Vol.2017-HPC-158 No.24
2017/3/9

低ランク近似を用いた深層学習の行列積の高速化

関谷 翠[†1,a)]　大沢 和樹[†1,b)]　長沼 大樹[†1,c)]　横田 理央[†1,d)]

概要：TensorFlow，Caffe，Torch をはじめとした深層学習のオープンソースフレームワークは世界中で広く用いられており，その高速化は大きな意味を持つ．これらのフレームワーク中では，計算時間の多くが畳み込み計算などの密行列積に費やされており，cuDNN などの高度にチューニングされたライブラリが高速化を担っている．一方で，これらのライブラリでは密行列を近似することなく畳み込み計算を行なっており，ここに低ランク近似を用いることでデータ量，計算量を大幅に削減する余地がある．本研究では，科学技術計算の分野で広く用いられている低ランク近似法を畳み込み計算に導入する手法を既存ネットワークに対して実装し，その識別精度に対する影響を調査した結果，認識精度に影響を与えることなく約 40%のランクを削減することができた．

Sekiya Akira[†1,a)]　Osawa Kazuki[†1,b)]　Naganuma Hiroki[†1,c)]　Yokota Rio[†1,d)]

1. はじめに

近年，自然言語処理や画像認識をはじめとした様々な分野において，深層学習を用いた手法が広く用いられている．特に画像認識の分野においては，Krizhevsky ら [1] のニューラルネットワークが大規模画像認識コンテスト ILSVRC [2] で 2012 年に既存手法に 10% 以上の差をつけて優勝して以来，VGGNet [3]，GoogLeNet [4]，ResNet [5] といった優れた性能を示すニューラルネットワークが数多く開発されている．

一方で，これらのニューラルネットワークはその層の多さが影響し，トレーニングに多大な時間がかかる．19 層で構築される VGGNet では，4 台の NVIDIA Titan Black GPU を用いた場合，トレーニングに 2 から 3 週間もの時間がかかったことが論文中に示されている．

また，層の数が増えるに従い，層中に含まれるパラメータの数も増えるため，モデルのサイズが大きくなり，特に車載などの組み込み分野においては大きな課題点となっている．

この問題を解決するために，ニューラルネットワークの持つノイズに対する耐性を利用する手法が複数提案されている．Courbariaux ら [6] は重みおよび活性を +1，−1 に制約して計算を行なった場合に，Gupta ら [7] は半精度固定小数点を計算に用いた場合に精度が低下しないことをそれぞれ示している．

また，ニューラルネットワーク中の計算時間の約 90% が畳み込み層での計算時間であることが知られている [8]．このことを受けて，畳み込みで用いられる行列積の高速化・最適化を行う研究が数多く存在する．maxDNN [9] では，NVIDIA Maxwell GPU 向けに高度に最適化された行列積演算ライブラリを使用することで，畳み込み計算の高速化を達成している．

科学技術計算の分野においても，行列積演算の高速化は大きな関心を集めている．Coppersmith ら [10] は $\mathcal{O}(n^{2.375477})$，Gall [11] は $\mathcal{O}(n^{2.3728639})$ の計算量での行列積を行うアルゴリズムを開発している．

本研究では，cuDNN [12] などで畳み込み計算に用いられる im2col [13] 手法によって得られる行列に対し，行列分解法の一種である 特異値分解 (SVD, Singular Value Decomposition) を用い，さらに低ランク近似を行う手法を提案し，実装する．さらに，上記の手法をニューラルネットワークの一つである ResNet に対し適用する．

また，特異値分解を行う際に，行列をブロック分割した場合に精度に対しどのような影響が出るかについて測定を行う．

†1 現在，東京工業大学
Presently with Tokyo Institute of Technology
a) sekiya.a@rio.gsic.titech.ac.jp
b) oosawak@rio.gsic.titech.ac.jp
c) hiroki11x@rio.gsic.titech.ac.jp
d) rioyokota@gsic.titech.ac.jp

2. 関連研究

2.1 Deep Residual Network

Deep Residual Network（ResNet）[5] は MSRA（Microsoft Research Asia）の K. He らによって考案された CNN のモデルである．ResNet は 2015 年開催の大規模物体認識，画像分類のベンチマークである ILSVRC（ImageNet Large Scale Visual Recognition Challenge）[2][*1] の物体検出部門（DET）[*2] と物体分類，ローカリゼーション部門（LOC）[*3] にて最高の性能を示した．

