Vol.2020-OS-148 No.4
2020/2/27

組込み向けディープラーニングフレームワークの評価

平森将裕[†1]　出口昌弘[†1]　大木英俊[†1]　水口武尚[†1]

概要：ディープラーニングフレームワークは，ディープラーニングにおけるモデル構築，学習，推論の実現に必要な構成要素を再利用可能な形で提供することで，ディープラーニングの実装を容易化するソフトウェアフレームワークであり，組込み向けディープラーニングフレームワークとしては TensoFlow Lite や ONNX Runtime が開発・提供されている．ディープラーニングフレームワークを活用して推論処理を組込み機器上に実装する際の，ディープラーニングフレームワーク選定のための指標を得ることを目的に，本研究では，これらの組込み向けディープラーニングフレームワークのリソース使用量を測定し，評価・比較した．結果，組込み機器の用途として考えられる下記 2 つに対して，以下の指標が得られた．(A) 電源投入後，最初の 1 回目の推論終了までの時間が重要な用途については，フレームワーク読み込み，モデル読み込み，1 回目の推論時間が重要，(B) 起動時間が長くとも繰り返す推論処理の処理時間が短いことが求められる用途については，2 回目以降の推論時間が重要であることが分かった．この指標をもとに TensorFlow Lite と ONNX Runtime を比較すると，(A)，(B)の両方の用途で TensorFlow Lite が有利であることが分かった. また, TensoFlow Lite の 1 回目の推論時間は, 2 回目以降と比較し 1.1 倍から 2.0 倍の時間である一方, ONNX Runtime は大きな差がないため，この原因を調査したところ，TensorFlow Lite では畳み込み演算実行前にフィルタの転置処理を実行していることが原因であることが判明した．一方 ONNX Runtime では，Conv 演算に用いるフィルタはあらかじめ転置された状態でモデルファイルに保存しているため，推論時に転置処理を呼出す必要がなく，1 回目の推論時間と 2 回目以降の推論時間に大きな差がなかったと考えられる．繰り返し行う推論処理の時間変化を抑える必要がある用途の場合，Conv 演算に使用するフィルタを転置した状態でモデルファイルに保存するように変更することで，TensorFlow Lite における 1 回目の推論を 2 回目以降と同等の処理時間に短縮可能である見込みを得た．

キーワード：ディープラーニング，　TensorFlow Lite, ONNX Runtime，Raspberry Pi, Benchmark

1. はじめに

ディープラーニングにおけるモデル構築，学習，推論の実現には，自動微分，多次元配列，活性化関数といった構成要素が必要となる．ディープラーニングフレームワークは, これらの構成要素を再利用可能な形で提供することで, ディープラーニングの実装を容易化するソフトウェアである．代表的なディープラーニングフレームワークとして，Google の TensorFlow[1]や Facebook の PyTorch[2]が OSS として開発・公開されている．

ディープラーニングを用いたアプリケーションを開発する場合，組込み機器で取得した情報を用いてクラウドや PC でディープラーニングの推論処理を実行すると, 処理のレイテンシが大きい，ネットワーク通信が必須といった制約がある．そのため，組込み機器上でディープラーニングの推論処理を実行したいという要求が高まっている．

しかし上記のディープラーニングフレームワークは，クラウドや PC 上での動作を前提としているため，リソース制限の厳しい組込み機器上で実行するには処理時間とメモリ使用量が課題となる．そのため，ディープラーニングの推論処理のみを実行可能な組込み向けディープラーニングフレームワークが開発されており, 代表的なものとして，Google の TensorFlow Lite[3] や Microsoft の ONNX Runtime[4]が OSS として開発・公開されている．

そこ本研究では，推論処理を組込み機器上に実装することを想定し、組込み向けディープラーニングフレームワークである TensorFlow Lite と ONNX Runtime の処理時間とメモリ使用量を測定し，比較・評価する．

2 章で関連技術として TensorFlow Lite と ONNX Runtime について解説し，3 章で調査に用いたモデルと組込み機器のハードウェア・ソフトウェア環境を示す．4 章で結果を示し，5 章で考察を述べる．

2. 関連技術

2.1 TensorFlow Lite

TensorFlow Lite は, Google が開発する TensorFlow の組込み向けディープラーニングフレームワークである．TensorFlow で学習したモデルを TensorFlow Lite 形式に変換し，TensorFlow Lite 上で変換後のモデルを読み込み実行する．CPU アーキテクチャは x86-64，ARM32v7，ARM64 に対応している．C++で実装されており，既存の C++プログラムとの統合が可能である. Python や Java とのインタフェースも用意されているため, 手軽に使うことが可能である．また，マルチスレッド処理と GPU に対応しているため，高速な処理が可能である．

