DEIM Forum 2016 E4-4

モバイルOS上での深層学習による画像認識システムの実装と比較分析

丹野　良介†　　柳井　啓司††

† 電気通信大学 情報理工学部
†† 電気通信大学大学院 情報理工学研究科
〒 182–8585 東京都調布市調布ヶ丘 1-5-1
E-mail: †tanno-r@mm.inf.uec.ac.jp, ††yanai@cs.uec.ac.jp

あらまし　近年，スマートフォンをはじめとするモバイル端末の急激な普及，高機能化・多機能化により，高い演算能力を必要とする高機能アプリケーションの実行が可能となってきた．また，画像認識や音声認識，自然言語処理などのパターン認識分野では，Deep Learning と呼ばれる多層のニューラルネットワークを用いた機械学習の手法の一つが，従来手法より極めて高い性能を見せ，様々なコンペティションで state-of-the-art な結果を得るなど大きな注目を集めている．これらの state-of-the-art な結果は，Graphics Processing Unit(GPU) の高い演算処理能力を前提にしており，ハードウェアリソースに制限が厳しいデバイスに Deep Learning を実装することは困難である．その為，今後はスマートフォンをはじめとしたリソースに制限があるデバイス上で動かすことができる Deep Learning の研究が求められる．本稿では，モバイルデバイス上に最新の画像認識手法である Deep Learning を用いて，高速かつ高精度に一般物体認識をすることができる認識エンジンを提案し，GPU やクラウドコンピューティングに依存しないスタンドアロンな認識システムを提案する．また，モバイル上に実装した画像認識システムを用いて，認識時間の計測を行い，認識システムの比較分析を行った．その結果，iOS では BLAS を用いた場合は 143ms，Android では NEON 命令を用いた場合は 251ms という結果が得られることがわかった．よって，iOS では BLAS の実装の Accelerate framework を用いることが最も最適な高速化手法であること，一方，Android では，NEON 命令による高速化が最も最適な手法であることが判明した．これにより，モバイルの特性を考慮した Deep Learning の実装をするための知見を明らかにすることができた．また，最新の CPU を搭載した iPhone 6s 上では 77ms で認識することが確認できた．

キーワード　画像認識，モバイル，アプリケーション，ディープラーニング，Deep Learning，

1. はじめに

近年，モバイル端末の高機能化により，高い演算能力を必要とするアプリの実行が可能となってきた．また，パターン認識分野では深層学習 (以下，Deep Learning) と呼ばれる多層のニューラルネットワークを用いた機械学習の手法の一つが，従来手法より極めて高い性能を見せるなど大きな注目を集めている．Deep Learning の応用としてモバイルデバイス上での画像認識が考えられるが，モバイル上に実装するには計算量がボトルネックとなる．その多くの計算量は各層の中でも特に畳込み層における畳込み演算の計算量が大部分を占めることから，畳込み演算を高速化することが，モバイル上で Deep Learning を実装する上で重要となる．しかし，モバイル上での Deep Learning に関する研究はあまり行われておらず，モバイルの特性を考慮した Deep Learning の実装について参考になる知見がほとんど存在しない．

そこで本研究では，モバイルデバイス上に Deep Learning によるスタンドアロン型の画像認識システムを実装することで認識システムの比較分析を行い，モバイル上で Deep Learning を実現するのに重要な要素を明らかにすることを目的とする．

2. 関連研究

Deep Learning を利用した物体認識システムはこれまで様々な研究がなされてきている．本研究ではモバイル上で Deep Learning を利用することから，ハードウェアリソースを考慮したシステムの設計が求められる．

そこで，本章では DCNN アーキテクチャ，Deep Learning を利用した物体認識アプリケーション，DCNN の高速化の 3 項目に関連した研究を紹介し，本研究との関係性や違いについて言及する．

2.1 DCNN アーキテクチャの研究

代表的な DCNN のアーキテクチャには AlexNet [1] や Network In Network(NIN) [2]，GoogleNet [3] などがある．また，現在，最先端の CNN アーキテクチャとして ImageNet Large Scale Visual Recognition Challenge 2015(ILSVRC 2015) の全ての部門でトップの精度を達成した Residual Network(ResNet) [4] がある．

AlexNet [1] は 5 つの畳込み層と 3 つのプーリング層，3 つの全結合層の全 11 層からなる CNN アーキテクチャであり，必要なパラメーター数は 6000 万と膨大な数である．AlexNet の

構造を図 1 として以下に示す．

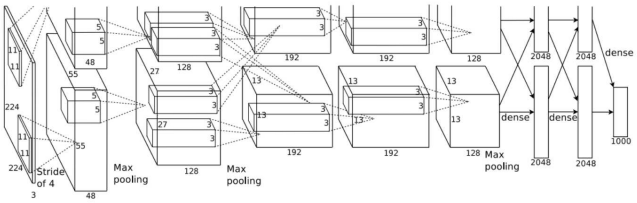

図 1　AlexNet のアーキテクチャ([1] から引用)