また，同年に開催された COCO 2015 Detection Challenge [14] においても画像検出，セグメンテーション（segmentation）の分野で優勝している．

ResNet の最大の特徴は，ネットワークを構成する層数にある．2015 年の ILSVRC で用いられた ResNet モデルの最大層数は 152 で，2014 年に同じく DET，LOC で最高の性能を示した GoogLeNet[4] の最大層数の 22 や，VGG net（very deep ConvNet）[3] の最大層数の 16 と比較するとネットワークを構成する層数が大幅に増加したことが分かる．

近年の報告では，ネットワークの層数を増やすことが画像認識問題において重要であることが明らかにされており，ResNet が発表される以前から，ILSVRC を始めとする多くの画像認識の問題において，ネットワークを構成する層数が多い（very deep: ∼30 程度）モデルが成果をだしてきた．

一方で，単純に層数を増やすと，ネットワークの重み $\mathbf{W} \in \mathbb{R}^N$（$N \in \mathbb{N}$）が属する実数空間 $\mathbb{R}^N$ 上で，ネットワークによる推定結果と真の値との差異を表現するために定義される損失関数 $l : \mathbb{R}^N \to \mathbb{R}$ の勾配が消失，発散し，認識精度がかえって低下してしまうことが知られている [15]．

この問題に対し，ResNet は，残差表現（Residual Representations）を CNN に取り入れ，ネットワークを構成する層数を大幅に増やし，高い認識精度を獲得している．

ネットワーク内の（必ずしも全てとは限らない）連続した複数の層への入力 $\mathbf{x} \in \mathbb{R}^n$（$0 < n \in \mathbb{N}$）に対し，複数の層による出力は関数 $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}^n$ を用いて $\mathcal{H}(\mathbf{x})$ で表される．残差表現では，$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ で定義される残差関数 $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$ を用いて，複数の層による出力を $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ で表す．これにより，入力画像を最も良く認識するために（ネットワーク内の連続した層のまとまり全てについて）最適な関数 $\mathcal{H}$ を推定する問題が，最適な残差関数 $\mathcal{F}$ を推定する問題に置き換えられる．

損失関数の勾配が消失，発散する問題は，複数の非線形層で恒等写像（identity mapping）を近似することの困難さに起因すると考えられているが，残差表現では，$\mathcal{F}(\mathbf{x}) \approx \mathbf{0}$ としてしまえば，容易に恒等写像を近似できる．実際に，K. He らの報告によると，ResNet の各層（層数: 20∼110）間の出力の標準偏差が，残差表現を用いていない単純なモデルと比べ，低いことが示されている．

この結果から，残差表現により恒等写像を容易に近似できる ResNet は，有効な初期値設定を獲得し，層数を増やすことによる恩恵を享受していることが分かる．

[*1] http://image-net.org/challenges/LSVRC/2015/
[*2] 200 カテゴリの Mean average precision: 0.6270741
[*3] 1,000 カテゴリの誤分類率: 0.03567，ローカリゼーション誤差率: 0.090178

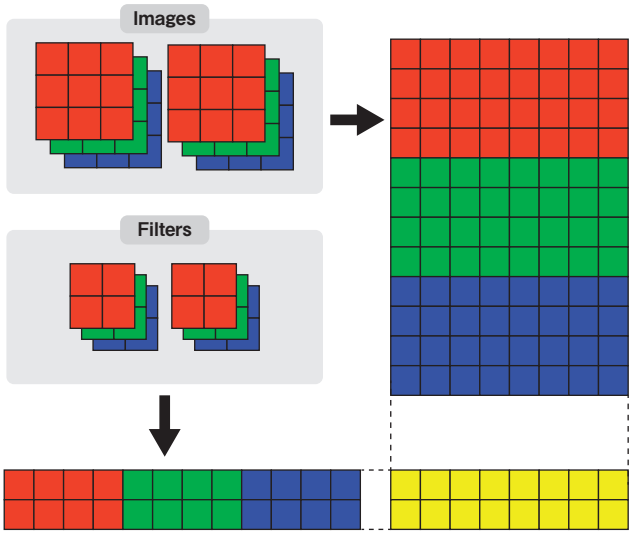

図 1 im2col による入力・重み行列の変換
Fig. 1 Translation of input and weight matrix using im2col