2.2 ONNX Runtime

Microsoft が開発する ONNX(Open Neural Network Exchange)[5]フォーマット実行エンジンである. ONNX は，ディープラーニングのモデルを表すために使用するフォーマットであり，Microsoft，Facebook，AWS などが中心となって開発している．ONNX Runtime は，ONNX 形式のモデルを読み込み実行する．CPU アーキテクチャは x86_32，x86-64，ARM32v7，ARM64 に対応している．C++で実装さ

†1 三菱電機株式会社　情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

れており，既存の C++プログラムとの統合が可能である．Python，Java，C#，Ruby とのインタフェースも用意されているため，手軽に使うことが可能である．また，マルチスレッド処理と GPU に対応しているため，高速な処理が可能である．

3. 評価方法

本研究では，3 つの異なる規模のモデルにおけるフレームワーク読み込み時間，モデル読み込み時間，推論時間，メモリ使用量を比較し，TensorFlow Lite と ONNX Runtime を評価した．評価に用いたモデルを表 1 に示す．

表 1 評価に用いたモデル

項目	Resnet50 v1.5	Mobilenet_v1 1.0_224	SqueezeNet v1.1
対象タスク	画像分類	画像分類	画像分類
発表元	Microsoft Research	Google	DeepScale
発表年	2015	2017	2016
学習済モデルフォーマット上のファイルサイズ(MB)	98	17	4.8
パラメータ数	25.6M	4.24M	1.24M
積和演算回数	3870M	569M	352M

3.1 評価環境

評価には Raspberry Pi 3 を用いた．Raspberry Pi 3 のハードウェア・ソフトウェア環境を表 2 に示す．

表 2 Raspberry Pi 3 のハードウェア・ソフトウェア環境

項目	Raspberry Pi 3
CPU	Broadcom BCM2837
アーキテクチャ	ARM Cortex-A53
命令セット	ARMv7l
コア数	4
周波数	1.2 GHz
主記憶	1GB
ストレージ	Class10 16GB microSDHC
OS	Raspbian Buster
Linux カーネル	4.19.75-v7
Python	3.5.9
NumPy	1.18.1

本研究で使用した TensorFlow Lite と ONNX Runtime のバージョン，コミット ID を表 3 に示す．

表 3 評価対象ディープラーニングフレームワーク

項目	TensorFlow Lite	ONNX Runtime
バージョン	2.0.0	1.1.0
コミット ID	64c3d38	b0019ac

3.2 測定方法

ディープラーニングの推論処理は，フレームワーク読み込み，学習済みモデル読み込み，入力データ取得，推論の 5 つの処理から構成される．

処理時間を測定するために，評価用プログラムを開発した．処理概要を図 1 に示す．

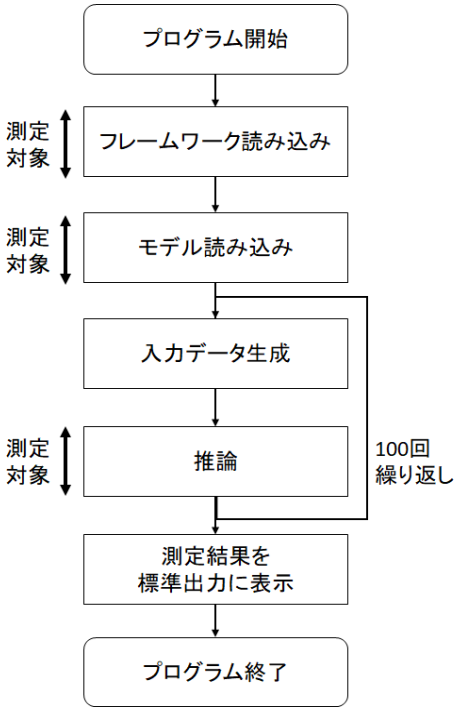

図 1 評価用プログラムの処理概要

評価用プログラムを開始すると，TensorFlow Lite もしくは ONNX Runtime のフレームワークを読み込む．次に，評価に用いるモデルファイルを読み込む．入力データ生成処理と推論処理を 100 回繰り返し，処理時間測定結果を標準出力に表示後，プログラムを終了する．

本研究では処理時間とメモリ使用量への影響について評価することが目的であるが，評価に用いた 3 つのモデルは推論では入力データの値による処理時間とメモリ使用量に大きな影響はない．このため，入力データは画像ではなく NumPy[8]で生成した乱数行列を用いた．この入力データ生成処理は測定対象に含まない．

評価用プログラムは，ディープラーニング学習環境との連携を考え，組込みシステム開発で一般的な C/C++ではなく Python で開発した．

評価用プログラムは 10 回実行し，最大最小を除く 8 回の平均を算出した. 実行時に nice コマンド[7]を用いて優先度を-20(-20~19 で指定可能，値が小さいほど優先度高)に設定し，優先度を高くして実行した.

