CC-002

2値化畳込みニューラルネットワークのニューロン刈りによるメモリ量削減とFPGA実現について

藤井　智也†　佐藤　真平†　中原　啓貴†

† 東京工業大学工学院情報通信系
E-mail: †t-fujii@eda.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深層畳み込みニューラルネットワーク (CNN: deep Convolutional Neural Network) の識別高速化と低消費電力化が求められている．一般的な CNN は前半部が畳込み層, 後半部がフル結合層で構成されている．先行研究より, 畳込み層では積和演算部がボトルネックであり，フル結合層ではメモリアクセスがボトルネックである．本論文では，フル結合層ではニューロンを刈ることで，重みを格納したメモリを削減し，フル結合層のメモリアクセスを高速化する．従って，FPGA のオンチップメモリ上にフル結合層の重みを全て格納でき，メモリアクセス問題を解決できる．また，本論文では FPGA のオンチップメモリと組み合わせた高速なフル結合層回路を提案する．提案する閾値ニューロン刈りにより，元の認識精度に対して 99%同等な場合は VGG-11 におけるフル結合層のニューロンを 87.4%削減できた．加えて，本論文では CNN の入力値と重みを 2 値 (-1/+1) に制限した 2 値化 CNN に対して閾値ニューロン刈りを適用し，フル結合層のニューロンを 60.2%削減できた．

A Memory Reduction with Neuron Pruning for a Binarized Deep Convolutional Neural Network: Its FPGA Realization

Tomoya FUJII†, Shimpei SATO†, and Hiroki NAKAHARA†

† Tokyo Institute of Technology, Japan
E-mail: †t-fujii@eda.ict.e.titech.ac.jp

1. まえがき

1.1 畳み込みニューラルネットワーク

近年，組み込み計算機システムでは 2 次元畳み込み層とフル結合層とで構成される畳み込みニューラルネットワーク (CNN: deep Convolutional Neural Network) が広く使用されている．CNN は人間の視覚を模擬するため画像認識精度が高く，手書き文字認識 [21]，顔認識 [27]，シーン判定 [24]，物体認識 [12] 等に応用されている．CNN はネットワークを深くするほど認識精度が上がる．従って高認識精度を実現するため，大規模な CNN が求められる．既存の CPU では識別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 [21]．ソフトウェア実現の多くは GPU を採用しているが [2]，[7]，[28]，[29]，GPU は消費電力が大きく組込みシステムに適さない [10]．そこで FPGA による低電力で応答の早い組み込み画像認識システムが求められている．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 [12]，FPGA 上で精度の低い固定小数点で演算しても問題ない．FPGA は GPU と比較して単位電力当りの演算効率で 1 桁以上優れている [10]．従って，FPGA を用いた高速化法が多数提案されている．

1.2 既存手法の問題点

一般的な CNN は畳込み層とフル結合層で構成されている．図 1 に畳込み層 (a) とフル結合層 (b) の複雑度分布を示す．図 1 が示すように，畳込み層では積和演算がボトルネックであり，フル結合層ではメモリアクセスがボトルネックである [32]．本論文ではフル結合層のメモリ量を削減することによりボトルネックを解消する．フル結合層のメモリアクセスはニューラルネットワークの枝に付けられた重みの読出しであり，枝を刈ることでメモリ量を削減できる．既存手法では，枝をランダムに

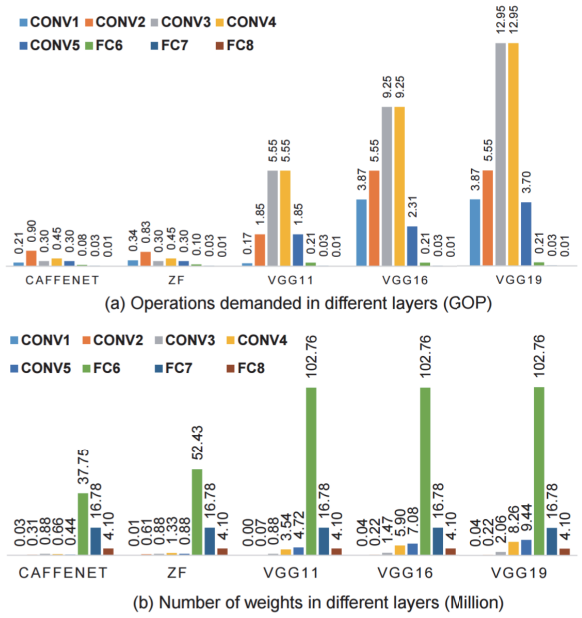

図 1　モダンな CNN モデルの複雑度分布: (a) 積和演算数分布; (b) 重み数分布 [19].

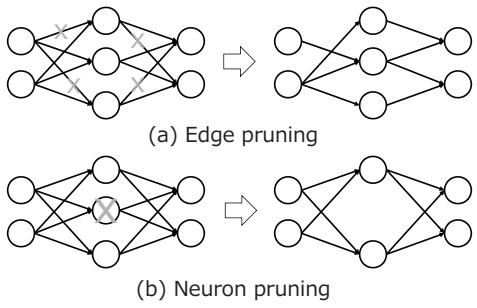

図 2　枝刈り手法の比較.