2.2 cuDNN

cuDNN [12] は，NVIDIA 社が自社の GPU アーキテクチャ向けに公開しているディープラーニング用計算ライブラリである．ニューラルネットワークで用いられる様々なレイヤ向けの計算を行う関数が含まれている．

cuDNN は TensorFlow をはじめとしたフレームワークにおいて，ニューラルネットワークの要素を実装する場面で広く用いられており，高速化のための様々なチューニングやアルゴリズムが実装されている．

このライブラリでは，1 回の行列積演算で畳み込み計算が行えるよう，im2col [13] と呼ばれる手法を用い，畳み込み層の入力と重みをそれぞれ **図 1** のような，定められた行列へと変換する．これにより，cuBLAS を始めとした GPU を利用する高速な行列積実装を使用することによって，畳み込みの計算時間を短縮することが可能となっている．

2.3 特異値分解

行列 $A \in \mathbb{R}^{m \times n} \backslash \{O\}$ に対して定義される実対称行列 $AA^{\mathrm{T}} \in \mathbb{R}^{m \times m}$ の固有値 λ_j $(j = 1, 2, \ldots, m)$ を順に

$$\lambda_1 \geq \cdots \geq \lambda_r > \lambda_{r+1} = \cdots = \lambda_m = 0$$

とすると，

$$\sigma_j := \sqrt{\lambda_j} > 0 \quad (j = 1, \ldots, r)$$

を A の特異値（singular value）という．AA^{T} の固有値 σ_j $(j = 1, \ldots, r)$ に対応する固有ベクトル $\mathbf{u}_j$ $(j = 1, \ldots, r)$ と，$A^{\mathrm{T}}A$ の固有値 σ_j $(j = 1, \ldots, r)$（AA^{T} と $A^{\mathrm{T}}A$ の正の固有値は完全に一致する）に対応する固有ベクトル $\mathbf{v}_j$ $(j = 1, \ldots, r)$ を用いて，A は，

$$\begin{aligned} A &= [\mathbf{u}_1 \ldots \mathbf{u}_r] \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{bmatrix} [\mathbf{v}_1 \ldots \mathbf{v}_r]^{\mathrm{T}} \\ &= \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}} \end{aligned}$$

と表現できる．これを特異値分解と呼ぶ．

ここで，行列 $A \in \mathbb{R}^{m \times n}$ と行列 $B \in \mathbb{R}^{n \times k}$ との行列積を考える．A, B の特異値分解をそれぞれ $A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}}, B = \sum_{i=1}^{r} \hat{\sigma}_i \hat{\mathbf{u}}_i \hat{\mathbf{v}}_i^{\mathrm{T}}$ とすると，$1 \leq r', r'' \leq r$ に対し，

$$AB \approx \left(\sum_{i=1}^{r'} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}} \right) \left(\sum_{i=1}^{r''} \hat{\sigma}_i \hat{\mathbf{u}}_i \hat{\mathbf{v}}_i^{\mathrm{T}} \right)$$

と近似される．これを低ランク近似と呼ぶ．

$$\begin{aligned} U_A &:= [\mathbf{u}_1 \ldots \mathbf{u}_{r'}] \in \mathbb{R}^{m \times r'} \\ \Sigma_A &:= \mathrm{diag}(\sigma_1 \ldots \sigma_{r'}) \in \mathbb{R}^{r' \times r'} \\ V_A &:= [\mathbf{v}_1 \ldots \mathbf{v}_{r'}] \in \mathbb{R}^{n \times r'} \\ U_B &:= [\hat{\mathbf{u}}_1 \ldots \hat{\mathbf{u}}_{r''}] \in \mathbb{R}^{n \times r''} \\ \Sigma_B &:= \mathrm{diag}(\hat{\sigma}_1 \ldots \hat{\sigma_{r''}}) \in \mathbb{R}^{r'' \times r''} \\ V_B &:= [\hat{\mathbf{v}}_1 \ldots \hat{\mathbf{v}}_{r''}]^{\mathrm{T}} \in \mathbb{R}^{k \times r''} \end{aligned}$$