一方，NIN [2] は CNN アーキテクチャが畳込み層のみとなっており，全結合層がないため，必要なパラメーター数が約 750 万と AleNet と比較して大幅に少ない．しかし，性能は同程度を維持していることから，如何に畳込み層に不必要なパラメーターが非常に多いことがわかる．NIN の構造を図 2 として以下に示す．

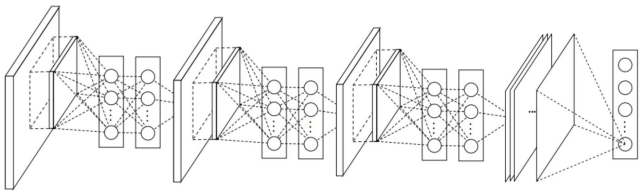

図 2　Network In Network(NIN) のアーキテクチャ([2] から引用)

この NIN をベースにして，更に層を深く設計したのが GoogLeNet [3] であり，22 層を持つ CNN アーキテクチャである．畳込み層における処理を多階層化し，異なるカーネルサイズの畳込み処理を並列に行うことで，より性能が高い CNN アーキテクチャを実現している．GoogLeNet の構造を図 3 として以下に示す．

最先端の CNN アーキテクチャとしては，ILSVRC 2015 の全ての部門で第一位を獲得した Microsoft の ResNet [4] がある．これは全 152 層からなる CNN アーキテクチャであり，残差 (Residual) を学習することで構造をより Deep にすることを可能としている．通常，CNN は層を深くすれば深くするほど特徴量の情報量が増えるが，“損失が急速に下がってしまう”，“エラー率が高くなってしまう” など深い層を収束させるのはこれまで困難であった．ResNet ではこの問題を残差 (Residual) により解決し，直接，各層を最適化するのではなく，残差 (Residual) を最適化することで，これまで以上に Deep な CNN アーキテクチャを構築した．ResNet の構造を図 4 として以下に示す．図 4 では残差 (Residual) と plain な CNN アーキテクチャの比較のため，層は 34 層で表示してある．

本研究ではモバイルでの実装を考慮し，パラメーター数が比較的少ない NIN の CNN アーキテクチャを利用している．

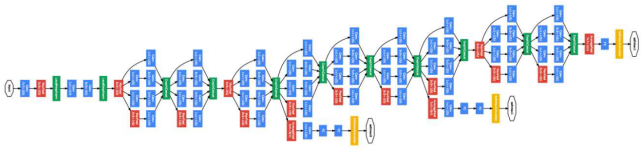

図 3　GoogLeNet のアーキテクチャ([3] から引用)

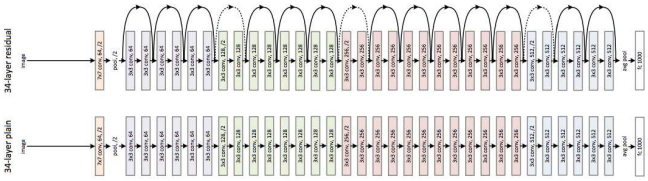

図 4　Residual Network(ResNet) のアーキテクチャ([4] から引用)

2.2　DCNN による物体認識アプリケーションの研究

Deep Learning の一種である Deep Convolutional Neural Network(DCNN) の性能を，気軽に体感できる物体認識アプリケーションとして，Toronto 大学がリリースした「Deep Learning」が存在する．アプリのデモを図 5(注1) として以下に示す．このアプリケーションは画像をサーバーに送信し，サーバー側で Deep Learning を行って，その結果を返して表示するサーバーアプリケーションであり，本研究が提案するスタンドアロンなアプリケーションとは属性が異なっている．

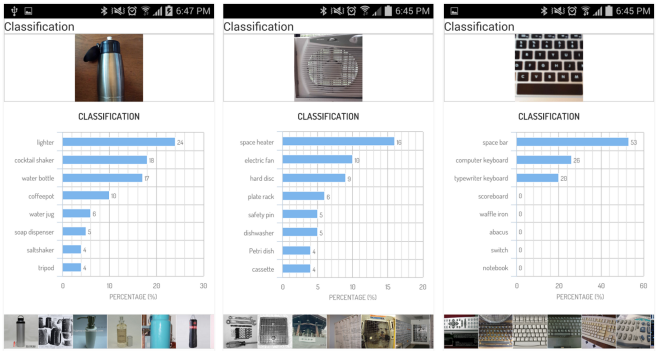

図 5　Toronto 大がリリースした物体認識アプリケーション「Deep Learning」

本研究で開発したアプリと同じ属性をもつスタンドアロンなアプリケーションとしては，JetPac 社の「JetPac Spotter(注2)」が過去に存在していた．2014 年 8 月に Google が JetPac 社を買収したことで，現在，アプリは非公開となっている．

