社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

CoreMLによるiOS深層学習アプリの実装と性能分析

丹野　良介† 　泉　 裕貴† 　柳井　啓司†

† 電気通信大学 大学院情報理工学研究科 情報学専攻
E-mail: †{tanno-r,izumi-y,yanai}@mm.inf.uec.ac.jp

あらまし　CoreMLとは，iOS11から新しく追加される機械学習API群である．既存のフレームワークであるKerasやCaffeで学習したモデルをiOS用にコンバートすることで，Basic Neural Network Subroutines(BNNS, CPU実行)やMetal Performance Shaders(MPS, GPU実行)を利用した推論処理をiOS上で実現可能である．CoreML自体はBNNSやMPSより高レイヤーな概念であり，複雑な計算処理を隠蔽して実装でき，どの程度のパフォーマンスが得られるのか，また，新しいAPI群のため文献がまだ少なく，実際にどういった実装ができるかなど未知数なところが大きい．そこで本報告では，発展目覚ましいCoreMLに対し，弊研究室が開発したオリジナル順伝搬ライブラリとの比較を通して，性能分析を行うとともに，実装事例の紹介も行う．

キーワード　深層学習, アプリケーション, iOS, CoreML

Implementation and Performance Analysis of iOS Deep Learning Application using CoreML

Ryosuke TANNO†, Yuki IZUMI†, and Keiji YANAI†

† Department of Informatics, The University of Electro-Communications
E-mail: †{tanno-r,izumi-y,yanai}@mm.inf.uec.ac.jp

1. はじめに

一般的に，Deep Learningの学習にはGPUが必須であり，多くの計算リソースを必要とする．しかし，学習済のモデルを用いて推論する場合は，そこまで多くの計算資源は必要ではなく，iPhone上でも十分に利用可能である．

通常，CaffeやChainerなどの深層学習フレームワークで学習したモデルをiOSで使う場合，主に次の3つの手段が考えられる．下に行くほど手順が複雑になり自身で用意するハードルが高くなる．しかし，その分細かい部分のチューニングも可能となるので高いパフォーマンスが期待できるが，深層学習に要する各レイヤの実装なども自分で実装するので難しい．

（1） フレームワーク自体がiOS上で動くように設計されている．または，ラッパーが用意されている．(例: Caffe for iOS, TensorFlow, OpenCV)**(Easy)**

（2） 既存のフレームワーク上に設計され，高レベルAPIを提供し，モバイルへのデプロイが考慮されているモジュラーがある．または，モバイル上で動作可能にする各種コンバータが存在する．Appleが提供するNeural Network APIを利用する．(例: Caffe2, Caffe2Go, MPS, BNNS)**(Intermediate)**

（3） 学習済みモデルファイルをiOS上で使えるように変換し，推論部分を独自で実装する．(例: Caffe2C(Chainer2C) [15])**(Advanced)**

このように，モバイル上で学習済みモデルを利用する際には工夫が必要であった．一方で，iOS11の一般公開 (日本時間2017年9月20日) に伴い，機械学習API群「CoreML」が新しく追加された．学習済みのモデルをCoreML用に変換するコンバータ「coremltools」も同時に公開し，これまでより簡単に深層学習をアプリに実装できるようになった．しかしながら，新しいAPI群のため文献がまだ少なく，実際にどういった実装ができるか，処理速度は実運用に耐えうるか，など未知数なところが大きい．

そこで本報告では，発展目覚ましいCoreMLに対し，弊研究室が独自に開発したオリジナル順伝搬ライブラリとの比較を通して，性能分析を行うとともに，CoreMLを用いたアプリ実装例の紹介も行う．

