CC-001

完全２値化畳み込みニューラルネットワークについて

下田　将之†　藤井　智也††　米川　晴義††　佐藤　真平††　中原　啓貴††

† 東京工業大学 工学部 情報工学科
†† 東京工業大学 工学院 情報通信系
E-mail: †{shimoda,fujii,yonekawa}@reconf.ict.e.titech.ac.jp, ††{satos,nakahara}@ict.e.titech.ac.jp

あらまし　画像識別等の組込み機器では, 学習済み畳み込みニューラルネットワーク (CNN:Convolutional Neural Network) の識別高速化と低消費電力化が求められている. CNN の演算の 90%以上は 2 次元畳み込みであり, 主に積和 (MAC: Multiply-Accumulation) 演算である. 従って外部メモリに格納するデータ量と積和演算回路の削減が必要である. 近年, CNN の重み, 入力及び活性化関数の出力を 2 値 (-1/+1) に制限した 2 値化 CNN が提案されている. しかし, 提案された手法では入力層への入力はカラー画像を扱うため依然整数値のままであった. そのため, 第一層での畳み込み演算は, 整数精度で行われている. 本稿は入力層の入力値も同様に 2 値化を行い, 全ての畳み込み演算をビット演算で行う全 2 値化 CNN を提案する. そして, 全 2 値化 CNN を既存の 2 値化 CNN や, 整数精度の CNN と比較を行った. 提案する全 2 値化 CNN は, 2 次元畳み込みを行う際の整数精度の積和演算回路が不要なため, FF を 13.7%, LUT を 2.2%削減し, 1.2 倍高速化できた.

キーワード　ニューラルネットワーク, ２値化畳み込みニューラルネットワーク

All Binarized Convolutional Neural Network

Masayuki SHIMODA†, Tomoya FUJII††, Haruyoshi YONEKAWA††, Shimpei SATO††, and Hiroki NAKAHARA††

† Department of Computer Science, School of Engineering, Tokyo Institute of Technology, Japan
†† Department of Information and Communications Engineering, School of Engineering, Tokyo Institute of Technology, Japan
E-mail: †{shimoda,fujii,yonekawa}@reconf.ict.e.titech.ac.jp, ††{satos,nakahara}@ict.e.titech.ac.jp

Abstract　A pre-trained convolutional neural network (CNN) is a feed-forward computation perspective, which is widely used for the embedded systems, requires power-and-area efficiency. This paper realizes a binarized CNN which treats only binary values (+1/-1) for the inputs, the weights and the activation value. In this case, the multiplier is replaced into an XNOR circuit instead of a dedicated DSP block. Both inputs and weights are more suitable for hardware implementation. However, first convolutional layer still calculates in integer precision, since input value is not binarized one. In this paper, we decompose input value into maps of which each pixel consists of 1 bit precision. The proposed method enables a binarized CNN to use bitwise operation in all layers. We call this all binarized CNN. We conduct experiment on comparing all binarized CNN, floating-point CNN and binarized CNN. Since all binarized CNN do not need a dedicated DSP block, all binarized CNN is smaller than that of conventional binarized CNN and it is 1.2 times faster.

Key words　Neural Network, Convolutional Neural Network, Binary Convolutional Neural Network

1. はじめに

1.1 ニューラルネットワーク

ニューラルネットワーク (NN:Neural Network) とは, ニューロンを多層に接続したネットワークである. NN は大きく分けて入力層, 中間層, 出力層により構成される. 深層ニューラルネットワーク (DNN:Deep Neural Network) は中間層が 3 層以上の NN を指す. 畳み込みニューラルネットワーク (CNN:Convolutional Neural Network) とは, 2 次元畳み込み層を含んだ NN のことである. CNN は画像認識において有効であり, 手書き文字認識 [5], 顔認識 [6], シーン判定 [27], 物体検出 [7] 等幅広く用いられている. また, CNN を解析する研究も多く行われている [19].