と定義し，

$$\begin{aligned} \left(\sum_{i=1}^{r'} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}} \right) \left(\sum_{i=1}^{r''} \hat{\sigma}_i \hat{\mathbf{u}}_i \hat{\mathbf{v}}_i^{\mathrm{T}} \right) &= U_A \Sigma_A V_A^{\mathrm{T}} U_B \Sigma_B V_B^{\mathrm{T}} \\ &= U_A \left(\Sigma_A V_A^{\mathrm{T}} U_B \Sigma_B \right) V_B^{\mathrm{T}} \end{aligned}$$

として近似値を計算するとき，近似に伴う計算量は十分小さな r', r'' が与えられれば，行列積の計算量 $\mathcal{O}(mnk)$ を下回る．

2.4 TensorFlow

TensorFlow [16] は数値計算をデータフローグラフと呼ばれる構造を用いて定義することにより，多次元配列の計算を行うオープンソースフレームワークであり，機械学習の分野ではニューラルネットワークを実装する際に広く用いられている．

データフローグラフは変数を表す `Variable`， 定数を表す `Constant`，計算を表す `Operator` などからなり，定義されたデータフローグラフを実行する環境である `Session` に 画像，重みなどの行列である `Tensor` とともに渡すことによって計算を実行する．

3. 実装

本研究での実装は，畳み込み層に対する入力と重みの 4 次元テンソルに対し im2col を適用し，その結果得られる 2 次元行列に対し低ランク近似法を適用し，さらに行列積を行うことによって畳み込み層の出力とする．

しかし，im2col によって出力される行列は辺の大きさが 10 万を超えるものもあり，そのまま特異値分解を行うと，$m \times n (m < n)$ 行列に対して $\mathcal{O}(mn^2)$ の計算量がかかり，$\mathcal{O}(m^2 + n^2 + mn)$ のメモリを使用する．

この問題を解決するために，im2col によって得られる行列を複数の小さなブロックに分割し，そのそれぞれに対して特異値分解を行う，ブロック化と呼ばれる手法を実装した．この手法を導入したアルゴリズムは，アルゴリズム 1 のようになる．

この時，行数・列数が block_size となるようなブロックに行列を分割すると，計算量は $\mathcal{O}(\text{block\_size} \times m \times n)$，メモリ使用量は $\mathcal{O}(m \times n)$ まで削減することが可能となる．

また，ブロック化を行うことによって，キャッシュヒット率の向上による高速化も期待することができる．

ブロック化を行う際には，元の行列の行数・列数が分割するブロックサイズの倍数となるよう 0 padding を行なっている．これにより，特異値分解を行なった後に低ランク近似を行う際，低いランクへと近似できるように見えることに注意する必要がある．

本実装での行列積，特異値分解は，それぞれ LAPACK ライブラリ [17] の `sgemm` 関数，`sgesvd` 関数を用いて行なっている．

TensorFlow との連携は Cython [18] を用いて行なった．ある畳み込み層に対し本手法を適用する際には，その畳み込み層以前までの結果を TensorFlow を用いて計算し，その結果を Cython を用いて C 言語で実装されたモジュールに渡し，その結果を再度 TensorFlow に入力することで以後の計算を行う．

4. 実験と評価

本研究では，10 クラスに分類された，50000 枚のトレーニング用画像と 10000 枚の評価用画像を含む CIFAR-10 [19] データセットににによって，あらかじめトレーニングされた ResNet に対し，畳み込み層での行列積に対してブロック分割する際のブロックの大きさと，低ランク近似を行う際のランク数を変化させ，認識精度の測定を行なった．対象とした ResNet のパラメータは**表 1** の通りである．ここで，scale1, 2, 3 中の 2 つの畳み込み層のまとまりを block，block 中の畳み込み層の前者を A，後者を B と呼ぶことに

