課題研究報告書

画像分類モデル保護手法の FPGA 実装と検証

小俣 直史

主指導教員　井口 寧 教授
審査委員主査　井口 寧 教授
審査委員　田中 清史 教授
青木 利晃 教授
冨田 尭 准教授

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

2025 年 3 月

概要

近年，社会における深層ニューラルネットワーク（Deep Neural Network: DNN）と，その活用が進んでいる．DNN モデルはデータセットの学習によって生成され，入力データから推論を実行し，出力データを生成するための”層構造アーキテクチャ”と，各層に含まれる重み・バイアス パラメータから構成される．特に，高性能な DNN モデルの学習には膨大なデータセットと計算資源，専門知識が必要とされ，学習済みの DNN モデルは知的財産に値する．しかし，組み込みコンピュータや FPGA 等のエッジデバイスをターゲットとし，サイドチャネル攻撃等の不正アクセスによって DNN モデルを不正取得される可能性が存在する．

このような経路によって取得されてしまった DNN モデルの更なる不正な利用・配布を防ぐため，エッジデバイス上の DNN モデルを保護する手法が提案されてきた．Manaar Alam らによって提案された NN-Lock アルゴリズム [1] は組み込みコンピュータを対象とした保護手法の 1 つである．DNN モデルのうち，ノード間接続の強さを示す係数重みパラメータ (Weight Parameter) を AES アルゴリズムによって全て暗号化し，推論時に正しい利用者が正しい暗号鍵を使用することで一時的に復号化する．一方で，窃取者等の不正な利用者が誤った暗号鍵を使用した場合，暗号化されたパラメータの値の変化が後続の全ての層に伝播し，最終的な推論結果が暗号化前のモデルと異なるものになる．これにより，正しい復号化キーを持たない攻撃者が DNN モデルを窃取しても，重みパラメータが暗号化されているため DNN モデルの推論精度が大幅に低減し，利用することは困難となり，DNN モデルの不正な利用・配布を防ぐことができる．

近年，FPGA（Field-Programmable Gate Array）を自動車の先進運転支援システムのプロセッサとして使用し，搭載した DNN モデルによって推論処理を実行する事例が増加傾向にある．本研究では，NN-Lock アルゴリズムを FPGA 上のハードウェア化された DNN モデルに初めて適用する．具体的には，重みパラメータが既に NN-Lock アルゴリズムによって暗号化された DNN モデルと NN-Lock アルゴリズムの復号化機構を FPGA 上に構築する手法を提案する．その際，NN-Lock アルゴリズムによって暗号化された重みパラメータを FPGA 上のメモリ BRAM に書込む直前 (BRAM 書込前 NN-Lock) と DNN 推論を実行する直前 (DNN 実行前 NN-Lock)，いずれのタイミングで重みパラメータを復号化するかという”NN-Lock の適用箇所”と DNN，NN-Lock モジュールの処理並列数を変更して実装する．あわせて，NN-Lock の FPGA 実装に適した DNN 推論演算モジュール単体の実装手法も提案する．

その上で，提案手法の運転支援システムとしての活用を想定し，提案手法の実装・評価から得られた回路量と推論処理時間より，実際に流通する FPGA ボードやカメラモジュールの性能を参考に，回路規模と推論処理時間の要求要件を考察する．また，NN-Lock 適用箇所における重みパラメータの窃取耐性についても，FPGA を対象とした代表的な攻撃手法を複数取り上げ考察する．以上の観点から，幅広いハードウェアを対象に，運転支援システムとしての活用の想定下で，FPGA

上で動作する DNN モデルに NN-Lock を適用する際に必要な情報を提供する.

Abstract

In recent years, Deep Neural Networks (DNNs) and their applications in society have been increasing. DNN models are generated by training datasets and consist of a “layered architecture” for performing inference from input data and generating output data, as well as weight and bias parameters contained in each layer. In particular, training a high-performance DNN model requires a huge amount of data sets, computational resources, and expertise, and a trained DNN model is considered intellectual property. However, there is a possibility that DNN models can be illegally obtained through unauthorized access such as side-channel attacks targeting edge devices such as embedded computers and FPGAs.

In order to prevent further unauthorized use and distribution of DNN models that have been obtained through such channels, methods have been proposed to protect DNN models on edge devices, and the NN-Lock algorithm[1] proposed by Manaar Alam et al. The AES algorithm encrypts all the coefficient weight parameters of the DNN model, which indicate the strength of the connections between nodes, and temporarily decrypts them by using the correct encryption key by the correct user at the time of inference. On the other hand, if an unauthorized user, such as a thief, uses the wrong encryption key, changes in the values of the encrypted parameters are propagated to all subsequent layers, and the final inference result is different from the model before encryption. This makes it difficult for an attacker without the correct decryption key to steal and use the DNN model because the accuracy of the DNN model inference is greatly reduced due to the encrypted weight parameters, thus preventing unauthorized use and distribution of the DNN model.

In recent years, field-programmable gate arrays (FPGAs) are increasingly being used as processors in advanced driver assistance systems for automobiles to perform inference processing using on-board DNN models. In this study, I apply the NN-Lock algorithm to a hardware DNN model on an FPGA for the first time. Specifically, I propose two methods to construct a DNN model whose weight parameters are already encrypted by the NN-Lock algorithm and a decryption mechanism for the NN-Lock algorithm on FPGA.

The first is “just before writing the weight parameters encrypted by the NN-Lock algorithm to the memory BRAM on FPGA (NN-Lock: Before writing to BRAM)” and the second is “just before executing DNN inference (NN-Lock: Before DNN execution).” I implemented and evaluated the logic patterns of the two aforementioned application points of NN-Lock modules and the number of parallel processes of the DNN module and the NN-Lock module. In addition, I also propose a method of implementing a stand-alone DNN inference computation module suitable for FPGA implementation of NN-Lock.

Next, based on the circuit size and inference processing time obtained from the implementation and evaluation of the proposed method, I consider the requirements for circuit size and inference processing time with reference to the performance of FPGA boards and camera modules that are actually available in the market. I also discuss the resistance to theft of weight parameters at the point where NN-Lock is applied, by discussing several typical attack methods targeting FPGAs. In light of the above, this paper provides information necessary for applying NN-Lock to DNN models running on FPGAs for a wide range of hardware under the assumption that they will be used in driver assistance systems.

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究背景

近年，自動車の先進運転支援システムでは，システムのコアとなる組込コンピュータ上への DNN(Deep Neural Network) モデルの搭載・活用が進んでいる．DNN モデルはデータセットの学習によって生成され，入力データから推論を実行し，出力データを生成するための層構造 ”アーキテクチャ”と，各層に含まれる重み・バイアス パラメータから構成される．

学習には膨大なデータセットと計算資源，専門知識が必要とされることから，エッジデバイスへの不正アクセスによって，パラメータを不正利用・再配布する脅威が想定されている．この脅威に対し，重みパラメータ (Weight Parameter) を暗号化し，推論時に正しい復号化キーを使用することで一時的に復号化し，推論終了後直ちに再暗号化する手法が提案されている．

その手法の 1 つとして，直近では， 2022 年に NN-Lock[1] アルゴリズムが Alam らによって提案されている．この手法は，モデルの全層の重みパラメータを AES 暗号化の一部手法を用いて暗号化する．

一方で，近年，スバル社の自動車に代表されるように，先進運転支援システム等の，画像分類処理を必要とする場面での FPGA(Field Programmable Gate Array) 組込プロセッサの採用事例が増加傾向にある．

FPGA は，設計者が論理回路の構成をプログラムできるデバイスである．製造後は容易に回路構成を変更できない LSI(Large Scale Integration) と異なり，回路構成を何度でも再構成することができるという特徴から，開発期間の短縮，回路の不具合・脆弱性の容易な修正を実現するため，上記の事例で採用が進んでいる．

先進運転システムにおける FPGA プロセッサの採用事例の増加を見越し，組込コンピュータ ソフトウェア向けの DNN モデル保護アルゴリズムである NN-Lock を，FPGA 上に構築された DNN モデルに対して複数の条件下で適用し評価を行うことで，当該アルゴリズムの FPGA における有用性を示す．

1.2　研究目的と成果

本研究では，重みパラメータが既に NN-Lock アルゴリズムによって暗号化された DNN モデルと NN-Lock アルゴリズムの復号化機構を FPGA 上に構築する．そ

の際，NN-Lock アルゴリズムによって暗号化された重みパラメータを FPGA 上のメモリ BRAM に書込む直前と DNN 推論を実行する直前，いずれのタイミングで重みパラメータを復号化するかという”NN-Lock の適用箇所”と DNN，NN-Lock モジュールの並列数を変更して実装する．

続いて，提案手法の運転支援システムとしての活用を想定し，提案手法の実装・評価から得られた回路量と推論処理時間より，実際に流通する FPGA ボードやカメラモジュールの性能を参考に，回路規模と推論処理時間の要求要件を考察する．また，NN-Lock 適用箇所における重みパラメータの窃取耐性についても考察する．以上の観点から，幅広いハードウェアを対象に，運転支援システムとしての活用の想定下で，FPGA 上で動作する DNN モデルに NN-Lock を適用する際に必要な情報を提供する．

本研究の成果は，第一に流通する FPGA ボードの価格帯と回路規模等の性能面に応じた実装手法を提案し，提案手法が幅広いボードで動作可能であることを示した．第二に運転支援システム採用時の要求要件について，リアルタイム処理を想定してカメラモジュールのキャプチャレートと提案手法の推論処理時間を比較し，採用に向けて解決すべき課題を明らかにした．第三に，FPGA における NN-Lock の適用箇所毎の重みの窃取耐性について，FPGA を対象とした攻撃手法を複数取り上げ，適用箇所毎の攻撃手法の実現可能性を議論した．

第2章　研究の背景と関連研究

2.1　はじめに

この章では，本研究の先行研究である NN-Lock アルゴリズム [1] と，その前提となる DNN モデルと AES Key Scheduling について説明する．2.4 章では，実際に NN-Lock アルゴリズムを再現実装し，論文中の評価結果と比較し再現できていることを確認する．最後に 2.5 章で，FPGA アーキテクチャの概要を説明する．

2.2　DNN モデルとデータセット

2.2.1　全結合層

はじめに，NN-Lock の保護対象である DNN モデルの構造について説明する．

全結合層とは，DNN モデルにおいて，層状に並べたパーセプトロンが隣接層間で結合された層のことであり，MLP (Multi-Layer Perceptron, 多層パーセプトロン) の 1 つである [2]． 図 2.1, 2.2 へ全結合層の概要図を示す．