刈っていく方法が提案されている [1]，[16]，図 2 (a) にフル結合層の枝刈りの例を示す．しかし，ハードウェア実現を考えた場合，同一パターンの連続アクセスが向いているため，ランダムに枝を刈った場合，性能低下が懸念される．

1.3　提 案 手 法

本論文ではフル結合層のニューロンを刈る方法を提案する．図 2 (b) にフル結合層のニューロン刈りの例を示す．該当するニューロンの入出力枝を全て刈ることでニューロン刈りが実現できるため，一般的には枝刈りのほうがニューロン刈りよりも多くの枝を刈ることができる．しかし，ニューロン刈りをしたとしてもメモリアクセスは連続になるため，ハードウェア実現に適している．提案手法のニューロン刈りでほとんどの枝を除去できるため，全ての残りの枝を FPGA のオンチップメモリに格納できる．本研究ではシリアル入力-パラレル出力フル結合層レイヤ回路を提案する．FPGA のオンチップメモリと DSP スライスを効率的に活用することによって，高性能な CNN 回路を実現する．実験では，CPU，GPU 実装との比較を行い，FPGA 実装の優位性を示す．

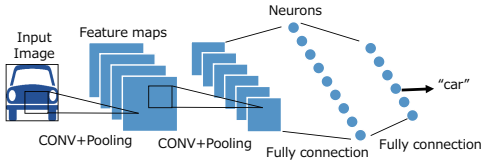

図 3　Convolutional Neural Network (CNN) の例.

1.4　本論文の貢献点

以前の論文 [13] の貢献点を以下に示す．

1. フル結合層のニューロン刈りを提案した．提案するニューロン刈りは枝刈りと比較してハードウェア化に向く．ニューロン刈りにより，認識精度をニューロン刈り前の 99%に維持しつつ，VGG-11 におけるフル結合層のニューロンを 76.4%削減できた．

2. FPGA のオンチップメモリと組み合わせた高速なフル結合層回路を提案した．ニューロン刈りによりフル結合層の重みを全てオンチップメモリに格納できるため，GPU などの広帯域メモリを持つプロセッサよりも高速なメモリアクセスが実現できる．提案手法は既存の畳込み層を高速化する手法と相補的であり，FPGA を利用した CNN の高性能化法も広げることができた．

3. VGG-11 のフル結合層に対してニューロン刈りを行い，Digilent 社 NetFPGA-1G-CML に実装した．ARM プロセッサ (CPU)，Jetson TK1 (GPU) と比較を行い FPGA 実現の優位性を示した．

先行研究 [13] では単精度パラメーターを持つ CNN に対してのみニューロン刈りを適用したが，本研究では 2 値化 CNN にも適用した．我々が知る限りでは，2 値化 CNN にニューロン刈りを適用したというのは本論文が最初である．加えて，本論文の新しい貢献点を以下に示す．

1. メモリサイズがボトルネックとなっている 2 値化フル結合層に対してニューロン刈りを行い，ニューロン数を減らした．

2. さらなるメモリ削減をするべく，ニューロン刈り適用後の 2 値化 CNN に対して再学習を行い，また，ニューロン刈りを適用した．この方法でニューロン数をさらに削減することに成功した．

3. 以前の研究ではフル結合層に対してのみハードウェア実装していたが，今回は 2 値化 CNN の全層に対して，Xilinx 社 Zynq UltraScale+MPSoC zcu102 評価ボードを用いて実装した．

本論文は以前の論文 [13] を改訂したものである．

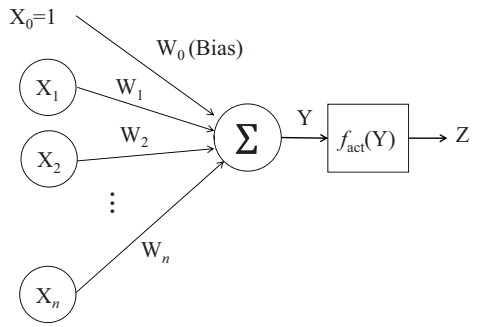

図 4　Artifical Neuron.

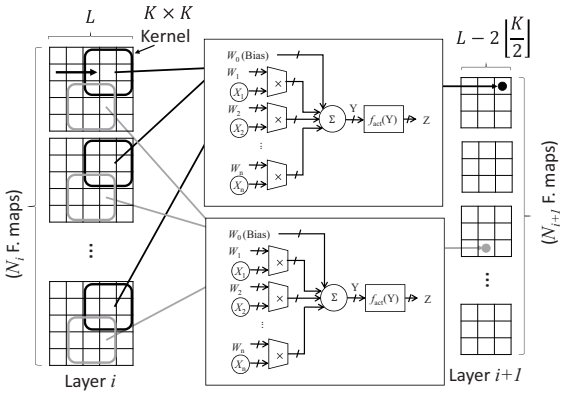

図 5　Convolutional Operation.

