一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-75,CPSY2018-85,RECONF2018-49(2019-01)

雑音畳込みニューラルネットワークとそのFPGA実装について

宗形　敦樹†　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: †munakata@reconf.ict.e.titech.ac.jp

あらまし　画像認識の分野で広く利用されている畳込みニューラルネットワーク (CNN: Convolutional Neural Network) は重みの数や乗算数が多いという問題がある．これらを解決するため, 本論文では雑音付加と point-wise (1×1) 畳込みを組み合わせた雑音畳込み層を用いる．既存研究の解析から, 雑音畳込み層だけでは入力データは偏っているため認識精度が低下することが判明している．本論文では, k 層までを既存の畳込み層で実現し, $k+1$ 層以降を雑音畳込み層で実現する Noise Convolutional Neural Network (NCNN) を提案する．これにより, 大部分の畳込み層を 1×1 畳込み層に換えることで重みの数と乗算数を削減しつつ, 雑音を加えることで認識精度劣化を抑えることができる．NCNN と既存 CNN の認識精度とパラメータ (重み) 量の比較を行った．CIFAR-100 データセットに関して, AlexNet ではパラメータを 88%, ResNet-18 では 96.2% 削減できた一方, 認識精度に関しては AlexNet では 2.2%, ResNet-18 では 1.8% に抑えることができた．また, 本論文では提案する NCNN を効率よく実行するアーキテクチャについて述べる．NCNN では $k+1$ 層以降は point-wise 畳込みのみ行われるため, 複雑なメモリアクセスアーキテクチャは不要であり, 単純かつ高速なアーキテクチャで実現可能である．提案 NCNN を Xilinx 社 ZCU102 FPGA 評価ボード上に実装した結果, クラス分類タスクに関しては Binary CNN と比較して同程度の速度を達成しつつ, 認識精度が約 10% 優れていた．

キーワード　Deep Learning, CNN, FPGA, 雑音畳込み Neural Network

A CNN with a Noise Addition for Efficient Implementation on an FPGA

Atsuki MUNAKATA†, Shimpei SATOU††, 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: †munakata@reconf.ict.e.titech.ac.jp

Abstract　This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.

Key words　Deep Learning, CNN, FPGA

1. はじめに

1.1 畳込みニューラルネットワーク

近年, 画像認識の分野では畳込みニューラルネットワーク（CNN : Convolutional Neural Network）が成功を収めている．その中でも, 畳込み層, プーリング層と全結合層で構成される CNN のモデルが広く使用されている．畳込み層で行われる（3 × 3）畳込み演算を図 1 に示す．畳込み演算は出力 y, 入力 $\boldsymbol{x}$, 重み $\boldsymbol{w}$, カーネル高さ h, 幅 w を用いると以下のように表せる．

$$y = \sum_{i=0}^{w} \sum_{j=0}^{h} x_{i,j} w_{i,j}$$

1.2 CNN の問題点

畳込み層では重みの数や乗算数が多いという問題がある．この問題を解決するために今まで様々な工夫が行われてきた．AlexNet [3] では大きなカーネルサイズ（11 × 11, 5 × 5 など）が用いられていたが, ResNet [4] 等では小さなカーネルサイズ（3 × 3）が用いられるようになり, 重みの数が削減された．また, 2 値化 CNN [5] やスパース化された CNN [6] によって計算効率が向上した．しかし, これらの工夫は全て計算量の多い畳

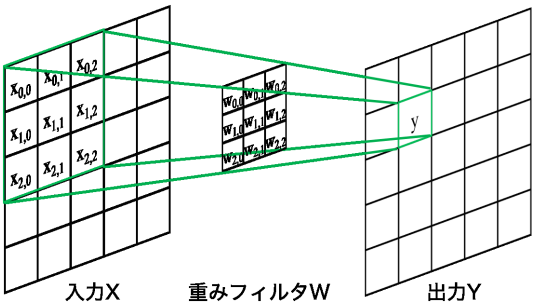

図 1　畳込み演算

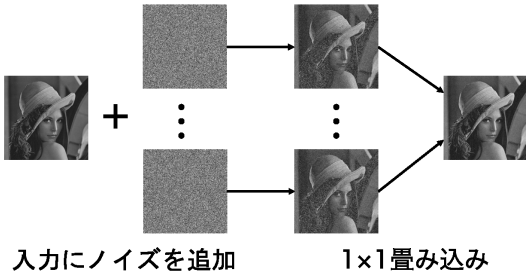

図 2　雑音畳込み層

込み演算を行うことを前提としている.