全結合層は 1 つの入力層と複数の中間層，1 つの出力層の順で構成され，データは入力から出力へ，一方向にのみ伝搬するネットワーク構造を有する．パーセプトロン間には，結合の強さを示す係数である，行列形式の重みパラメータとベクトル形式のバイアスパラメータが設定される．

図 2.1 に全結合層中間層の概要図を示す．中間層第 t 層 第 i 要素における 1 個当たりのパーセプトロンの重み付き和 $u_i^{(t)}$ と出力 $h_i^{(t)}$ は， $h_i^{(t-1)}$ を前の層の出力，$w_i^{(t)}$ を重みパラメータ，$b_s^{(t)}$ をバイアスパラメータ，f を活性化関数，N を入力層の出力数とすると，

$$\boldsymbol{u}_i^{(t)} = \sum_{j=0}^{N} h_j^{(t-1)} w_{ji}^{(t)} + b_i^{(t)} \tag{2.1}$$

$$\boldsymbol{h}_i^{(t)} = f(u_i^{(t)}) \tag{2.2}$$

で表される．

また，行列の積を用いて，中間層第 t 層目の重み付き和のベクトル $\boldsymbol{U}^{(t)}$ は，中間層 $t-1$ 層目の出力のベクトル $\boldsymbol{H}$，重みパラメータの行列 $\boldsymbol{W}^{(t)}$ ，バイアスパラメータのベクトル $\boldsymbol{B}^{(t)}$ は以下の式 2.3 でまとめて表すことができる．

$$\boldsymbol{U}^{(t)} = \boldsymbol{H}\boldsymbol{W}^{(t-1)} + \boldsymbol{B}^{(t-1)} \tag{2.3}$$

但し，$\boldsymbol{U}^{(t)}$，$\boldsymbol{H}$，$\boldsymbol{W}^{(t)}$，$\boldsymbol{B}^{(t)}$ は下記のとおりである．ここで，N は入力数，M は出力数である．

$$\boldsymbol{U}^{(t)} = \left(u_1^{(t)},\ \cdots\ u_i^{(t)},\ \cdots\ u_M^{(t)}\right), \boldsymbol{H^{(t)}} = \left(h_1^{(t-1)},\ \cdots\ h_j^{(t-1)},\ \cdots\ {}_N^{(t-1)}\right), \tag{2.4}$$

$$\boldsymbol{W}^{(t)} = \begin{pmatrix} w_{11}^{(t)} & \cdots & w_{1i}^{(t)} & \cdots & w_{1M}^{(t)} \\ \vdots & \ddots & & & \vdots \\ w_{j1}^{(t)} & & w_{ji}^{(t)} & & w_{jM}^{(t)} \\ \vdots & & & \ddots & \vdots \\ w_{N1}^{(t)} & \cdots & w_{Ni}^{(t)} & \cdots & w_{NM}^{(t)} \end{pmatrix}, \boldsymbol{B}^{(t)} = \left(b_1^{(t)},\ \cdots\ b_i^{(t)},\ \cdots\ b_M^{(t)}\right) \tag{2.5}$$

活性化関数の 1 つに ReLU (Rectified Linear Unit)[3] が存在する． ReLU は，式 2.6 で表されるように単純な演算のみで計算できるため，計算コストが小さく高速に学習を行うことができる．

$$\mathrm{ReLU}(\mathrm{u_i^{(t)}}) = \begin{cases} u_i^{(t)} & \text{if } u_i^{(t)} > 0 \\ 0 & \text{if } u_i^{(t)} \leq 0 \end{cases} \tag{2.6}$$

以上ののパラメータと関数は図 2.2 のように結合される．

参考までに，本研究の実験・評価に使用したモデルの概要図を図 2.3 に示す．このモデルは，0∼9 の手書き数字画像を識別するためのデータセット MNIST と，中間層として 全結合層 2 層から構成される．推論対象の画像のパラメータを入力として取り，最終中間層の 10 パラメータより，最も値の大きいパラメータの番号を推論結果のラベルとして返す．

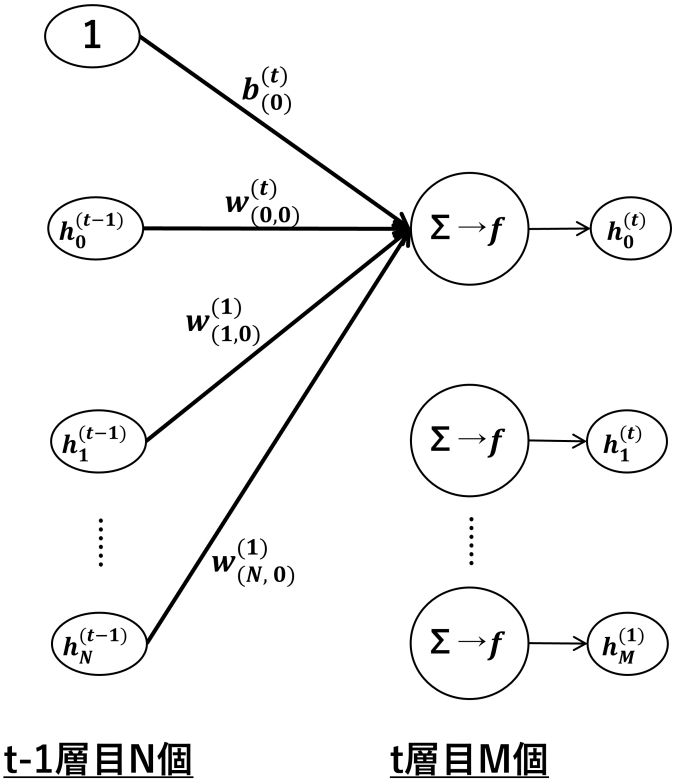

図 2.1: 全結合層 (1 演算単位) の概要図

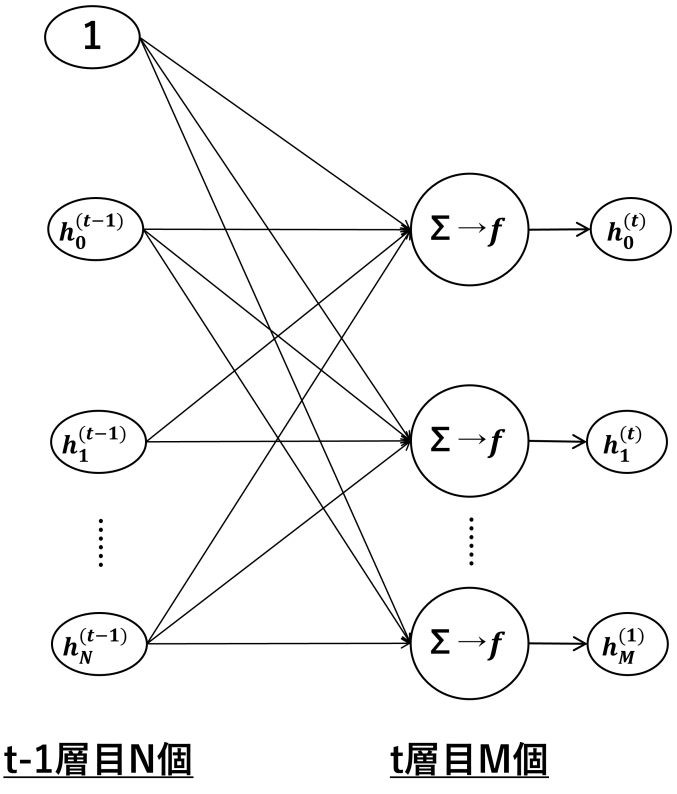

図 2.2: 全結合層 全体の概要図

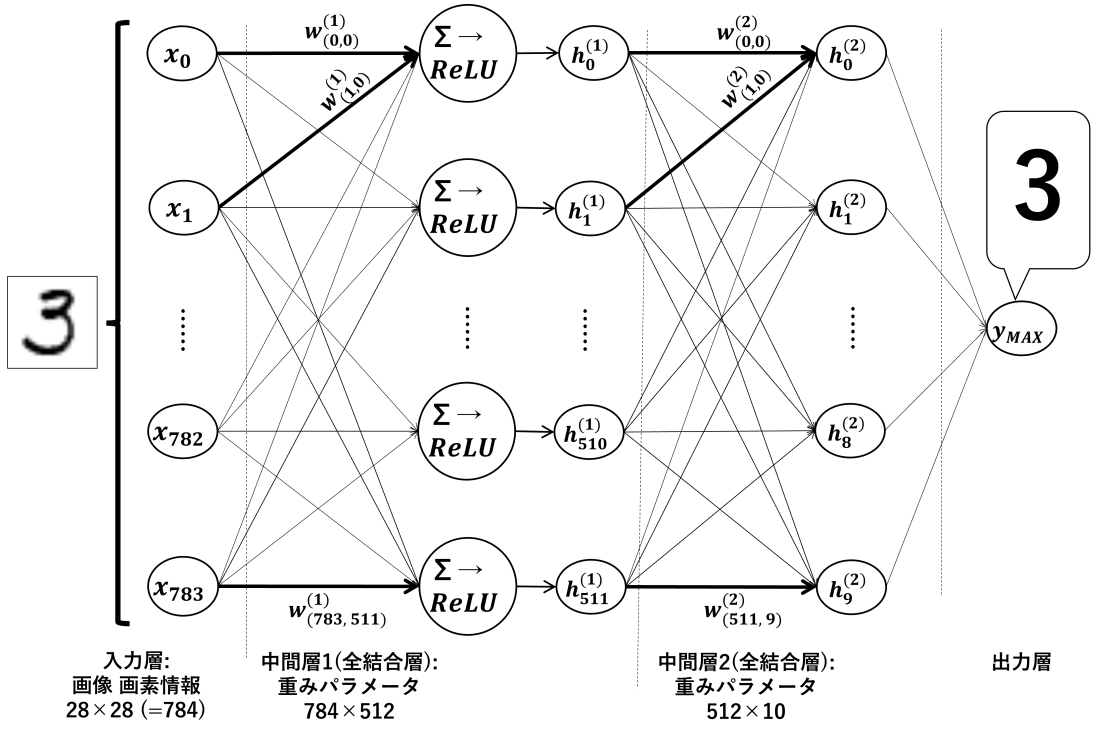

図 2.3: 全結合層 2 層構成 MNIST DNN モデルの概要図

2.2.2 DNNの量子化

続いてDNNモデルの量子化について説明する．量子化とは，通常単精度浮動小数点等の高精度な形式で扱われる画像パラメータや重みパラメータ等の各種パラメータを，8bit整数等の低精度の値に変換してモデル内に保存し，推論時に精度を復元して演算する手法である[4]．この手法によりDNNモデルのサイズを削減できる．NN-Lockでも重みパラメータが符号付8bit整数に量子化されたモデルを対象に保護を行う．

