

低コスト・低消費電力での深層学習アプリケーションの実現

村松 沙那恵^{1,a)} 山田 佑二^{1,b)} 江田 毅晴^{1,c)}

概要：昨今の AI ブームにより、深層学習の適用範囲が拡大されてきているが、計算リソースの価格や消費電力が問題となっている。深層学習の推論に特化したデバイスを使うと、それらの問題を解決する可能性があるが、それらのデバイスのスケーラビリティなどは不明である。そこで本研究では、安価で省電力な Neural Compute Stick(NCS) を利用し、NCS のスケーラビリティを確認するための性能評価を行った。NCS を 14 本接続し推論させたところ、NCS の本数に応じて線形に性能が向上し、最大 107fps で領域検出できることがわかった。また、1 本あたりに実行させるグラフ数を変化させたところ、2 つのグラフまでは性能が向上することがわかった。この結果をもとに、深層学習アプリケーションを作成し性能評価したところ、NCS の本数に応じて性能が向上することを確認することができた。

1. はじめに

昨今の AI ブームにより、深層学習の適用範囲が拡大されてきているが、深層学習の演算を高速に行うためには、高価な GPU が必要となっている。一方で、深層学習をエッジで推論する場合には、安価で省電力なデバイスで推論したいというニーズがあり、推論に特化したデバイスも開発されてきている。

深層学習の推論に特化したデバイスは、DeePhi DPU[2]、edge-TPU(2018 年秋発売予定)[3]、Neural Compute Stick(NCS)[1] など様々あるが、本研究では安価で省電力な USB 型の NCS に着目した。

NCS の単体性能は知られている [4] が、スケーラビリティなどの基礎評価や設計指針がない。そのため、深層学習アプリケーションに NCS を利用する方法が不明である。

そこで本研究では、NCS の本数や、1 本あたりに実行させるグラフ数を変化させた基礎検証を行い、その結果をもとに深層学習アプリケーションを作成し性能評価を行った。

2. Neural Compute Stick

NCS には、低消費電力でハイパフォーマンスの Movidius Vision Processing Unit (Myriad2 VPU) というプロセッサが搭載されている。Myriad2 は、画像認識に重きを置いたプロセッサで、SHAVE (Streaming Hybrid Architecture Vector Engine) と呼ぶ 128 ビットの SIMD 構成の VLIW

プロセッサを 12 基持っている。Myriad2 は約 1W の消費電力にも関わらず、100G FLOPS (Floating-point Operations Per Second) の性能を持ち、省電力かつ高速な深層学習の推論処理を行うことができる。

NCS 上で深層学習の推論を行うためには、深層学習モデルを専用の開発キット (SDK) でコンパイルする必要がある。また、NCS を操作するための NCS API も提供されており、モデルやデータの転送や実行などは全て API 経由で行う。

3. 基礎性能検証

本検証では、図 1 のように、NCS の本数を増やしたマルチスティック検証と、1 つの NCS に同じモデルを複数実行させるマルチグラフ検証の 2 つの基礎検証を行う。それぞれ処理性能がどのように変化するかを確認する。

利用するモデルは、領域検出を行う TinyYOLO[5] を利用する。全フレームをメモリ上に読み込んだ状態から全ての処理が終わるまでの経過時間を計測し、全フレーム数で除した fps(frame per second) を性能指標とする。



図 1 検証環境

¹ 日本電信電話株式会社
180-8585, 東京都武蔵野市緑町 3-9-11

a) muramatsu.sanae@lab.ntt.co.jp

b) yamada.yuji@lab.ntt.co.jp

c) eda.takeharu@lab.ntt.co.jp

3.1 検証環境

検証環境として以下を利用する。

- ハードウェア

CPU	Intel Core i7-8700K CPU
メモリ	32GB
HDD	1TB
GPU	GeForce GTX 1050
USB HUB	USB3.0 Hub 名人 十六段 (CHM-U3P16)

- Neural Compute Stick(NCS)

プロセッサ	Intel Movidius VPU
接続形式	USB 3.0 Type-A

- ソフトウェア

OS	Ubuntu 16.04.4 LTS
NCS SDK	2.05.00.02
NCS ファームウェア	2.4.9296.228

- 入力データ

解像度	448x448
画像に映る人物数	10人
フレーム数	5400枚

3.2 マルチスティック検証

本検証では、NCSの本数を増やした場合に処理性能がどのように向上するかを確認する。NCSを14本用意し、NCSを1本ずつ増やしたときのfpsを測定する。実装の概要は、図2に示す。入力データを読み込み、領域検出を行い、推論結果は破棄する。マルチスレッドとマルチプロセスの両方を実装し、それぞれ測定する。