2.　雑音畳込みニューラルネットワーク

1.2 節で挙げた問題点を解決するために, 計算量の多い畳込み演算を行わない手法として, 畳込み層の代わりに摂動と呼ばれる微小雑音を利用した Perturbative Neural Networks (PNN) [1] が提案された. これは畳込み層を雑音畳込みに換える以外は CNN と同様の操作を行うものである. 本論文では回路実現方式に着目し, 摂動演算を雑音加算で実現することから, 以降, 雑音畳込み演算と呼ぶこととする. また, 雑音加算を行う畳込み演算を雑音畳込み演算と定義する.

2.1　雑音畳込み層

雑音畳込み層では入力の各画素に雑音を加え, 1×1 畳込みを行う. 各画素の雑音の値は手で設定した範囲において連続一様分布からサンプリングされた値を用い, 学習は行わない. そのため, 雑音畳込みにおける重みは 1×1 畳込みの重みのみである. 雑音畳込みで行う操作を図 2 に示す. なお, 図 2 では簡単のためバッチ正規化や活性化関数は省略している. 出力 $\boldsymbol{y}$、入力 $\boldsymbol{x}$、雑音 $\boldsymbol{n}$、 重み $\boldsymbol{v}$、雑音フィルタの数 N を用いると雑音畳込みの計算は以下のように表せる.

$$\boldsymbol{y} = \sum_{i=1}^{N} (\boldsymbol{x} + \boldsymbol{n}_i)\, v_i \tag{1}$$

2.2　重みの数の削減

畳込み層の問題点であった重みの数について考察する. 畳込み層のカーネル高さを h, 幅を w, 入出力チャネル数を p, q とすると, 雑音畳込み層と畳込み層の重みの数の比は以下のように表せる.

$$\frac{\text{雑音畳込み層の重みの数}}{\text{畳込み層の重みの数}} = \frac{pq}{phwq} = \frac{1}{hw}$$

雑音畳込み層では畳込み層に比べて重みの数を hw 分の 1 に削減, つまり畳込み層のカーネルサイズの分だけ削減できる.

2.3　乗算数の削減

畳込み層の問題点であった乗算数について考察する. 入力画像の高さを h', 幅を w' とすると, 雑音畳込み層と畳込み層の乗算数の比は以下のように表せる.

$$\frac{\text{雑音畳込み層の乗算数}}{\text{畳込み層の乗算数}} = \frac{ph'w'q}{phwh'w'q} = \frac{1}{hw}$$

雑音畳込み層では畳込み層に比べて乗算数を hw 分の 1 に削減, つまり畳込み層のカーネルサイズの分だけ削減できる.

しかし, GPU で実行する場合, DNN の演算ライブラリである cuDNN は 3×3 畳込み演算が高速化されるように最適化されており, 雑音を並列に計算できないことから実際のプログラム実行時間は hw 分の 1 にはならない.

2.4　雑音畳込み層の重みの導出

雑音畳込み層の重み $\boldsymbol{v}$ や雑音 $\boldsymbol{n}$ を導出する. 式 (1) を変形すると雑音畳込み層の計算は以下のように表せる.

$$\begin{aligned}\boldsymbol{y} &= \sum_{i=1}^{N} (\boldsymbol{x} + \boldsymbol{n}_i)\, v_i = \hat{\boldsymbol{X}}\boldsymbol{v} \\ &= (\boldsymbol{X} + \boldsymbol{N})\,\boldsymbol{v} = \left(\boldsymbol{x}\mathbf{1}^{\mathsf{T}} + \boldsymbol{N}\right)\boldsymbol{v}\end{aligned} \tag{2}$$

まず, 重み $\boldsymbol{v}$ を導出する. 入力の次元 d が $N = d$ の時は以下のように表せる.

$$\begin{aligned}\boldsymbol{v} &= \hat{\boldsymbol{X}}^{-1}\boldsymbol{y} = \left(\boldsymbol{x}\mathbf{1}^{\mathsf{T}} + \boldsymbol{N}\right)^{-1}\boldsymbol{y} \\ &= \left(\boldsymbol{N}^{-1} - \frac{\boldsymbol{N}^{-1}\boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}}{1 + \mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{x}}\right)\boldsymbol{y}\end{aligned}$$

ただし, 最後の式変形は Sherman-Morrison の公式 [7] による. また, $N < d$ の時は以下のように表せる.

$$\boldsymbol{v} = \left(\hat{\boldsymbol{X}}^{\mathsf{T}}\hat{\boldsymbol{X}}\right)^{-1}\hat{\boldsymbol{X}}^{\mathsf{T}}\boldsymbol{y}$$

このように, 出力 $\boldsymbol{y}$, 入力 $\boldsymbol{x}$, 雑音 $\boldsymbol{n}$ が与えられると重み $\boldsymbol{v}$ を求めることができる.