単精度浮動小数点数値である重みパラメータ $W^{(t)} \in \mathrm{FP32}^{\mathrm{M}\times\mathrm{N}}$ について，$W^{(t)}$ の要素の最大値を α，最小値を β とおく．$W^{(t)}$ を q bit整数値 $-2^{q-1} \leq w_q \leq 2^{q-1}-1$ に変換する際に必要なパラメータであるScale Factor($s^{W^{(t)}}$), およびZero Point($z^{W^{(t)}}$)は以下の式で表される．

$$s^{W^{(t)}} = \frac{2^q - 1}{\alpha - \beta} \tag{2.7}$$

$$z^{W^{(t)}} = \mathrm{round}\left(\alpha - \frac{2^{\mathrm{q}} - 1}{\mathrm{s_t}}\right) \tag{2.8}$$

この時，Scale Factor は入力値の範囲と量子化後の整数の範囲の比率を表し，Zero Point は浮動小数点数の0が量子化後の整数でどの値に対応するかを表す．

例えば，符号付き8bit整数への量子化を行う時，w_q の値の範囲は $w_q \in [-128, 127]$ となる．

これらの値を用いて，量子化操作 qutantize は

$$\boldsymbol{W_q^{(t)}} = \mathbf{quantize}(\boldsymbol{W^{(t)}}) = \mathbf{round}(\boldsymbol{s^{W^{(t)}}W^{(t)}} + \boldsymbol{z^{W^{(t)}}}) \tag{2.9}$$

量子化復元操作 dequtantize は

$$\hat{\boldsymbol{W}}^{\boldsymbol{(t)}} = \mathbf{dequantize}(\boldsymbol{W_q^{(t)}}) = \frac{\mathbf{1}}{\boldsymbol{s^{W^{(t)}}}}(\boldsymbol{W_q^{(t)}} - \boldsymbol{z^{W^{(t)}}}) \tag{2.10}$$

とそれぞれ行う．なお，この際 $\hat{W^{(t)}}$ は丸め処理の影響で $W^{(t)}$ の近似値となる．

続いて，量子化を適用した際の値の伝搬について考える．

中間層 $t-1$ 層目の出力のベクトル $\boldsymbol{H}^{(t-1)}$，量子化前の重みパラメータの行列 $\boldsymbol{W}^{(t)}$，中間層第 t 層目の重み付き和のベクトル $\boldsymbol{U}^{(t)}$ を

$$\boldsymbol{U}^{(t)} = \boldsymbol{H^{(t-1)}W}^{(t)} \tag{2.11}$$

と表し，量子化操作 qutantize を適用すると

$$\boldsymbol{U_q^{(t)}} \simeq \mathbf{quantize}(\boldsymbol{H^{(t-1)}})\mathbf{quantize}(\boldsymbol{W^{(t)}}) \tag{2.12}$$

が成り立つ．

また，中間層 t 層目の出力ベクトル $\boldsymbol{H}^{(t)}$ は活性化関数 ReLU による $\boldsymbol{U}^{(t)}$ の活性値であり，

$$\boldsymbol{H}_q^{(t)} = \mathrm{quantize}(\mathrm{ReLU}(\boldsymbol{U}_{\mathrm{q}}^{(\mathrm{t})})) \tag{2.13}$$

から，量子化された活性値のベクトル $\boldsymbol{H}_q^{(t)}$ が得られる．なお，活性化の量子化は

$$n = \log_2(s^{U_q^{(t)}}) - 1 \tag{2.14}$$

を満たす n によって，算術右シフト演算により近似することで高速に計算できる．

$$\hat{\boldsymbol{H}_q^{(t)}} \simeq \mathrm{ReLU}(\boldsymbol{U}_{\mathrm{q}}^{(\mathrm{t})}) >> \mathrm{n} \tag{2.15}$$

2.2.3 MNIST

MNIST データセットとは，Yann Lecun らによって作成された，手書き数字画像のクラス識別用のデータセットである [5].

グレースケールの 1 桁の手書き数字 (0 ～ 9) が撮影された，60,000 枚の学習用の画像と 10,000 枚の評価用の画像から構成される．画像は 1 枚あたり縦 28 ピクセル，横 28 の計 784 ピクセルで構成される． また，それぞれの画像には，0 ～ 9 の識別用のカテゴリラベルが設定されている．

2.3 DNN モデルの保護手法

2.3.1 AES Key Scheduling[6]

Advanced Encryption Standard(AES) 暗号化アルゴリズムとは，アメリカ国立標準技術研究所によって標準暗号として定められた共通鍵暗号アルゴリズムである．暗号鍵長は 128bit・256bit・512bit の 3 種類から選択する．

AES Key Scheduling Algorithm[6] は，AES 暗号化アルゴリズムの中核的な処理で，単一の暗号鍵をマスターキーとして，マスターキーを暗号化・復号化処理で使用される複数のラウンドキーに展開する手法である．この手法により，暗号化の各ラウンドでマスターキーから派生した異なるキーを使用することが保証され，追加の複雑性と非線形性を導入することで暗号化処理の安全性を向上させる．後述する既存研究 NN-Lock アルゴリズムでも，AES のうち，AES Key Scheduling Algorithm を使用して DNN モデルの重みパラメータの暗号化を行う．図 2.4 へ Key Scheduling の概要図を，アルゴリズム 1 に Key Scheduling のアルゴリズムを示す [7].

AES Key Scheduling Algorithm の操作は，マスターキー K を入力として受け取り，ラウンドキー $\{L^0, L^1, ..., L^{n-1}\}$ を出力する． NN-Lock アルゴリズムでは

AES-128 を使用するため，$n = 11$ となり，128bit (16byte) 長のマスターキー K を入力として，128bit (16byte) 長 11 個のラウンドキー $\{L_0, L_1, ..., L_10\}$ を出力する．

これより AES-128 における Key Scheduling によるラウンドキーの生成方法を説明する．この図解では，128bit (16byte) 長のラウンドキーの値 $L_i(i = 0 \cdots 10)$ を 4Word(32byte) に分割し，各要素を $l^i_j(j = 0 \cdots 3)$ とする．また，l^i_j を 8bit(1byte) に分割し，$l^i_{j,k}(k = 0 \cdots 3)$ として表す．

初めに，マスターキー K をラウンドキー L_0 とした上で，4Word(32byte) に分割し，$L^0_j(j = 0 \cdots 3)$ とする．

生成過程は，反復インデックス i の値によって以下の通りに異なる．

$$L^i = \begin{cases} K & \text{if } i < 1 \\ L^{i-1}_0 \oplus SubWord(RotWord(L^{i-1}_3)) \oplus rcon_i & \text{if } i \geq 1 \, and \, j = 0 \\ L^{i-1}_j \oplus L^i_{j-1} & \text{Otherwise} \end{cases} \tag{2.16}$$

ここで if $i \geq 1 \, and \, j = 0$ の条件で使用する関数と定数について説明する．

関数 RotWord と SubWord は以下の通りに定義される．

$$\text{RotWord}\left([L^{i-1}_{3,0} \quad L^{i-1}_{3,1} \quad L^{i-1}_{3,2} \quad L^{i-1}_{3,3}]\right) = [L^{i-1}_{3,1} \quad L^{i-1}_{3,2} \quad L^{i-1}_{3,3} \quad L^{i-1}_{3,0}] \tag{2.17}$$

$$\text{SubWord}\left([L^{i-1}_{3,1} \quad L^{i-1}_{3,2} \quad L^{i-1}_{3,3} \quad L^{i-1}_{3,0}]\right) = [SBox(L^{i-1}_{3,1}) \quad SBox(L^{i-1}_{3,2}) \quad SBox(L^{i-1}_{3,3}) \quad SBox(L^{i-1}_{3,0})] \tag{2.18}$$

また，ラウンド定数 $Rcon[0...9]$ 以下の通りに定義される．

$$rcon_i = [rc_i \quad 00_{16} \quad 00_{16} \quad 00_{16}] \tag{2.19}$$

ここで，rc_i は表 2.1 と式 2.20 から導出される．

表 2.1: ラウンド定数 $rcon$ とラウンド i の対応表

i	1	2	3	4	5	6	7	8	9	0
rc_i	01	02	04	08	10	20	40	80	1B	36

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \; and \; rc_{i-1} < 80_{16} \\ 2 \cdot rc_{i-1} \oplus 11B_{16} & \text{if } i > 1 \; and \; rc_{i-1} \geq 80_{16} \end{cases} \tag{2.20}$$

S-Box は 8-bit の入力を別の 8-bit の出力に置き換える変換テーブルであり，平文と暗号文の相関 (線形性) を破壊するために用いられる．S-Box の内容を表 2.2 に示す．

S-Box 変換は $Sbox[x] = y$ として定義され，y は

$$y = b + (b \lll 1) + (b \lll 2) + (b \lll 3) + (b \lll 4) + 63 \tag{2.21}$$

から導出できる．ここで，b は $GF(2^8)$ における x の逆元で，$\lll$ は左循環シフトを表す．

このアルゴリズムにより，マスターキーから暗号化対象のパラメータに対して一意の鍵が導出される．また，マスターキーのわずかな変更に対しても鍵に大きな変化をもたらし，復号化のための可逆性も維持される．

Algorithm 1 AES Key Schedule Operation

Require: 128-bits(16-bytes) Master key K
Ensure: Round keys $\{L^0, L^1, ..., L^{n-1}\}$ for n parameters
$L^0 \leftarrow K$ ▷ 1st Round Key = Master Key
$Rcon[0...9] \leftarrow \{01, 02, 04, 08, 10, 20, 40, 80, 1B, 36\}$ ▷ in hex ▷ Rcon are round constants in $GF(2^8)$
Sbox[x] ← y where $y = b + (b \lll 1) + (b \lll 2) + (b \lll 3) + (b \lll 4) + 63$
▷ b is inverse of x in $GF(2^8)$, $\lll$ is left circular shift
for $i \leftarrow 1$ to 10 **do**
    for $j \leftarrow 0$ to 3 **do**
        if $i \geq 1$ AND $j = 0$ **then**
            $temp \leftarrow$ SubWord(RotWord($\mathrm{L}_3^{\mathrm{i}-1}$)) $\oplus$ Rcon[i]
            ▷ SubWord applies Sbox to each byte
            ▷ RotWord performs one-byte circular left shift
        else
            $temp \leftarrow L^i_{j-1}$
        end if
        $L^i \leftarrow L^{i-1} \oplus temp$
    end for
end for