(注1)：http://gallery.mobile9.com/asf/iMbVvMIF2th9/spotter-by-jetpac-object-recognition-real-time/ から引用
(注2)：http://spotterapp.tumblr.com/

しかし，「JetPac Spotter」に用いられていた認識エンジンはオープンソースの Deep Learning Framework「DeepBeliefSDK(注3)」として公開されており，このフレームワークを用いることで，簡単に既存のアプリケーションに Deep Learning を実装することができる．ただ，Convolutional Neural Network(CNN) のアーキテクチャに AlexNet [1] を用いているため，学習済パラメータが約 6000 万個も必要であることから，モバイルに実装するにはアプリ容量 (サイズ) が必然的に大きくなってしまい実装する上で問題になる．

また，本研究では画像認識システムとして，モバイル食事画像認識システムを開発した．本研究のような食事画像の認識に関する研究としては [5]，[6] などがある．

河野の研究 [5] では認識手法に HOG 特徴量と色特徴の 2 つの局所特徴量を Fisher Vector で表現し，線形 SVM で分類する認識エンジンでスマートフォン上でのリアルタイム高精度物体認識システムを実現している．図 6 が「FoodCam」であり，食事画像を認識して，カロリー付きの結果を表示する機能や，食事の量を右下のスライダーで調整することでカロリーの量を調整することもできる．また，食事記録をシステム内に登録することで，食事管理支援の機能も備わっている．

岡元の研究 [6] では河野 [5] の認識エンジンを Deep Learning に変更し，畳込み層の計算の工夫と NEON 命令 4core 並列実行により，従来手法である Fisher Vector+ 線形 SVM による認識エンジンと同程度の認識速度で，より高精度な物体認識を実現している．認識エンジンに Fisher Vector+ 線形 SVM を用いている従来版 Android アプリを「FoodCam」，認識エンジンを Deep Learning に改良した最新版 Android アプリを「DeepFoodCam」として FOODCAM(注4) で一般公開されている．図 7 に「DeepFoodCam」の認識画面を示す．

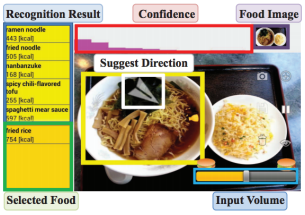

図 6　FoodCam [5]

図 7　DeepFoodCam [6]

本研究では岡元らの研究 [6] で利用している認識エンジンを用いて iOS 上に DeepFoodCam を実装し，iOS 及び Android 両 OS 上で Deep Learning を利用した画像認識システムの比較分析を行った．なお，本研究で開発したアプリは Android 版と同様に近日公開予定である．

(注3)：https://github.com/jetpacapp/DeepBeliefSDK
(注4)：http://foodcam.jp/

2.3　DCNN の高速化の研究

DCNN をリソースが制限されるモバイルコンピューティング上で使うために DCNN の高速化は必須であり，この分野は盛んに研究が行われている．例えば，処理に大部分の時間を要する畳込み層の計算の工夫に関する研究として [7]，[8]，[9] などがある．また，学習時のパラメーターにかかる重みに着目した高速化の研究として [10]，[11] などがある．

2.3.1　畳込み層の計算の工夫に関する研究

Gong ら [7] は Deep Learning をモバイル端末でも利用できるように，Vector 量子化によるパラメーター圧縮を行っている．1000 種類カテゴリ分類で，僅か 1%の損失に抑えて 16~24 倍の圧縮を達成する研究成果を残している．Liu ら [8] はスパース分解による DNN のパラメーター圧縮を行っている．また，CPU 上での効率的なスパース行列演算アルゴリズムとして Sparse Convolutional Neural Networks(SCNN) を提案し，物体検出に SCNN モデルを適用することで大幅な高速化を実現している．Jaderberg ら [9] は低ランク近似やフィルタ近似で，文字認識タスクにおいて精度低下 1%未満で 4.5 倍の高速化を実現している．

2.3.2　学習時のパラメーター重みに着目した高速化の研究

Courbariaux ら [10] は順方向及び逆方向伝搬時における重みを −1 or 1 に二値化することで，本来必要な演算の 2/3 を除去し，トレーニング時間を 3 倍にするという新しい高速化手法として “BinaryConnect” を提案している．Lin ら [11] は畳込み層の乗算をビットシフトに置き換える手法として “quantized back propagation” を提案し，先行研究である “BinaryConnect” を上回る性能を見せた．

3.　画像認識システム

本論文が提案する画像認識システムでは Deep Learning を認識手法に利用する．そこで，本章では DeepFoodCam(BLAS) 及び DeepFoodCam(NEON) の認識エンジン部分について説明する．

