Vol.2020-ARC-239 No.16
Vol.2020-SLDM-190 No.16
2020/1/23

コーナー検出プログラムのニューラルネットワーク化による高速化と出力精度に関する検討

竹内 一登[1,a)]　谷本 輝夫[1,b)]　川上 哲志[1,c)]　井上 弘士[1,d)]

概要：計算機性能向上化技術の一つとして，計算結果の精度低下を意図的に許容することで計算時間や消費電力の削減を可能にする近似コンピューテングが注目されている．この近似コンピューテング手法の 1 つである，プログラムのニューラルネットワーク化が注目を集めている．プログラム内で頻出する高負荷な関数やループボディといった「ホットコード」を，それらの入出力データをもとに訓練したニューラルネットワークに置換え実行することで，当該コード部分の実行における精度を犠牲に実効性能を向上する手法である．これまでに幾つかのアプリケーションに対してニューラルネットワーク化の効果が示されたものの，未だ限定的である．したがって，様々な適用事例を積み重ね，その有効性を評価する必要がある．本研究では，FAST (Feature from Accelerated Segments Test) アルゴリズムを用いたコーナー検出プログラムを対象とし，ニューラルネットワーク化による高性能化に向けた評価・解析を行う．具体的には，1) コーナー検出プログラムが主に 3 つの関数で構成されることに着目したニューラルネットワーク化対象箇所，ならびに，2) ニューラルネットワーク構成に関する探索を行い，実行時間と出力精度のトレードオフを解析する．

1. はじめに

一般の計算機においては，計算精度と性能 （電力効率も含む）の間にトレードオフ関係が存在する．この性質を利用した性能向上技術の一つとして，計算結果の精度低下を許容することで，実行時間の短縮や消費電力の削減を可能にする近似コンピューテングが注目されている [10]．適応領域としては，特に，演算精度の低下に対し比較的耐性を持つ画像処理や信号処理，機械学習などの分野が挙げられる．

これまでに，近似コンピューティングに関する様々な手法が提案されてきた [11]．例えば，不完全一致許容型の計算結果再利用 （いわゆるメモ化）やループボディ内処理の省略といったソフトウェアレベルから，狭ビット幅演算器の実装や DRAM におけるリフレッシュレートの低下といったハードウェアレベルまで，多様な手法が存在する．その中でも特に近年，プログラムのニューラルネットワーク化が注目を集めている [7]．ソースコード全体または一部（例えば関数）を対象とし，これらに対する入出力データをもとに訓練することでニューラルネットワークモデルを獲得する．そして，当該ソースコード部分を獲得した推論モデルで置換する．これにより，ニューラルネットワーク化にともなう精度低下が生じる反面，当該ソースコード部分が高負荷な場合には，積和演算と活性化関数という単純な基本演算で処理可能なニューラルネットワーク実行により性能向上を達成できる．また，近年ではニューラルネットワーク専用アクセラレータ開発が進んでおり，プログラムのニューラルネットワーク化はこれら新技術の恩恵を直接的に受けることが可能である．これまでに幾つかのアプリケーションに対してニューラルネットワーク化の効果が示されたものの，未だ限定的である [2]．したがって，今後は様々な適用事例を積み重ね，その有効性を評価する必要がある．

そこで本研究では，FAST (Feature from Accelerated Segments Test) アルゴリズムを用いたコーナー検出プログラムを対象とし，ニューラルネットワーク化による高性能化に向けた評価・解析を行う．具体的には，1) コーナー検出プログラムが主に 3 つの関数で構成されることに着目したニューラルネットワーク化対象箇所，ならびに，2) ニューラルネットワーク構成に関する探索を行い，実行時間と出力精度のトレードオフを解析する．

本稿の構成は以下の通りである．まず第 2 節でプログラムのニューラルネットワーク化による高速化の一例として，

[1] 九州大学
[a)] kazuto.takeuchi@cpc.ait.kyushu-u.ac.jp
[b)] tteruo@kyudai.jp
[c)] satoshi.kawakami@cpc.ait.kyushu-u.ac.jp
[d)] inoue@ait.kyushu-u.ac.jp