CNN は階層を深くするほど認識精度が上がる. そのため, 認識精度を高くするには大規模な CNN が求められる. CPU では学習・推論に時間がかかるため, 実時間での応答が求められる画像認識分野には適さない [12]. そのため, ソフトウェア実現の多くは GPU を採用しているが [1]〜[4], GPU は消費電力が大きい [8]. 幸いにも, CNN は GPU のような倍精度演算を用いなくても固定小数点演算でほぼ同等の認識精度が得られるため [28], FPGA 上で精度の低い固定小数点で演算しても問題ない. 加えて, FPGA は GPU と比較して単位電力あたりの演算効率で 1 桁以上優れている [8]. 従って, FPGA を用いて CNN を高速化する手法が多数提案されている. 近年, 重み, 入力及び活性化関数の出力を 2 値 (-1 と +1) に限定した BinaryNet [10] が提案されている. BinaryNet は浮動小数点精度の CNN と大差ない認識精度をもち, また, BinaryNet 内のパラメータが 2 値に制限されているため, ハードウェア実装に適している. BinaryNet の他にも, 1 ビットや 2 ビット等低い精度で量子化された CNN が多く提案されている [17] [18]. これらの CNN は量子化しても高い認識精度を保ち, 2 値化の場合と同様に回路面積を大幅に削減することができる.

FPGA へ CNN を実装する際のアーキテクチャは様々な種類のものが提案されてきた. 大まかに分類すると, シストリックアレイ方式 [22] [23], ストリーミングアーキテクチャ [24] [25], ベクトルプロセッサ方式 [9], ニューロンシナプスプロセッサ [26] となる. 最近では, ヘテロジニアス 2 値化ストリーミングアーキテクチャ [21] や可変長ラインバッファ [20] を用いた実装が提案されている.

1.2 提案手法

既存の 2 値化 CNN である BinaryNet は, 入力データが画像ピクセル 24 ビットである. そのため, 入力層と中間層とを結合する畳み込み演算を整数精度で行なっている. 従って, 第 1 層目は他の層に比べ, 推論時の計算時間が多くかかり, 回路面積も大きくなっている. 本稿では 2 値化 CNN の入力層への入力を 1 bit に変換し, 全ての畳み込み演算をビット演算と加算器のみで行う全 2 値化 CNN を提案する. 全 2 値化 CNN は, 畳み込み演算における整数精度の積和演算回路を全て省略することができるため, 高速かつ小型な CNN を実現できる.

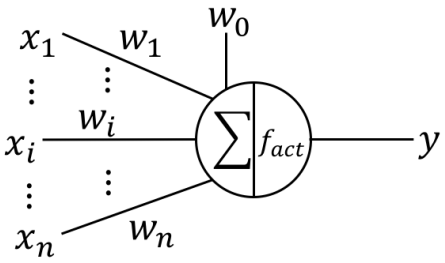


図 1　人工ニューロン

2. ニューラルネットワーク

2.1 ニューラルネットワークの構成

$(x_0, x_1, ..., x_{n-1})$ を入力変数, $(w_1, ..., w_{n-1})$ を重み, w_0 をバイアス, s を中間変数, $f_{act}(s)$ を活性化関数, z を出力変数とする. 図 1 に人工ニューロン (AN:Artificial Neuron) を示す. AN で行われる計算式を以下に示す.

$$s = \sum_{i=0}^{n} w_i x_i \tag{1}$$

$$z = f_{act}(s)$$

上の式 (1) において, $x_0 = 1$ である. AN は入力変数に対し, 重みを掛け合わせて総和を求め, 活性化関数を通し出力を得る. 活性化関数にはシグモイド関数, tanh 関数, ReLU(Rectified Linear Unit) 関数等がよく用いられる. この AN を多層に接続したネットワークのことを NN という. NN は大きく分けて入力層, 中間層, 出力層により構成され, 各層では全結合層, 2 次元畳み込み層, プーリング層がよく用いられる. 図 2 に 2 次元畳み込み層で行う演算を示す. AN の入力ニューロンを $L \times L$ サイズの 2 次元上に並べたものを特徴マップという. i 層における N_i 個の $W_i \times H_i$ サイズの特徴マップに対して, サイズが $K \times K$ のカーネルをシフトしながら積和演算を行う. その後, 活性化関数を通して i+1 層目の特徴マップとする. この演算を N_{i+1} だけ繰り返す. ある i+1 層目の特徴マップの座標 (x, y) における 2 次元畳み込み演算は次の式で表される.