Linux カーネルの governor[6]を ondemand で評価用プログラムを実行したところ，CPU 温度の上昇によりサーマルスロットリング機能が動作し，CPU 動作周波数が 700MHz 付近へ低下する現象が発生した. この CPU 動作周波数の低下を回避するために，governer を ondemand から powersave へ変更し，CPU 動作周波数を 600MHz へ固定し実行した.

3.3 フレームワーク読み込み

フレームワーク読み込みは，Python の import 文で行う. Python の time モジュール[9]を用いてフレームワーク読み込み処理の前後で時刻を取得し，差分を計算することで処理時間を測定する.

3.4 モデル読み込み

モデル読み込み処理の呼出しには，TensorFlow Lite では tflite_runtime.interpreter() 関数，ONNX Runtime では onnxruntime.InferenceSession()関数を使用する．Python の time モジュール[9]を用いてモデル読み込み処理の前後で時刻を取得し, 差分を計算することで処理時間を測定する.

3.5 推論

推論処理の呼出しには，TensorFlow Lite では interpreter.Invoke()関数，ONNX Runtime では session.run()関数を使用する. Python の time モジュールを用いて推論処理の前後で時刻を取得し，差分を計算することで処理時間を測定する．処理時間の変化の調査には，100 回推論処理を実行し，推論時間の推移を測定する.

3.6 メモリ使用量

GNU time コマンド[10]を用いて，評価用プログラムの実行時の Max RSS を計測した.

4. 結果

表 4 に Resnet50 v1.5 の測定結果，表 5 に Mobilenet_v1 1.0_224 の測定結果，表 6 に SqueezeNet v1.1 の測定結果をそれぞれ示す.

100 回の推論処理の処理時間の推移をグラフにプロットした. 図 2 に Resnet50 v1.5, 図 3 に Mobilenet_v1 1.0_224, 図 4 に SqueezeNet v1.1 の推移を示す.

処理時間の評価にあたり、組込み機器の用途として、重視される時間には次の 2 つのタイプがあると考えた.

(A) 電源投入後，最初の 1 回目の推論終了までの時間が重要な用途

(B) 起動時間が長くとも，繰り返す推論処理の処理時間が短いことが求められる用途

(A)の用途に対しては, ディープラーニングフレームワーク読み込みから 1 回目の推論終了までの時間が重要，(B)の用途に対しては，2 回目以降の推論時間が重要である.

(A)の用途に対する比較のために, フレームワーク読み込み，モデル読み込み，1 回目の推論時間の合計値をモデルごとにプロットした図を，図 5 に Resnet50 v1.5，図 6 に Mobilenet_v1 1.0_224，図 7 に SqueezeNet v1.1 をそれぞれ示す.

図 5 から図 7 を見ると，TensorFlow Lite は ONNX Runtime に比べて最初の 1 回目の推論終了までの時間が，Resnet50 v1.5 で 9%，Mobilenet_v1 1.0_224 で 13%，SqueezeNet v1.1 で 32%短く，(A)の用途に対しては TensorFLow Lite が有利であることが分かった.

表 4 から表 6 に示す 2 回目以降の平均推論時間を比較すると，Resnet50 v1.5 は TensorFlow Lite が 17%短い，Mobilenet_v1 1.0_224 は TensorFlow Lite が 24%短い，SqueezeNet v1.1 は同等であり，(B)の用途に対しては TensorFlow Lite が有利であることが分かった.

上記の結果より，(A)，(B)のどちらの用途に対しても TensorFlow Lite が有利であることが分かった．TensorFlow Lite の 1 回目と 2 回目以降の平均推論時間の差を比較すると，最小で 1.1 倍，最大で 2 倍と，大きな差があることが分かり, この差を解消することで(A)の用途に対する改善が可能であると考える．一方 ONNX Runtime は 1 回目と 2 回目以降の平均推論時間に大きな差はなく，この原因について調査した結果は考察で述べる.