3.1　DCNN の学習

DCNN には一般的には AlexNet [1] を用いるが，モバイルに実装するにはアプリ容量に限りがあるなど問題が生じる．そこで，全結合層をもたない Network-In-Network(NIN) [2] のモデルを認識エンジンに利用している．

これにより AlexNet では約 6000 万もの大規模パラメーター数が必要だったところを，NIN を利用することで約 750 万のパラメーター数で済むなど，約 87.5%のパラメーター数の圧縮が可能である．NIN のモデルを利用することで大幅なパラメーター数の削減を実現しているが，認識性能については，1000 種類認識において AlexNet と同程度の認識性能を維持しており，モバイル実装を考慮すると，NIN のネットワークアーキテク

チャを利用することが妥当であると考えられる．

NIN のモデルを利用して，ILSVRC1000 種類と食事に関連した 1000 種類の ImageNet 画像 2000 クラス，計 210 万枚で pre-train を行い，そのモデルを UEC-FOOD100 の食事 100 クラスと，主に Twitter から収集した非食事画像 1 万枚，計 101 クラスで fine-tuning している．学習には最も有名な Deep Learning Framework である Caffe [12] を用いた．

3.2 DeepFoodCam(BLAS) の高速化手法

ここでは DeepFoodCam(BLAS) で用いられている高速化の工夫について説明する．[13] を元にして AlexNet の各層にかかる計算時間をグラフ化したものを図 8 として以下に示す．Deep Learning において，図 8 にあるように，各層の計算で最も計算量が大きい層は畳込み層であることから，各畳込み層における畳込み演算を工夫することで，Deep Learning 全体の計算時間の削減に大きく貢献し，システム全体の高速化に繋がる．よって，本項では畳込み層の演算の工夫として GEMM，Im2col，BLAS などを説明する．

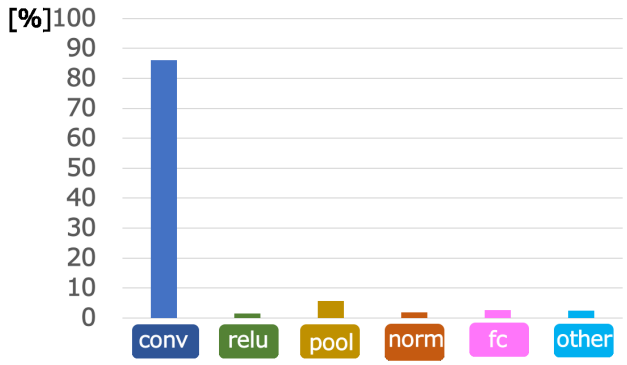

図 8　AlexNet の各層にかかる計算時間の内訳

3.2.1 Im2col

画像の 3 次元配列を行列のような 2 次元配列に変換することができれば，畳込み演算が行列積の計算に落としこむことができる．それを可能にするのが im2col である．im2col とは image-to-column の略であり，図 9(注5) にあるように，画像に畳込むフィルタのカーネルサイズと同じ大きさのパッチに画像を切り分け，画像のパッチを列行列に変換する操作のことである．この操作により，畳込み層における畳込み演算を図 10(注6) の行列積での計算に変換することができる．

3.2.2 Basic Linear Algebra Subprograms(BLAS)

im2col により Deep Learning における畳込み層の畳込み演算を，行列積で計算可能な形に変換した後は，行列積をいかに高速

(注5)：http://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/ から引用

(注6)：http://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/ から引用

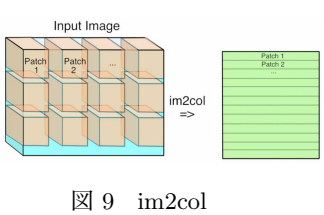

図 9　im2col

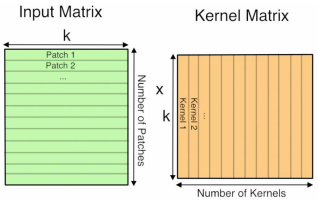

図 10　畳込み演算の行列積

に計算するかが高速化の鍵となる．DeepFoodCam(BLAS) ではこの行列積の計算に，線形代数演算ライブラリである BLAS の GEneric Matrix Multiplication(GEMM) 関数を用いている．

BLAS とは線形代数演算で用いられる基本的な演算を API 化したものであり，BLAS の演算性能によって以下の 3 つの演算に分類される．

Level1 BLAS

ベクトルの内積やベクトルの定数倍の加算などの演算を行う関数群である．

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y} \tag{1}$$

Level2 BLAS

行列とベクトルの積などの演算を行う関数群である．

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{A}\boldsymbol{x} + \beta \boldsymbol{y} \tag{2}$$