Vol.2020-ARC-239 No.16
Vol.2020-SLDM-190 No.16
2020/1/23

Parrot Transformation [3] について述べる．第 3 節では画像から特徴点の一種であるコーナーを検出する FAST アルゴリズムについて説明し，第 4 節で本研究で採用したニューラルネットワーク化フローを示す．第 5 節では，関数部分をニューラルネットワーク化した際の出力の精度と実行時間を評価する．第 6 節で関連研究について説明し，最後に第 7 節でまとめる．

2. Parrot Transformation

本節では，ソースコードのニューラルネットワーク化の一例として，Hadi らが提案した *Parrot Transformation* について述べる [3]．なお，その他の関連研究に関しては第 6 節で言及する．

Parrot Transformation とは，プログラムの一部をニューラルネットワーク推論モデルに置換え，NPU (Neural Processing Units) で加速実行することにより，計算速度の向上や消費電力の削減を可能にする手法である．本稿では，ソフトウェアレベルでの近似コンピューティング手法に着目したニューラルネットワーク化適用法の探索と性能／精度評価に着目する（専用アクセラレータを用いた高性能化は今後の課題）．そこで以降では，Parrot Transformation におけるソースコードのニューラルネットワーク化に焦点を絞り説明する．

一般に，プログラム内には頻繁に実行される「ホットコード」が存在する場合が多い．Parrot Transformation では，ホットコード部分をそれぞれのニューラルネットワーク推論モデルに置換え，該当コード部分の実行における精度を犠牲にすることで実効性能を向上する．推論モデルの生成においては，当該コード部分に対する入出力データを収集し学習する．ニューラルネットワークへの置換え対象となるソースコード部分は，基本的に以下の条件を満たす必要がある．

- ホットコードであること．
- プログラム実行の最終出力に深刻な精度低下をもたらさないこと．
- 静的に判定可能な明確な入出力を有すること（変換コードの入出力サイズが既知かつ固定，入力以外のデータを参照しない，システムコールを呼び出さない，など）．
- ニューラルネットワーク訓練用の入出力データを準備できること．

また，ニューラルネットワーク化の手順は以下の通りである．

(1) ニューラルネットワーク化対象コード部分の決定．
(2) 訓練に必要な入出力データの収集．例えば，訓練データ用の入力でオリジナルのソースコードによりニューラルネットワーク化対象部分を実行し，出力データを保存するなど．
(3) 置換えに使用するニューラルネットワークの訓練．
(4) ニューラルネットワーク推論モデルによるオリジナル・ソースコード部分の置換え．この際，プログラムのアルゴリズムに変更を加える必要はない．

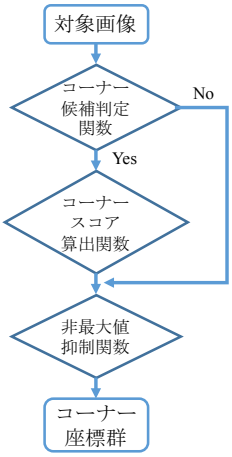

図 1　FAST によるコーナー検出の流れ

3. FAST アルゴリズムによる画像中のコーナー検出

3.1 コーナー検出

コーナーとは，画像中に存在する様々な「境界点」であり，例えば物体の輪郭点などが挙げられる（境界線ではない）．画像中のコーナーは特徴点の一種と捉えることができ，輝度の変化が大きい角もしくはその周辺に現れる傾向にある．コーナー検出とは，画像データ中から自動でコーナーを検出する処理であり，物体追跡や 3D モデリングなど様々な応用が存在するため，その高速化（とくにリアルタイム処理）に対する要求は大きい．

3.2 FAST アルゴリズムによるコーナー検出

様々なコーナー検出アルゴリズムが存在するが，本研究では Rosten と Drummond によって提案された FAST (Feature from Accelerated Segments Test) に着目する [8], [9]．本研究で対象とする FAST の実装は，「コーナー候補判定関数」「コーナースコア算出関数」「非最大値抑制関数」の 3 つの関数で構成される．FAST によるコーナー検出の流れを図 1 に示す．以下，各関数の詳細を説明する．