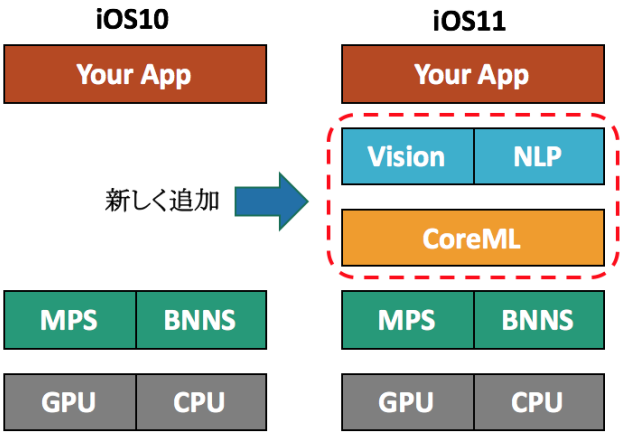

図 1　CoreML は MPS 及び BNNS のラッパー的立ち位置.

2.　深層学習 iOS アプリ実装法

学習済みモデルを利用する手段は 1. 章で述べた通り色々な方法がある．本報告では，CoreML を用いた実装とオリジナル順伝搬ライブラリを用いた 2 種の実装の処理速度ベンチマークを取ることから，この 2 種について言及する．

2.1　CoreML

CoreML(ML: Machine Learning) とは，iOS11 から追加された機械学習 API 群のことであり，学習済みモデルを iOS 上で推論できるようにモデルを変換して利用する．本報告では，深層学習部分のみ言及するが，本来は，scikit-learn, XGBoost, libsvm などもモデル変換をサポートしている．

注意して欲しいのが，iOS11 から何か特別できる事が増えたというわけではなく，CoreML が登場する前の iOS10 でも学習済みモデルを利用できる点である．図 1 に iOS10 と iOS11 の差異を示すが, CoreML はあくまで Metal Performance Shaders(MPS) と Basic Neural Network Subroutines(BNNS) のラッパーに過ぎない．iOS には，GPU で高速計算を可能にする Metal，CPU で高速計算を可能にする Accelerate という両フレームワークが存在し，iOS10 ではそれぞれにニューラルネットワーク API 群が追加された．それが MPS(GPU 利用) と BNNS(CPU 利用) の 2 つである．よって，iOS10 時点でも深層学習ライクなアプリを実装することは可能ではあったが，CoreML ほど話題にはならなかった．理由は以下の点が考えられる．

- 1. 章の手段 3 に該当し，MPS や BNNS を用いてレイヤを自分で書く必要がある．
- ネットワークの実装のみでプログラムコードが数千行になることがある．

上記のネック部分を改善したのが CoreML であり，実装が面倒な部分を隠蔽して学習済みモデルを利用できるため，開発者側からすると非常に助かる仕様となった．その為，以前と比べて格段に話題に挙がっているのだと推測できる．

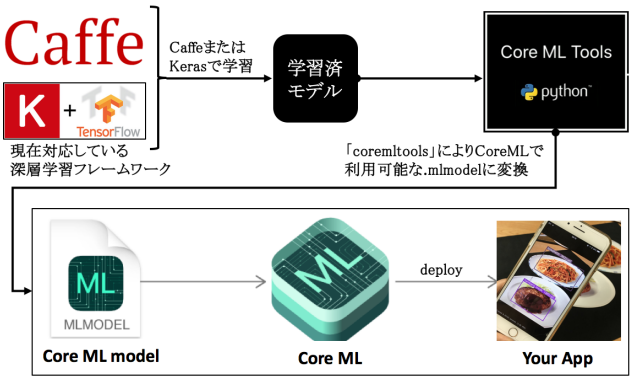

図 2　CoreML を用いた iOS アプリ deploy フロー.