Level3 BLAS

行列と行列の積などの演算を行う関数群である．

$$\boldsymbol{C} \leftarrow \alpha \boldsymbol{A}\boldsymbol{B} + \beta \boldsymbol{C} \tag{3}$$

本研究では iOS 上に画像認識システムを実装していることから，BLAS ライブラリを Mac OS X 向けに実装した Accelerate Framework の SGEMM 関数 (cblas_sgemm) を使用している．SGEMM 関数は単精度浮動小数に対する行列積演算を行う関数のことである．

3.3 DeepFoodCam(NEON) の高速化手法

ここでは DeepFoodCam(NEON) で用いられている高速化の工夫について説明する．DeepFoodCam(NEON) は畳込み層での計算を SIMD 命令セットである NEON 命令による記述で高速化をしている．よって，本項では畳込み層の演算の工夫として NEON を説明する．

3.3.1 NEON

NEON とは ARM プロセッサの Single Instruction Multiple Data(SIMD) 命令セットであり，一の命令で複数の処理を可能にするベクトル処理機構のことである．畳込み演算を主に以下の式 4 及び式 5 の NEON 命令を使って高速化している．

ベクタ乗算

2 つのベクタの対応する要素を乗算して，デスティネーショ

ンベクタに結果を返す NEON 命令である．

$$V_r[i] := V_a[i] \times V_b[i] \tag{4}$$

ベクタ積和

2 つのベクタの対応する要素を乗算して、結果をデスティネーションベクタの要素に累積する NEON 命令である．

$$V_r[i] := V_a[i] + V_b[i] \times V_c[i] \tag{5}$$

式 4 及び式 5 の NEON 命令により，同時に 4 つの 32bit 単精度浮動小数点を演算させることができる．また，マルチプロセッサにより，iPhone 5s では 2 コア同時実行の合計 8 演算を同時に実行することができる．Android ではコア数が QuadCore の 4 コアであることが一般的なので，合計 16 演算を同時実行でき，畳込み演算を高速化している．

4. iOS 上への実装

画像認識システムに用いる認識エンジンは C++言語で書かれていることから，実装には Objective-C++ により実装を行った．本研究で開発した画像認識システムの構成画面を図 11 に示す．食事を認識して，カロリーを調整する機能があり，次の 6 つの GUI から構成されている．また，システムの流れを図 12 として以下に示す．

Recognition Result

認識結果を表示する部分である．認識した食事のカロリー，食事名，確率値の順に一行で表示し，トップ 1 からトップ 5 までの認識結果を表示する．

Select Recognition Result

Recognition Result に表示された食事認識結果の中からユーザは自由に結果を選択し，カロリーを保存することができる．

Total Calorie

Select Recognition Result で選んだ食事のカロリーの合計を表示する．

Select Mode

使用する認識エンジンを認識精度重視なのか認識速度重視なのか選ぶことができる．

Input Volume

ミディアムサイズを基準にして，スモールサイズからラージサイズまで 5 段階でカロリーを調整することができる．

Recognition Time

食事の認識にかかった時間を表示する．

5. 実　　験

本研究ではモバイル上での Deep Learning による画像認識システムの比較のために，今回は，“モバイルデバイス上での認識時間” に着目し，実装した画像認識システムを用いて画像認識時間の計測実験を行った．便宜上，実装したシステムを