3.2.1 コーナー候補判定関数

各画素に対しコーナー候補となり得るか否かを判定する関数である．コーナーの判定方法は以下の通り．

(1) ある注目画素 P を基準とし，それを中心として半径 3 ピクセル分離れた円周上 16 ピクセルに注目する (図 2 参照)．
(2) 注目画素 P に対し，式 (1) に従って円周上の各 16 個

Vol.2020-ARC-239 No.16
Vol.2020-SLDM-190 No.16
2020/1/23

		1	2	3		
	16				4	
15						5
14			P			6
13						7
	12				8	
		11	10	9		

図 2　FAST によるコーナー検出の際に参照されるピクセルの配置

の画素それぞれにおいて輝度値が高い (明るい)，低い (暗い)，または同程度のいずれかに分類する．

$$\begin{cases} brighter & (I_x > I_p + t) \\ darker & (I_x < I_p - t) \\ samelevel & \text{上記以外} \end{cases} \quad (1)$$

ここで，I_x は円周上に存在する画素 x の輝度，I_p は注目画素 P の輝度，t は閾値を表す．

(3) 上記 (1) で定めた円周上において，brighter または darker と判定された画素が 9 個以上連続して検出された場合に注目画素 P をコーナー候補であると判定する．

上記のフローに基づきコーナー候補と判定された注目画素 P に対して，コーナースコア算出関数が実行される．一方，コーナー候補と判定されなかった画素に関しては，コーナースコアが 0 として出力され，以降はコーナー検出処理の対象外となる（つまり，最終的にコーナーと判定されることはない）．

3.2.2　コーナースコア算出関数

コーナー候補と判定された画素のコーナーらしさを示すスコアを算出する関数である．注目画素のコーナースコアは以下の式 (2) で算出される．

$$score = max\left(\sum_{x \in brighter} |I_x - I_p| - t, \sum_{x \in darker} |I_x - I_p| - t \right) \quad (2)$$

3.2.3　非最大値抑制関数

冗長にコーナーと判断された画素を棄却する関数である．非ゼロのコーナースコアをもつ画素に注目し，隣接している画素がより大きいコーナースコアを持っていた場合，注目した画素のコーナースコアを 0 にする．

FAST アルゴリズムの問題点は，コーナーであると判定した画素に隣接した画素もコーナーと判定される場合が多く，冗長にコーナーを検出することである．この問題点を解決するために，コーナースコアが導入されている．

4. FAST アルゴリズムの関数部分のニューラルネットワーク化

FAST (Feature from Accelerated Segments Test) アルゴリズムに基づくコーナー検出プログラムを構成する 3 つの関数は，第 2 節で述べたニューラルネットワーク化の適用基準を満たしている．本節では，第 2 節にて解説した手順に基づく各関数のニューラルネットワーク実装に関して説明する．

表 1　実装したニューラルネットワークの名称と再現した関数名，使用した訓練データ数

ニューラルネットワーク名	再現した機能(斜線部分)			訓練データ数
	a.コーナー候補判定関数	b.コーナースコア算出関数	c.非最大値抑制関数	
a	▨			300,000
b		▨		76,814
c			▨	76,814
ab	▨	▨		300,000
bc		▨	▨	76,814
abc	▨	▨	▨	300,000

4.1　ニューラルネットワーク化対象関数

ニューラルネットワーク化の対象となる関数の選択に関しては，各関数を個別選択するのみならず，複数関数を統合して単一ニューラルネットワーク推論モデルで表現することも可能である．そこで本研究では，表 1 に示すように，合計 6 種類の推論モデルを生成した．ここで，単一英文字は各関数を個別にニューラルネットワーク化した場合を示している．すなわち，ニューラルネットワークの表記として，コーナー候補判定関数は”a”，コーナースコア算出関数は”b”，非最大値抑制関数は”c”としている．ここで，ニューラルネットワーク”a”と”c”は注目画素がコーナーであるか否かを分類問題として，”b”はコーナースコアを回帰問題として処理する．