表 4 から表 6 の最大メモリ使用量を比較すると，すべてのモデルにおいて TensorFlow Lite と ONNX Runtime で同等のメモリ使用量であることが分かった．これは，メモリ上に展開するモデルの重みパラメータが支配的なためと思われる.

表 4 Resnet50 v1.5 の測定結果

項目	TensorFlow Lite	ONNX Runtime
フレームワーク読み込み時間(ms)	31	157
モデル読み込み時間(ms)	406	3118
推論時間	-	-
1 回目(ms)	5596	3361
2 回目以降の平均(ms)	2788	3337
最大メモリ使用量(MB)	232.4	232.1

表 5　Mobilenet_v1 1.0_224 の測定結果

項目	TensorFlow Lite	ONNX Runtime
フレームワーク読み込み時間(ms)	31	157
モデル読み込み時間(ms)	71	334
推論時間	-	-
1 回目(ms)	1006	787
2 回目以降の平均(ms)	585	763
最大メモリ使用量(MB)	66.3	62.1

表 6　SqueezeNet v1.1 の測定結果

項目	TensorFlow Lite	ONNX Runtime
フレームワーク読み込み時間(ms)	31	158
モデル読み込み時間(ms)	24	172
推論時間	-	-
1 回目(ms)	419	362
2 回目以降の平均(ms)	367	362
最大メモリ使用量(MB)	45.1	42.8