2. 2 値化畳み込みディープニューラルネットワーク

図 3 に畳み込み層，プーリング層，フル結合層から構成される CNN を示す．本研究の目標は訓練済みの CNN モデルが与えられたとき，ハードウェア量の小さい回路で応答時間の早い推論を行うことである．本節では 2 値化 CNN の紹介をする．

2.1　畳み込み層

畳み込み層もフル結合層も図 4 で示している人工ニューロンのバリエーションの一種である．人工ニューロンの出力は以下のように計算できる．

$$Z = f_{act}(\sum_{l=0}^{n} W_l X_l),$$

ただし， Z は出力, X_l は入力, W_l は重みを表す．ここで，留意すべきは $n = 0$ のとき， W_0 は定数となることである．このとき，これをバイアスと呼ぶ．バイアスはニューロンの出力を調整することで，モデルとしての認識精度を維持する役割がある．f_{act} はシグモイド関数, tanh 関数, ReLU 関数といった活性化関数を表している．

図 5 で示している通り，CNN における 2 次元畳み込み層とフル結合層は似た演算を行っている．人工ニューロンによる積和演算を特徴マップに対して $K \times K$ サイズのカーネルに適用している．これにより，演算数が大きく減らせ，局所的な特徴が抽出でき，過学習も防げる．畳み込み層 i^{th} の出力 $Z_{l,r,c}$ は位置 (r, c) 毎に入力として $K \times K$ の画像（特徴マップ）が N_i 個あり，以下のように計算できる．

表 1　2 値 (-1/+1) 乗算の真理値表

$y = w \times x.$

w	x	$w \times x$
-1	-1	+1
-1	+1	-1
+1	-1	-1
+1	+1	+1

表 2　2 値 (0/1) 乗算の真理値表

$y = \overline{w \oplus x}.$

w	x	$\overline{w \oplus x}$
0	0	1
0	1	0
1	0	0
1	1	1

$$Z_{l,r,c} = f_{act}(\sum_{s=0}^{N_i}\sum_{j=0}^{K}\sum_{l=0}^{K} W_{i,j,l,s} X_{i-1,s,r+j,c+k}),$$

2.2　2 値化 CNN

Courbariaux らは 2 種の 2 値化 CNN 手法を提案している [8]. 1 つ目は重みのみを 2 値化した手法であり，2 つ目は重みと入力の両方を 2 値化する手法である．他の先行研究で， [20], [25], [35] も全 2 値化 CNN を提案している．しかしながら，これらの 2 値化 CNN は単精度 CNN に比べて認識精度が低下する．例えば，これらの手法の中で ImageNet の最も良い認識精度は全 2 値化で 43%, 部分的 2 値化で 53%である．一方で，Courbariaux らの 2 値化手法はバッチ正規化を用いることで認識精度の低下を防いでいる．バッチ正規化はバッチごとに平均が 0, 分散が 1 になるように正規化し，正規化された値をスケーリング 及びシフト することで，低精度での情報損失を減少させている．つまり，バッチ正規化の導入によって 2 値化 CNN のエラー率を低下させることができる．これにより，MNIST，SVHN，CIFAR-10 等のタスクで実用的な認識精度が得られる．しかしながら，正規化は推論時にも必要な過程であるため，正規化がボトルネックとなり，専用ハードウェアが追加で必要となる．活性化関数は符号ビットを判定すればよいだけなので，正規化操作はバイアスでまとめて計算できる [31]. つまり，ハードウェア実装においては 2 値バイアスではなく，多値バイアスを用いて計算する．表 1 は 2 値 (-1/+1) 乗算の真理値表で，表 2 は 2 値 (0/1) 乗算の真理値表である．後者の場合，乗算は XNOR 回路のみで実行できる．つまり，2 値の畳込み演算は以下のように表現できる．

$$z_{l,r,c} = f_{sgn}(\sum_{s=0}^{N_i}\sum_{j=0}^{K}\sum_{l=0}^{K} \overline{w_{i,j,l,s} \oplus x_{i-1,s,r+j,c+k}}),$$

このとき $f_{sgn}(Y)$ は以下のように表せる．

$$f_{sgn}(Y) = \begin{cases} 1 & (if\ Y \geqq 0) \\ 0 & (otherwise) \end{cases}$$

表 3　VGG-11 の各パラメータ [33].

Layer	Output Dim.	Input # Fmaps	Output Fmaps
Conv1	32× 32	3	64
Conv2	32× 32	64	64
Max Pool	16× 16	64	64
Conv3	16× 16	64	128
Conv4	16× 16	128	128
Conv5	16× 16	128	128
Max Pool	8× 8	128	128
Conv6	8× 8	128	256
Conv7	8× 8	256	256
Conv8	8× 8	256	256
Max Pool	4× 4	256	256
FC1	1× 1	4096	4096
FC2	1× 1	4096	4096
FC3	1× 1	4096	10

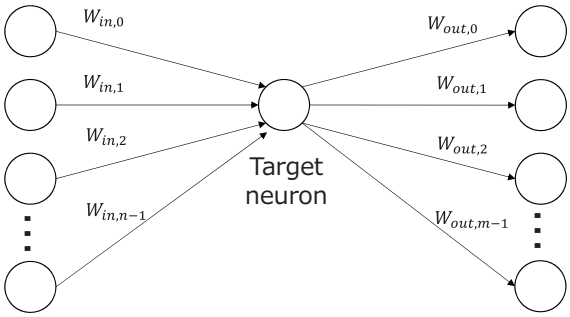

図 6　閾値ニューロン刈りのモデル.