Algorithm 1 ブロック低ランク近似行列の生成アルゴリズム

Require: $A \in \mathbb{R}^{m\times k}, B \in \mathbb{R}^{k\times n}$
1: 行列 A, B の行数，列数が block_size の倍数となるように 0 padding
2: 行列 A, B を行数，列数が block_size であるブロックに分割する
3: $K_A^*, K_B^* \leftarrow$ 行列 A, B を近似するランク
4: **for** $I \leftarrow 1, \dots m/$block_size **do**
5: **for** $L \leftarrow 1, \dots k/$block_size **do**
6: $A_{I,L} \leftarrow A$ の I 行 L 列のブロック
7: SVD($A_{I,L}$)
8: LRA(K_A^*)
9: **end for**
10: **end for**
11: **for** $L \leftarrow 1, \dots k/$block_size **do**
12: **for** $J \leftarrow 1, \dots n/$block_size **do**
13: $B_{I,J} \leftarrow B$ の L 行 J 列のブロック
14: SVD($B_{I,J}$)
15: LRA(K_B^*)
16: **end for**
17: **end for**
18: **for** $I \leftarrow 1, \dots m/$block_size **do**
19: **for** $J \leftarrow 1, \dots n/$block_size **do**
20: **for** $K \leftarrow 1, \dots k/$block_size **do**
21: 行 8, 15 の結果を用いて行列積を計算する
22: **end for**
23: **end for**
24: **end for**

レイヤー名	出力マップサイズ	パラメータ
scale1	32×32	3×3, 16, batch normalization
		$\begin{bmatrix} 3\times 3, 16 \\ 3\times 3, 16 \end{bmatrix} \times 3$
scale2	16×16	$\begin{bmatrix} 3\times 3, 32 \\ 3\times 3, 32 \end{bmatrix} \times 3$
scale3	8×8	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 3$
	1×1	average pool, 10-d fc, softmax

表 1 使用した ResNet のパラメータ
Table 1 Parameter of ResNet

する．

4.1 畳み込み層の行列のランクの調査

実際に畳み込み層に対し本研究を適用し認識精度を測定するに先立ち，畳み込み層で用いられる行列が持つ特異値の分布について調査した (**図 2**)．縦軸は特異値を，横軸は全体を 1 とした時のランクの割合を表している．

このグラフから，入力層に近い畳み込み層においては，重み行列の特異値が素早く減衰することが見て取れる．一方で，出力層に近い畳み込み層においては，行列の特異値の減衰は遅くなっている．これにより，特に入力層に近い畳み込み層において，より低いランクを閾値として低ランク近似を行えることが分かる．

4.2 ブロック・ランクと認識精度の関係

低ランク近似による影響がどの程度発生するかを測定するために，分割するブロックのサイズおよび低ランク近似行列作成の際のランクの閾値を変化させながら推論を行なった際の認識精度を測定した (**図 3**)．

この時，C, R, S はそれぞれ重み行列のチャンネル数，行数，列数を表す．縦軸はネットワークの認識精度を表しており，横軸はブロックサイズに対して何%のランクを閾値として低ランク近似を行なったかを表している．

水色線は，低ランク近似を行わなかった場合の認識精度である．緑・赤色の線はそれぞれブロックのサイズを全ての層で 32, 64 に固定した場合の認識精度を表し，青色の線はブロックサイズをそれぞれの層の $C \times R \times S$ に設定した場合の認識精度を表している．

この結果から，約 6 割のランクを確保するよう閾値を設定した場合には，認識精度ほぼ変化しないことがわかる．また，3%程度の認識精度の低下を許容した場合には，約 5 割のランクを確保するよう閾値を設定すれば良いことがわかる．

また，ブロックサイズ間の比較では，ブロックサイズが増加するほど認識精度が増加する．これは，低ランク近似を行う際に，対象となる行列が大きいほど，小さい特異値を無視することによる近似誤差が小さくなることが影響していると考えられる．ブロックサイズを $C \times R \times S$ に設定した場合に認識精度が向上する結果に対しても同様のことが言える．それぞれの層で低ランク近似を行う対象となる行列のサイズは**表 2** のようになっている．

5. おわりに

本研究によって，畳み込み計算に用いられる行列に対し，ニューラルネットワーク全体の認識精度に悪影響を与えることなく低ランク近似法を適用できることがわかった．

一方で，本研究の手法を高速化に繋げるためには，以下のような課題がある．

本研究の手法を TensorFlow の Operator として実装する

現在の実装では，TensorFlow のセッションの実行におけるオーバーヘッドが大きく，実行時間に大きな影響を与えている．また，ネットワークグラフのソースコードを改変する必要があり，多大な労力を必要としている．TensorFlow の Operator として定義することによって，これらの問題を解決することが可能である．