CoreML を用いたアプリ実装フローを図 2 として以下に示す．CoreML が現在サポートしている深層学習フレームワークは Caffe 及び Keras(TensorFlow backend のみ．Keras は Tensor 演算の backend に Theano も選べるが未対応なので注意) の 2 つであり，学習したモデルを「coremltools(注1)」により CoreML 用に変換して利用する．学習済みモデルを変換するコンバータの存在，また，CNN の各レイヤー実装などは CoreML により隠蔽して利用できるため，iOS10 と比べて簡単に利用できるようになった．上述の通り，coremltools がサポートしているのは Caffe 及び Keras であるが，「torch2coreml(注2)」を利用することで，Torch7 で学習したモデルも CoreML 用に変換できる．

一方で，1. 章 手順 3 にある通り，自分で全て担う方法も存在し，2. 2 節では我々の実装手順を紹介する．

2.2　オリジナル順伝搬ライブラリ

本節では，CoreML を用いない場合の CNN の利用法について言及する．CNN をモバイルに実装する上で高速化は必須であり，我々は今まで深層学習モデルコンバータ [15] や効率的な CNN のモバイル実装 [17] の研究成果がある．

2.2.1　深層学習モデルコンバータ: Caffe2C(Chainer2C)

iOS で学習済みモデルを利用するためには，CoreML 登場以前では，モデルの変換やモデルのラッパーなどを使うなど工夫が必要であった．しかし，オープンソースで利用できるライブラリはモバイル用の最適化が不十分であり，処理速度が十分とは言えない．

そこで，本研究では Caffe(Chainer) で学習した後に生成されるパラメータファイルから，モバイルでも実行可能な C 言語コードを自動生成することができるモデルコンバーター Caffe2C(Chainer2C) を作成した．コンバータにより生成される C 言語インターフェイスは，モデル定義ファイルが除去され，全ての学習済みパラメータが定数配列として C コードに組込まれる．よって，本コンバータを使用することで，学習済みモデルを実行するために必要なファイルを全て含んだ C 言語インターフェイスを生成できるため，C 言語動作環境上であれば

(注1)：https://pypi.python.org/pypi/coremltools
(注2)：https://pypi.python.org/pypi/torch2coreml/0.0.4

学習済みモデルを使用することが可能になる．

2.2.2 CNN の高速化

CNN の演算の大部分は畳込み演算が占めることから，図 4 の im2col 操作を行い，畳込み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ことで，畳込み演算を行列積 (Generic Matrix Multiplication, GEMM) で計算することができる．変換後は BLAS ライブラリにその処理を任せることになるが，iOS の BLAS 実装である Accelerate Framework を利用した．

2.2.3 CoreML との差異

2.2.2 項にある通り，我々の順伝搬ライブラリでは Accelerate Framework を使っていることから，CPU のみでの実行となっており GPU は使っていないことに注意してほしい．

3. ベンチマーク

実用的なアプリを作成する際，予測精度の他，処理速度，メモリ使用量は利用する深層学習モデル選択の重要な物差しである．今回は，画像認識タスク及び画像変換タスクにおける端末上での処理速度の計測を行うことにした．認識では iPhone7Plus，変換では iPad Pro2017 を用いた．また，画像認識では代表的な認識モデル 11 種，画像変換ではスタイル変換ネットワークを題材に取り上げた．認識ネットワークから Network-In-Network，画像変換では ConvDeconvNet を用いて，CoreML と前章で取り上げたオリジナル順伝搬ライブラリによる処理速度の比較を行った．

3.1 画像認識ネットワーク

多種多様なネットワーク構造がある中で，今回は，良く利用される以下の 11 種のモデルを CoreML 上で用いた．(NIN: Network-In-Network, BN: Batch Normalization と略記．) 学習には Keras を利用し，元論文ほぼそのままの実装を心がけた．

- AlexNet [7]
- AlexNet+BN [7]
- NIN [8]
- NIN+BN [8]
- GoogLeNet [13]
- VGG16 [12]
- VGG19 [12]
- Inception-v3 [14]
- ResNet50 [2]
- SqueezeNet [4]
- MobileNet [3]