また，複数英文字 (積項) は複数の関数を統合して単一のニューラルネットワークで置換えることを意味する. 例えば，“ab” はコーナー候補判定関数とコーナースコア算出関数を単一ニューラルネットワークで置換えることに相当する. したがって，単一ニューラルネットワークで置換する関数の組み合わせは，” a”,” b”,” c”,” ab”,” bc”,” abc” の 6 種類となる. 表 1 の斜線部分は，各ニューラルネットワークで置換対象である関数を示す. 置換対象関数に対応する入力と出力の組みを用いてニューラルネットワークの訓練を行うことで，各ニューラルネットワークでの関数機能の近似を実現する．

4.2　ニューラルネットワークの訓練データ

各ニューラルネットワークの訓練データは，各画素の輝度を乱数生成した $500 \times 600[pixel]$ 画像を基に生成する (図 3). オリジナルのコーナー検出プログラムに対し乱数生成した画像を入力することで，各関数（図 1）毎の入力/出力のデータの組が得られる．これをニューラルネットワークの入力/正解データセット （訓練データ）とすることで訓

Vol.2020-ARC-239 No.16
Vol.2020-SLDM-190 No.16
2020/1/23

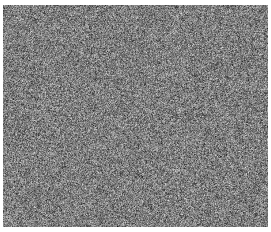

図 3　訓練データ収集用画像

表 2　実装したニューラルネットワークの設定

batchsize	64
最適化手法	Adam[6]
epoch 数	75
隠れ層数が 1 のトポロジー (括弧内はニューラルネット ワーク名)	17 - 40 - 2(abc, bc, a) 17 - 40 - 1(bc, b) 5 - 40 - 2(c)
隠れ層数が 2 のトポロジー (括弧内はニューラルネット ワーク名)	17 - 40 - 45 - 2(abc, bc, a) 17 - 40 - 45 - 1(bc, b) 5 - 40 - 45 - 2(c)

練を行う．表 1 に，各ニューラルネットワークの訓練データ数を示す．図 3 で示した 500 × 600[*pixel*] 画像をオリジナルのコーナー検出プログラムに入力した場合，各関数の入出力データセット数は，それぞれコーナ候補判定関数は 300,000，スコア算出関数および非最大抑制関数は 76,814 となる．

4.3　ニューラルネットワーク訓練

表 2 にニューラルネットワーク訓練時のハイパーパラメータ一覧，ならびにアーキテクチャを示す．本研究では，近似対象関数 （表 1）に依らず全て同一のハイパーパラメータに基づいてニューラルネットワークを訓練する．また，ニューラルネットワークアーキテクチャは隠れ層数が 1 と 2 の場合を想定し，各層数のニューロン数は optuna[1] を用いて最適化を実施する．

4.4　各関数のニューラルネットワーク置換え

プログラムのニューラルネットワーク化は，各関数が実行される際にその関数機能を近似するニューラルネットワークを呼び出すことで実現する. 置換に使用したニューラルネットワーク名と置換された関数の関係を表 3 に示す. ここで，複数のニューラルネットワークによる関数近似を和により表現している. 例えば，“ab+c” はコーナー候補判定関数とコーナースコア算出関数の機能を統合した単一のニューラルネットワーク，ならびに，非最大値抑制関数の機能を近似した単一ニューラルネットワークの 2 つを用いてコーナー検 出プログラムを実現する. また，ネットワーク名に” a”,” b”,” c” いずれかの英文字が現れない場合には，当該関数は近似を行わないオリジナルの関数で実行する. 例えば，“a+b” の場合には，コーナー候補判定関数，ならびに，コーナースコア算出関数はニューラルネットワークで実現するが，非最大抑制関数はオリジナルの関数で実行される．

表 3　置換えに使用したニューラルネットワーク名と再現された関数

ニューラルネットワーク名	再現された関数		
	コーナー候補判定関数	コーナースコア算出関数	非最大値抑制関数
abc	abc		
ab	ab		
bc		bc	
a	a		
b		b	
c			c
ab+c	ab		c
a+bc	a	bc	
a+b	a	b	
a+c	a		c
b+c		b	c
a+b+c	a	b	c