同じように，2 値のフル結合層は次のように計算される．

$$z = f_{sgn}(\sum_{l=0}^{n} \overline{w_l \oplus x_l}), \tag{1}$$

$Y = \sum_{l=0}^{n} \overline{w_l \oplus x_l}$ とすると，式 (1) は多数決関数 $f_{major}(Y)$ で表せる．多数決関数は入力の半数が 1 ならば 1 を返し，入力の半数が 0 ならば 0 を返す．

3. 閾値ニューロン刈り

3.1 定　　義

本論文では枝刈りではなくニューロン刈りを提案する．ニューロンの入出力の枝を全て刈れば，ニューロン刈りそのものになる．したがって，一般的には枝刈りのほうがニューロン刈りよりも多くの枝を刈れる．しかしながら，枝刈りはランダムに枝を刈るためにハードウェア化に向かない．一方で，ニューロン刈りを行ったとしてもまとめて枝を刈るため，連続的な枝読み込みは維持される．従って,，ハードウェア実装に向く．まず，ニューロン刈りの定義を行う．

［定義 3.1］ **ニューロン刈り**とは，あるニューロンへの全ての入力枝と出力枝をすべて刈ることである．

［定義 3.2］ **閾値ニューロン刈り**とは，設定した閾値以下の条件を満たす場合，ニューロンを刈ることである．

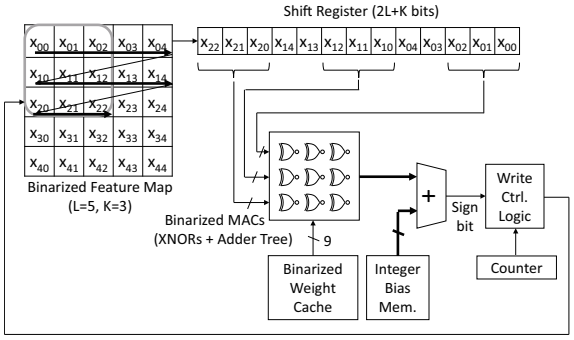

図 7　ストリーミング 2 値化 2 次元畳み込み回路.

様々な閾値の設定の仕方があるが，本論文では入出力の枝の重みの総和がどちらかでも設定した閾値を下回っていた場合，ニューロンを刈ることとする．あるニューロンへの入力枝数を n，出力枝数を m とする．すなわち，以下の条件をどちらかでも満たす場合，閾値ニューロン刈りが行われる．

（1） $\sum_{k=1}^{n} |w_{i,k}| < \mu_i \times n$

（2） $\sum_{k=1}^{m} |w_{o,k}| < \mu_o \times m$

ここで，$w_{i,k}$ とはあるニューロンへの k 番目の重みであり，$w_{o,k}$ とはあるニューロンからの k 番目の重みであり，μ_i, μ_o を閾値とする (図 6)．本論文では入出力にそれぞれ閾値を設定する．

3.2 閾値ニューロン刈り後の再学習

ニューロン刈りはシナプス刈り込みと似ている．ヒトの脳は本来，シナプスを刈り込むようにできていて，幼児期から成年期までにの間に 5 回シナプスを刈り込むと言われている [16]．このような知見が人工ニューラルネットワークにも応用できる．ニューロン刈りはネットワークの複雑度と過学習を減らす有効な手段であることが分かった [17]．この手法は一般的なニューラルネットワークでも同じように機能する．まず，普通に 2 値化ネットワークモデルの学習を行う．次に，重みの小さい結合部分を提案手法であるニューロン刈りによって刈り，閾値以下の重みを持つ全ての結合部分をネットワークから取り除く．最後に，取り除かれずに残ったネットワークを再学習させる．このニューロン刈りと再学習の操作を繰り返すことで，ニューロン数が飽和していく．実験では，2 値化 CNN のフル結合層にこれらの操作を適用している．

4. FPGA 向け 2 値化 CNN アーキテクチャ

4.1 ストリーミング演算向け共有 XNOR-MAC 回路

本研究では，浮動小数点の MAC 演算ではなく 2 値の MAC 演算を使っているが，XNOR-MAC 演算を全並列で実装するには多大なハードウェアリソースを使用する．一般的に CNN はレイヤごとに異なった数の特徴マップを持っていることから，ヘテロジーニアスなストリーミング回路を実現する上では多くの LUT と 大規模な XNOR 回路 が必要である．

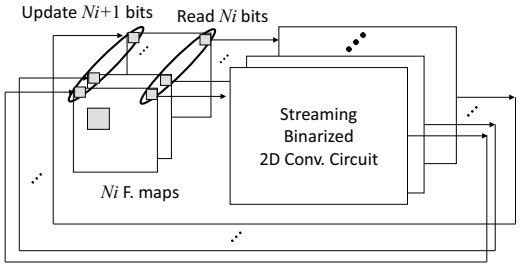

図 8　共有ストリーミング 2 値化 2 次元畳み込み回路.