$$\begin{aligned} s_{x,y}^{i+1} &= w_{bias} \\ &\quad + \sum_{ch=0}^{N_i-1} \sum_{r=0}^{K-1} \sum_{c=0}^{K-1} z_{ch,x+r,y+c}^{i} w_{ch,r,c} \\ z_{x,y}^{i+1} &= f_{act}(s_{x,y}^{i+1}) \end{aligned} \tag{2}$$

ここで, z^i は i 層目の特徴マップ, s は中間変数を表す. 上の式 (2) より, i+1 層目の特徴マップサイズが 1 , カーネルサイズが i 層目の特徴マップのサイズと同じ場合, 全結合層で行う演算と同じになる. プーリング層は, 特徴マップの圧縮を行う目的で使用される, 非線形な低画像化処理を行う層である. 平均値プーリング等様々なプーリング層が存在するが, 本稿では多くの CNN で用いられている最大値プーリングを用いる.

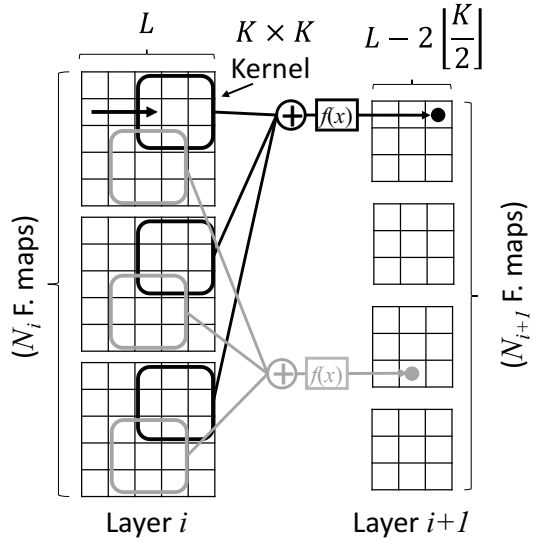


図 2　畳み込み演算

2.2 学　　習

NN における学習とは, 式 (1) における重みやバイアスなどの内部パラメータを最適化することを指す. 学習アルゴリズムで最もよく知られた手法の一つに確率的勾配降下法 (SGD:Stochastic Gradient Descent) がある. SGD では, 学習データに対しての NN の推論結果と正解の誤差を目的関数として最小化する. NN のパラメータを $w_{i,j}$, 入力を x, 誤差関数を $f_{error}(x)$ とすると, SGD は次の式で表される.

$$w_{i,j}^{(t+1)} = w_{i,j}^{(t)} - \varepsilon \frac{\partial f_{error}(x)}{\partial w_{i,j}^{(t)}}$$

ϵ は学習率といい, $0 < \varepsilon < 1$ を満たす実数である. また, SGD より良い解を得ることができる手法として Adam [11] が知られている. Adam のパラメータ更新アルゴリズムは次の式で表される.

$$\begin{aligned} g_{i,j} &= \frac{\partial f_{error}(x)}{\partial w_{i,j}^{(t)}} \\ w_{i,j}^{(t+1)} &= w_{i,j}^{(t)} - \varepsilon \frac{E[g_{i,j}]}{\sqrt{E[g_{i,j}^2]}} \end{aligned} \tag{3}$$

上の式 (3) の $E[g_{i,j}]$ は $g_{i,j}$ の期待値を表す. 本稿では, Adam を用いて学習させる.

3. 2 値化 CNN

3.1 定　　義

近年, 重み, 入力及び活性化関数の出力を 2 値 (-1 と +1) に限定した BinaryNet [10] が提案されている. 以降 BinaryNet を 2 値化 CNN とする. 2 値化 CNN 内の AN が行う計算を以下に示す.

$$s = \sum_{i=1}^{n} w_i^b x_i^b + w_0$$

$$z^b = f_{sign}(s)$$

Algorithm 1 (バッチ正規化)

Input: ミニバッチ　：$\mathcal{B} = \{x_1, ..., x_m\}$
学習パラメータ：γ, β

Output: $y_i = BN_{\gamma,\beta}(x_i)$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i)$$

ここで, $x^b, w^b, z^b \leftarrow \{0, 1\}$ である. w_0 はバイアスであり整数値, つまり多ビットである. 活性化関数 $f_{sign}(Y)$ は

$$f_{sign}(x) = \begin{cases} +1 & \text{if x} \geqq 0 \\ -1 & \text{otherwise} \end{cases}$$

となる. 学習時に勾配を用いるため, 微分は

$$\frac{df_{sign}(x)}{dx} = \begin{cases} 1 & \text{if } |\text{x}| \leqq 1 \\ 0 & \text{otherwise} \end{cases}$$

と定義する.