次に, 雑音 $\boldsymbol{n}$ を導出する. 畳込み演算は重みを用いた行列 $\boldsymbol{A}$ を用いて、以下のように表せる.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{3}$$

雑音畳込み層の式 (2) と畳込み層の式 (3) を連立して式変形すると, 以下のように表せる.

$$\begin{aligned}\boldsymbol{y} &= \left(\boldsymbol{x}\mathbf{1}^{\mathsf{T}} + \boldsymbol{N}\right)\boldsymbol{v} \\ &= \left(\boldsymbol{x}\mathbf{1}^{\mathsf{T}} + \boldsymbol{N}\right)\left(\boldsymbol{N}^{-1} - \frac{\boldsymbol{N}^{-1}\boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}}{1 + \mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{x}}\right)\boldsymbol{y} \\ &= \boldsymbol{A}\boldsymbol{x}\end{aligned}$$

$$\Rightarrow \boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{A}\boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{x} = \boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{x}\mathbf{1}^{\mathsf{T}}\boldsymbol{N}^{-1}\boldsymbol{A}\boldsymbol{x}$$

$$
\begin{aligned}
&\Rightarrow \boldsymbol{A}\boldsymbol{X}\boldsymbol{N}^{-1} = \boldsymbol{X}\boldsymbol{N}^{-1}\boldsymbol{A} \\
&\Rightarrow \left(\boldsymbol{X}^{+}\boldsymbol{A}\boldsymbol{X}\right)\boldsymbol{N}^{-1} = \boldsymbol{N}^{-1}\boldsymbol{A} \\
&\Rightarrow \left(\boldsymbol{X}^{+}\boldsymbol{A}\boldsymbol{X}\right)\boldsymbol{N}^{-1} - \boldsymbol{N}^{-1}\boldsymbol{A} = \boldsymbol{0} \\
&\Rightarrow \left(\boldsymbol{I}\otimes S_a + S_b^{\mathsf{T}}\otimes \boldsymbol{I}\right)\boldsymbol{N}^{-1} = S_c \qquad (4)
\end{aligned}
$$

ただし, $S_a = \boldsymbol{X}^{+}\boldsymbol{A}\boldsymbol{X}$, $S_b = \boldsymbol{A}$, $S_c = \boldsymbol{0}$ とする. 式 (4) はシルベスター行列方程式 [8] の形となっているため, S_a と $-S_b$ の固有値が異なる場合, 入力 $\boldsymbol{x}$, 行列 A が与えられると雑音 $\boldsymbol{n}$ を計算により求めることができる.

2.5 雑音畳込み層と畳込み層の等価性について

確率分布の観点から雑音畳込み層と畳込み層の関係について考える. 畳込み層において, 畳込みを行う入力の中心ピクセル x_c, 近傍ピクセル x_i, 相関係数 ρ, 中心ピクセルと近傍ピクセルの差 $\epsilon = x_i - x_c$, 近傍ピクセルの集合 $\mathbb{N}_c$ を用いる. ここで, 前提条件として $E(x_i) = 0$, $E\left(x_i^2\right) = \sigma^2$ を仮定する.

まず, 今後の計算に使用するため, $E(\epsilon_i)$, $E\left(\epsilon_i^2\right)$, $E(\epsilon_i\epsilon_j)$ を求める.

$$
\begin{aligned}
E(\epsilon_i) &= E(x_i - x_c) = 0 \\
E\left(\epsilon_i^2\right) &= E\left[(x_i - x_c)^2\right] = E\left(x_i^2 + x_c^2 - 2x_i x_c\right) \\
&= E\left(x_i^2\right) + E\left(x_c^2\right) - 2E(x_i x_c) \\
&= 2\sigma^2 - 2\rho\sigma^2 = 2\sigma^2\delta
\end{aligned}
$$

これらを用いると,

$$
\begin{aligned}
&E(\epsilon_i\epsilon_j) \\
&= E\left[(x_i - x_c)(x_j - x_c)\right] \\
&= E\left(x_c^2\right) - E(x_i x_c) - E(x_j x_c) + E(x_i x_j) \\
&= 2\sigma^2 - 2\rho\sigma^2 = 2\sigma^2\delta' \\
&= 2\sigma^2 - \rho\sigma^2 - \rho\sigma^2 - \hat{\rho}\sigma^2 \\
&= 2\sigma^2 - \rho\sigma^2 = \sigma^2\delta = (1/2)\,E\left(\epsilon_i^2\right)
\end{aligned}
$$

ただし, $\hat{\rho} \approx \rho$ と仮定する.