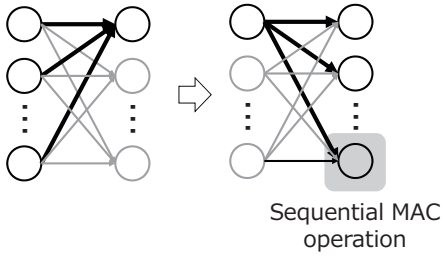

図 9　シリアル入力パラレル出力フル結合層レイヤ [11].

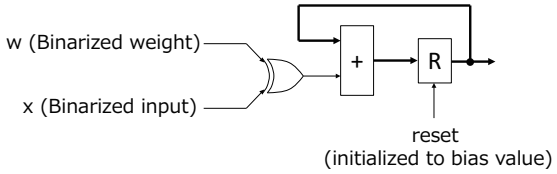

図 10　逐次方式 MAC 演算回路.

本研究では，少ないハードウェアでの高性能を実現するために，図 7 で示すようなストリーミング回路に対応した共有 XNOR-MAC 回路を提案する．メモリアクセス削減のため，特徴マップの情報を格納するメモリからシフトレジスタを用いて，ストリーミングデータフローを生成する．

そして，様々な大きさの XNOR-MAC 演算を一つの共有 XNOR 回路を使って行い，加算器ツリー，バイアス加算器，書き込み制御回路へと続く．シフトレジスタから適切な入力を読み出し，ビットごとに 2 値の MAC 演算を適用する．次に，予め計算しておいたバイアスを加算する．このバイアスは訓練後のバイアス値とバッチ正規化のパラメータの両方から得られる．特徴マップの境界にカーネルがまたがるので，書き込み制御を回路の出力部におく．

さらにパフォーマンスを向上させるため，図 8 に示す 2 値化共有ストリーミング 2 次元畳み込み回路を提案する．全特徴マップへのアクセスを柔軟に行うため，オンチップの BRAM を用いて大きな帯域幅を持つマルチポートを実現する．一方で，重みを読み出すのには，畳み込み命令がそれぞれの特徴マップで重みを読み出すのにインターバルがあるため，オフチップメモリを使う．本研究では，2 値化 CNN を使用する上，フル結合層の枝を刈るため，通常の CNN に比べてかなりのメモリ削減が期待できる．

4.2　ニューロン刈りを考慮したフル結合層の回路実現

図 9 にシリアル入力パラレル出力フル結合層レイヤ [11] を

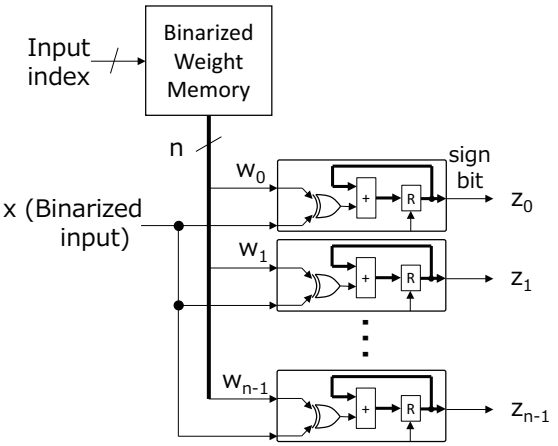

図 11　シリアル入力パラレル出力フル結合層.

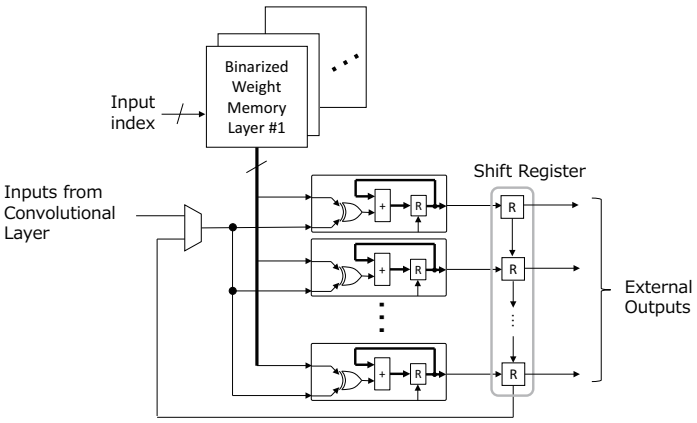

図 12　シリアル-パラレル方式フル結合層回路.