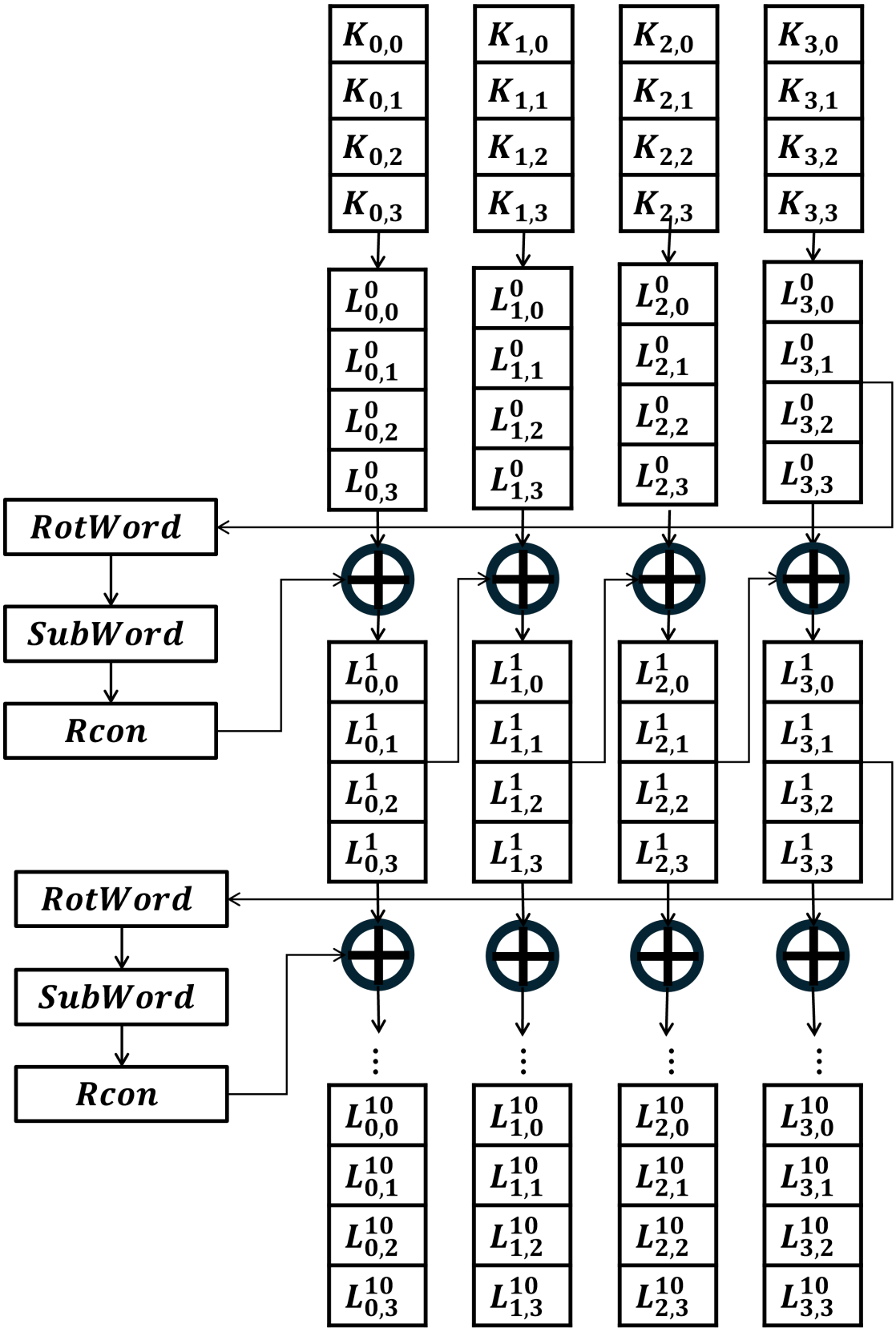

図 2.4: AES-128 における AES Key Scheduling Algorithm[7]

		入力: 下位 4-bit															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
入力: 上位 4-bit	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

表 2.2: Rijndael AES Key Scheduling における S-Box [7]

		入力: 下位 4-bit															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
入力: 上位 4-bit	**0**	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

表 2.3: Rijndael AES Key Scheduling における Inverse S-Box [7]

2.3.2 NN-Lock[1]

NN-Lock は，DNN モデル内の重みパラメータを，先述の AES Key Schedule を使用して暗号化・復号化するアルゴリズムである．

暗号化は，n 個の int8 量子化された重みパラメータ $\{w_0, w_1, ..., w_{n-1}\}$ と暗号鍵 K を入力として行う．アルゴリズム 2 に暗号化処理のアルゴリズムを示す．前述の AES Key Scheduling Algorithm を使用して， K をマスターキーとしてラウンドキーのセット $\{L^0, L^1, \ldots, L^{10}\}$ を生成する．

各パラメータ $w_m(m = 3i + 3j + k)$ に対して，まずパラメータをバイナリ表現に変換し，対応するラウンドキー $L^i_{j,k}(i = m \pmod{11})$ との XOR 演算を実行する．続いて以下の操作によって S-box 置換を行う．

$$w'_m = Sbox[w_m \oplus L^i_{j,k}] \tag{2.22}$$

最後に，暗号化された結果を 8bit 整数値 w'_i に変換し，モデル内の元のパラメータと置き換える．

復号化は，Inverse S-box 演算 InvSbox を使用して暗号化操作を逆順に実行する．アルゴリズム 3 に復号化処理のアルゴリズムを示す．

Inverse S-box は S-Box と同様に 8-bit の入力を別の 8-bit の出力に置き換える変換テーブルであり，構造は表 2.3 の通りである．暗号化された各パラメータ w'_m に対して，まず $InvSbox$ を適用し，続いてマスターキーから導出されたラウンドキー $L^i_{j,k}$ との XOR を行う．

$$w^d_m = InvSbox[w'_m] \oplus L^i_{j,k} \tag{2.23}$$

このアルゴリズムにより，正しい暗号鍵 (Correct Key) K を使用した場合，$w^d_m = w_m$ となり，暗号化されていた DNN モデル内の重みパラメータが元の重みパラメータの値に復号化され，推論精度は元の DNN モデル (Original) と同一になり，正しく推論を行える．しかし，誤った暗号化鍵 (Wrong Key) を使用した場合，$w^d_m \neq w_m$ となり，重みパラメータが不正確に復号化され，結果として推論精度が大幅に低減し，正しく推論処理を行えなくなる．

Algorithm 2 NN-Lock Encryption

Require: Set of n int8 quantized trained DNN weight parameters: $\{w_0, w_1, \ldots, w_{n-1}\}$; A master key K; A Key_Schedule algorithm; An Sbox mapping

Ensure: Locked DNN parameters: $\{w'_0, w'_1, \ldots, w'_{n-1}\}$

$\{L^0, L^1, \ldots, L^{10}\} = Key\_Schedule(K)$
for $i \leftarrow 0$ to $\lfloor n/(11 \cdot 16) \rfloor$ **do**
  for $j \leftarrow 0$ to 3 **do**
    for $k \leftarrow 0$ to 3 **do**
      $m \leftarrow 3i + 3j + k$
      $w'_m =$ converted int8 value of $Sbox[w_m \oplus L_{j,k}^{i \pmod{11}}]$
    end for
  end for
end for

Algorithm 3 NN-Lock Decryption

Require: Set of n int8 quantized encrypted DNN weight parameters: $\{w'_0, w'_1, \ldots, w'_{n-1}\}$; A master key K; A Key_Schedule algorithm; An $InvSbox$ mapping

Ensure: Unlocked DNN parameters: $\{w^d_0, w^d_1, \ldots, w^d_{n-1}\}$

$\{L^0, L^1, \ldots, L^{10}\} = Key\_Schedule(K)$
for $i \leftarrow 0$ to $\lfloor n/(11 \cdot 16) \rfloor$ **do**
  for $j \leftarrow 0$ to 3 **do**
    for $k \leftarrow 0$ to 3 **do**
      $m \leftarrow 3i + 3j + k$
      $w^d_m = InvSbox[(w_m)'] \oplus L_{j,k}^{i \pmod{11}}$
    end for
  end for
end for

