ONNX2MPSNNGraph:モバイル深層学習コードジェネレータの実装と評価

泉 裕貴[1,a)]　柳井 啓司[1,b)]

概要

近年，モバイル端末の性能向上により，サーバーを介さない端末上での深層学習の実行が可能となり，実装を容易にするようなツールなども存在している．しかし，それらは実装がブラックボックス化され拡張が困難であるものや特定のフレームワークで学習したモデルしか対応してないものであった．そこで本研究では，多数の深層学習フレームワークに対応し，アプリケーションの実装後の修正や最適化が可能な深層学習モデルコンバータ ONNX2MPSNNGraph を作成した．

1. はじめに

iPhone を始めとするモバイル端末の性能は年々向上し高い演算能力を持つようになったため，サーバーを介さない端末上での学習済みモデルを用いた深層学習の実行が可能となってきた．しかし，最新のネットワークは枝分かれのあるものや非常に深い構造を持ったものなど複雑なものが多い．このような複雑なネットワークを用いたアプリケーションを開発しようとした場合，ネットワークの部分のみで数百行にも及ぶような長文且つ難解なコードを書く必要があり，人手で書くことは困難である．そこで，大変な作業量となるコーディングやパラメータの受け渡しなどといった部分を自動的に行う CoreML[*1] や Chainer2MPSNNGraph[2] といったフレームワークやコンバータが存在する．しかし，CoreML は内部の実装をブラックボックス化しているために拡張が困難であり，Chainer2MPSNNGraph[2] は利用できる学習済みモデルが Chainer[3] で学習されたもののみで適応範囲が限られているといった問題がある．

本研究では，Open Neural Network Exchange(ONNX)[*2] のモデルから MPSNNGraph API を用いたコードやパラメータファイルの生成を行う，深層学習モデルコンバータ ONNX2MPSNNGraph を作成した．これにより，既存の数多くの深層学習フレームワークからの変換が可能になり，GPU を用いた実行で高速性を保ちつつ，コードの修正や最適化を可能とした．

[1] 電気通信大学 大学院情報理工学研究科 情報学専攻
[a)] izumi-y@mm.inf.uec.ac.jp
[b)] yanai@cs.uec.ac.jp

[*1] https://developer.apple.com/documentation/coreml
[*2] https://onnx.ai

2. 実装方法

iPhone 上で深層学習の演算処理を行うモバイルアプリケーションを作成する実装方法は複数存在する．

2.1 Metal Performance Shaders

Metal Performance Shaders (MPS) は，iOS9 で追加され iPhone に搭載されている GPU である Metal GPU を利用し，GPU 上で画像処理や行列演算などを実行させる計算カーネルのライブラリである．MPS フレームワークを用いることで，Metal GPU を容易に利用できるようになっている．

MPS の中には，iOS11 で追加された MPSNNGraph というクラスがある．これを用いてニューラルネットワークのグラフを構築することで，グラフ内のノードのうち出力を得るために必要のないノードを無視するようになり，ノード同士が内部で連結するといったことを自動的に行うことによって，より高いパフォーマンスを見込めるものとなっている．

2.2 CoreML

CoreML は iOS11 から新しく追加された機械学習向けライブラリであり，図 1 にあるように GPU を利用した高速演算処理を実現する MPS と CPU 上でエネルギー効率の高い計算を提供できるように最適化された Accelerate ライブラリのラッパーである．CoreML では，coremltools を用いて学習済みモデルを CoreML 用のモデルファイルに変換することで，それまでニューラルネットワークを MPS などで実装する場合に数百行にも及ぶことがあるコーディングを必要とせず，実装内でそのモデルファイルを読み込むのみで容易に深層学習を実装できるようになった．しかし，CoreML でどのような実装，処理を行っているかは不明であり，モデルファイル作成後に内部に手を加えることはできない．

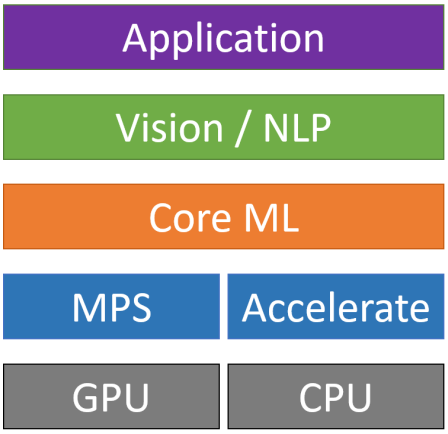

図 1　CoreML の位置づけ

3. ONNX2MPSNNGraph

深層学習のアプリケーションを作成しようとした場合に，モデルネットワークによってはニューラルネットワークの部分のみで数百行のコーディングが必要であり，その中でモデルパラメータの受け渡しやオプションの設定なども必要であるため，非常に手間がかかる．そこで本研究では，その複雑な部分を自動的に生成するコンバータ ONNX2MPSNNGraph を作成した．