ネットワークの入力とする画像サイズは基本的には 224x224 を Input とするが，Inception-v3 は 299x299，SqueezeNet は 227x227 を Input とすることに注意してほしい．

CoreML による推論結果及び NIN+BN のオリジナル順伝搬ライブラリによる推論結果を図 3 として以下に示す．CoreML では GPU を用いているにも関わらず，CPU 実行と大差ない結果となった．これは GPU のオーバーヘッドに起因するものと推測される．

3.2 画像変換ネットワーク

近年，Variational AutoEncoder(VAE) や Generative Adversarial Network(GAN) をはじめとする生成モデルが注目を集めている．生成モデル系では，訓練データと生成データの分布が一致するように学習することで，新しいデータを生成することを可能とするようなモデルを指す．生成系に関するネットワークは多数存在するが，今回は，高速に画風変換を行う手法 [6] を iOS 上に実装し，図 5 のような ConvDeconvNet のベンチマークを計測した．

表 1 に CoreML 及びオリジナル順伝搬ライブラリで用いる ConvDeconvNet のネットワーク構成及び推論結果を示す．CoreML の実装では GPU 利用が前提なので元論文 [6] そのままのネットワークを deploy するのに対し，オリジナル順伝搬ライブラリでは CPU のみでの実行なので，[6] のネットワークから以下の縮小を行うことで，モバイル上でリアルタイムに動作するようにした．

（1） down-sampling layer と up-sampling layer を追加．

（2） 最初と最後の conv layer のカーネルを 9x9 を 5x5 に変更，他全ての conv のカーネルを 3x3 → 4x4 に変更．

推論結果を参照すると，CoreML で用いた ConvDeconvNet はネットワーク縮小を行っていないにも関わらず，オリジナル順伝搬ライブラリよりも処理速度が早い結果となった．画像変換ネットでは 640x480 の認識ネットと比べてかなり高解像度の画像を入力としているため，GPU のオーバーヘッドによる遅延が処理速度のパフォーマンスに比べてかなり小さくなったことが要因と推測される．

表 1 ベンチマークに用いるネットワーク構成の比較．Input は 640x480.

CoreML	オリジナル順伝搬ライブラリ
Reflection Padding (40x40)	5x5x16 conv, stride 1
9x9x32 conv, stride 1	4x4x32 conv, stride 2
3x3x64 conv, stride 2	4x4x64 conv, stride 2
3x3x128 conv, stride 2	4x4x128 conv, stride 2
Residual block, 128 filters	Residual block, 128 filters
Residual block, 128 filters	Residual block, 128 filters
Residual block, 128 filters	Residual block, 128 filters
Residual block, 128 filters	Residual block, 128 filters
Residual block, 128 filters	Residual block, 128 filters
3x3x64 conv, stride 1/2	4x4x64 conv, stride 1/2
3x3x32 conv, stride 1/2	4x4x32 conv, stride 1/2
9x9x3 conv, stride 1	4x4x16 conv, stride 1/2
	5x5x3 conv, stride 1
512[msec]	841[msec]

4. 実 装 例

4.1 マルチスタイル変換 [16]