4.5　ニューラルネットワーク化の効果に関する定性的評価

コーナー候補判定関数においては，コーナーであるはずの画素を誤って非コーナーと判定した場合，この誤りはプログラム最終出力へと伝搬する．そのため，本関数がニューラルネットワーク化により精度が劣化した場合には，プログラム全体の出力精度も大幅に低下することが予想される．しかしながら，本関数はすべての画素に対して実行されるため処理負荷が高く，ニューラルネットワーク化による性能向上効果が期待できる．一方，コーナースコア算出関数および非最大値抑制関数は冗長なコーナー検出を回避することを目的としており，これらの関数で精度が劣化したとしても大半のコーナー検出は可能であると考えられる．すなわち，これらの関数がニューラルネットワーク化されても，プログラム全体の出力精度は大幅には悪化しないと予想できる．しかしながら，これらの関数はコーナー候補判定関数によってフィルタリングされた結果に対する処理であるため負荷はさほど高くなく，ニューラルネットワーク化による性能向上効果は限定的となる．

5.　評価

本稿では，FAST アルゴリズムを対象としたニューラルネットワーク化における実行時間と出力精度を解析する．また，その結果に基づき，適切なニューラルネットワーク化対象部分の選択やニューラルネットワークの構成について考察する．評価に用いた実験環境を表 4 に示す．

5.1　評価用画像と精度指標

評価に使用した画像を図 4 と図 5 に示す．画像サイズはいずれも 512 × 512[pixel] である．ニューラルネットワークの訓練に使用した図 3 と，評価に使用した図 5 は，サイ

表 4　評価に用いた実験環境

CPU	name: AMD Opteron™ Processor 6282 SE CPU MHz: 14000MHz = 14 GHz cache size: 2048KB = 2 MB cpu cores: 64
メモリ	MemTotal: 98970784 kB = 94.4 GB
OS	Ubuntu 14.04
プログラム言語	Python 3.4.3
ライブラリ	cv2(画像読み込みのみ), chainer, numpy, optuna, time

ズおよび使用した乱数が異なっている．ニューラルネットワーク化を適用しないオリジナルのコーナー検出プログラム（以降，正確なコーナー検出プログラムと呼ぶ）を用いて図 4 ならびに図 5 を入力したところ，抽出されるコーナー数はそれぞれ 530 ならびに 42,057 であった．これは全画素数に対しそれぞれ 0.20%ならびに 16.04%であり，極めて少ない数の画素がコーナーとして抽出されることが分かる．

このように，検出対象の出現率に大きな偏りが存在する場合，精度評価に正答率（すなわち，正解を正しく検出できる確率）を使用することは不適切となる．なぜなら，例えば「全ての画素がコーナーではない」と回答した場合でも上記の例では正答率がそれぞれ 99.79%ならびに 83.95%となり，適切な評価ができないためである．そこで本精度評価における指標として F 値 (F-measure) を使用する．本実験では，各画素をコーナーか非コーナーかへと振り分ける 2 クラス分類問題とみなし，図 5 に示す混合行列を定義する．TP，FN，FP，TN は，それぞれ真陽性，偽陰性，偽陽性，真陰性を指す．F 値とは，混合行列での数値を用いた推論結果の評価尺度の一種であり，式 (3) で示すように適合率と再現率の調和平均として定義される．ここで，適合率 (precision) とは，式 (4) で示すように，正と予測したデータのうち，実際に正であるものの割合である．また，再現率 (recall) とは，式 (5) で定義されるように，実際に正であるもののうち，正であると予測されたものの割合である．F 値を用いることで，入力に偏りが存在する場合においても適合率と再現率の両面からの出力精度評価が可能となる．

$$F\text{-}measure = 2Precision \times Recall/(Precision+Recall) \quad (3)$$

$$Precision = TP/(TP + FP). \quad (4)$$

$$Recall = TP/(TP + FN). \quad (5)$$

5.2　出力精度評価

図 4 の画像を評価用データとし，各関数をニューラルネットワーク化したコーナー検出プログラムの実行による F 値を測定した．その結果を図 6 に示す．横軸は表 3 で定義したニューラルネットワーク化に関する選択肢であり，