低ランク近似法として，特異値分解以外の手法を用いる

現在低ランク近似法として用いている LAPACK ライブラリの特異値分解は実行に長い時間がかかり，分解後の行列積の計算量低下による高速化を上回ってしまう．そこ

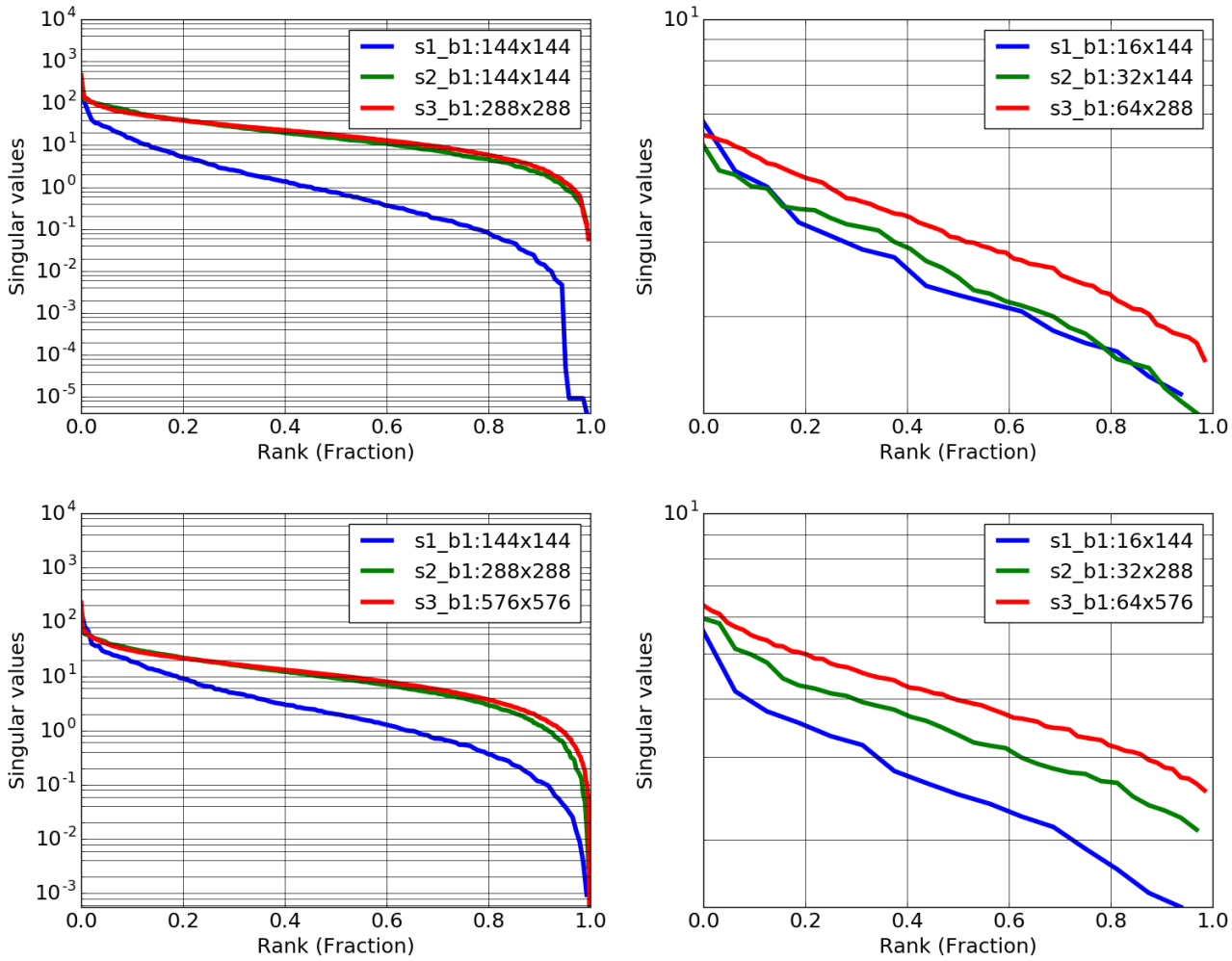

図 2 低ランク近似を適用する行列のランク．左上から，レイヤー A の入力行列のランク，レイヤー A の重み行列のランク，レイヤー B の入力行列のランク，レイヤー B の重み行列のランク．

Fig. 2 Rank of each matrices to low rank approximate.