ONNX2MPSNNGraph は，ONNX の学習済みモデルを用いてアプリケーション作成に必要なモデルのネットワークコードとパラメータファイルを生成する．Chainer2MPSNNGraph[2] では，対応している学習済みモデルは Chainer のみであったが，ONNX2MPSNNGraph では，ONNX という Chainer[3] や Pytorch[*3]などの深層学習フレームワークの共通プラットフォームの学習済みモデルに対応することによって，多数のフレームワークで学習したモデルからモバイルアプリケーションを作成できる．

3.1 ONNX のネットワーク構造

Chainer2MPSNNGraph[2] では，Chainer で学習したモデルにしか対応していなかった．Chainer は，Define-by-Run という特徴を持っている．これは，計算グラフの構築をデータを流しながら行うため，順伝搬計算を一度行いその出力結果から計算グラフを遡るかたちで重みなどのパラメータやネットワーク構造を取得し，コードを生成している．そのため，コンバータにはモデルファイルの他にネットワークファイルが必要である．

それに対し，ONNX は Define-and-Run の特徴を持った Keras[*4]などのフレームワークにも対応できるため，ONNX のモデルファイル自体にネットワーク構造を持っているため，モデルファイルのみでのコード生成が可能である．読み込んだモデルの graph コンポーネントの node プロパティによってネットワーク構造の全体を参照することができる．また，graph コンポーネントの initializer プロパティによって，重みやバイアス等のパラメータを取得することが可能である．ここで，ノードとパラメータを結びつけたリストを作成し，リストからアプリケーション作成に必要なコードを生成する．

3.2 アプリケーション作成の流れ

本研究で作成したコンバータ ONNX2MPSNNGrapgh を用いた深層学習モバイルアプリケーションの作成は以下の手順で行い，図 2 に示す．また，Pytorch で実装した Resnet50[1] から ONNX のモデルを経由してコンバータを用いてモバイルアプリケーションを作成した時の，Pytorch のネットワークコードの一部とコンバータによって生成された MPSNNGraph のコードの一部を図 3 と図 4 に示す．

(1) ONNX に対応した深層学習フレームワークを用いてネットワークを学習し，モデルファイルを作成
(2) ツールを用いて ONNX モデルを作成
(3) モデルファイルからネットワークの情報を取得
(4) 取得した情報から MPSNNGraph API を用いたネットワークコードを生成
(5) 取得した情報からパラメータファイルを生成
(6) 生成したコードと手書きの GUI コードを組み合わせてアプリケーションを作成

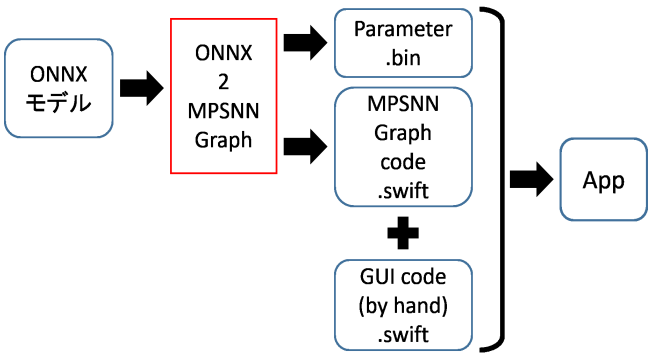

図 2　ONNX2MPSNNGraph のワークフロー

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)
```

図 3　Pytorch の Resnet のコードの一部

*3 https://pytorch.org
*4 https://keras.io