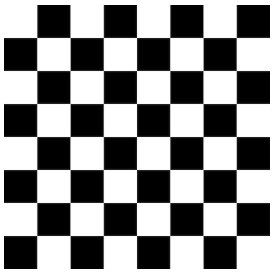

図 4　評価用画像 1:チェス盤画像

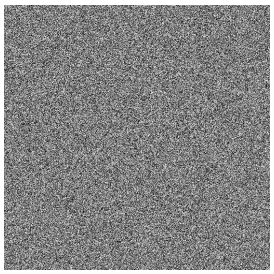

図 5　評価用画像 2:各画素の輝度を乱数で生成した画像

表 5　混合行列

		ニューラルネットワーク化後の出力 ○	ニューラルネットワーク化後の出力 ×
正確なプログラムの出力	○	TP	FN
正確なプログラムの出力	×	FP	TN

○:コーナー
×:非コーナー
TP,FN,FP,TN:
各状態に当てはまる画素数

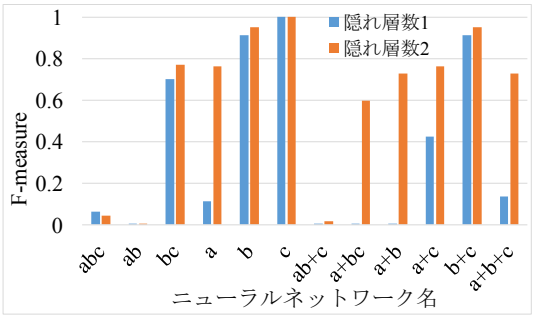

図 6　使用したニューラルネットワークに対する F 値

縦軸がそれぞれの選択肢における F 値である．この結果より，隠れ層数が 1 の場合，コーナー候補判定関数をニューラルネットワーク推論モデルに置換えることにより（すなわち，横軸にて”a”を含む場合），F 値が大きく低下していることが分かる．このことから，出力精度はコーナー候補判定関数に大きく依存していると考えられる．しかしながら，同様の傾向が見られるものの，隠れ層数を 2 へ増加することでこの問題を大幅に改善できており（例えば，a，a+bc，a+b，a+c，a+b+c），ニューラルネットワークを適切に設計することでこの問題をある程度回避できると考える．

5.3　実行時間評価

図 4 ならび図 5 の画像を評価用データとし，各関数をニューラルネットワーク化したコーナー検出プログラムの

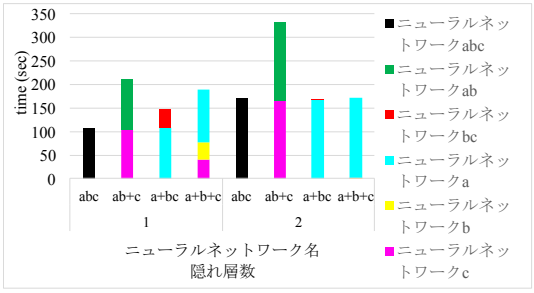

図 **7**　図 4 における実行時間

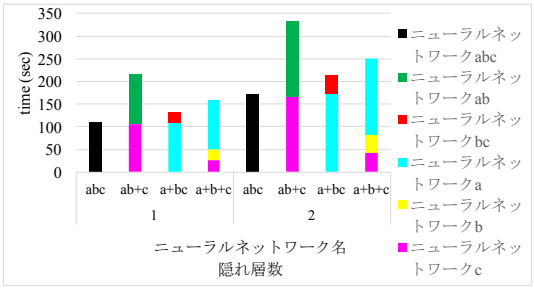

図 **8**　図 5 ににおける実行時間

性能を評価した．その結果を図 7 ならびに図 8 に示す．横軸は表 3 で定義したニューラルネットワーク化に関する選択肢であり，縦軸がそれぞれの選択肢における実行時間である．なお，ここでは 3 つの関数全てがニューラルネットワーク化される場合（すなわち，横軸のニューラルネットワーク名にて”a”，”b”，”c”の全てが必ず現れる）に限定し考察する．