事前に特定のスタイルの変換を feed-forward で行う CNN

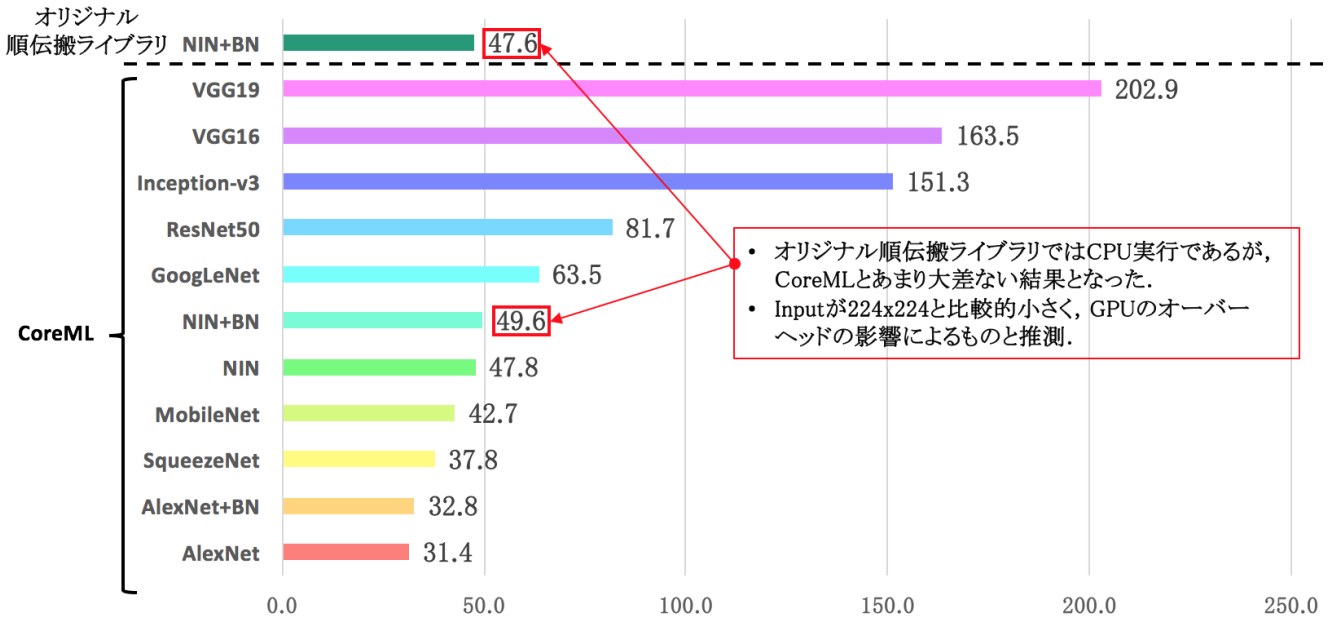

図 3　画像認識ネットワークの推論結果．単位は [msec]．

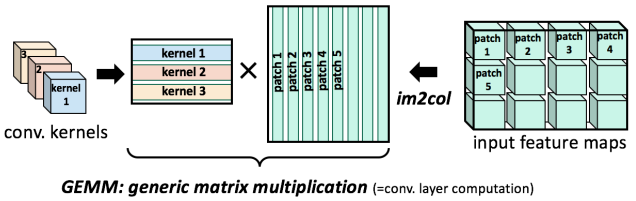

図 4　im2col 操作により畳込み演算を行列積に変換する．

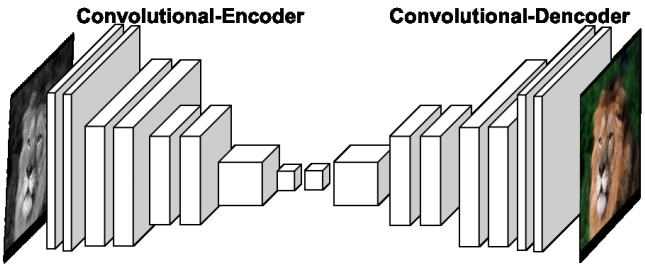

図 5　ConvDeconvNet の構造

を学習しておくことで，ほぼリアルタイムで画像を変換可能にする手法が提案された [6]．この手法では，事前に学習した単一のスタイルのみを転送することしかできなかった．そこで，本実装では，[6] のネットワークを複数種類のスタイルを任意の重みで合成して，リアルタイムで合成したスタイルを転送する手法に拡張した．アプリの動作例を図 6 として以下に示す．

4.2　マテリアル変換

pix2pix [5] は GAN で提案された Adversarial Loss と ConvDeconvNet を組み合わせることで，画像のペア集合間の変換方法を学習することができる．(図 7 参照) しかし，画像変換ペアを用意する必要があるなどデータセット上の縛りが存在した．一方，CycleGAN [18] は pix2pix では入出力に画像ペアが必要