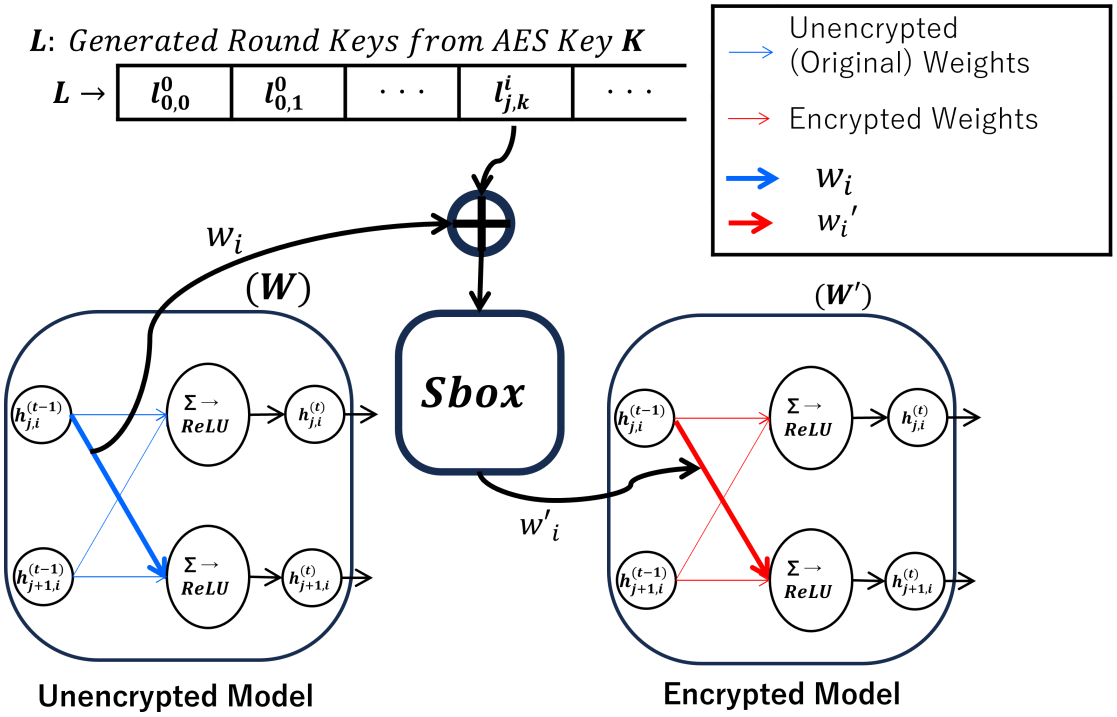

図 2.5: NN-Lock 暗号化処理

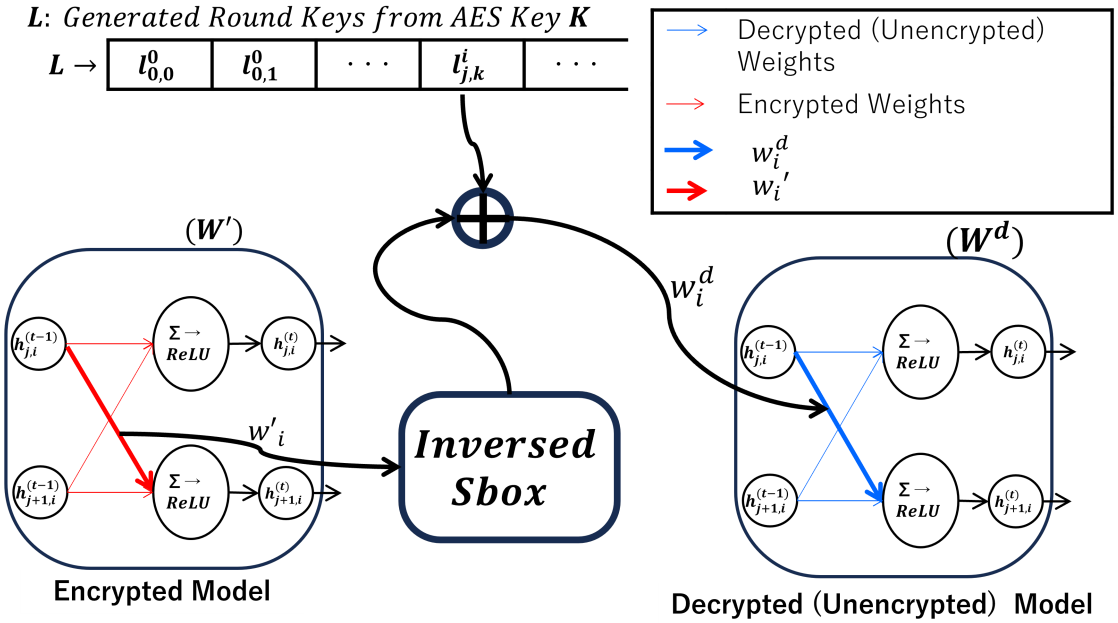

図 2.6: NN-Lock 復号化処理

2.4 先行手法 NN-Lock 再現実験

図 2.7 に，NN-Lock 論文中の評価結果と，筆者の再現実装の評価結果 (小俣実装) の比較を，表 2.4 に筆者の再現実験環境を示す [8].

両者とも，正しい暗号鍵 (Correct Key) K を使用した場合，推論精度は元の DNN モデル (Original) と同一になり，正しく推論を行えることが分かる．一方で，誤ったマスターキー (Wrong Key) を使用した場合，重みパラメータが不正確に復号化され，結果として推論精度が大幅に低減し，正しく推論処理を行えていないなることが分かる．

FPGA ボード	AMD Kria KR260
CPU	AMD Zynq UltraScale+ EV (Quad Arm Cortex-A53 + Dual Arm Cortex-R5F)
フレームワーク	Pytorch 2.1.0 [10]
使用モデル	MNIST 全結合層 2 層 int8 量子化モデル

表 2.4: NN-Lock 再現実験環境

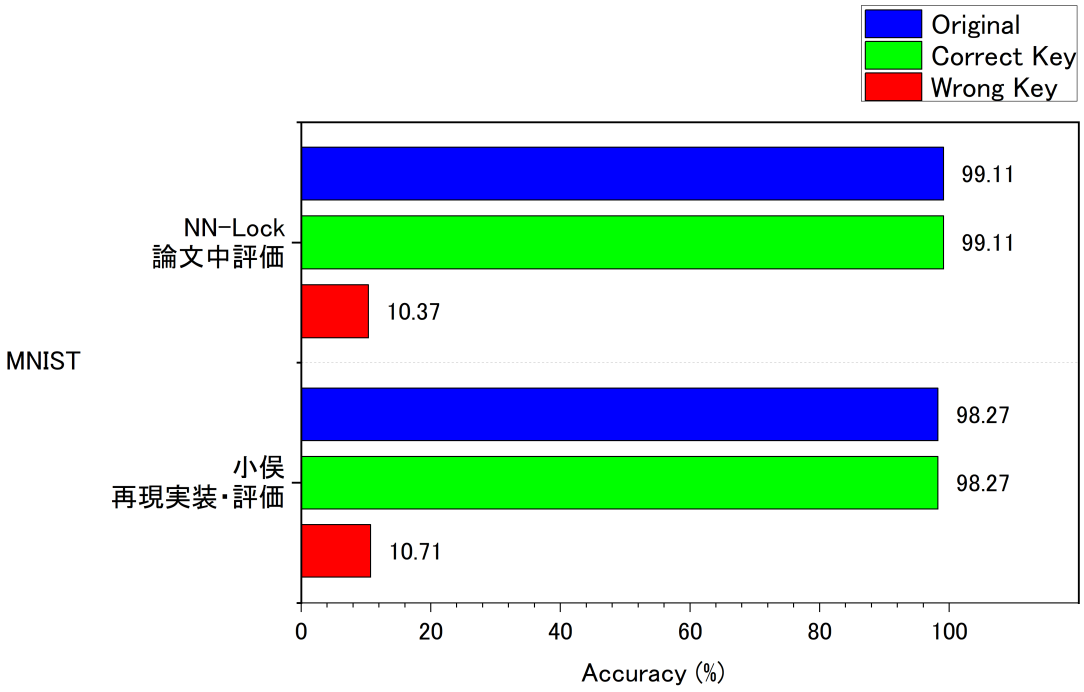

図 2.7: NN-Lock 論文と再現実装の推論精度の比較
(論文中の値は [1] から引用)

2.5 FPGA(Field Programmable Gate Array)

FPGA とは，プログラムにより内部構成を変更可能な論理ブロックを格子状に接続し，集積した構造を持つデバイスである．論理ブロックの機能と入出力は，SRAM で構成された Look Up Table によって管理される．論理ブロック間の配線は，配線路に設けられたスイッチマトリクスによって有効/無効を決定され，その内容は SRAM で構成されたトランスファーゲートによって管理される．

2.5.1 SoC-FPGA

FPGA-SoC は，プログラマブルロジック（PL）部に FPGA を，プロセッシングシステム（PS）部に SoC(System on Chip) を搭載し，1 つのチップに統合したデバイスである．

PS 側でオペレーティングシステムと PL 側の回路制御を実行し，PS 部と PL 部間の通信は AXI インターフェースを介して行うことで，オペレーティングシステムによる柔軟なソフトウェアと FPGA ハードウェアの協調動作を可能にする．

2.5.2 BRAM(Block Random Access Memory)

今回の評価で使用する FPGA 内には，専用回路として，BRAM と呼ばれる 36Kb の SRAM メモリが 144 個搭載されている．BRAM は，DRAM(Direct Random Access Memory) 程大規模なメモリ空間を確保することはできない一方，レイテンシが小さく高速にアクセス可能である．また，複数のメモリを用意し，ロジック毎に接続して使用したり，或いは複数のメモリを束ねて大容量のメモリやデータ幅の広いメモリとして利用するなどの柔軟な使い方が可能である．この時，それぞれのメモリアクセスは並行して動作する．そのため，FPGA 上で頻繁に使用するデータを，BRAM のようなメモリバンド幅が大きく，遅延の小さなローカルなメモリに格納することで，高速な処理が期待できる．

2.6 まとめ

本章では本研究に関連する先行研究と，その背景情報について論じた．

2.2 節では，画像分類モデルの基本的な構造と，画像の推論の仕組みを説明した．また，モデル内の重みパラメータを低精度値に変換することでモデルのサイズを削減する手法である量子化について説明した．2.3 節では NN-Lock アルゴリズムと，その中心となる AES Key Scheduling Algorithm を紹介した．2.4 章で実際にソフトウェアでの再現実験を行うことで，NN-Lock による暗号化時の DNN モデルの推論精度が論文の評価結果と同様になることを示した．2.5 節では FPGA の

アーキテクチャについて説明した．FPGA は BRAM と呼ばれるメモリを有しており，複数の単位に分割して並列にデータの入出力を行うことが可能となる．

次章ではこのアーキテクチャを基に NN-Lock と DNN モデルをどのように FPGA 上に構築するかという，具体的な提案手法の説明を行う．

第3章　NN-LockによるFPGA上での重みパラメータ復号化のタイミングの検討

3.1　はじめに

本章では，DNNモデルを保護するアルゴリズムであるNN-Lockについて，DNNモデルと共にFPGA上に数通りの適用箇所を提案・実装し，DNNモデルの推論回路のうち最適な適用箇所を考察する．これにより，モデルの推論実行を行う際，どのタイミングでNN-Lockアルゴリズムによる復号化を行うかの選択肢を列挙し，セキュリティ的な観点から考察を行う．また，並列数を変えることによる回路規模や消費リソース等から評価を行う．

3.1.1　モデルの構成

データセットは，0~9の手書き数字画像のクラス識別用データセットであるMNISTを使用し，DNNは中間層を全結合層計2層で構成した．内訳として，第1層の重みパラメータは$784 \times 512 = 401,408$，第2層の重みパラメータは$512 \times 10 = 5,120$である．

3.2　NN-Lock 回路の実装

NN-Lockの適用箇所として，第1に，BRAMに重みパラメータを書込む時点での復号化 (BRAM書込前NN-Lock), 第2に，DNN推論処理を行う直前での重みパラメータの復号化 (DNN実行前NN-Lock) を実施する．また，両者とも並列数 1, 2, 4, 8, 16の5通りの並列実装を提案する．

3.2.1　実装対象のFPGAの構成

適用箇所の概要図を3.1に示す．実装を行うAMD Kria KR260[8]では，SoCやDRAMから構成され，OSが動作するProcessing System(PS)とFPGAの論

理回路領域である Programmable Logic(PL) を搭載し，Processing System から Programmable Logic を操作する．具体的には，PS 側の OS 上で動作させるプログラムによって，SoC を中心に，DRAM と PL 間で AXI Interconnect[9] を経由してデータを転送する．この際，事前に AXI Interconnect の配線を PL 上に構成する必要がある．