FPGA 上に実装するには, 負数 -1 を直接実現できないため論理値 0 を割り当てる. このとき, 乗算は XNOR ゲートで表現できるため, 浮動小数点精度の乗算回路に比べ大幅に面積を削減できる. また, 2 値化 CNN 内の中間値, 重みが 1 bit になるため, メモリ量も大幅に削減できる.

3.2 バッチ正規化 [13]

NN は学習データをミニバッチと呼ばれるセットでまとめて, パラメータの更新を行うミニバッチ学習法がよく用いられる. ミニバッチ毎のデータの分布が異なるため, 内部共変量シフトが発生する. 内部共変量シフトが発生すると, 学習速度の低下や, 学習結果が重みの初期値に依存してしまう問題が生じる. この問題を解決する手法としてバッチ正規化が知られている.

Algorithm 1 にバッチ正規化のアルゴリズムを示す. ϵ は安定化のための定数であり, 学習時に調整する. γ はスケール, β はシフトを表し, それぞれ正規化された値の平均及び分散を学習によって調整するためのものである.

バッチ正規化を導入した時に, 2 値化 CNN の AN が行う計算を次に示す.

Algorithm 2 (1 ビット分解アルゴリズム)

Input: x: 入力画像
x^i : i ビット目の値
Ch : 画像のチャンネル数
L: 入力画像の画素値のビット長
W: 画像の幅
H: 画像の高さ
Output: f : 変換後の変数

```
for ch = 1 to Ch do
  for c = 1 to W do
    for r = 1 to H do
      for i = 1 to L do
        f_{ch×L+i,r,c} ← x^i_{ch,r,c}
      end for
    end for
  end for
end for
```

$$s = \sum_{i=1}^{n} w_i^b x_i^b$$

$$y = BN(s + w_0)$$

$$z^b = f_{sign}(y)$$

2 値化 CNN では +1 と -1 のみ扱うため, 平均値が偏ってしまう. そのため, 活性化関数の出力が偏ってしまい, 学習が収束しない. バッチ正規化を導入することにより, 2 値の活性化関数の出力の割合が調節され, 学習が収束することが報告されている [14].

3.3 2 値化 CNN の FPGA 実装

バッチ正規化は 2 値化 CNN を実現するために必須であるが, アルゴリズム 1 が示すように, パラメータ γ, β, μ_B, σ_B^2 が必要である. 従って, それらを計算するための乗算回路と加算器が余分に必要になる. この問題を解決する手法として, バッチ正規化フリーな CNN が提案されている [16]. これは, バッチ正規化後の値と 2 値化活性化関数の関係から, バッチ正規化を導入した 2 値化 CNN は整数値のバイアスを持つ 2 値化 CNN と等価であることを示したものである. バッチ正規化を導入した 2 値化 CNN におけるバッチ正規化後の中間変数 s' の値は, 以下の式で表せられる.

$$\begin{aligned} s' &= \sum_{i=0}^{n} w_i^b x_i^b + \left\lfloor \frac{\beta\sqrt{\sigma_B^2+\epsilon}}{\gamma} - \mu_B \right\rfloor \\ &= \sum_{i=1}^{n} w_i^b x_i^b + \left(w_0 + \left\lfloor \frac{\beta\sqrt{\sigma_B^2+\epsilon}}{\gamma} - \mu_B \right\rfloor \right) \\ &= \sum_{i=1}^{n} w_i^b x_i^b + W_{bias} \end{aligned}$$

ここで w_{bias} は整数値を取る. この処理を行う AN をバッチ正規化フリー 2 値化 AN という. 図 3 にバッチ正規化フリー 2 値化 AN の構成を示す. バッチ正規化を整数化されたバイアス値で置き換えることができるため, 回路が単純となり面積を小さくすることができる.