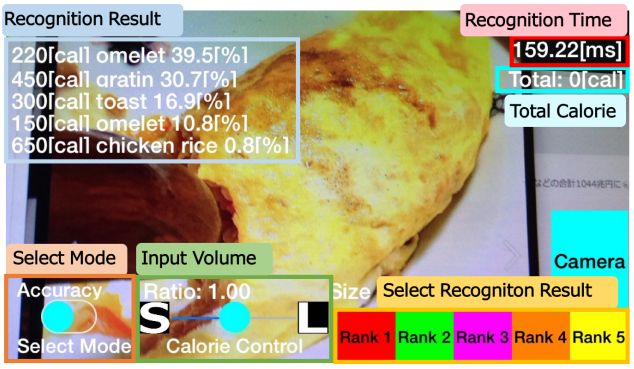

図 11　システム構成画面

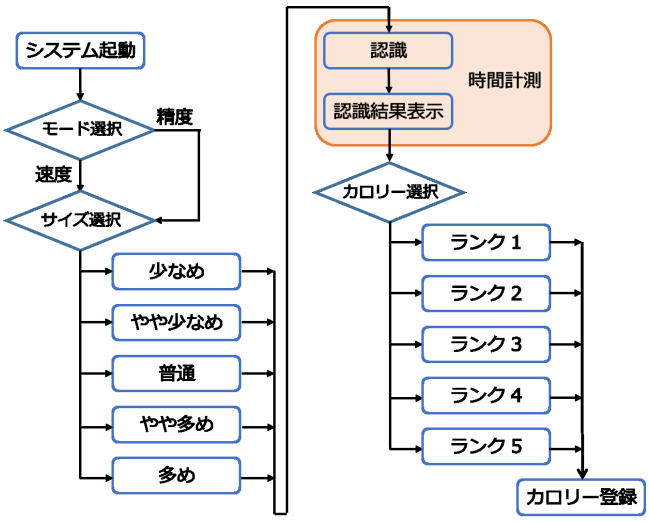

図 12　システムの流れ

iOS(BLAS) のように “OS 名 (フレームワーク名)” の形で呼ぶことにする．

5.1 実験の設定

実験に使用するフレームワークを以下に示す．

- DeepFoodCam(BLAS) 及び DeepFoodCam(NEON)
- DeepBeliefSDK(オープンソース)

3 つのフレームワークを利用してモバイル上に実装した画像認識システムによる認識速度の計測を行った．

DeepFoodCam による画像認識システムは 101 クラス (食事画像 100 クラス+非食事画像) の食事画像を認識することができるフレームワークを利用していることから，実験では食事画像を認識させ，画像 1 枚を認識するのにかかる処理時間を計測する．食事画像の例を図 13 として以下に示す．

実験では食事画像認識の対象物体として “実物の食品” ではなく，図 14 のような “食品サンプル” を認識させる．本来，認識には “実物の食品” を用いるべきではあるが，“食品サンプル” を用いても認識精度及び認識速度に違いは見られないこと，また，認識速度の計測回数が多く時間がかかることから，今回

図 13　認識することができる食事画像 100 種類 ([5] から引用)

図 14　実験で使用する食品サンプル群

は “食品サンプル” を認識対象とした.

一方，DeepBeliefSDK による画像認識システムは ILSVRC の 1000 クラス物体識別タスクと同じ 1000 クラスの物体を認識することができるフレームワークであることから，実験では非食事画像を認識させ，画像 1 枚を認識するのにかかる処理時間を計測した．非食事画像の例を図 15 として以下に示す.

図 15　ILSVRC の 1000 クラス物体識別タスクの例 ([14] から引用)

このように，DeepFoodCam は 101 種類の物体を，DeepBeliefSDK は 1000 種類の物体を認識することができる画像認識システムであり，分類種類数は大きく異なっている．しかしながら，実際には，DeepFoodCam に用いている 101 種類認識用の NIN は 1000 種類のネットワークを 101 種類にファインチューニングしたものであり，最終の畳込み層のカーネルサイズが 1000 から 101 に減少している以外は 1000 種類認識と同等のネットワークアーキテクチャである．よって，各層の計算量や計算時間は 1000 種類の場合とほぼ同等であると考えることができるため，本研究では DeepFoodCam 及び DeepBeliefSDK の画像認識システムの認識時間の比較を行うこととする.

また，用いるフレームワークにより認識可能な画像が異なることから，実験の詳細な設定を “共通設定” “DeepFoodCam の設定” “DeepBeliefSDK の設定” の 3 つに分けて記述する.

共通設定

実験の共通な設定として，実装に使用する評価用デバイスは同一のものを利用する．また，認識時間の計測はシステムの内部に記述し，認識の処理にかかる部分だけ計測する．計測回数は 1 実験につき 20 回行い，認識時間の平均を計測する．共通設定をまとめると以下のようになる.

（1） 評価用デバイスは同一のものを利用
（2） 計測時間は認識にかかる処理部分のみに限定
（3） 計測回数は 20 回

DeepFoodCam(BLAS) 及び DeepFoodCam(NEON) の設定

当研究室で開発した認識エンジンの性能を確かめるために，CPU で使用するコア数を iPhone 5s では 2 コア，GALAXY Note 3 では 4 コアとしデバイスに最適な CPU コア数を使用した．また，認識エンジンは “認識速度重視”，“認識精度重視” の 2 タイプ存在するが，今回は認識時間に着目していることから “認識速度重視” にエンジンを設定し実験を行った.．本設定をまとめると以下のようになる.

（1） 食事画像を認識させる
（2） CPU で使用するコア数はデバイスに最適なものとする
（3） 認識重視タイプを認識速度重視に固定する

DeepBeliefSDK の設定

認識速度の比較のために，オープンソースのフレームワークでは特別な設定を一切せずに，デフォルトのままのフレームワークを使用する．フレームワークで用意されている関数やファイルを単純に呼び出して画像認識させるシステムとし，ユーザ側では高速化の工夫などはしないこととする．本設定をまとめると以下のようになる.

（1） 非食事画像を認識させる
（2） デフォルトのままフレームワークを使用
（3） 高速化の工夫などはフレームワークの開発者に依存

認識速度計測実験の設定まとめ

これまでの実験の設定をまとめ，一覧表示したものを表 1 として以下に示す.