示す．図 9 により，外部入力メモリの帯域を狭めることができる．このフル結合層レイヤを実現するには，図 4 に示した AN の MAC 演算を逐次行う回路が必要である．図 10 に逐次方式 2 値化 MAC 回路を示す．各フル結合層レイヤの演算ごとにレジスタの値をバイアス値に初期化する．そして各ニューロンの重みを逐次加算する．最後にその結果を出力する．逐次方式 MAC 回路の MAC 演算部は LUT による EXNOR ゲートで実現できる．図 11 にシリアル入力-パラレル出力フル結合層レイヤ回路を示す．各枝の重みを **2 値化重みメモリ**に格納しておき，入力毎に同時に読み出す．逐次方式 MAC 回路で逐次ニューロンの値を更新する．図 12 にシリアル入力-パラレル出力フル結合層フル結合層回路を示す．ほとんどの重みはニューロン刈りにより除去できるので，残りの僅かな重みのみが重みメモリに格納される．各レイヤ毎の重みを複数の重みメモリに格納する．FPGA は任意のサイズのメモリを実現できるので，ニューロン刈りにより適切なメモリで実現できる．これを各層毎に同時に読み出し，出力ニューロンの値を並列に更新する．全ての入力値が評価された後，出力ニューロンの値をシフトレジスタに同時に転送する．そして, 次層のフル結合層レイヤの入力値としてシフトしながら転送する．フル結合層が評価された後，最終層のニューロンの値を外部に出力する．

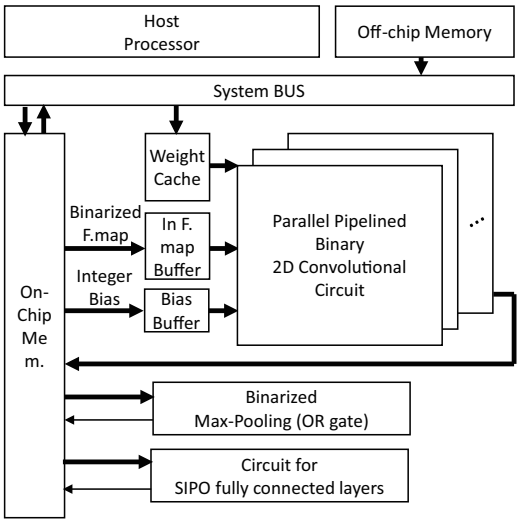

図 13　全体のアーキテクチャ.

4.3　全体のアーキテクチャ

図 13 に提案手法である 2 値化 CNN 向けのアーキテクチャを示す．メモリアクセス用の回路は既存のものとほとんど同じであるが，メモリ部分はオンチップのブロック RAM (BRAM) のみで実装されている．本研究では，2 値の乗算を XNOR ゲートで実装しているので，畳込み演算には DSP ブロックを使用しない．オンチップメモリ (BRAM) は全ての特徴マップの入出力を格納し，オフチップメモリ (DDR3SDRAM) は重みを格納する．このように，BRAM を効率的に使用することで，メモリを節約しつつパフォーマンスを維持している．また，256 枚の特徴マップに対する畳込み演算を同時に行うため，この実装は既存のものより実行速度において優れている．

5. 実 験 結 果

5.1　再学習 - 閾値ニューロン刈り

本研究では，ディープニューラルネットワークのフレームワークである Chainer [3] を用いて CNN を設計し，画像認識タスクである CIFAR-10 [5] を用いて性能評価を行った．実験では，適切な閾値 μ をフル結合層のレイヤごとに手作業で探索し，設定した．ニューロン刈り適用後，刈られずに残ったフル結合層部分を再学習させた．目標認識精度を維持できなくなるまでこの操作を繰り返した．

表 4 は n 回操作でのニューロン数を示している．ここで，一般的にニューロン数が減少すれば認識精度も減少することに留意してほしい．本実験では，元モデルの認識精度に対して 99% の認識精度を維持するようにニューロン刈りを適用する．ニューロン刈りと再学習を繰り返していくうちにニューロン数が飽和してくる．今回は 13 回目で飽和したようなので，そこでニューロン刈りの操作を終了した．表 4 に示す通り，ニューロン数は 39.8% まで減少した．本実験においては，VGG-11 CNN のフル結合層の重みメモリを FPGA のオンチップメモリに格納することに成功している．つまり，広い帯域幅でのメモリアクセスが可能なため，メモリアクセスがボトルネックとなっているフル結合層の演算が高速化する．また，オフチップメモリを必要としないため，消費電力や費用的コストも削減できる．ニューロン刈りを一度しか適用しなかった前回の論文 [13] に比べて，提案する再学習 - 閾値ニューロン刈りはよりニューロンを削減でき，使用ハードウェア量を小さくできる．

5.2　実 装 結 果

本研究では，2 値化版 VGG-16 を Xilinx 社 Zynq UltraScale+ MPSoC zcu102 評価ボード (Xilinx Zynq UltraScale+ MPSoC FPGA (ZU9EG, 68,520 Slices, 269,200 FFs, 1,824 18Kb BRAMs, 2,520 DSP48Es 搭載) に実装した．また，タイミング制約を 100MHz とし，ビットストリームの生成には Xilinx Inc. SDSoC 2016.4 を使用した．実装にはあたっては，14,509 の LUT， 545 の 18Kb BRAM, を使用し，DSP48Es は使用しなかった．また，リタイミング制約を満たしていて，遅延時間は 2.456 ミリ秒であった．