```
let conv2 = MPSCNNConvolutionNode(source:pool1.resultImage,
                                  weights: DataSource3("conv2", 1, 1, 64, 64, 1,
                                                       "bn2", 64, useBias: true))
let relu2 = MPSCNNNeuronReLUNode(source: conv2.resultImage)
let conv3 = MPSCNNConvolutionNode(source:relu2.resultImage,
                                  weights: DataSource3("conv3", 3, 3, 64, 64, 1,
                                                       "bn3", 64, useBias: false))
let relu3 = MPSCNNNeuronReLUNode(source: conv3.resultImage)
let conv4 = MPSCNNConvolutionNode(source:relu3.resultImage,
                                  weights: DataSource3("conv4", 1, 1, 64, 256, 1,
                                                       "bn4", 256, useBias: true))
```

図 4　ONNX2MPSNNGraph を用いて生成した MPSNNGraph のコードの一部

4. 実験

ONNX2MPSNNGraph による MPS 実装と CoreML 実装の速度比較の実験を行った．モデルは ResNet50[1] を用いて認識時間を測定し，20 回の平均時間を認識時間とした．デバイスは以下のものを用いた．

- iPhone8: iOS11.4, A11 Bionic
- iPhone8Plus: iOS12.3.1, A11 Bionic
- iPhoneXs Max: iOS12.3.1, A12 Bionic
- iPad Pro 11inch 2018: iOS12.3.1, A12X Bionic

4.1 結果

表 1 に各デバイスでの MPS 実装と CoreML 実装の認識時間の結果を示す．

iPhone8 と iPhone8Plus の結果を見ると，MPSNNGraph と CoreML のどちらの実装方法においても同様の結果が得られたことから，iOS11 と iOS12 のバージョンの違いによって認識速度は変化しないことがわかる．また，MPSNNGraph による実装と CoreML による実装では，MPSNNGraph による実装の方が CoreML による実装に比べて約 3[msec] 高速に認識しているという結果が得られた．これは，図 1 にあるように CoreML が MPS のラッパーに位置しているため，そこにオーバーヘッドが存在し，認識速度の差として出てきていると考えられる．

iPad Pro 11inch (2018) と他のデバイスの MPSNNGraph による実装の認識速度を見ると，iPad Pro 11inch (2018) の速度は他のデバイスと比べて約 2.1 倍となっている．これは，A11 Bionic や A12 Bionic に比べて A12X Bionic のチップセットの主に GPU に関する性能が大幅に向上したことによる速度向上であると考えられる．

iPhoneXs Max において，CoreML による実装の認識速度は，MPSNNGraph による実装に比べて約 3.7 倍高速になっている．また，iPad Pro 11inch (2018) においても CoreML による実装の方が約 2.1 倍高速に認識可能であるという結果が得られた．これだけの速度差を生み出した要因は，チップセットに搭載されている Neural Engine によるものだと考えられる．A11 Bionic のチップセットにもこの Neural Engine は搭載されているが，Face ID やアニ文字などの機能でしか Neural Engine を使用せず，ユーザーが Neural Engine を利用することはできなかった．しかし，A12 Bionic 以降のチップセットでは，CoreML が Neural Engine を利用することが可能となり，GPU のみによる演算処理と比較してより高速な演算処理が可能となっている．しかし，A12 Bionic 以降のチップセットでも CoreML を用いた実装以外では Neural Engine を用いる API といったものは公開されておらず，MPSNNGraph による実装では Neural Engine を用いない GPU のみの演算処理のため，これ程の差が生まれていると推測できる．

表 1　Resnet50[1] による物体認識の認識時間 [msec]

実装方法	MPSNNGraph	CoreML
iPhone8	86.02	89.12
iPhone8Plus	86.06	89.18
iPhoneXs Max	81.33	23.90
iPad Pro 11inch (2018)	40.94	19.29

5. おわりに

深層学習フレームワークの共通フォーマットである ONNX のモデルを用いて，深層学習モバイルアプリケーションを作成するために必要なモデルのネットワークコードやパラメータファイルを生成する深層学習モデルコンバータ ONNX2MPSNNGraph を作成した．

実験では，A11 Bionic が搭載されたデバイスでは，MPSNNGraph を用いた実装の方が CoreML による実装よりも高速であり，A12 Bionic 以降のチップセットが搭載されたデバイスでは，Neural Engine の活用の差により CoreML による実装のほうが高速であった．現状では，Neural Engine を利用するための API は公開されていないため CoreML による実装が有用であるが，Neural Engine を利用できるようになれば，MPSNNGraph による実装の方が高速になると予想でき，作成したコンバータの有用性が見込める．

参考文献

[1] He, K., Zhang, X., Ren, S. and Sun, J.: Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016).

[2] Izumi, Y., Horita, D., Tanno, R. and Yanai, K.: Real-Time Image Classification and Transformation Apps on iOS by "Chainer2MPSNNGraph", *Proc. of NIPS WS on Machine Learning on the Phone and other Consumer Devices (MLPCD)* (2018).

[3] 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)* (2015).