図 6　multi-style feed-forward network を用いたアプリの動作例．画面右側のスライダーを動かすことで任意の重みでスタイルを合成し，リアルタイムに変換可能である．

だったところを，ペアが無くとも上手く画像生成できるように cycle consistency loss を導入して拡張することで，画像変換のペア集合間の写像を学習可能となった．画像のペアではなく集合を用意すれば良いので，データセットを用意するのも用意である．

よって，マテリアル変換では，CycleGAN を用いて ConvDeconvNet を学習することで，質感画像の材質を変換可能なアプリを実装した．例えば，本アプリを用いることで図 9 のような Metal → Fabric, Leather → Glass といった質感の変換を行える．また，CycleGAN の学習イメージを図 8 として以下に示す．

4.3　文字隠蔽

情景画像中の文字を隠蔽するネットワーク [10] を実装した．文字有画像を入力すると，文字部分が隠蔽されて出力されるように学習を行う．CNN の構造は図 10 の U-Net を利用し，画像サイズが同じものを深い層から段階的に統合することで，局

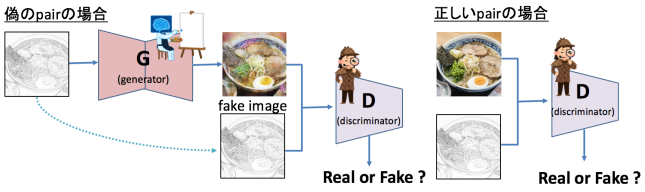

図 7　pix2pix の学習イメージ．pair 画像間の違いを学習して，その差を補う形で画像を出力．

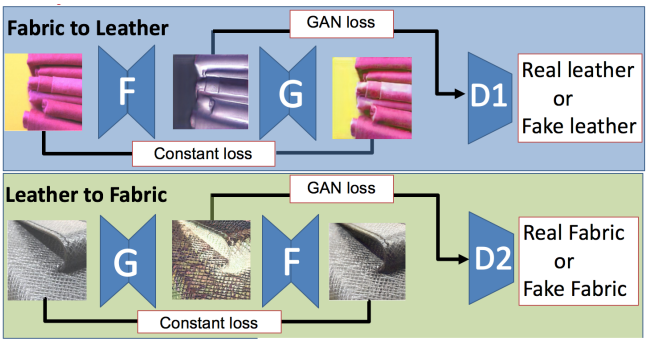

図 8　Cycle GAN による素材クラス変換 (マテリアル変換)

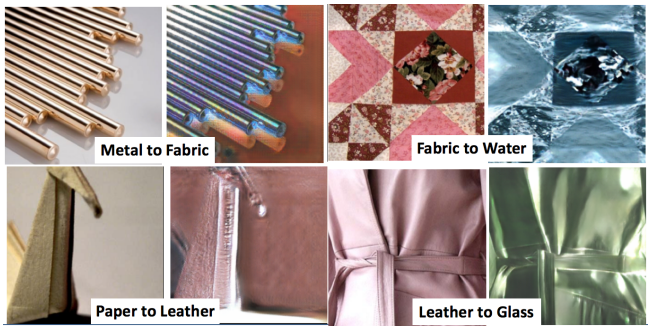

図 9　マテリアル変換例

所的特徴を保持したまま全体的位置情報の復元が可能となる．図 11 に変換結果を記載した．

4.4　複　合　系

食事検出とカロリー推定を組合せたアプリ「DeepCalorieCam」を紹介する．本アプリの処理イメージが図 12 であり，カロリー値を推定するワークフローは以下の通りである．また，アプリの使用例を図 13 として以下に示す．