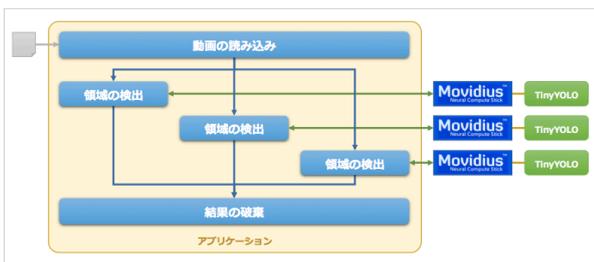


図2 マルチスティック構成

マルチスティック検証の結果を、図3に示す。マルチスレッド、マルチプロセス構成で、どちらもNCSの本数に対して線形にスケールし、14本のNCSでは、最大で107fpsの映像の領域検出ができることがわかる。マルチスレッドとマルチプロセスの結果を比較すると、マルチプロセスの方が高速化することがわかる。これは、PythonAPIを利用しているため、マルチスレッドではGILの影響をうけていると考えられる。

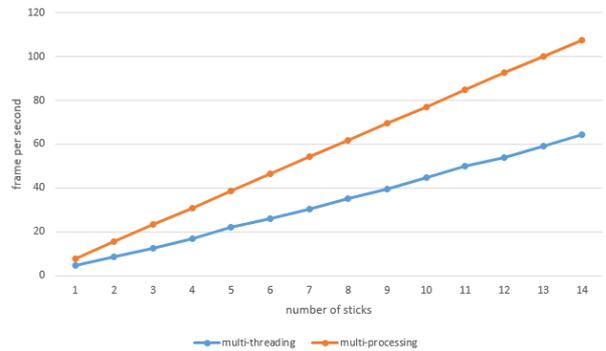


図3 マルチスティック検証の結果

3.3 マルチグラフ検証

本検証では、同じ処理を行うモデル(グラフ)を1本のNCSで複数実行させた場合に処理性能がどのように変化するかを確認する。NCSを1本用意し、図4のように領域検出処理を複数作成し、推論処理を同じNCSに割り当てる。NCS APIの制約上、マルチプロセス実装はできなかったため、マルチスレッドのみの実装を行い、測定する。

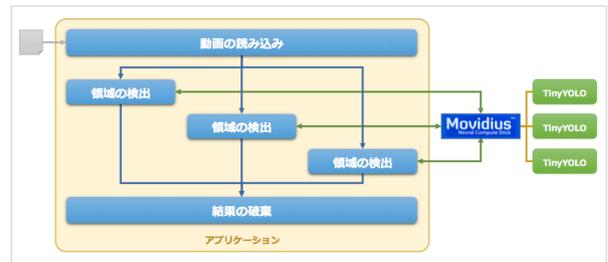


図4 マルチグラフ構成

マルチグラフ検証の結果は、図5となる。NCSに割り当てるグラフ数を増やしたところ、2つまでは性能向上するものの、それ以降は頭打ちとなる。この結果より、NCSには計算リソースの上限があり、グラフ数が2までは計算リソースに空きがあるが、グラフ数が3以上のケースでは計算リソースを奪い合っていると考えられる。NCSの計算リソースを活用するためには、1本のNCSに複数のグラフを実行した方が良いことがわかる。

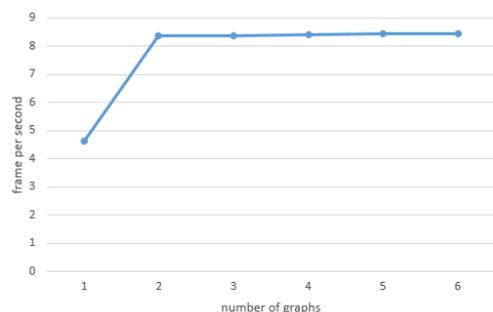


図5 マルチグラフ構成のマルチスレッドの結果

4. 深層学習アプリケーションの検証

4.1 深層学習アプリケーション検証の目的

基礎検証の結果を元に、深層学習アプリケーションを作成する。本検証では、深層学習アプリケーションが、マルチスティックやマルチグラフによって、処理性能がどのように変化するかを確かめる。

4.2 想定するユースケース

深層学習アプリケーションとして、複数種類のモデルを推論するアプリケーションを想定する。複数種類のモデルを推論するアプリケーションとは、人検出後に属性推定や顔照合をする場合や、車を検出した後に車種や車のナンバーを検出するような複数の分析を場合が挙げられる。また、顔検出という1つの分析をする場合においても、軽量で精度の低いニューラルネットワークと処理が重い高精度なニューラルネットワークを組み合わせるようなカスケード型な構成で高精度、高性能化している研究もある [7]。そのため、分析タスクが1つでも、複数種類のモデルを推論する場合も考えられる。

4.3 アプリケーション全体像

本検証では、防犯目的の映像解析アプリケーションを想定し、人検出後に顔照合を行うアプリケーションを作成する。人検出には、TinyYOLO、顔照合にはFaceNet[6]を利用する。マルチスティック検証の結果より、マルチスレッドよりもマルチプロセスの方が性能が高いことから、マルチプロセスでの実装を行う。マルチグラフ検証の結果より、1本のNCSには2つのグラフを実行した方が性能向上することから、顔照合処理は画像数(領域数)の子スレッドを起動し1本のNCSで顔照合用グラフを複数実行できるようにする。実装したアプリケーションは、図6のようになる。領域検出と顔照合はqueueを介してつなぎ合わせることで、処理パイプラインを構築している。このような構成を取ることで、処理を集約することができ、効率的にNCSを使うことができる。