ここで, 畳込み演算は次のように式変形できる.

$$
\begin{aligned}
&y = \sum_{i=1}^{N} x_i w_i = x_c + \sum_{i\in\mathbb{N}_c} x_i w_i \\
&\Rightarrow x_c w_c + \sum_{i\in\mathbb{N}_c}(x_c + \epsilon_i) w_i = y \\
&\Rightarrow x_c\left(w_c + \sum_{i\in\mathbb{N}_c} w_i\right) + \sum_{i\in\mathbb{N}_c}\epsilon_i w_i = y \\
&\Rightarrow x_c + \sum_{i\in\mathbb{N}_c}\epsilon_i\left(\frac{w_i}{\sum_i^N w_i}\right) = \frac{y}{\sum_i^N w_i} \\
&\Rightarrow y' = x_c + \sum_{i\in\mathbb{N}_c}\epsilon_i w_i'
\end{aligned}
$$

ただし, $y' = \frac{y}{\sum_i^N w_i}$, $w_i' = \frac{w_i}{\sum_i^N w_i}$ とする.

ここで, $\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'$ に関して, $E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right)$, $E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right)^2$ を求める.

表 1　ANCNN と CNN の比較（ResNet-18, CIFAR-100）

	ANCNN	CNN
Accuracy(%)	73.6	73.2
Weight(MB)	5.1	134.9
ratio	0.038	1.000

$$
\begin{aligned}
&E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right) = 0 \\
&E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right)^2 \\
&= E\left(\epsilon_i^2 w_i'^2 + \cdots + \epsilon_1\epsilon_2 w_1' w_2' + \cdots\right) \\
&= E\left(\epsilon_i^2\right)\sum_{i=1}^{N} w_i'^2 + E(\epsilon_i\epsilon_j)\sum_i\sum_{j\neq i} w_i' w_j' \\
&= \left(2\sigma^2\delta\right)\|\boldsymbol{w}'\|_2^2 + \left(\sigma^2\delta\right)\sum_i\sum_{j\neq i} w_i' w_j' \\
&= 2\sigma^2\delta\left[\|\boldsymbol{w}'\|_2^2 + (1/2)\sum_i\sum_{j\neq i} w_i' w_j'\right] \\
&= 2\sigma^2\delta'
\end{aligned}
$$

ただし, $\delta' = \left[\|\boldsymbol{w}'\|_2^2 + (1/2)\sum_i\sum_{j\neq i} w_i' w_j'\right]$ とする.

以上より, $E(n_c) = E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right) = 0$, $E\left(n_c^2\right) = E\left(\sum_{i\in\mathbb{N}_c}\epsilon_i w_i'\right)^2 = 2\sigma^2\delta'$ となる雑音 n_c を用いると, 統計の観点から見て雑音畳込み層は畳込み層と等価であると考えられる.

2.6 ANCNN と CNN の比較

全て雑音畳込み層で構成した CNN を All NCNN (ANCNN) とする. ANCNN と CNN の認識精度を比較した結果と全層における重みを表 1 に示す. 表中の Weight は畳込み層と全結合層を合わせた全層の重みを示す. また, ratio は以下のように計算する.

$$
\text{ratio} = \frac{\text{ANCNN 全層の重みの数}}{\text{CNN 全層の重みの数}}
$$

今回の条件において NCNN は CNN に比べて同等の認識精度を示しており, 重みの数を 0.038 倍に減らすことができている。

3. NCNN

3.1 評 価 方 法

本稿では認識精度は以下のように計算する (注1).

$$
\text{認識精度 (\%)} = \frac{\text{正しく認識した画像の枚数}}{\text{全画像の枚数}}
$$

この評価方法における ANCNN と CNN の認識精度を比較した結果を表 2 に示す. 本稿の評価方法において ANCNN は CNN に比べて認識精度が 8.4% 低下した. よって, 本稿の評価方法における ANCNN について検討する必要がある.