図 7 において，ニューラルネットワークの隠れ層数が 1 の時，コーナースコア算出関数および非最大値抑制関数に相当するニューラルネットワークの実行時間は，ニューラルネットワークの隠れ層数が 2 の時と比較して大きかった．この理由は，コーナー候補判定関数の精度悪化により他関数の実行回数が増加したことが原因と考えられる．ニューラルネットワークの隠れ層数が 1 の時，コーナー候補算出関数実行後の非ゼロのコーナースコアを持つ画素数は 96,680 に対し，隠れ層数が 2 の時は 857 であった．このことから，不正確なコーナー候補判定関数を使用した場合，コーナースコア算出関数および非最大値抑制関数の実行時間が増加する可能性がある．

5.4　実行時間と出力精度の相関評価

図 4 ならび図 5 の画像を評価用データとし，各関数をニューラルネットワーク化したコーナー検出プログラムの実行時間と F 値の関係を図 9 ならびに図 10 に示す．各プロットは，隠れ総数が 1（青ドット）ならびに 2（橙ドット）における各ニューラルネットワーク化に関する選択肢（表 3）に対応する．また，赤直線は正確なコーナー検出プログラムを実行した際の実行時間である．これらの結果より，ニューラルネットワーク化対象箇所の選択，ならびに，各ニューラルネットワークの設計を適切に行うことで，ニューラルネットワーク・アクセラレータを用いない状況を想定したとしても 0.75 程度の F 値を達成しつつ，実行時間の短縮を見込めることが分かる．例えば，図 9 では隠れ層数 2 のニューラルネットワーク a を使用した場合の F 値は 0.76 であり，この時の実行時間削減率は 7.2%となった．また，図 10 においては，隠れ層数 2 のニューラルネットワーク a を使用した場合 0.73 の F 値を達成し，この時の実行時間削減率は 28%である．

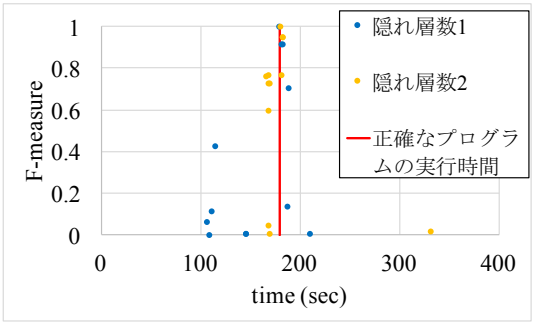

図 **9**　図 4 における実行時間に対する F 値

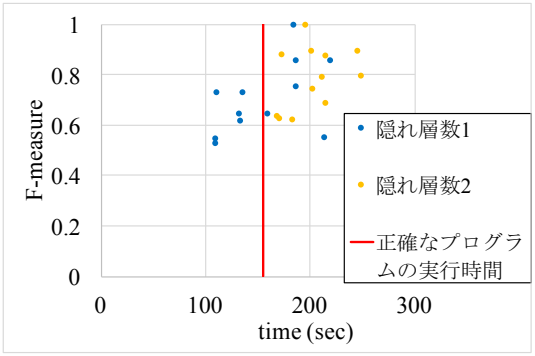

図 **10**　図 5 における実行時間に対する F 値

6.　関連論文

6.1　プログラムのニューラルネットワーク化のアナログ回路への実装

Amant らによりプログラムの一部をニューラルネットワーク化し，それをアナログ回路でハードウェア実装する手法が提案された [2]．アナログ回路ならびにデジタル回路でニューラルネットワークを実現する場合，シナプスは乗算回路，ニューロンは加算回路と非線形関数による写像を行う変換回路で構成される．アナログ乗算器は，デジタル回路と比較しエネルギー効率が高くチップの小型化が可能であるため，高集積化の観点から有利である [5]．しかし，入力と重みの設定は 8 ビットまで，各ニューロンへの入力数は 8 まで，訓練時には CDLM (customized continuous-discrete learning method) の導入による離散化への適応が必要と，再現できる機能に制約がある．