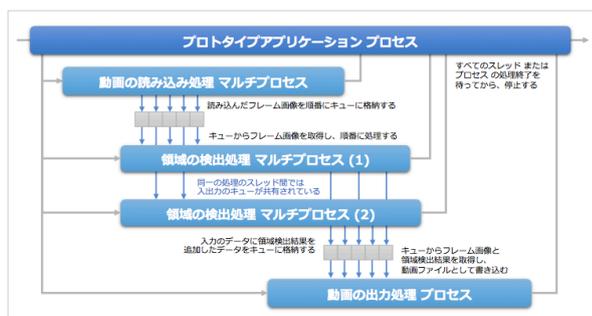


図6 アプリケーション全体像

4.4 検証環境

深層学習アプリケーションの検証では、以下のデータを利用する。

- 入力データ

解像度	448x448
画像に映る人物数	10人
フレーム数	5400枚
顔照合用クエリ画像	1枚
顔照合対象画像	各フレームから検出した領域の上部40%を利用

4.5 複数種類のモデルのNCS割当て

複数種類のモデルを複数本のNCSで実行する場合、どのモデルを何本のNCSに割り当てるかを考える必要がある。事前に人検出で1本、顔照合用に1本のNCSを割り当てて実行したところ、人検出では6.297fps、顔照合では0.666fpsという結果となる。そのため、人検出用に1本のNCSに対して、顔照合用に9本のNCSが必要となる。今回の検証では最大12本のNCSを使うため、人検出用に2本のNCSを割り当て、残りの10本を顔照合用とし、顔照合用のNCSを1本ずつ増やしたときに性能向上するかを確かめる。

4.6 検証結果

深層学習アプリケーションの検証結果は図7となる。NCSの本数に応じて、性能が線形にスケールすることがわかる。ただし、グラフ数が1の場合と、2の場合で性能が変化しないことから、マルチグラフの効果がないことがわかる。マルチグラフの効果が見られない原因として、スレッド数の増加によりホスト側の処理が遅くなっている可能性が考えられる。顔照合をマルチグラフ化するために、領域数ごとに子スレッドを起動している。今回の検証では、画像に映る人物数が10人であるため、1つの顔照合処理あたり、10スレッド起動している。そのため、顔照合用にNCSを10本割り当てている場合には、100スレッド起動している計算となる。このような結果より、マルチグラフ化する場合はアプリケーション全体を通して設計する必要がある。

5. まとめ

本検証では、NCSの本数を増やしたマルチスティック検証、1本のNCSに割り当てるグラフを増やしたマルチグラフ検証を行い、複数種類のモデルを推論する深層学習アプリケーションの検証を行った。

マルチスティック検証では、NCSの本数に応じて線形に性能がスケールすることが確認でき、最大107fpsで領域検出することを確認することができた。マルチグラフ検証では、1本のNCSに2つのグラフを実行させると性能向上

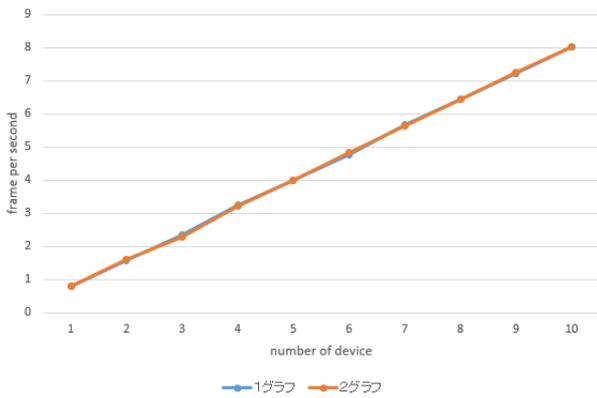


図 7 アプリケーションの性能検証結果

することが確かめられたが、ホスト側のスレッド数が増加するようなアプリケーションでは、マルチグラフィックの効果は得られなかった。

本検証を通じて、NCS を活用して低コスト、低消費電力な深層学習アプリケーションを実現できることを示せた。今後の課題として、NCS の状態監視、メモリ制約や処理量を考慮したアプリケーション全体の最適化などが考えられる。

参考文献

- [1] Intel Movidius Neural Compute Stick, available from <https://developer.movidius.com>
- [2] Deephi DPU available from <http://www.deephi.com/technology.html>
- [3] Edge TPU available from <https://aiyprojects.withgoogle.com/edge-tpu/>
- [4] Dexmont Pena et.al, Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications, 2017
- [5] Joseph Redmon et.al, You Only Look Once: Unified, Real-Time Object Detection, CVPR2016
- [6] Florian Schroff et.al, FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR2015
- [7] Kaipeng Zhang et.al, Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, SPL2016