本研究の実装ではオンチップのみ使用するので，オンチップの消費電力を計測するのに zcu102 ボードの pre-built power advantage tool を実行した．消費電力を計測するのには ZynqusPowerTool モニターを使用し，消費電力は 313.3 mW であった．遅延時間は 2.456 ミリ秒 で，パフォーマンスは 407 (frames per second) であり，電力効率は 1300.3 (FPS/Watt) であった．

5.3　既存の 2 値化 CNN との比較

表 6 は同じ FPGA での 2 値化 CNN 実装を比較したものである．Zhao らの実装と比較すると認識精度はほとんど変わらないが，電力効率 (FPS/Watt) では 36.42 倍優れている．FINN と比較すると，メモリ効率は 2.28 倍劣っているが，電力効率では 7.30 倍優れている．本研究の実装は全ての演算がオンチップで行われるので，電力効率において良い結果が出ている．

5.4　CPU，GPU との比較

本研究の 2 値化 CNN と他の組み込みプラットフォームとで比較を行った．本研究では，組み込みデバイスとして CPU (ARM Cortex-A57) と GPU (Maxwell GPU) の両方を持つ NVidia Jetson TX1 を使用した．Caffe [2] バージョン 0.14 を使用し，VGG11 を実行するというベンチマーク方法 [23] で，比較評価した．また，全体の消費電力も合わせて計測した．尚，レイテンシを計測する際にはバッチサイズを 1 に設定した．

表 7 は本研究の FPGA 実装と他のプラットフォームを比較したものである．ARM Cortex-A57 と比較して，1773.0 倍高速で，22.3 倍消費電力が少なく，電力効率は 40634.4 倍優れていた．また，Maxwell GPU と比較して，11.1 倍高速で，54.3 倍消費電力が少なく，電力効率は 591.0 倍優れていた．したがって，2 値化 CNN を組み込みシステムで実装する上では FPGA が適している．

6. 結　　論

本論文では，CNN の後半部のフル結合層のメモリアクセスを高速化する手法を提案した．すなわち，閾値に基づきニュー

表 4　ニューロン刈り適用後のニューロン数（*元認識精度の 99%を維持しつつ再学習）.

Step	1	2	3	4	5	6	7	8	9	10	11	12	13
FC 1	4,096	2,259	1,578	1,458	1,457	1,457	1,457	1,454	1,454	1,454	1,454	1,454	1,454
FC 2	4,096	3,853	3,826	3,754	3,716	3,716	3,534	3,456	3,456	3,447	3,426	3,421	3,395
FC 3	4,096	3,438	1,149	1,059	498	373	193	102	89	57	54	51	37
FC 4	10	10	10	10	10	10	10	10	10	10	10	10	10
Total	12,298	9,560	6,563	6,281	5,681	5,556	5,194	5,022	5,009	4,968	4,944	4,936	4,896
Ratio	1.00	0.78	0.53	0.51	0.46	0.45	0.42	0.41	0.41	0.40	0.40	0.40	0.39

表 5　2 値化 VGG-11 の比較 (*整数値の畳み込み層 （IConv1） は 1 ビットの重みと 8 ビットの入力を使用している.）.

Layer	Baseline				Neuron Pruning with Retraining			
	Output Dim.	Input # Fmaps	Output Fmaps	Weight [bits]	Output Dim.	Input # Fmaps	Output Fmaps	Weight [bits]
IConv1	32× 32	3	64	1.7K	32× 32	3	64	1.7K
BConv2	32× 32	64	64	36.8K	32× 32	64	64	36.8K
Max Pool	16× 16	64	64		16× 16	64	64	
BConv3	16× 16	64	128	73.7K	16× 16	64	128	73.7K
BConv4	16× 16	128	128	147.4K	16× 16	128	128	147.4K
Max Pool	8× 8	128	128		8× 8	128	128	
BConv5	8× 8	128	256	294.9K	8× 8	128	256	294.9K
BConv6	8× 8	256	256	589.8K	8× 8	256	256	589.8K
Max Pool	4× 4	256	256		4× 4	256	256	
BFC1	1× 1	4096	4096	16.7M	32× 32	1454	3395	4.4M
BFC2	1× 1	4096	4096	16.7M	32× 32	3395	37	125.6K
BFC3	1× 1	4096	10	40.9K	32× 32	37	10	370
(fc total)				(33.6M)				(5.1M)
Total				34.7M				16.5M

表 6　FPGA における他の 2 値化 CNN 実装との比較.