（1）　YOLOv2 [11] を用いて食事検出をする．

（2）　検出された各食事領域のバウンディングボックス情報から画像をクロップする．

（3）　クロップした各食事画像をカロリー値を推定する CNN の入力とする．

食事検出器の学習は YOLOv2 [11] を UEC FOOD-100 データセット [9] で fine-tune を行った．カロリー値を推定する Multi-task CNN は [1] に詳細が記載してあるが，端的に述べると，[1] にあるカロリー量付き食事画像データセットを用いて，Inception-v3 を fine-tune している．フレームワークは Keras を利用し，「coremltools」で.mlmodel に変換後のモデル

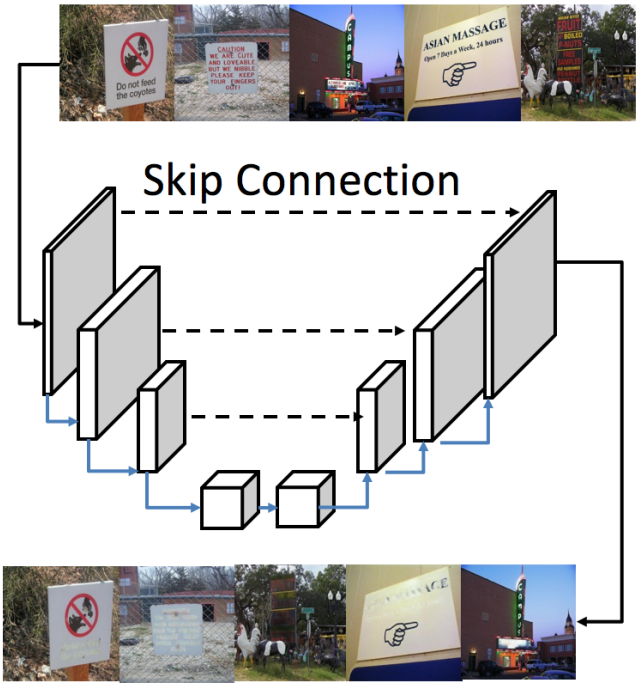

図 10　文字隠蔽ネットワーク [10]

を CoreML で利用している．

5.　ま　と　め

本報告では，深層学習 iOS アプリの実装法について言及した．また，Apple が 2017 年 9 月に公開した iOS11 から追加される機械学習 API 群「CoreML」について，オリジナル順伝搬ライブラリとの比較を通して実機上での処理速度に焦点を絞りベンチマークを測定することで性能分析を行った．

その結果，低解像度画像 (224x224) をネットワークの入力とする場合は CoreML では GPU を用いているにも関わらず，CPU 実行と大差ない 結果となった．これは GPU のオーバーヘッドによるものと考えられる．一方で，高解像度画像 (640x480) を用いる場合は，GPU のオーバーヘッドによる遅延が処理速度のパフォーマンスに比べてかなり小さくなることがわかった．

CoreML の登場以降，iOS 上で深層学習を用いたアプリの開発は今後加速すると予測される．モバイル+深層学習の今後の展望として，AR(拡張現実) との融合は有望であると考える．よって，今後の課題として，4.4 節で述べたアプリに AR を組合せることで，スマートフォンをかざすだけで空間上にカロリー値を記録する AR カロリー推定を実装予定である．

文　　献

[1] Ege, T. and Yanai, K.: Simultaneous Estimation of Food Categories and Calories with Multi-task CNN, *Proc. of ACPR International Conference on Machine Vision Applications (MVA)* (2017).

[2] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *Proc. of IEEE Computer Vision and Pattern Recognition* (2016).