本実験ではニューラルネットワークの実行に CPU を使

用したため，上記の制約は存在しない．よって，多様なプログラムの機能のニューラルネットワーク化が可能となっている．

6.2 プログラム全体のニューラルネットワーク化

Hashmi らにより NISA (Neuromorphic Instruction Set Architecture) を使用することで，その命令を利用するプログラムを高速化する手法が提案された [4]．プログラムを正確に近似する場合は，巨大なニューラルネットワークを用意する場合が多い．しかし，CPU や GPU といった汎用プロセッサは巨大なニューラルネットワークの実行に必要な大量の計算の実行に適しておらず，そのまま実行しても高速化が期待できない．NISA は，あるプログラムを推論する巨大なニューラルネットワークのパラメータを取得し，それを元に許容できる出力精度を維持しつつ，汎用プロセッサでの実行でも性能向上が期待できる構成を持つニューラルネットワークを再定義する．用意するハードウェアに合わせてニューラルネットワークを再定義することで，プログラムのニューラルネットワーク化を容易に適用することができる．

本研究ではプログラム全体のニューラルネットワーク化だけでなく，機能の一部分のみのニューラルネットワーク化も行なっている．実行時間の大部分を占める処理だけをニューラルネットワーク化することにより，プログラム全体をニューラルネットワーク化した場合に近い性能向上を得ながら，出力精度低下を抑制できる可能性がある．

7. おわりに

本稿では，FAST アルゴリズムを用いたコーナー検出プログラムを対象にニューラルネットワーク化による近似をした際の実行時間と出力精度を評価した. 具体的には，プログラムの機能を 3 つの関数に分割し 1) ニューラルネットワーク化対象関数の組み合わせ，ならびに，2) ニューラルネットワークアーキテクチャを変化させ，プログラムの実行時間と出力精度のトレードオフ解析を行った．その結果，実行時間・出力精度ともに近似適用対象関数の選定やニューラルネットワークアーキテクチャに大きく依存しており，これら 1),2) の最適化が極めて重要であることが分かった．なお，本評価ではプログラムの最適なニューラルネットワーク化により F 値 0.73 で最大 28%の性能向上が達成できた．今後は正確なプログラムとニューラルネットワーク化したプログラムの実行時間の比較のため，GPU と機械語に翻訳可能なプログラミング言語を使用した実行時間評価を行う．また，アプリケーション内での機能のニューラルネットワーク化による出力精度の影響の調査のため．コーナー検出を利用したアプリケーション内でコーナー検出をニューラルネットワーク化した場合の出力精度評価を行う．

謝辞　本研究をご支援頂いた株式会社富士通研究所に心から感謝致します．なお，本研究は一部，JSPS 科研費 JP19H01105，未来社会創造事業 JPMJMI18E1 による．

参考文献

[1] Akiba, T. et al.: Optuna: A Next-generation Hyperparameter Optimization Framework, *In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &Data Mining*, ACM, pp. 2623–2631 (2019).
[2] Amant, R. S. et al.: General-Purpose Code Acceleration with Limited-Precision Analog Computation, *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 1–12 (2014).
[3] Esmaeilzadeh, H. et al.: Neural Acceleration for General-Purpose Approximate Programs, *In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 449–460 (2012).
[4] Hashmi, A. et al.: A Case for Neuromorphic ISAs, *In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ACM, pp. 145–158 (2011).
[5] Hasler, J. and Marr, B.: Finding a roadmap to achieve large neuromorphic hardware systems, *frontiers in Neuroscience*, Vol. 7, pp. 1–29 (2013).
[6] Kingma, D. P. and Ba, J. L.: Adam: a Method for Stochastic Optimization, *arXiv:1412.6980*, pp. 1–15 (2015).
[7] Mittal, S.: A Survey of Techniques for Approximate Computing, *ACM Computing Surveys*, Vol. 48, No. 4, pp. 1–33 (2016).
[8] Rosten, E. and Drummond, T.: Fusing points and lines for high performance tracking., *In Proceedings of the IEEE International Conference on Computer Vision*, ICCV '05, Vol. 2, pp. 1508–1511 (2005).
[9] Rosten, E. and Drummond, T.: Machine learning for high-speed corner detection, *In Proceedings of the 9th European Conference on Computer Vision*, ECCV '06, Vol. 1, pp. 430–443 (2006).
[10] Sekanina, L.: Introduction to Approximate Computing: Embedded Tutorial, *In Proceedings of the 2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, IEEE, pp. 1–6 (2016).
[11] Xu, Q. et al.: Approximate Computing: A Survey, *IEEE Design & Test*, Vol. 33, No. 1, pp. 8–22 (2016).