3.2.2 DNN 回路の動作

DNN 回路の動作の流れを説明する．PS 側からプログラムによって，事前に PL 上の BRAM に重みパラメータを転送・配置する．続いて，推論実行毎に，推論を行う画像を PL 上のレジスタに転送し，転送が完了次第 PL 上に実装した DNN 回路で推論を行う．なお，DNN 回路の詳細な実装手法は 4 章で説明する．

3.2.3 適用方法の検討

続いて，DNN 回路周辺に NN-Lock 回路を適用する箇所について検討する．

NN-Lock 自体の処理については 図 4.3 の処理概要図の通りである．入力値として，復号化対象の重みパラメータ (encrypted_params) の準備ができ次第，top モジュールから nn_rst = 1 が入力される．ここで，ラウンドキーの生成が完了していない (keygen_init = 0) 場合，状態が INIT に遷移し，入力された AES 暗号鍵 (aes_key) から AES Key Scheduling アルゴリズム (アルゴリズム 1) を基にラウンドキー round_keys を生成する．ラウンドキーの生成完了 (keygen_init = 1) 以降は状態が DECRYPT に遷移し，NN-Lock 復号化アルゴリズム (アルゴリズム 3) を基に重みパラメータを復号していく．

続いて上記 NN-Lock の適用箇所を考える．適用方法 1 として，BRAM への書込時点で重みパラメータの復号化 (図 3.1 中 ①) を，適用方法 2 として，DNN 直前で重みパラメータの復号化 (図 3.1 中 ②) を行う．

ここで，適用方法 1,2 いずれにおいても重みパラメータを NN-Lock (アルゴリズム 2) で事前に暗号化したモデルを使用するため，提案手法では復元処理のみ実装する．

適用方法 1,2 での実装手法の差異と各々の利点・欠点については，以下に示す．

適用方法 1: BRAM への書込時点での重みパラメータの復号化 (BRAM 書込前 NN-Lock)

この適用方法では，Processing System 側の DRAM から AXI バス経由で，NN-Lock によって暗号化された重みパラメータを FPGA 上の BRAM に転送する際，BRAM に格納される直前で NN-Lock 復号化処理を適用する．復号化された重み

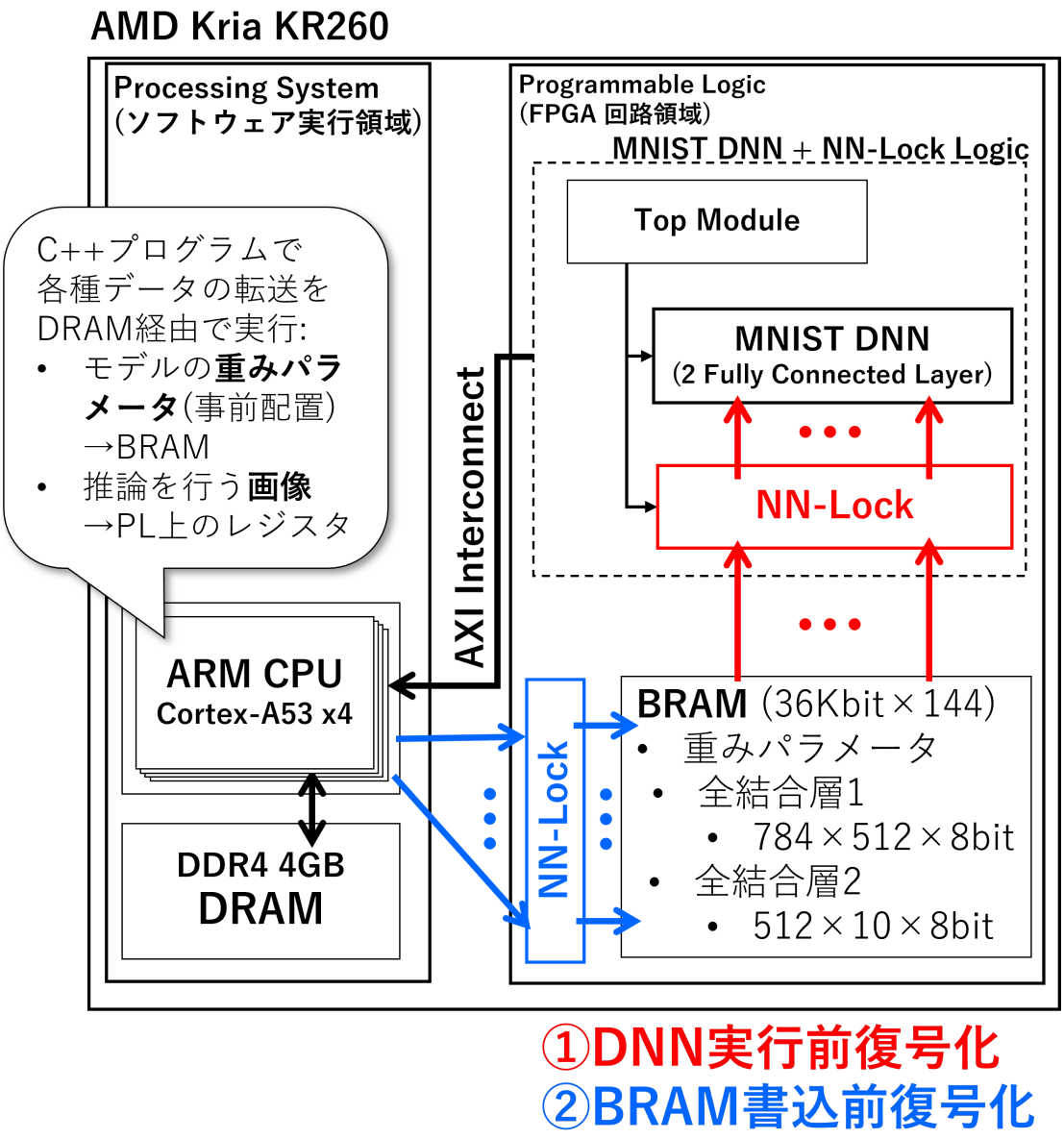

図 3.1: 提案手法 (NN-Lock を適用した DNN モデルの FPGA への実装) の構成 概要図

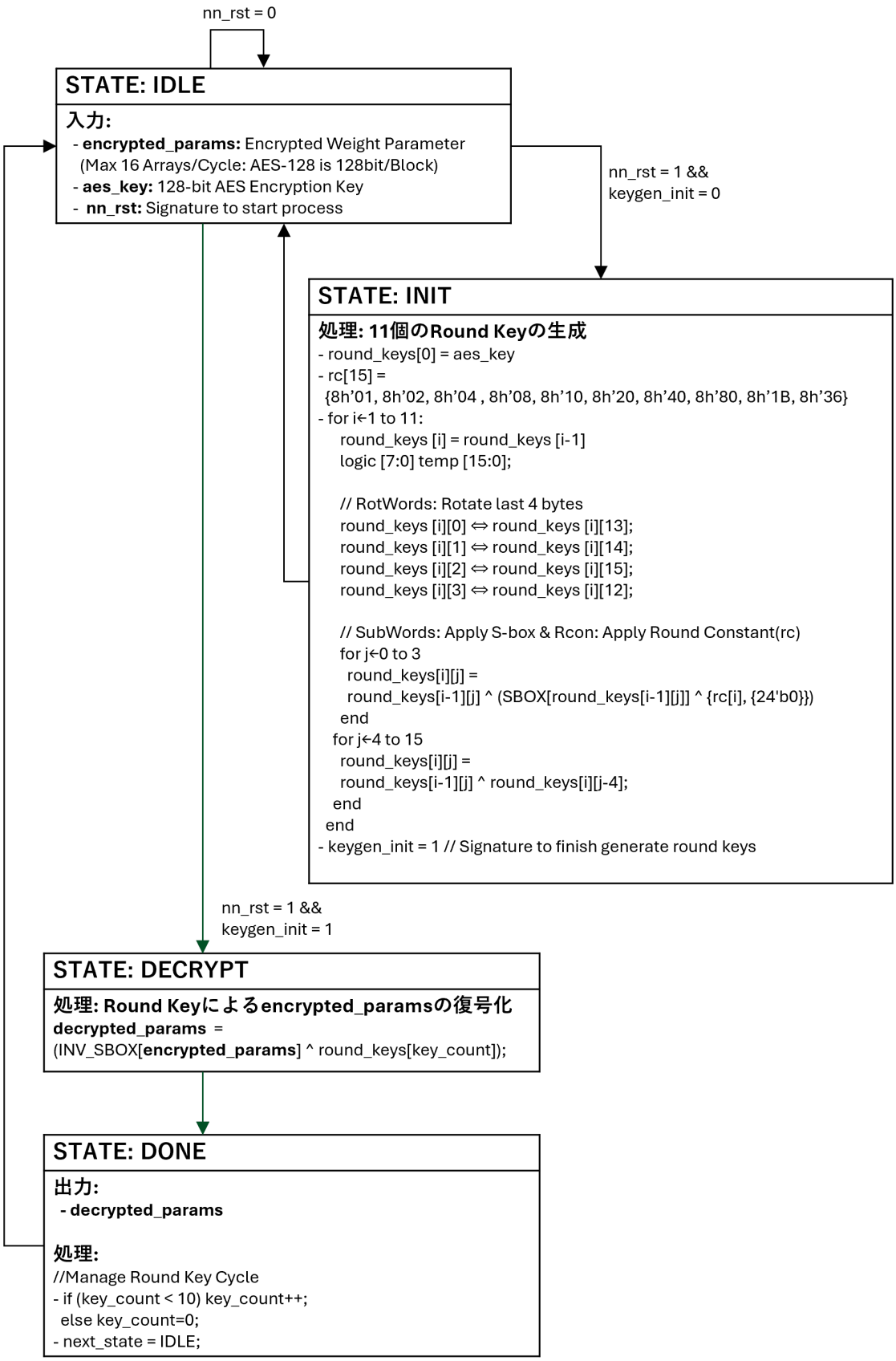

図 3.2: NN-Lock FPGA 実装 処理概要図

パラメータはBRAM上で保持し続け，そのまま入力画像データを受け取り次第推論処理を実行する．

上記のタイミングでNN-Lock 復号化処理を適用する利点として，第一にProcessing System上と比較して外部攻撃手段が限られるFPGA側で重みパラメータを復号し，保持することで保護手法としての頑健性を高める．第二に，推論処理毎に重みパラメータの復号化処理を行う必要がないため，処理速度，回路規模，消費電力を小さくできることが予測される．この2点を両立可能な点において，適用方法2と比較して優れると考えられる．

欠点として，PS側の制御権限が奪取された場合，Bitstreamの上書きや改ざんによるPL側回路の再構成によって，BRAM内の暗号化された重みパラメータへのアクセスも可能となり，本方法は脆弱となる．