表 1　使用したモデル

type	filter	kernel size	padding
conv	128	3	1
conv	128	3	1
maxp	0	2	0
conv	256	3	1
conv	256	3	1
maxp	0	2	0
conv	512	3	1
conv	512	3	1
maxp	0	2	0
fc	1024	4	0
fc	1024	32	0
fc	10	32	0

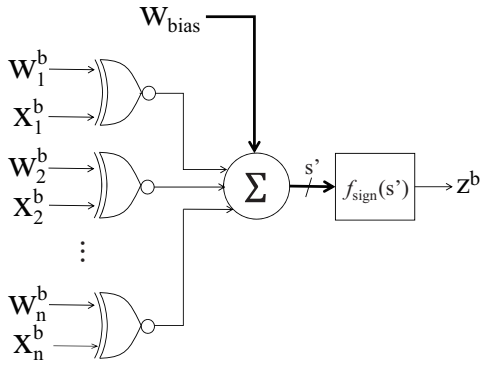


図 3　BN フリー 2 値化 AN

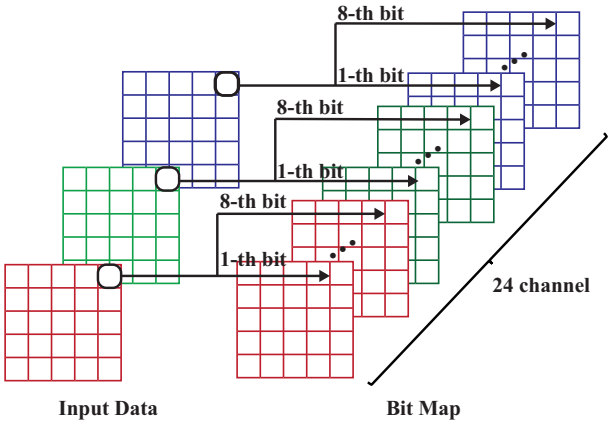


図 4　1 bit 画像への分解

4. 畳み込み演算の全 2 値化の実現

4.1 畳み込み演算の全 2 値化

CNN の応用は入力がカラー画像であることが多いため, 第 1 層での畳み込み演算は整数精度で行われている. この演算をビット演算と加算器のみで実現するには, 入力データのビット

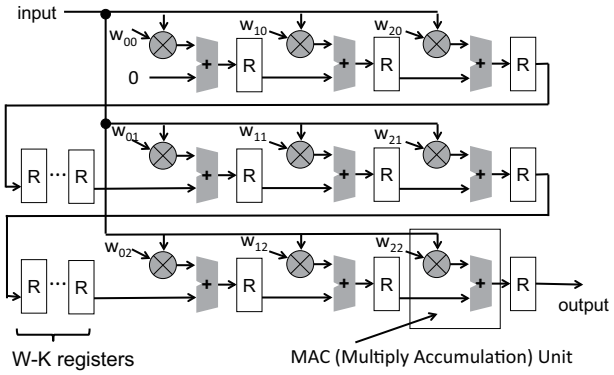


図 5　メモリアクセスを効率的に行う積和演算回路

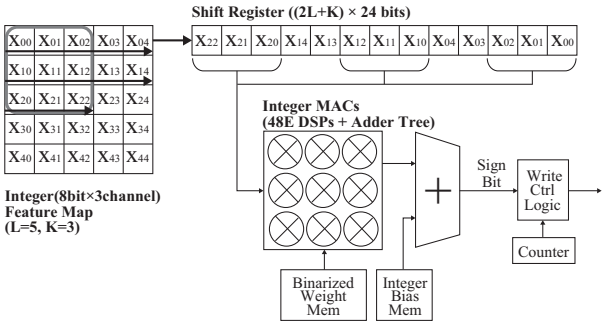


図 6　2 値化 CNN の第 1 層目の畳み込み演算処理要素

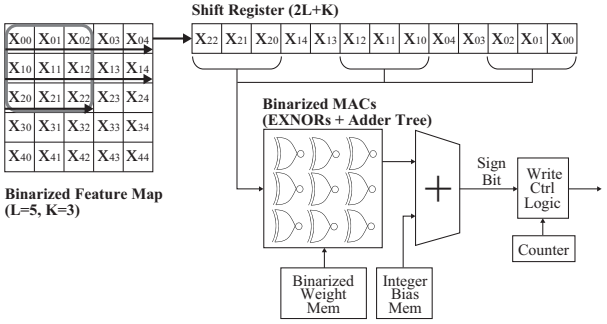
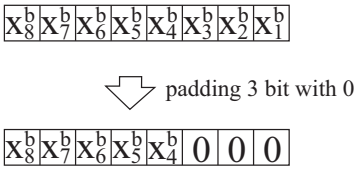