表 1　認識時間の計測実験の設定一覧

OS	実験番号	高速化手法	フレームワーク
iOS	(1)	BLAS	iOS(BLAS)
iOS	(2)	NEON	iOS(NEON)
Android	(3)	BLAS	Android(BLAS)
Android	(4)	NEON	Android(NEON)
iOS	(5)	BLAS	DeepBeliefSDK

5.2　評価用デバイス

認識速度の計測には iOS では iPhone 5s(CPU A7 1.3GHz RAM1GB DualCore iOS9.1)，Android では GALAXY Note 3(2.3GHz RAM3GB QuadCore Android5.0) を使用した．

5.3　実 験 結 果

iPhone 5s 及び GALAXY Note 3 を用いて，表 1 にある各実験の設定毎に計測を 20 回行い，平均の認識時間 [ms] を求めた結果を表 2 として以下に示す．表 2 では iOS 及び Android で認識時間が早かった方を青色に，遅かった方を赤色にしてある．

表 2　実 験 結 果

実験番号	平均認識時間 [ms]	最速 [ms]	最遅 [ms]	高速化手法
(1)	143	140	147	BLAS
(2)	503	494	509	NEON
(3)	1652	1564	1755	BLAS
(4)	251	224	281	NEON
(5)	418	396	443	BLAS

6.　考　　察

本章ではまず実験前の予想について記述し，実験前と実験後での予想の違いを明らかにする．その後，画像認識システムの比較分析を行う．

6.1　実験前の予想と結果

認識時間の計測実験を実施するまでは，iOS については

$$DeepBeliefSDK < iOS(NEON) < iOS(BLAS) \quad (6)$$

式 6 のように iOS(BLAS) が最も高速であり，次に iOS(NEON) のような結果が得られると予想していた．

iOS(DeepBeliefSDK) については BLAS による高速化により，C++コードが高度に最適化されており，GitHub の DeepBeliefSDK 公式ドキュメント (注7) によると iPhone 5s 上で 1 枚の画像認識に 300ms 程度で認識可能と記述してある．また，iOS(BLAS) 及び iOS(NEON) は PC 上の CPU において，前者は 100ms 程，後者は 200ms 程の実行時間を計測していたことから，上記のような認識時間の順位となると予想していた．

(注7)：https://github.com/jetpacapp/DeepBeliefSDK

また同様に Android についても，

$$Android(NEON) < Android(BLAS) \quad (7)$$

と iOS と同じ結果になると予想していた．

しかし，実験の結果，iOS 上での認識時間は

$$iOS(NEON) < DeepBeliefSDK < iOS(BLAS) \quad (8)$$

式 8 の結果となり，式 6 と反する結果が得られた．また，Android については

$$Android(BLAS) < Android(NEON) \quad (9)$$

式 9 の結果となり，式 7 と反する結果が得られた．表 2 の結果をグラフ化したものを図 16 として以下に示す．

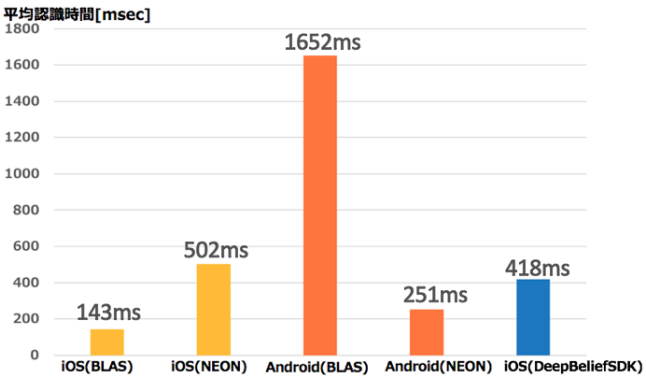

図 16　高速化の結果に違いがある

6.2　iOS と Android の認識時間に関する考察

iOS では BLAS の実装として Apple が提供している Accelerate framework を利用している．一方，Android では BLAS の実装として OpenBLAS を利用している．両者の違いとしては，デバイスに用いた BLAS の実装がデバイスに最適化されているか否かの違いにあると考えられる．Android 端末は汎用的な OS であり，Android 端末の開発元により実装の中身が異なっていることから，結果に違いが生じたと思われる．

表 3　iOS と Android の結果

	NEON	BLAS	デバイス	BLAS 実装
iOS	502[ms]	143[ms]	iPhone 5s	Accelerate
Android	251[ms]	1652[ms]	GALAXY Note 3	OpenBLAS

7.　お わ り に

7.1　ま　と　め

Deep Learning をモバイルに実装する上でボトルネックとなるのが計算量であり，モバイルに実装されているような CPU では Deep Learning に要する計算量を処理するのに時間がかかりすぎてしまう．その多くの計算量は各層の中でも特に畳込