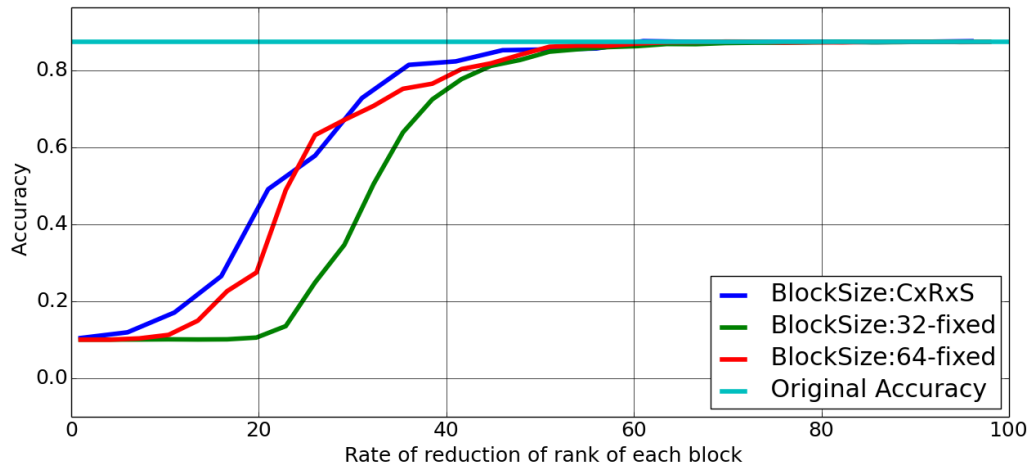

図 3 低ランク近似行列を畳み込み演算に適用させた際の採用したランクの閾値との割合と認識精度の関係

Fig. 3 Relationship between accuracy and rate of reduction of rank for each matrix

scale	block	A or B	重み行列サイズ	入力行列サイズ
1			16 × 27	27 × 147968
1	1	A	16 × 144	144 × 147968
1	1	B	16 × 144	144 × 147968
1	2	A	16 × 144	144 × 147968
1	2	B	16 × 144	144 × 147968
1	3	A	16 × 144	144 × 147968
1	3	B	16 × 144	144 × 147968
2	1	A	32 × 144	144 × 147968
2	1	B	32 × 288	288 × 32768
shortcut			32 × 16	16 × 165888
2	2	A	32 × 288	288 × 41472
2	2	B	32 × 288	288 × 41472
2	3	A	32 × 288	288 × 41472
2	3	B	32 × 288	288 × 41472
3	1	A	64 × 288	288 × 41472
3	1	B	64 × 576	576 × 12800
shortcut			64 × 32	32 × 51200
3	2	A	64 × 576	576 × 12800
3	2	B	64 × 576	576 × 12800
3	3	A	64 × 576	576 × 12800
3	3	B	64 × 576	576 × 12800

表 2 層ごとの低ランク近似を行う行列のサイズ
Table 2 Size of matrices for convolution layers

で，特異値分解を行うアルゴリズムに対し乱択アルゴリズムの手法を用いて高速化を行う Randomized SVD [20] などの手法を用いることが考えられる．

GPU を利用した実装を行う

近年のニューラルネットワーク実装では，GPU を用いた計算の高速化が広く行われている．また，行列積，特異値分解といった計算を行うライブラリも公開されている [21] [22]．これらを利用することで，より高速な実装を行うことが可能であると考えられる．

TensorFlow 以外のフレームワークへの実装

ニューラルネットワーク実装に用いられるフレームワークとしては，TensorFlow 以外にも Torch [23]，Caffe [24]，Chainer [25] などが存在する．これらのフレームワークに対する実装を行うことで，様々なネットワークに対して本研究の手法を適用することが可能となる．

謝辞 本研究は，JST，CREST の支援を受けたものである．本研究は JSPS 科研費 16H05859 の助成を受けたものである．

参考文献

[1] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Advances in Neural Information Processing Systems 25* (Bartlett, P., Pereira, F., Burges, C., Bottou, L. and Weinberger, K., eds.), pp. 1106–1114 (online), available from ⟨http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf⟩ (2012).

[2] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge, *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252 (online), DOI: 10.1007/s11263-015-0816-y (2015).

