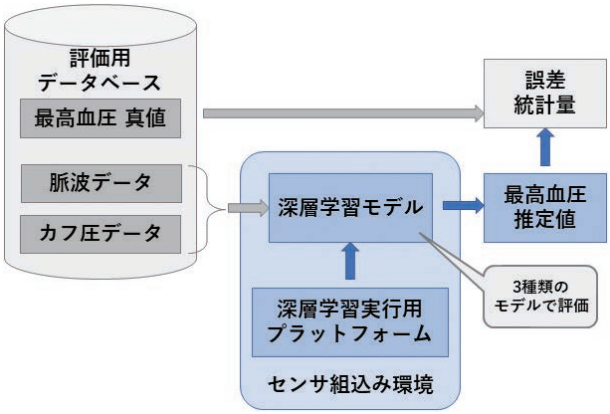


図 15　評価方法

3 種類の評価対象に対して実行した結果を表 8 に示す。

表 8　評価結果

指標	ハイパーパラメータチューニングモデル		iii. 量子化モデル
	i. サイズ無視	ii. バランス	
精度（MAE）	6.66	6.74	8.16
精度（SDE）	7.83	7.97	10.71
実行時間［ms］	12,214	1,938	348
ROM 容量［MB］	18.37	3.72	0.98
RAM 容量［kB］	663.4	221.0	63.7

・サイズ無視モデルでは，精度の評価はアルゴ開発環境で実施した。ROM 容量・RAM 容量は大きいものの今回のターゲットに実装できる範囲内だったため，ターゲット上でも実行可能であった。しかし，時間が 12 秒超と血圧計として実用に耐えない速度であった。
・サイズと精度のバランスを取ったモデルでは，ROM 容量は 2 MB を超えるため外付け Flash ROM が必要となるが，RAM 容量は 221 kB でマイコン内蔵 RAM だけで実行できるサイズとなった。また，実行時間も 2 秒以内と現行の市販品より短く，許容可能な速度であった。
・量子化モデルでは，ROM 容量 0.98 MB，RAM 容量 63.7 kB でどちらもマイコン内蔵メモリの範囲内であるとともに，実行時間も 348 ms（0.348 秒）と実用的な速度で実行できた。精度については大きく劣化し，血圧計の国際規格 SDE＜8 を満たさなくなっている。
・すべてのモデルで，平均絶対誤差の指標 MAE＜5 を満たしていない。本稿の目的は従来技術からの精度向上ではなく，深層学習技術が組込み可能であることを示すことであるため，この指標を満たさなくても本稿の目的は達せられる。

以上より，深層学習アルゴリズムを，血圧計を想定した組込み環境で動作させることを実現できた。また，量子化を行うことで実行時間と ROM・RAM 容量を大きく削減できることが分かった。

5. むすび

5.1 本稿での到達点

本稿では，深層学習アルゴリズムを組込み環境で実行するための手順を確立した。

深層学習アルゴリズム開発環境である PyTorch で開発された深層学習モデルを，組込み用の深層学習アルゴリズムの実行環境である TensorFlow Lite for Microcontrollers 向けに変換し，変換時に発生する幾つかの問題に対して具体的な解決手段を示した。また，組込み環境の要求であるメモリや実行時間の制約に応えられるよう，モデルの軽量化の手段についても，ハイパーパラメータチューニングとモデル量子化の 2 つの手段を示した。最後に，マイコン上で軽量化したモデルを実行し，組込み環境での深層学習アルゴリズム動作の実現性があることを示した。

本稿の 3 章で示した手順は，様々な開発に適応可能である。本稿では血圧計を事例として実装検討を行い，一定の実現性があることを検証した。血圧計に限らず，一次元のデータ列に対して信号処理を行うようなセンサに対しても，同様に深層学習アルゴリズムを適用できる可能性がある。候補として，例えば光や温度など物理量を計測するセンサが考えられる。

5.2 今後の課題

ハイパーパラメータチューニングでは，精度をある程度保つことはできたものの，マイコン内蔵メモリの範囲内に収めるには至らず，改善の余地がある。

量子化では，本稿の手法では精度を大きく落とした。量子化しつつ精度を保つ手段として，例えば開発環境でモデルに学習させる時点から量子化した状態で進め，量子化した状態で精度を保つように開発するという手法がある。今後の実用化に向け，サイズ削減と精度の両立に向けて検討を継続する。

参考文献

1）Katsuya Hyodo. “onnx2tf.” GitHub. https://github.com/PINTO0309/onnx2tf (Accessed: Nov. 1, 2024).

2）Google. “TensorFlow Lite コンバータ.” TensorFlow. https://www.tensorflow.org/lite/convert?hl=ja（Accessed: Nov.1, 2024）.

3）Open Neural Network Exchange. “TensorFlow Backend for ONNX.” GitHub. https://github.com/onnx/onnx-tensorflow（Accessed: Nov. 1, 2024）.

4）Dabal Pedamonti. “Comparison of non-linear activation functions for deep neural networks on MNIST classification task.” arXiv. https://arxiv.org/abs/1804.02763（Accessed: Apr. 8, 2018).

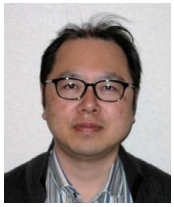
5）Google. “Protocol Buffers - Google’s data interchange format.” GitHub. https://github.com/protocolbuffers/protobuf （Accessed: Nov. 5, 2024）.

6）T. Akiba et al., “Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, 2019, pp. 2623–2631.

7）*Non-invasive sphygmomanometers - Part 2: Clinical investigation of intermittent automated measurement type*, ISO 81060–2, 2018.

執筆者紹介

小河原 徹　KOGAWARA Toru
技術・知財本部
アドバンストテクノロジーセンタ
アドバンストテクノロジー開発部
専門：ソフトウェア科学
所属学会：システム制御情報学会



渡辺 泰久　WATANABE Yasuhisa
技術・知財本部
アドバンストテクノロジーセンタ
専門：ソフトウェア科学
所属学会：電子情報通信学会，システム制御情報学会，計測自動制御学会