(注1)：参考にした元の論文 [1] では評価指標をスムージング精度で算出しており, 特にクラス分類タスクの精度を表す指標としては必ずしも適切であるとは言えない.

表 2　本稿の評価方法における ANCNN と CNN の比較（ResNet-18, CIFAR-100）

	ANCNN	CNN
Accuracy(%)	66.5	74.9

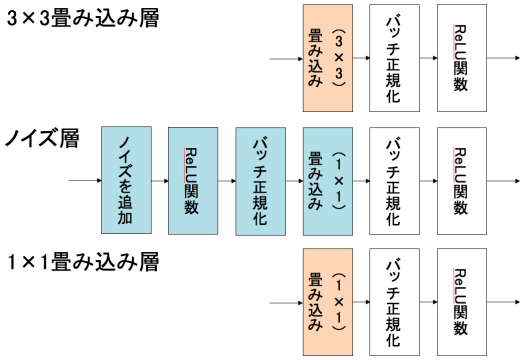

図 3　3 × 3 既存畳込み層, 雑音畳込み層, 1 × 1 畳込み層

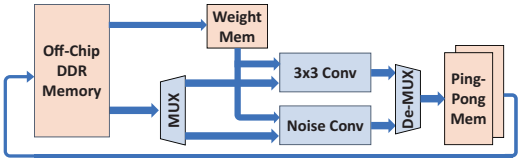

図 4　NCNN アーキテクチャ全体の構成

3.2　本稿の方針

本稿では以下の 3 点に関して考察を行った.

（1）　本稿の評価方法における NCNN の認識精度

（2）　NCNN が有効な場面

（3）　雑音畳込み層において雑音を加える必要性

3.3　NCNN (Noise Convolutional Neural Network)

本稿では畳込み層と雑音畳込み層を組み合わせた NCNN を提案する. これは, 1〜k 層目を畳込み層のままとし, k+1 層目以降の畳込み層を雑音畳込み層に換える手法を指す. この手法の k の値を変化させて様々なモデルで実験を行うことにより, 3.2 節の考察の初めの 2 点に関して考察を行った.

また, 雑音畳込み層において雑音を加えることの影響を考察するため, 畳込み層を雑音畳込み層に換えるのではなく 1 × 1 畳込み層に換える手法として Pointwise Convolutional Neural Network（PCNN）を提案し, NCNN との比較を行った. これにより, 3.2 節の考察の最後の 2 点に関して考察を行った.

CNN, NCNN, PCNN でそれぞれ使用する 3 × 3 畳込み層, 雑音畳込み層, 1 × 1 畳込み層の詳細を図 3 に示す.

4.　NCNN 専用回路の実装

4.1　全 体 構 成

前章で示した通り, NCNN では既存畳込み層と雑音畳込み層のハイブリッドで実現する必要がある. 図 4 に NCNN アーキテクチャの全体構成を示す. 本論文では第 1 層を既存畳込み層で構成し, 2 層以降を雑音畳込みで実現する. クラス分類タスクに関して, 多くの代表的な CNN はフル結合層を最後に持つ

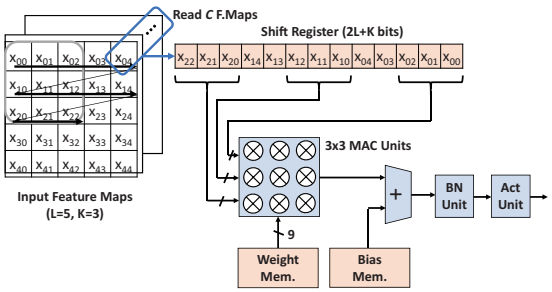

図 5　Sliding window 方式既存畳込み回路

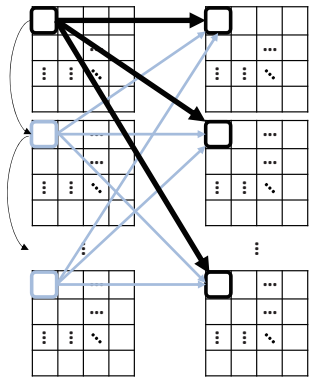

図 6　メモリアクセスパターン (SCMK 方式)

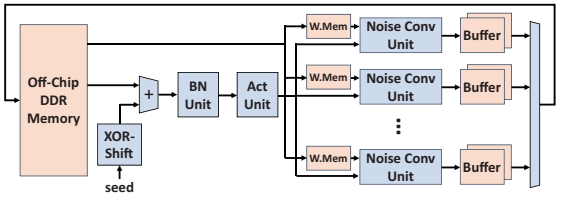

図 7　雑音畳込み回路

が, Global average pooling 層と point-wise 畳込み層で等価になることが知られており (Nakahara FPL'17), 本論文でもフル結合層はこれらの層で実現することとする.