[3] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *CoRR*, Vol. abs/1409.1556 (2014).

[4] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper with Convolutions, *Computer Vision and Pattern Recognition (CVPR)*, (online), available from ⟨http://arxiv.org/abs/1409.4842⟩ (2015).

[5] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *CoRR*, Vol. abs/1512.03385 (online), available from ⟨http://arxiv.org/abs/1512.03385⟩ (2015).

[6] Courbariaux, M. and Bengio, Y.: BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, *CoRR*, Vol. abs/1602.02830 (online), available from ⟨http://arxiv.org/abs/1602.02830⟩ (2016).

[7] Gupta, S., Agrawal, A., Gopalakrishnan, K. and Narayanan, P.: Deep Learning with Limited Numerical Precision, *CoRR*, Vol. abs/1502.02551 (online), available from ⟨http://arxiv.org/abs/1502.02551⟩ (2015).

[8] Jia, Y.: Learning Semantic Image Representations at a Large Scale, PhD Thesis, EECS Department, University of California, Berkeley (2014).

[9] Lavin, A.: maxDNN: An Efficient Convolution Kernel for Deep Learning with Maxwell GPUs, *CoRR*, Vol. abs/1501.06633 (online), available from ⟨http://arxiv.org/abs/1501.06633⟩ (2015).

[10] Coppersmith, D. and Winograd, S.: Matrix multiplication via arithmetic progressions, *Journal of Symbolic Computation*, Vol. 9, No. 3, pp. 251 – 280 (online), DOI: http://dx.doi.org/10.1016/S0747-7171(08)80013-2 (1990).

[11] Gall, F. L.: Powers of Tensors and Fast Matrix Multiplication, *CoRR*, Vol. abs/1401.7714 (online), available from ⟨http://arxiv.org/abs/1401.7714⟩ (2014).

[12] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. and Shelhamer, E.: cuDNN: Efficient Primitives for Deep Learning, *CoRR*, Vol. abs/1410.0759 (online), available from ⟨http://arxiv.org/abs/1410.0759⟩ (2014).

[13] Chellapilla, K., Puri, S. and Simard, P.: High Performance Convolutional Neural Networks for Document Processing, *Tenth International Workshop on Frontiers in Handwriting Recognition* (Lorette, G., ed.), La Baule (France), Université de Rennes 1, Suvisoft, (online), available from ⟨https://hal.inria.fr/inria-00112631⟩ (2006). http://www.suvisoft.com.

[14] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C. L.: Microsoft COCO: Common Objects in Context, *CoRR*, Vol. abs/1405.0312 (online), available from ⟨http://arxiv.org/abs/1405.0312⟩ (2014).

[15] Glorot, X. and Bengio, Y.: Understanding the difficulty

of training deep feedforward neural networks, In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS' 10). Society for Artificial Intelligence and Statistics (2010).

[16] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Man, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Vigas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2015).

[17] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition (1999).

[18] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S. and Smith, K.: Cython: The Best of Both Worlds, *Computing in Science Engineering*, Vol. 13, No. 2, pp. 31–39 (online), DOI: 10.1109/MCSE.2010.118 (2011).

[19] Krizhevsky, A. and Hinton, G.: Learning multiple layers of features from tiny images (2009).

[20] Halko, N., Martinsson, P. G. and Tropp, J. A.: Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, *SIAM Review*, Vol. 53, No. 2, pp. 217–288 (online), DOI: 10.1137/090771806 (2011).

[21] Tomov, S., Dongarra, J. and Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Computing*, Vol. 36, No. 5-6, pp. 232–240 (online), DOI: 10.1016/j.parco.2009.12.005 (2010).

[22] Yalamanchili, P., Arshad, U., Mohammed, Z., Garigipati, P., Entschev, P., Kloppenborg, B., Malcolm, J. and Melonakos, J.: ArrayFire - A high performance software library for parallel computing with an easy-to-use API (2015).

[23] Collobert, R., Kavukcuoglu, K. and Farabet, C.: Torch7: A Matlab-like Environment for Machine Learning, *BigLearn, NIPS Workshop* (2011).

[24] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding, *arXiv preprint arXiv:1408.5093* (2014).

[25] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, (online), available from ⟨http://learningsys.org/papers/LearningSys_2015_paper_33.pdf⟩ (2015).