図 7　2 値化 CNN の 2 層目以降の畳み込み演算処理要素

精度が 1 ビットである必要がある. そのため, 入力層にデータを入力する前に, データを複数の 1 ビット精度画像へと分解する. Algorithm 2 に 1 ビット画像へ分解するアルゴリズムを示す.

RGB の 3 チャンネル, 各画素のビット長が 8 ビットの画像を例にして説明する. 1 ビット分解の概要を図 4 に示す. あるチャンネルの, 各画素の i ビット目を取り出し, それを 1 つの特徴マップとする. この作業を各チャンネル, ビット毎に行う. そうすると, 8 ビット 3 チャンネルの画像が 1 ビット 24 チャンネルの特徴マップ集合となる. 入力層へ入力する前に, このアルゴリズムを入力データに適応することで, 2 値化 CNN 内の全ての畳み込み演算をビット演算と加算器のみで実現することができる.



x_i^b: i-th bit value

図 8　0 パディング

表 2　CIFAR-10 に対する認識精度

type	acc /%
浮動小数 CNN	89.8
2 値化 CNN	83.7
全 2 値化 CNN	79.0

5. アーキテクチャ

5.1 既存アーキテクチャ

畳み込み演算では, AN が同じ入力を繰り返しアクセスする. そのため, メモリアクセスを効率化するために長いバッファを用意し, 画像データや動画データをシーケンスにアクセスしながら受け取りながら積和演算回路に毎クロック供給する回路が提案されている [9]. 図 5 にその回路の概要図を示す. この回路を利用した, 畳み込み演算処理要素が提案されている [15]. また, この畳み込み演算処理要素を改良し, 2 値化 CNN の畳み込み演算を行えるようにしたものも提案されている [16]. 2 値化 CNN の畳み込み処理要素は浮動小数点精度のものと違い, パラメータである重みとバイアスがメモリ上に全て保持しておくことができる. 従って, 外部 I / O を介さないため, 低消費電力で実行することができる. 図 6 には 2 値化 CNN の第 1 層目を, 図 7 に第 2 層目以降の畳み込み演算処理要素を示す. 既存の 2 値化 CNN では, 第 1 層目の積和演算回路を整数精度の乗算回路を用いていることに注意されたい. 本稿では, この二つの畳み込み演算処理要素で 2 値化 CNN を実装する.

5.2 全 2 値化 CNN のアーキテクチャ

全 2 値化 CNN の第 1 層目の畳み込み演算は, 入力データが 1 ビット精度のため, 1 ビット積を行う. 従って, 全 2 値化 CNN の全ての畳み込み層を, 図 7 に示した 1 ビット精度の 2 値化畳み込み演算処理要素を用いて実装する.

6. 実　　験

本稿では次の 2 つの実験を行なった.

- CIFAR-10 を用いて, 浮動小数点精度, 既存 2 値化 CNN, 全 2 値化 CNN の 3 つの認識精度を比較した.
- 全 2 値化 CNN を FPGA 評価ボード上に実装し, 既存 2 値化 CNN と面積・速度で比較した.