4.2　既存畳込み層

畳込み層の実装法は Binary CNN に代表される低ビット化に適した Sliding Window 方式 [12] と, スパース性を利用した間接メモリアクセス方式 [13] に大別される. 既存研究や本論文でも言及した通り, 入力データはほぼ全て偏っておりスパース率が上がらないことが多い. 既存研究 [14] が示す通り, 前半部分を低ビット化しても後段のビット数を十分に持たせた混合精度方式で精度劣化は抑えることが可能である. 従って, 本論文ではスパース化よりも低ビット化によるパラメータ削減を想定した Sliding Window 方式を採用する. 図 5 に Sliding Window 方式畳込み回路を示す. 入力データを保持するラインバッファを $k-1$ 本と k 個のバッファを用意し, カーネルサイズ $k \times k$ に適合する入力データを同時アクセスする. $k \times k$ 個の積和演算を入力チャネル数 c 個並列に実行する. 第 1 層では入力チャネル数はカラー画像であることから $c = 3$ であり, 同時実行してもハードウェア量は増加しない. 出力はオフチップメモリへ格納する.

4.3 雑音畳込み層

図 3 が示すように, 雑音畳込み層は雑音を生成する乱数生成部, 活性化関数部, point-wise 畳込み, バッチ正規化で構成される. 乱数生成に関しては様々なハードウェア実装が報告されているが, 本論文では単純な演算で周期の長い XOR-Shift 法を用いて実装する. 本論文ではページ数制限のため 32 ビット精度で設計し, 乱数の周期性の調査 (ビット精度) は別稿で述べることとする. 雑音畳込みの学習時に雑音を定数として扱う (即ち偏微分値をゼロとする) ことで学習を容易に実装できるため, 本論文では乱数の初期値をメモリから読み出し, 疑似乱数を生成することとする. 一方で, 畳込み演算に関しては point-wise なのでラインバッファは不要である. 図 6 に示すように, 入力データを外部メモリから読出す (SCMK: Single channel multiple kernels) 方式を採用するため, m 個の出力チャネルを同時に計算する方式を採用する [15]. 図 7 に雑音畳込み回路を示す. 各チャネルの値を逐次読出すことで m 個の point-wise 畳込みが完了する. 予めオフチップメモリにはチャネル方向に逐次アクセス可能になるようにデータを転置して格納しておく. これらの処理を特徴マップサイズ分行うことで 1 層分の処理を完了する. 出力値はバッチ正規化処理, 活性化関数処理を行い, ping-pong バッファに格納する. 従って, 畳込み演算中に外部メモリに書き出しを同時に行えるので, 高速な処理が可能である.

5. 実験結果

本稿では Classification の実験を行い, データセットは CIFAR-100 [9], PASCAL VOC [11], モデルは AlexNet, ResNet-18 を使用した. なお, PASCAL VOC は Object Detection 用のデータセットであるが, クラス分類タスクに関しては対象となる物体を長方形の枠で切り取り, 128 × 128 のサイズとなるように黒い画像を用いてパディングをすることでデータセットを作成する.

5.1 AlexNet

AlexNet における NCNN と ANCNN, CNN の認識精度を比較した結果と全層における重みを表 3 に示す. 表中の Difference は NCNN と ANCNN の認識精度の差, つまり雑音を加えることによりどれだけ認識精度の低下が抑えられているかを示す.

conv$k$-NCNN を k 層まで既存畳込み, $k+1$ 層以降を雑音畳込みで構成した NCNN とする. 認識精度に注目したとき, CIFAR-100 において conv0-NCNN, conv1-NCNN の場合, 雑音を加えたときは雑音を加えなかったときに比べて認識精度の低下を抑えられた. conv0-NCNN の場合は認識精度の低下を 2.1% 抑えられているが, CNN と比べて認識精度が 20.7% 低下しているため, 1 層目を雑音畳込み層や 1 × 1 畳込み層とすることは有効ではない. conv1-NCNN の場合は 2.2% 抑えられており, CNN と同等の認識精度を示した. conv2-NCNN の場合. 認識精度の低下を抑えられていないため, 雑音を加えることによるメリットはなかった.