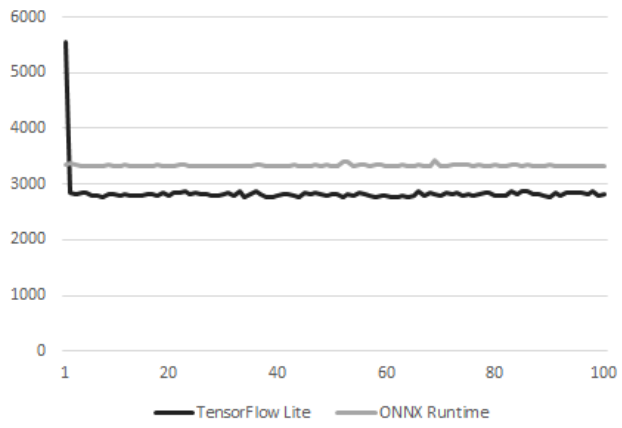

図 2　Resnet50 v1.5 における 100 回の推論時間(ms)の推移

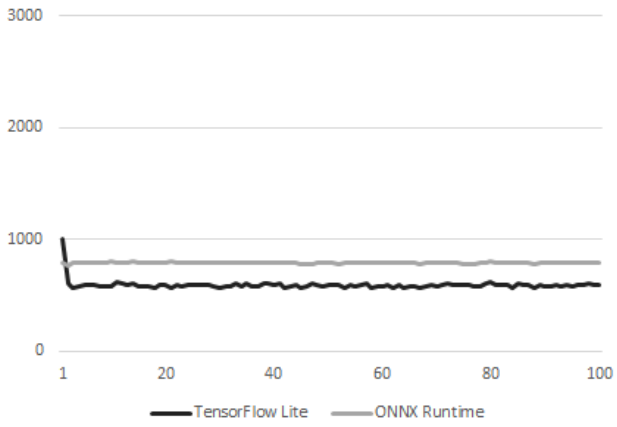

図 3　Mobilenet_v1 1.0_224 における 100 回の推論時間(ms)の推移

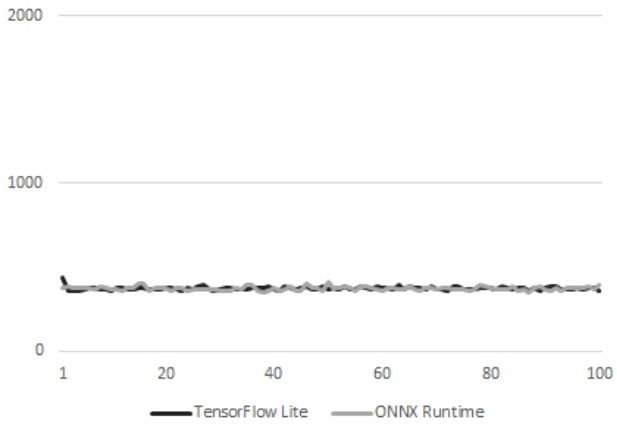

図 4　SqueezeNet v1.1 における 100 回の推論時間(ms)の推移

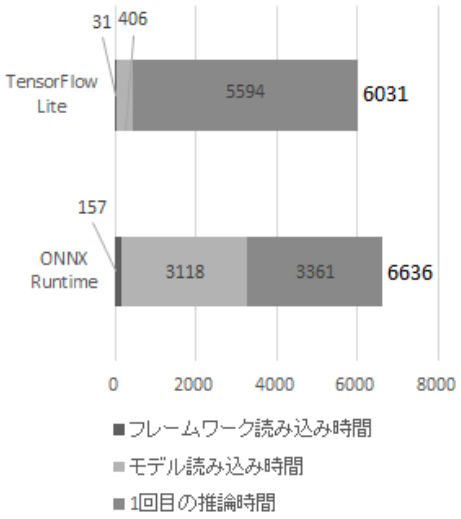

図 5　Resnet50 v1.5 におけるフレームワーク読み込みからモデル読み込み，1 回目の推論時間

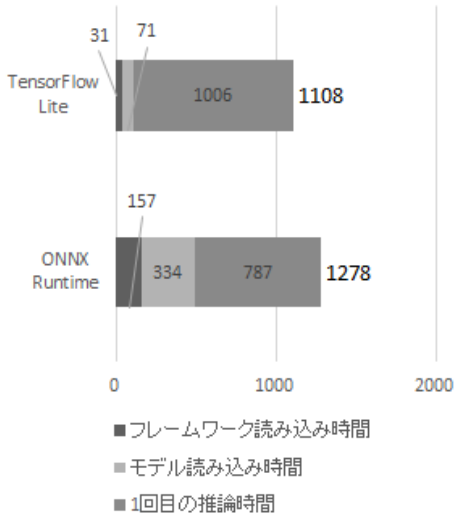

図 6　Mobilenet_v1 1.0_224 におけるフレームワーク読み込みからモデル読み込み，1 回目の推論時間

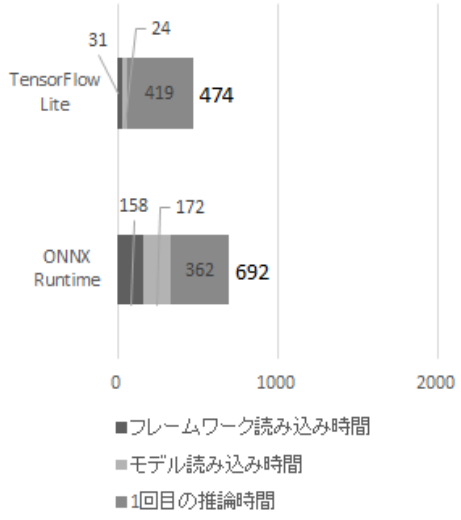

図 7　SqueezeNet v1.1 におけるフレームワーク読み込みからモデル読み込み，1 回目の推論時間

5.　考察

本章では，4 章で得られた結果における，TensorFlow Lite の 1 回目と 2 回目以降の推論時間の差について考察する．

モデルによらず 1 回目の推論時間が 2 回目以降より長いため, 1 回目と 2 回目の推論時間の差が最も大きい Resnet50 v1.5 を対象に調査する．

TensorFlow Lite の推論処理で実行する各種演算の実装は，TensorFlow ソースコード[11]ディレクトリの tensorflow/tensorflow/lite/kernels 以下にある．CPU 向けの実装はアーキテクチャによらず共通である．

TensorFlow Lite における Resnet50 v1.5 の推論時間を PC 上で測定したところ，Raspberry Pi 3 の結果と同様に，1 回目と 2 回目の推論時間に差があることが分かった．そのため, 推論時間の差は実行する CPU によらないものであると推測する．そこで，調査効率化のため，以下では PC 上で推論時間を測定した．使用した PC のハードウェア・ソフトウェア環境を表 7 に示す．

表 7　PC のハードウェア・ソフトウェア環境

項目	PC
CPU	Intel Core i9-7900X CPU @ 3.30GHz
アーキテクチャ	Skylake
命令セット	x86, x64
コア数	10
周波数	3.3 GHz
主記憶	64GB
ストレージ	512GB M.2 SSD
OS	Ubuntu 18.04 LTS
Linux カーネル	5.3.0-26
Python	3.5.9
NumPy	1.18.1

1 回目と 2 回目の推論時間を，Resnet50 v1.5 に含まれる演算種類ごとに測定した．結果を表 8 に示す．

表 8　Resnet50 v1.5 に含まれる演算種類ごとの推論時間測定結果(PC, TensorFlow Lite)