6.1 CIFAR-10 に対しての認識精度

CIFAR-10 とは, 車や犬, 馬など 10 カテゴリの物体が写った 32×32 ピクセルのカラー画像である. 実験では, この CIFAR-

表 3　ビット長に対する CIFAR-10 の認識精度

ビット長	acc /%
1-8BitMap	79.0
2-8BitMap	79.1
3-8BitMap	79.4
4-8BitMap	79.5
5-8BitMap	80.1
6-8BitMap	79.3
7-8BitMap	76.2

表 4　パディングの数に対する CIFAR-10 の認識精度

0 padding ビット長 /bit	acc /%
0	89.8
2	89.8
3	89.6
4	88.7
5	88.1

表 5　FPGA への実装結果

	2 値化 CNN	全 2 値化 CNN
18 Kbit BRAM 数	34	34
DSP48E 数	1	1
FF 数	38191	32993
LUT 数	70681	69158
Clock Cycle 数	508011	423622

10 を 10 カテゴリに分類する認識タスクをベンチマークとする. 表 1 に実験で使用する, CNN の一種であるモデルを示す. 浮動小数点精度, 2 値化 CNN 及び全 2 値化 CNN で同じ構造のモデルを使用するが, CNN 内部で保持するパラメータの精度はそれぞれ異なることに注意されたい. 学習に関しては, 重みの初期値には He の初期値, 学習アルゴリズムには Adam を使用し, エポック数 500, ミニバッチ数 50 の条件で統一した. また, 学習率の初期値を 0.001, 学習終了時の学習率を 0.0000003 とし, 1 エポックごとに学習率を減らした. 実験結果を表 2 に示す. 全 2 値化 CNN が既存の 2 値化 CNN に比べ 4 %精度が落ちた. これは, 8 ビットの精度で表現されていたものを 1 ビットに分解したため, 入力値が持っていた情報量が損なわれたからだと考える. また, 1 ビットに分解して学習をすると, 経験的に 8 ビット精度で学習するよりも勾配消失が発生しやすくなった. このことから, 2 値化 CNN とは違うモデルと学習率を使用した方が高い精度が出るのではないかと考察する.

また, 認識精度に対して下位ビットはあまり関与しないと考え, 下位ビットを無視して 1 ビットへ変換した場合の認識率を測定した. ここで, 下位 i ビット目から 8 ビット目まで使用して変換した特徴マップのことを i-8BitMap と定義する. 実験結果を表 3 に示す. また, 下位ビットを含めたものより, 含めないものの方が精度がわずかに高くなった. 2 値化された重みでは, 上位ビットと下位ビットが持つ情報量に対して重みの差をつけられないため, 同じモデルを使用した場合では上手く学習できなかったものと推測する.

下位ビットと上位ビットが持つ認識に関する情報量の差を調べるために, 入力データを図 8 に示すように, 下位数ビットを 0 にしたデータを認識する実験を行なった. この実験では, 1 ビット精度へ分解は行わず, 8 ビット精度のままで学習をさせた. 表 4 に実験結果を示す. この結果から, 上位と下位ビットが持つ認識に関わる情報量には大幅な差があることが分かる.

6.2　FPGA への実装

提案手法である全 2 値化を FPGA 評価ボード上に実装し, 既存の 2 値化 CNN との比較を行なった. 実験に用いた評価ボードは Digilent 社 Nexys Video (Xilinx 社 Artix-7 FPGA XC7A200T, 18Kb BRAM 数:730, DSP48E 数:740 搭載) である. 表 5 に実験結果を示す. 2 値化 CNN に比べ, FF を 13.7%, LUT を 2.2%削減し, 1.2 倍高速化できた. 1 ビット精度の乗算は 1 つのゲートで表せるため, 8 ビット精度の乗算にかかるクロック数や, 中間値を保持する FF を大幅に削減することができた. 2 割高速化することができたことから, 実時間での応答が求められる組み込み用途により適していると考える.

7. ま と め

入力層への入力画像を 1 ビット画像へと分解し, 全ての畳み込み演算をビット演算と加算器のみで実現した全 2 値化 CNN を提案した. 1 つ目の実験では, 浮動小数点精度の CNN, 2 値化 CNN 及び全 2 値化 CNN に対してそれぞれ認識精度を比較した. 2 つ目の実験では, 2 値化 CNN と全 2 値化 CNN を FPGA 評価ボード上に実装し, 二つの実行速度と回路面積を比較する実験を行なった. 認識精度に関しては, CIFAR-10 のデータセットを使用し, 3 つの手法全て同じ条件で学習させた. その結果, 提案手法が 2 値化 CNN と比べ 4 %認識精度が落ちた. 実行速度・回路面積に関しては, 全 2 値化 CNN が 2 値化 CNN と比べ FF を 13.7%, LUT を 2.2%削減し, 1.2 倍高速化できた. これらの結果から, 全 2 値化 CNN は実時間での応答が求められる組み込み用途に適していると言える.