PASCAL VOC では, CIFAR-100 と比べて雑音畳込みの有無により差の生じるかどうかに変化がなかった. 提案手法は適したモデルにおいて, データセットのサイズを 32 × 32 から 128 × 128 に大きくしても認識精度の低下を抑えられた.

表 3 クラス分類タスクの比較 (AlexNet)

dataset		conv0	conv1	conv2	CNN
CIFAR-100	Accuracy(NCNN)(%)	29.1	49.4	49.5	49.8
	Accuracy(ANCNN)(%)	27.0	47.2	50.0	-
	Difference(%)	2.1	2.2	-0.5	-
PASCAL VOC	Accuracy(NCNN)(%)	43.6	72.2	74.8	75.2
	Accuracy(ANCNN)(%)	42.6	70.9	74.5	-
	Difference(%)	1.0	1.3	0.3	-
	Weight(MB)	1.1	1.2	2.4	10.0
	ratio	0.11	0.12	0.24	1.00

表 4 クラス分類タスクの比較 (モデル: ResNet-18, データセット: CIFAR-100)

	conv1	conv3	conv7	conv11	conv15	CNN
Accuracy(NCNN)(%)	66.5	68.1	72.2	74.2	75.4	74.9
Accuracy(PCNN)(%)	64.7	66.6	71.0	74.1	75.5	-
Difference(%)	1.8	1.5	1.2	0.1	-0.1	-
Weight(MB)	5.1	5.4	6.4	10.6	47.8	134.9
ratio	0.038	0.040	0.048	0.079	0.354	1.000

全層の重みに注目したとき, NCNN は CNN と比べて重みの数を 0.11〜0.24 倍に減らすことができた.

5.2 ResNet

ResNet-18 は 1〜17 層目を 3 × 3 畳込み層, 18 層目を全結合層とし, 1, 5, 9, 13, 17 層目の後にプーリング層を挿入したモデルである. なお, 1 層目を 7 × 7 畳込み層とするモデルが広く使用されているが, 本稿では使用するデータセットに対して精度を出すために 3 × 3 畳込み層としている. ResNet-18 における NCNN と ANCNN, CNN の認識精度を比較した結果と全層における重みを表 3 に示す.

認識精度に注目したとき, conv1-NCNN の場合, 雑音を加えたときは加えなかったときに比べて, 認識精度の低下を 1.8% 抑えられた. k の値を増やし, 畳込み層の数を増やしていった場合, 認識精度の低下の抑制は徐々に減少していき, conv11-NCNN において認識精度の低下を抑えることができなくなった. これは, モデルが大きくなり複雑度が増した場合, 雑音を加えることによる影響が少なくなってしまうためであると考えられる.

全層の重みに注目したとき, 3 × 3 畳込み層が 11 層までの場合は CNN と比べて重みの数を 0.038〜0.079 倍に減らすことができた. NCNN は ResNet-18 のようにモデルが大きいほど重みの数を減らすことができた.

5.3 学習時間

ResNet-18 の NCNN と CNN の 1 epoch 毎の学習時間を比較した結果を表 5 に示す. なお, CPU は i7-8700K, GPU は GeForce GTX 1080 Ti, OS は Ubuntu 16.04.3 LTS, ディープラーニングフレームワークは PyTorch 0.4.0, Python3.6.5 を使用し, batch size=10 として実験を行った 畳込み層が 11 層までの場合は, CNN と比べて学習時間を 0.60〜0.69 倍にできた.

conv1-NCNN が conv3-NCNN と比べ, 雑音畳込み層が増えたにも関わらず学習時間が増えたのは, 3 × 3 畳込み層を 1 ×

表 5　1epoch 毎の学習時間（NCNN, ResNet-18, CIFAR-100）

	conv1	conv3	conv7	conv11	conv15	CNN
学習時間 (s)	50.3	48.8	45.1	46.4	59.1	75.2

表 6　Comparison with other binarized CNN realizations on the FPGA.

Implementation (Year)	Zhao et al. (2017)	FINN (2017)	中原ら (2017)	提案手法
CNN	Binary	Binary	Binary	Noise Conv.
Clock (MHz)	143	166	143	199
# LUTs	46900	42823	14509	51136
# 18Kb BRAMs	94	270	32	256
# DSP Blocks	3	32	1	256
Test Error	12.27%	19.9%	18.2%	8.2%
Time [msec]	5.94	2.24	2.37	0.85
(FPS)	(168)	(445)	(420)	(1167)
Power [W]	4.7	2.5	2.3	2.5
FPS/Watt	35.7	178.0	182.6	466.8

1 畳込み層にした際に短縮される時間よりも, 追加したバッチ正規化にかかる時間の方が長いためである.