適用方法 2: DNN 直前での重みパラメータの復号化 (DNN 実行前 NN-Lock)

DNN回路の推論処理では，この計算の際，BRAMから重みパラメータを読み出し，入力画像パラメータとの積和演算を行う．この際，重みパラメータはNN-Lockで暗号化されているため，BRAMから重みパラメータを読み出し，レジスタに一時的に格納するタイミングでNN-Lock 復号化処理を適用する．復号された重みパラメータは，積和演算の終了後，直ちに破棄する．

この適用方法の利点は，第一に適用方法1に比べて，上記の推論処理の積和演算が開始・終了する一時的なタイミングでのみ，暗号化された重みパラメータを部分的にのみ復号化するため，保護手法としての頑健性に優れる．

欠点として，BRAMから重みパラメータを読み出し積和演算を実行する度に復号化処理を繰り返すため，適用方法1に比べて処理速度，回路規模，消費電力の増大が予測される．

3.3 まとめ

FPGAでのDNNモデルの動作と，NN-Lockによるモデルの保護手法の提案を行った．保護手法の適用箇所は”BRAM書込前NN-Lock”と”DNN実行前NN-Lock”の2種類を提案し，両者ともに並列数 1, 2, 4, 8, 16の5通りの並列実装を行う．これにより，NN-Lockの適用箇所によって各種リソースの消費量が変動し，複数の想定される利用場面に応じて適した適用手法が存在することが期待される．

本章で提案した適用方法と並列数による回路規模と処理時間は第5章に掲載し，FPGAボードや運転支援システムの要件と比較しながら評価する．

第4章　NN-Lockに適したFPGA上でのDNNの検討・実装

4.1　はじめに

本章では，3 章で検討した NN-Lock に適した DNN の演算回路 単体の実装手法を提案する．続いて，DNN の演算回路のみを FPGA 上に実装し，回路規模と推論処理時間を評価する．

4.2　評価環境

モデルの学習と，先行研究のソフトウェア · ハードウェアそれぞれによる提案手法の実装には以下に示す表 4.1 の環境を使用した．

また，今回 評価で使用した FPGA ”AMD Kria KR260”[8] では，クロックは Processing System 側の SoC から供給される．そしてクロックは，図 4.1 に示すように 99.999901MHz の値が自動的に割り当てられていたため，その値を使用する．

なお，学習に使用する環境の構成は，学習して生成したモデルのみをデータとして使用するため，先行研究のソフトウェアによる再現実験と提案手法の評価には影響しない．

4.3　実装手法 全体の構成

MNIST 全結合層 2 層 DNN モデルを FPGA に実装する．DNN 実装の際の FPGA 概要図を 4.2 に示す．構成は図 3.1 から NN-Lock 回路を外したものとなり，重みパラメータは BRAM から DNN 回路に直接読み出す．

続いて，DNN の処理概要図を 図 4.3 に示す．今回の DNN モデルは全結合層のみから構成されているため，ベクトル行列積の積和演算のみを行う回路となる．入力として，in_vec に 8bit 784 要素の入力画像パラメータ，または 512 要素の第 1 層の出力値を，in_part_mat に並列数に応じて最大 16 個の重みパラメータをとる．最大の個数が 16 個である理由は後述する．

入力値の準備ができ次第，top モジュールから fc_rst = 1 が入力され，状態が RESET から INNER_PRODUCT に遷移する．INNER_PRODUCT では 2 つの入

FPGA ボード	AMD Kria KR260
▼ Processing System 側 DNN モデル学習 · 提案手法実装環境	
CPU	AMD Zynq UltraScale+ EV (Quad Arm Cortex-A53 + Dual Arm Cortex-R5F)
DRAM	4,096 MB
使用モデル	MNIST 全結合層 2 層 int8 量子化モデル
▼ Programmable Logic 側 提案手法実装環境	
LUT	256,000
BRAM	36Kb × 144
使用モデル	MNIST 全結合層 2 層 int8 量子化モデル

表 4.1: NN-Lock・DNN モデル 実装環境 [8]

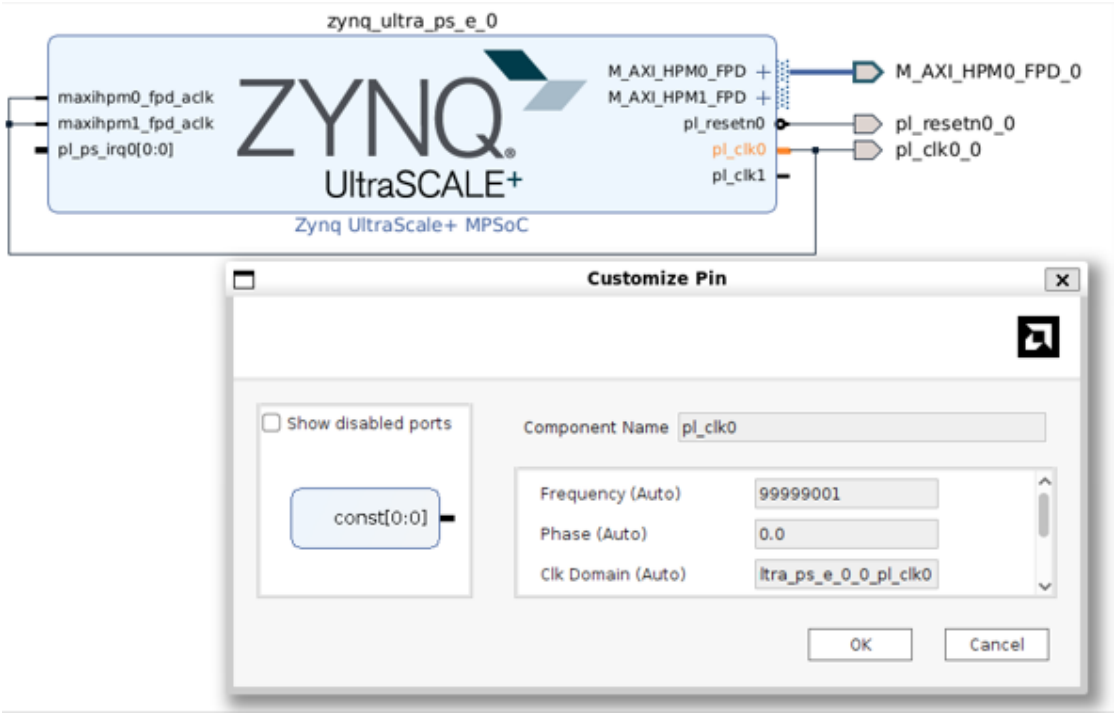

図 4.1: NN-Lock・DNN モデル実装環境 FPGA 周波数

力のベクトル積 prod_out を計算し，状態 ACCUMULATE では prod_out の総和を算出して出力 acc_result とする．その後，状態は再び RESET に戻る．

ここでベクトル積の総和は 4 段パイプラインの加算で算出するため，DSP を使用しない．

重みパラメータは Block Memory に格納する．今回の実験で使用する FPGA ボード AMD Kria KR260 においては，BRAM の 1Block 単位のサイズは 36Kb であり，144 個搭載しているので，合計で 648KB のメモリを確保できる．今回は符号付 8bit 整数に量子化された重みパラメータのみ BRAM に搭載するので，$648 \times 1024 = 663,552$ エントリ確保できる．今回使用するモデルの重みパラメータの総数は，全結合中間第 1 層で $784 \times 512 = 401,408$, 全結合中間第 2 層で $512 \times 10 = 5,120$, 合計で $401,408 + 5,120 = 406,528$ であり，BRAM に十分収まる．一方入力画像データは，パラメータの総数が 784 と比較的小さいため，MNIST データセットの手書き数字画像データを符号なし 8bit 量子化したものを FPGA 上のレジスタに直接保持し，BRAM への書き込み・読み出しを不要とする．

4.3.1 量子化の手法

本研究の実装・評価で使用する DNN モデルは，簡単のため，全結合層 2 層の符号付 8bit 整数の重みパラメータと，Scale Factor S のみから構成される．

第 2 部 第 2 章の量子化手法の章で述べたように，中間パラメータの再量子化処理は，中間パラメータが符号付 32bit 整数のまま S 分のシフト演算によって 8bit 整数に近似できることを利用し，推論処理で行われる計算を全て整数値で実行する．

この手法で学習したモデルは，第 2 部 第 3 章の先行手法の再現結果グラフ 2.7 で示すように，十分な精度を維持できることを示している．

そのため，今回の提案手法ではこの手法を採用し，パラメータの量子化処理の FPGA 上への実装を行う．これにより，レイテンシの増加や回路量の増大の原因となる浮動小数点演算を回避し，整数演算のみで推論を行うことを実現する．

4.3.2 並列数に関する検討

続いて，DNN 回路の並列数について検討する．この並列数とは，一度に BRAM から重みパラメータを取込み，同時に演算処理を行う数である．

今回使用する DNN モデルは全結合演算のみを行うため，入力値の画像パラメータ，または中間値と重みパラメータの内積の計算機の集合で構成される．

しかし，直前に接続する NN-Lock 回路の処理内容は AES アルゴリズムであり，AES-128 を使用するため，1 個当り 8bit のサイズである重みパラメータをブロック毎に処理する場合，16 並列が限界となる．