み層における畳込み演算の計算量が大部分を占めることから，畳込み演算を高速化することが，モバイルデバイス上で Deep Learning による画像認識システムを構築する上で重要となっている．

そこで本研究では，まず，モバイルデバイス上で Deep Learning による画像認識システムを構築し，画像認識にかかる時間を実験により計測した．実験の結果，iOS では線形代数演算ライブラリの BLAS を使って高速化した認識エンジンが iPhone 5s 上で平均 143ms という認識時間の結果が得られた．一方，高速化手法に SIMD 命令セットの NEON 命令を使っている認識エンジンの平均認識時間は 502ms という結果が得られた．iOS 上では NEON 命令より BLAS の方が約 3.5 倍高速となることを実験により示した．また，Android では NEON 命令を使用した認識エンジンの平均認識時間は GALAXY Note 3 上で 251ms という結果が得られた．一方，BLAS により高速化工夫がしてある認識エンジンによる平均認識時間は 1652ms という結果が得られた．Android 上では NEON 命令の方が BLAS を使った場合より約 6.5 倍高速となることを実験により示した．

このように iOS 及び Android で正反対の実験結果が得られたことにより，モバイルデバイス上で高速に画像認識させるのに必要な高速化の工夫がモバイル OS で異なることを明らかにした．iOS では BLAS の実装の Accelerate framework を用いることが最も最適な高速化手法であるのに対し，Android では NEON 命令による高速化が最も最適な手法であることを明らかにした．

7.2 今後の課題

現在，普及しているスマートフォンなどのモバイルデバイス上には GPU が搭載されていることが一般的となっている．GPU は CPU よりもベクトル演算や並列処理に適していることから，様々な汎用演算を高速化するために利用されている．その為，今後は CPU だけではなく，モバイル GPU も利用した CPU と GPU による並列演算を行うことで認識エンジンの更なる高速化をしたいと考えている．単に物体を認識するのみならず，領域分割 (セグメンテーション) や動画像のリアルタイム画像処理など，より複雑な処理をシステムに実装するには更なる高速化が必要不可欠となってくるためである．また，現段階では学習モデルに Network In Network(NIN) を採用しているが，更に認識エンジンを高速化することができれば，GoogLeNet や VGG など，NIN よりも複雑な学習モデルを利用することができ，更なる精度向上も期待できる．よって，認識エンジンの更なる高速化は必須であると考える．

また，認識エンジンの応用も重要であり，その中でもカロリー量推定は非常に難しい分野である．今後は，2 次元画像から 3 次元形状を復元することにより，食べ物のカロリー量の推定に関する研究も課題としたい．

7.3 最新モバイル CPU での認識時間の参考値

本研究を踏まえ，DeepFoodCam(BLAS) 及び DeepFoodCam(NEON) を用いて，最新のモバイル CPU を搭載した iPhone 6s(CPU A9 1.84GHz RAM2GB DualCore iOS9.1) 上で認識時間を計測した結果を参考値として表 4 にまとめる．最新のモバイル CPU を用いることで 100ms を切ることができた．依然として，NEON 命令による高速化は BLAS よりも 3 倍ほど遅いが，iOS デバイスが Quad Core に変更になれば効果を発揮すると思われる．

表 4 iPhone 6s 上での認識時間の計測結果

デバイス	NEON	BLAS
iPhone 6s	255[ms]	77[ms]

文　献

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[2] M. Lin, Q. Chen, and S. Yan. Network in network. In *Proc. of International Conference on Learning Representations*, 2014.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc.of IEEE Computer Vision and Pattern Recognition*, 2015.

[4] K. He, X. Zhang, S. Ren, and J Sun. Deep residual learning for image recognition. In *Proc. of arXiv:1512.03385*, 2015.

[5] Y. Kawano and K. Yanai. Real-time mobile food recognition system. In *Proc. of CVPR International Workshop on Mobile Vision (IWMV)*, 2013.

[6] 岡元晃一, 柳井啓司. DeepFoodCam: DCNN による 101 種類食事認識アプリ. 画像の認識・理解シンポジウム (MIRU), 2015.

[7] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. In *Proc. of International Conference on Learning Representations*, 2015.

[8] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proc. of IEEE Computer Vision and Pattern Recognition*, 2015.

[9] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. of arXiv: 1405.3866*, 2014.

[10] M. Courbariaux, Y. Bengio, and J. P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 2015.

[11] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. In *Proc. of arXiv:1510.03009*, 2015.

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of arXiv:1408.5093*, 2014. http://caffe.berkeleyvision.org/.

[13] Y. Jia. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, EECS Department, University of California, Berkeley, May 2014.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision (IJCV)*, 2015.