5.4 FPGA 実装と既存手法との比較

NCNN を Xilinx 社 ZCU102 評価ボード (Xilinx Zynq UltraScale+ MPSoC FPGA (ZU9EG) 搭載, 274,080 LUTs, 548,160 FFs, 1,824 18Kb BRAMs, 2,520 DSP48Es) に実装した. 用いたツールは Xilinx 社 SDSoC2018.2 であり, タイミング制約を 199.9 MHz に設定した. 実装した NCNN は, 18Kb BRAM を 256 個, DSP48E を 256 個, FF を 32384 個, LUT を 51136 個使用した. また, 処理速度は 1167 FPS (Frames Per Second) であった. このとき, 動作時の電力は 2.5 W であった.

表 6 に既存手法との比較を示す. クラス分類タスクでは単純な Binary CNN が多用されており, 高速かつ低電力な推論ハードウェアが提案されている [16]〜[18]. しかし, Binary 化により認識精度が低下する欠点がある. 提案手法は認識精度をほぼ維持しつつ, 高速な回路を実現可能である. 既存の Binary CNN と比較して, ハードウェア量が増加するものの, 高速かつ高精度な認識率を達成できた.

6. 結　　論

本稿では, 1〜k 層目を畳込み層とし k+1 層目以降を雑音層とする NCNN を提案し, 畳込み層を 1 × 1 畳込み層とすることで重みの数と乗算数を削減し, 本稿の評価方法において雑音を加えることで認識精度の低下を抑えることができた. この手法は小さく複雑でないモデルでは有効であり, conv1-NCNN は, CIFAR-100 において, AlexNet では CNN と比べて重みの数を 0.12 倍に削減し, conv1-PCNN と比べて認識精度の低下を 2.2% 抑えることができ, ResNet-18 では CNN と比べて重みの数を 0.038 倍に削減し, conv1-PCNN と比べて認識精度の低下を 1.8% 抑えることができた.

また, 本論文では雑音畳込み CNN に適したアーキテクチャを提案した. 提案手法を ZCU102 FPGA 評価ボードに実装した. その結果, 既存の Binary CNN と比較して, ハードウェア量が増加するものの, 高速かつ高精度な認識率を達成できた.

文　　献

[1] F. Juefei-Xu, V. N. Boddeti, and M. Savvides. Perturbative Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 1, 2018.

[2] F. Juefei-Xu. Perturbative Neural Networks (PNN). https://github.com/juefeix/pnn. pytorch. Acceced: 2018-9-3.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (NIPS), pages 1097-1105, 2012.

[4] K.He, X.Zhang, S.Ren, and J.Sun. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770-778, 2016.

[5] 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 (NIPS), pages 3105-3113, 2015.

[6] J. Park, S. Li, W. Wen, H. Li, Y. Chen, and P. Dubey. Holistic SparseCNN: Forging the Trident of Accuracy, Speed, and Size. arXiv preprint arXiv:1608.01409, 2016.

[7] J. Sherman and W. J. Morrison. Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row of the Original Matrix (abstract). Annals of Mathematical Statistics, 20:621, 1949.

[8] J.Sylvester. Surl 'equations en matrices px = xq. C.R.Acad. Sci. Paris, 99(2):67-71,115-116, 1884.

[9] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. CIFAR, 2009.

[10] PyTorch. https://pytorch.org. Acceced: 2018-9-5.

[11] The PASCAL Visual Object Classes Homepage, http://host.robots.ox.ac.uk/pascal/VOC/. Acceced: 2018-12-14.

[12] B. Bosi, G. Bois and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 7, No. 3, pp. 299-308, 1999.

[13] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," *ISCA*, 2017, pp.27-40.

[14] H. Nakahara, H. Yonekawa, T. Fujii and S. Sato, "A Lightweight YOLOv2: A Binarized CNN with A Parallel Support Vector Regression for an FPGA," *FPGA*, 2018, pp.31-40.

[15] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "CNP: An FPGA-based processor for Convolutional Networks," *FPL*, 2009, pp.32-37.

[16] 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.

[17] 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.

[18] H. Nakahara, T. Fujii and S. Sato, "A fully connected layer elimination for a binarizec convolutional neural network on an FPGA," *FPL*, 2017, pp.1-4.