Implementation (Year)	Zhao et al. (2017)	FINN (2017)	Ours
FPGA Board (FPGA)	Zedboard (XC7Z020)	PYNQ board (XC7Z020)	Zynq UltraScale+ MPSOC (XCZU9EG)
Clock (MHz)	143	166	143
# LUTs	46900	42823	14509
# 18Kb BRAMs	94	270	545
# DSP Blocks	3	32	0
Test Error	12.27%	19.9%	18.2%
Time [msec]	5.94	2.24	2.45
(FPS)	(168)	(445)	(408)
Power [W]	4.7	2.5	**0.313**
FPS/Watt	35.7	178.0	**1300.3**
FPS/LUT	35.8×10^{-4}	103.9×10^{-4}	$\mathbf{289.4}\times10^{-4}$
FPS/BRAM	1.8	1.6	0.7

表 7　VGG-11 順伝搬における組み込みプラットフォーム比較 (バッチサイズは 1).

Platform	Embedded CPU	Embedded GPU	FPGA
Device	ARM Cortex-A57	Maxwell GPU	Zynq UltraScale+ MPSOC
Clock Freq.	1.9 GHz	998 MHz	100 MHz
Memory	16GB eMMC Flash	4GB LPDDR4	16.5 Mb 18KbBRAM
Time [msec]	4210.0	27.23	**2.45**
(FPS)	(0.23)	(36.7)	**(408.1)**
Power [W]	7	17	2.3
Efficiency [FPS/W]	0.032	2.2	**1300.3**

ロンを刈ることで，重みを格納したメモリを削減した．従って，閾値ニューロン刈りにより FPGA のオンチップメモリですべてのフル結合層の重みメモリを実現した．その結果，FPGA のオンチップメモリを低遅延で並列アクセスすることによりメモリアクセス問題を解決した．また，本論文では FPGA のオンチップメモリと組み合わせた高速なフル結合層回路を提案した．閾値ニューロン刈りにより，元モデルに対し 99%認識精度を維持しつつ，VGG-11 におけるフル結合層のニューロンを 68.4%削減できた．さらに，ニューロン刈りを適用した CNN を Xilinx UltraScale+ MPSoC zcu102 評価ボードに実装した．ARM Cortex-A57 と比較して，1773.0 倍高速で，22.3 倍消費電力が少なく，電力効率は 40634.4 倍優れていた．また，Maxwell GPU と比較して，11.1 倍高速で，54.3 倍消費電力が少なく，電力効率は 591.0 倍優れていた．したがって，2 値化 CNN を組み込みシステムで実装する上では FPGA が適している．

文　献

[1] S. Anwar, K. Hwang and W. Sung, “Structured pruning of deep convolutional neural networks,” *Computer Research Repository (CoRR)*, Dec., 2015.

https://arxiv.org/ftp/arxiv/papers/1512/1512.08571.pdf
[2] Caffe: Deep learning framework, http://caffe.berkeleyvision.org/
[3] Chainer: A powerful, flexible, and intuitive framework of neural networks, http://chainer.org/
[4] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *Annual Int'l Symp. on Computer Architecture (ISCA)*, 2010, pp.247-257.
[5] The CIFAR-10 data set, http://www.cs.toronto.edu/ kriz/cifar.html
[6] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *In Proc. CVPR*, 2012.
[7] CUDA-Convent2: Fast convolutional neural network in C++/CUDA, https://code.google.com/p/cuda-convnet2/
[8] 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)*, Mar., 2016, http://arxiv.org/pdf/1602.02830v3.pdf
[9] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *In Proc. CVPR*, 2015.
[10] 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.
[11] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," *FPL2009*, 2009, pp.32-37.
[12] 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.
[13] T. Fujii, S. Sato, H. Nakahara, and M. Motomura, "An FPGA Realization of a Deep Convolutional Neural Network using a Threshold Neuron Pruning," *Int'l Symp. on Applied Reconfigurable Computing (ARC2017)*, 2017, pp. 268-290.
[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *In Proc. CVPR*, 2014.
[15] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and Vi. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv prprint arXiv: 1312.6082*, 2013.
[16] S. Han, H. Mao and W. J. Dally, "Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR2016*, 2016.
[17] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580*, 2012.
[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 1, pp.221-231, 2013.
[19] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li, "Large-scale video classification with convolutional neural networks," *In Proc. CVPR*, pp.1725-1732, 2014.
[20] M. Kim and P. Smaragdis, "Bitwise neural networks," *CoRR*, abs/1601.06071, 2016.
[21] 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.
[22] V. Nair and G.E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *ICML*, 2010, pp. 807-814.
[23] https://github.com/charlyng/Embedded-Deep-Learning/tree/master/Benchmark-Performance
[24] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *ICCD2013*, 2013, pp.13-19.
[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," https://arxiv.org/pdf/1603.05279.pdf
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR2015*, 2015.
[27] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," *In Proc. CVPR*, pp.1701-1708, 2014.
[28] Theano, http://deeplearning.net/software/theano/
[29] Torch: A scientific computing framework for LUTJIT, http://torch.ch/
[30] A. Toshev and C. Szegedy, "Deeppose: Human pose estimatiion via deep neural networks," *In Proc. CVPR*, 2014.
[31] 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.
[32] 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," *FPGA2016*, 2016, pp. 26-35
[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.
[34] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *FPGA2015*, 2015, pp.161-170.
[35] 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," http://arxiv.org/pdf/1606.06160v2.pdf