[3] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang,

図 11　文字隠蔽ネットワークによる変換結果．(左) 入力画像 (中) 変換結果 (右) 正解画像

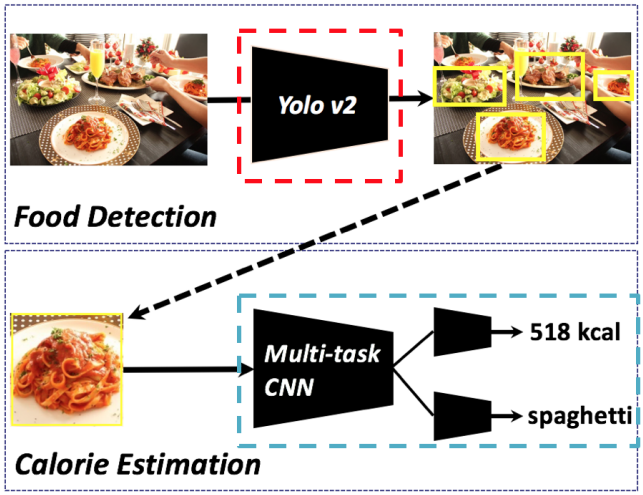

図 12　「DeepCalorieCam」の処理イメージ．

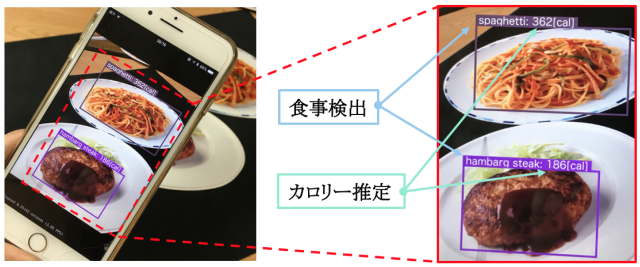

図 13　「DeepCalorieCam」の使用例．

W., Weyand, T., Andreetto, M. and Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, *Proc. of arXiv:1704.04861* (2017).

[4] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. and Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, *Proc. of arXiv:1602.07360* (2016).

[5] Isola, P., Zhu, J., Zhou, T. and Efros, A. A.: Image-to-Image Translation with Conditional Adversarial Networks, *Proc. of IEEE Computer Vision and Pattern Recognition* (2017).

[6] Johnson, J., Alahi, A. and Fei, L.: Perceptual Losses for Real-Time Style Transfer and Super-Resolution, *Proc. of European Conference on Computer Vision* (2016).

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

[8] Lin, M., Chen, Q. and Yan, S.: Network In Network, *Proc. of International Conference on Learning Representations* (2014).

[9] Matsuda, Y., Hoashi, H. and Yanai, K.: Recognition of Multiple-Food Images by Detecting Candidate Regions, *Proc. of IEEE International Conference on Multimedia and Expo (ICME)* (2012).

[10] Nakamura, T., Zhu, A., Yanai, K. and Uchida, S.: Scene Text Eraser, *Proc. of The 14th International Conference on Document Analysis and Recognition* (2017).

[11] Redmon, J. and Farhadi, A.: YOLO9000: Better, Faster, Stronger, *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[12] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *Proc. of International Conference on Learning Representations* (2015).

[13] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper with Convolutions, *Proc.of IEEE Computer Vision and Pattern Recognition* (2015).

[14] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z.: Rethinking the Inception Architecture for Computer Vision, *Proc. of arXiv:1512.00567* (2015).

[15] Tanno, R. and Yanai, K.: Caffe2C: A Framework for Easy Implementation of CNN-based Mobile Applications, *Proc. of International Workshop On Mobile Ubiquitous Systems, Infras- tructures, Communications, And AppLications (MUSICAL 2016)* (2016).

[16] Tanno, R. and Yanai, K.: DeepStyleCam: A Real-time Style Transfer App on iOS, *Proc. of International MultiMedia Modeing Conference (MMM)* (2017).

[17] Yanai, K., Tanno, R. and Okamoto, K.: Efficient Mobile Implementation of A CNN-based Object Recognition System, *Proc. of ACM Multimedia* (2016).

[18] Zhu, J. Y., Park, T., Isola, P. and Efros, A. A.: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, *Proc. of IEEE International Conference on Computer Vision (ICCV)* (2017).