演算種類	1 回目	2 回目	差
Pad	0.6	0.3	0.2
Conv	195.4	36.2	159.2
Add	5.2	3.3	1.9
Mean	0.1	0.1	0.0
FullyConnected	1.1	0.6	0.5
Softmax	0.0	0.0	0.0
MaxPool	0.7	0.4	0.2

結果，Conv 演算(畳み込み演算)の処理時間の差が最も大きいことが分かった. Conv 演算の実装[12]を確認した結果, 1 回目の推論実行時にリスト 1 の下線で示すフィルタの転置処理を実行していることが分かった．この処理は Conv 演算実行前に呼出され，結果をメモリに保存する．2 回目以降の Conv 演算実行時には，この保存した転置後のフィルタを用いる．そのためフィルタの転置処理を呼出すタイミングは, 1 回目の Conv 演算実行前であればいずれのタイミングで実行しても問題ない呼出し処理である．

一方 ONNX Runtime の場合, Conv 演算に用いるフィルタはあらかじめ転置された状態でモデルファイルに保存されている．そのため，推論時に転置処理を実行する必要がなく，1 回目の推論と 2 回目以降の推論時間の平均で大きな

差がなかったと考えられる.

```
if (data->need_hwcn_weights &&
   !data->have_weights_been_transposed) {
    TransposeFloatTensor(filter, hwcn_weights);
    data->have_weights_been_transposed = true;
}
```

リスト 1　Conv 演算内のフィルタ転置処理呼出し

繰り返し行う推論処理の時間の変化を抑える必要がある用途の場合, Conv 演算に使用するフィルタを転置した状態でモデルファイルに保存するように変更することで，1 回目の推論を 2 回目以降と同等の処理時間に短縮可能である.

6. おわりに

本研究では，組込み向けディープラーニングフレームワークである TensorFlow Lite と ONNX Runtime の推論処理における処理時間とメモリ使用量を測定し，評価・比較を行った．評価は推論処理におけるモデル読み込み時間，推論時間について，3 つの異なるサイズのモデルを対象とした.

結果，組込み機器の用途として考えられる下記 2 つに対して，以下の指標が得られた.

(A) 電源投入後，最初の 1 回目の推論終了までの時間が重要な用途

→フレームワーク読み込み，モデル読み込み，1 回目の推論時間が重要

(B) 起動時間が長くとも，繰り返す推論処理の処理時間が短いことが求められる用途

→2 回目以降の推論時間が重要

上記の指標で TensorFlow Lite と ONNX Runtime を比較すると，(A)，(B)の両方の用途で TensorFlow Lite が有利であることが分かった.

また，TensorFlow Lite の 1 回目の推論時間は 2 回目以降と比べての 1.1 倍から 2 倍の時間である一方，ONNX Runtime では大きな差がないため，この原因を調査した．結果，TensorFlow Lite では 1 回目の Conv 演算実行時にフィルタを転置し，結果をメモリに保存，2 回目以降の Conv 演算実行時には保存した転置後のフィルタを用いることが分かった．一方 ONNX Runtime の場合，Conv 演算に用いるフィルタはあらかじめ転置された状態でモデルファイルに保存されていることが分かった．そのため，推論時に転置処理を呼出す必要がなく，1 回目の推論と 2 回目以降の推論時間の平均で大きな差がなかったと考えられる.

繰り返し行う推論処理の時間変化を抑える必要がある用途の場合, Conv 演算に使用するフィルタを転置した状態でモデルファイルに保存するように変更することで，TensorFlow Lite における 1 回目の推論を 2 回目以降と同等の処理時間に短縮可能である見込みを得た.

7. 参考文献

[1] TensorFlow: https://www.tensorflow.org/
[2] PyTorch: https://pytorch.org/
[3] TensorFlow Lite: https://www.tensorflow.org/lite
[4] ONNX Runtime: https://microsoft.github.io/onnxruntime/
[5] ONNX: https://onnx.ai/
[6] CPU frequency scaling: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
[7] nice: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/nice.html
[8] NumPy: https://numpy.org/
[9] time: https://docs.python.org/3.5/library/time.html
[10] GNU Time: https://www.gnu.org/software/time/
[11] tensorflow/tensorflow at v2.0.0: https://github.com/tensorflow/tensorflow/tree/v2.0.0
[12] tensorflow/conv.cc at v2.0.0 · tensorflow/tensorflow: https://github.com/tensorflow/tensorflow/blob/v2.0.0/tensorflow/lite/kernels/conv.cc#L667-L718