謝辞　本研究は, 一部, 日本学術振興会・科学研究費補助金 (若手 (A): 課題番号 15H05304), 科学技術振興機構 ACCEL プロジェクトによる.

文　献

[1] Caffe: Deep learning framework, http://caffe.berkeleyvision.org/
[2] CUDA-Convent2: Fast convolutional neural network in C++/CUDA, https://code.google.com/p/cuda-convnet2/
[3] Theano, http://deeplearning.net/software/theano/
[4] Torch: A scientific computing framework for LUTJIT, http://torch.ch/
[5] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. of the IEEE*, 1998.
[6] Y. Taigman, M. Yang, M. A. Ranzato and L. Wolf., “DeepFace: Closing the Gap to Human Level Performance in Face Verification” , *Computer Vision and Pattern Recognition(CVPR)*, 2014, 1701-1708.
[7] S. Ren, K. He, R. Girshick, J. Sun. Faster, “R CNN: Towards Real Time Object Detection with Region Proposal Networks”, 2015.

https://arxiv.org/pdf/1506.01497.pdf
[8] A. Dundar, J. Jin, V. Gokhale, B. Martini and E. Culurciello, “Memory access optimized routing scheme for deep networks on a mobile coprocessor,” *HPEC2014*, 2014, pp. 1-6.
[9] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, “CNP: An FPGA-based processor for convolutional networks”, *FPL*, 2009, 32-37.
[10] M. Courbariaux, I. Hubara, D. Soudry, R.E.Yaniv, Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1”, *Computer Research Repository (CoRR)*, 2016.
[11] Diederik Kingma, Jimmy Ba, "Adam: A method for stochastic optimization", *ICoRR*, 2014.
[12] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition”, *Proc. of the IEEE*, Vol. 86, No. 11, 1998, pp.2278-2324.
[13] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, *CoRR*, 2015.
[14] 米川晴義, 中原啓貴, 本村真人, “電力効率に優れた二値化ディープニューラルネットワークの FPGA 実装”, リコンフィギャラブルシステム研究会, 2017.
[15] B. Bosi, G. Bois, and Y. Savaria, “Reconfigurable Pipelined 2D Convolvers for Fast Digital Signal Processing”, *VLSI, vol. 7, no. 3*, 1999, pp.299-308.
[16] Haruyoshi Yonekawa and Hiroki Nakahara, “An On-chip Memory Batch Normalization Free Binarized Convolutional Deep Neural Network on an FPGA”, *IPDPS*, 2017.
[17] M. Rastegari. V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net:ImageNet Classification Using Binary Convolutional Neural Networks”, 2016.
https://arxiv.org/pdf/1603.05279.pdf
[18] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, “DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”, 2016.
https://arxiv.org/pdf/1606.06160.pdf
[19] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, “Going deeper with embedded FPGA platform for convolutional neural network”, *ISFPGA*, 2016, pp. 26-35.
[20] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta and Z. Zhang, “Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs”, *ISFPGA*, 2017, pp. 15-24.
[21] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference”, *ISFPGA*, 2017.
[22] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks”, *In Proc. ACM/SIGDA ISFPGA*, pp. 161-170, 2015.
[23] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss and E. Chung, “Accelerating Deep Convolutional Neural Networks Using Specialized Hardware”, 2015.
[24] H. Alemdar, N. Caldwell, V. Leroy, A. Prost-Boucle and F. Petrot, “Ternary Neural Networks for Resource-Efficient AI Applications”, *CoRR*, abs / 1609.00222, 2016.
[25] S. I. Venieris and C.-S. Bouganis, “fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs”, *Proc. IEEE FCCM*, 2016, pp. 40-47.
[26] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al, “Convolutional networks for fast, energy-efficient neuromorphic computing”, *Proc. CoRR*, abs/1603.08270, 2016.
[27] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks”, *ICCD2013*, 2013, pp.13-19.
[28] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello, “Hardware accelerated convolutional neural networks for synthetic vision systems”, *ISCAS2010*, 2010, pp.257-260.