そのため，今回は DNN 回路についても 1 並列から最大 16 並列までの 1，2，4，8，16 の 5 通りで実装を行った．

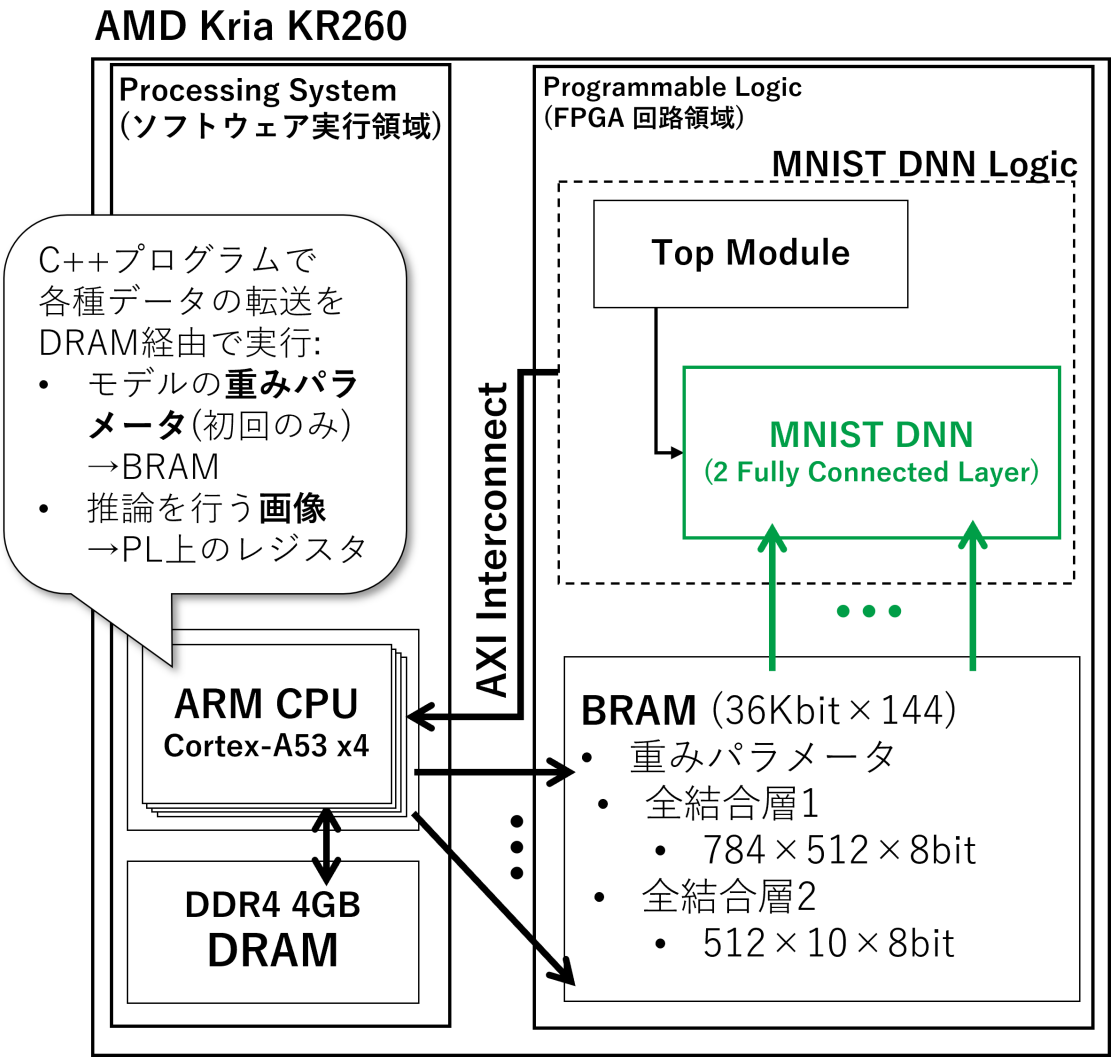

図 4.2: DNN モデル 単体の FPGA への実装 構成概要図

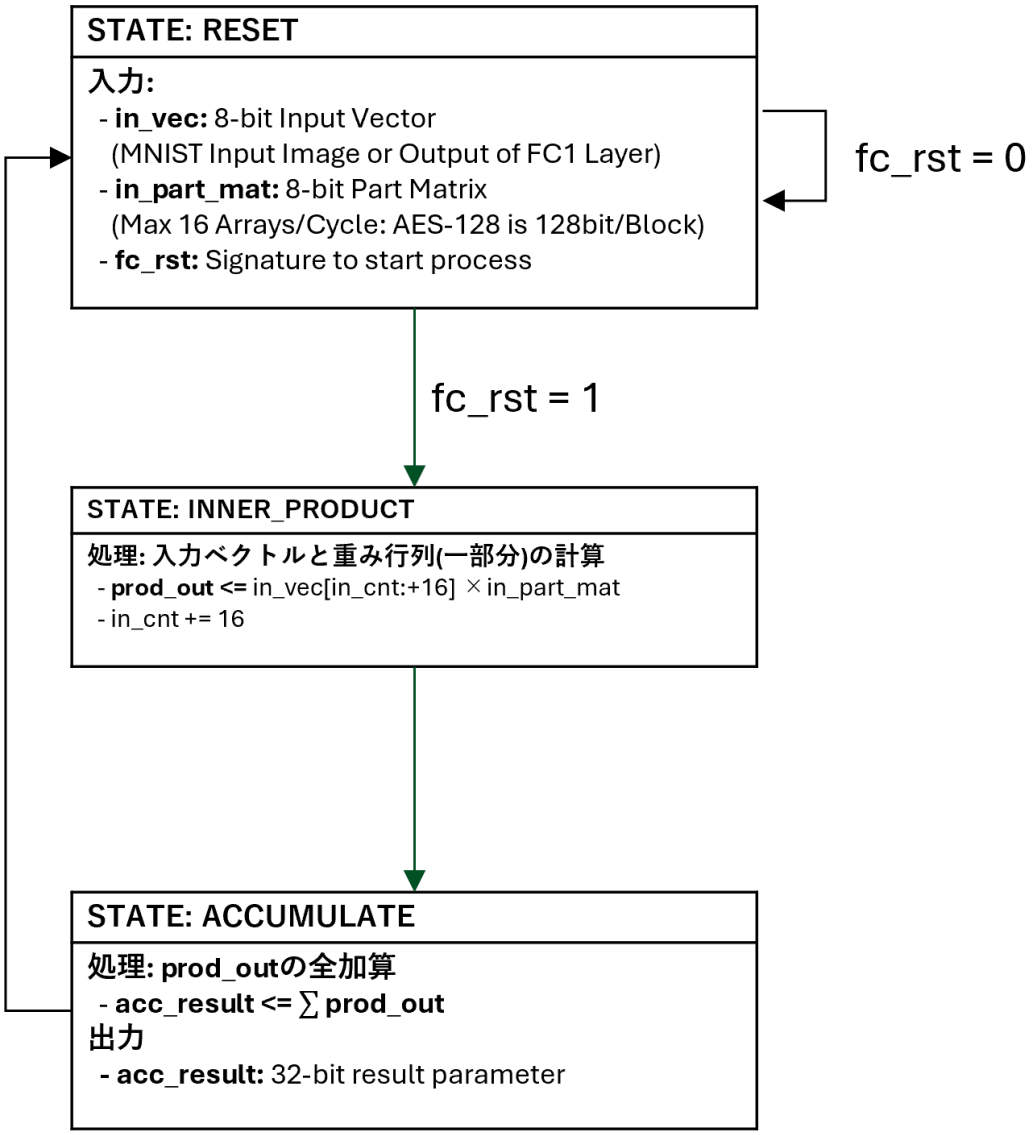

図 4.3: DNN 演算 FPGA 実装 処理概要図

4.3.3 DNN モデルの精度の評価

モデル内のパラメータは環境に依らず同一のため，先行研究のソフトウェアによる再現実験と FPGA 実装上の DNN モデルの評価では推論精度は一律である．表 4.4 に，実際に両者の精度が一致した結果を示す．

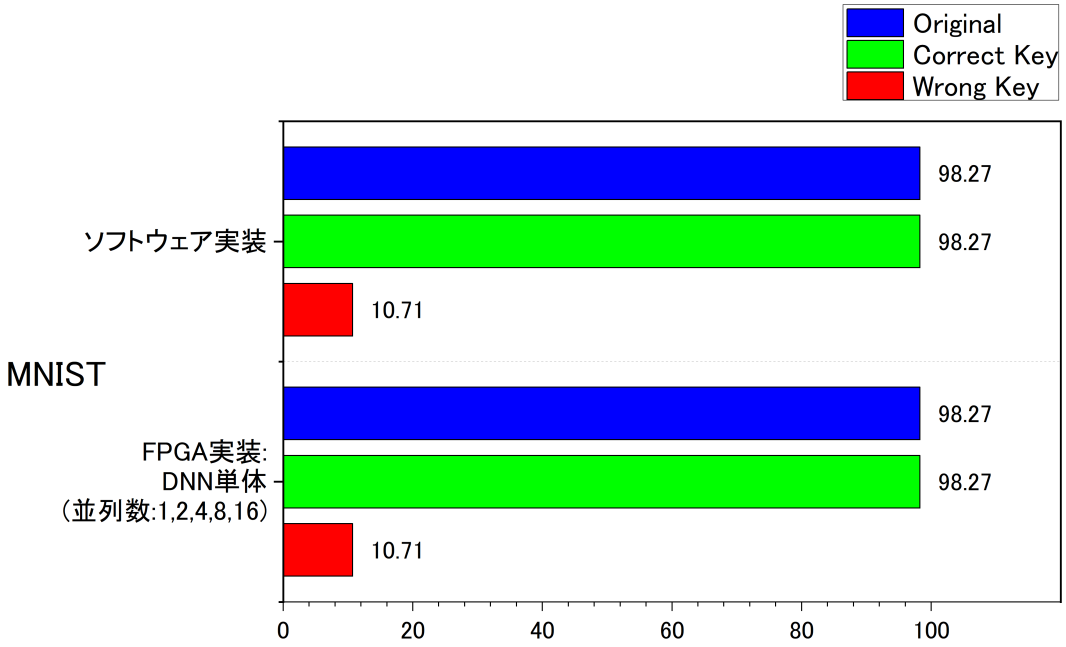

図 4.4: DNN モデルのソフトウェア実装と FPGA 実装の精度の比較

4.4 DNN モデル FPGA 単体実装 回路量・処理時間の評価

続いて，DNN モデルのみを複数の並列度で FPGA に単体で実装した際の評価を，第 3 章で提案した BRAM 書込前 NN-Lock 搭載回路と比較しながら行う．ここで，BRAM 書込前 NN-Lock 搭載回路は推論中 DNN 回路のみを実行するという点で DNN 単体回路と同じ条件を持つため，比較対象の両者が正しく実装できていれば，DNN モデル単体回路とほぼ同じ推論処理時間，回路量は NN-Lock 回路程度の差が生じるはずであり，評価を通して確認していく．

はじめに，回路規模として LUT 数を図 4.5 と表 4.2 に示す．”その他”の部分については，LUT を 1bit FF の RAM として使用していると考えられる．

BRAM 書込前 NN-Lock 搭載回路と比較して，ほぼ NN-Lock の回路分だけ回路量が減少していることがわかる．

次に，DNN の画像 1 枚当たりの処理時間を図 4.6 と表 4.3，4.4 に示す．BRAM 書込前 NN-Lock 搭載回路は事前に重みパラメータの復号化を行い，DNN 単体実装と同様，推論中には一切復号化操作を行わないため，処理時間がある程度同じになることがわかる．

この評価結果より，DNN 単体回路の FPGA 実装が十分に実装できていることが確認できる．

4.5 まとめ

本章では，NN-Lock を FPGA 上で動作させるにあたり，最適な DNN の演算回路実装手法を提案した．続いて，DNN の演算回路のみを FPGA 上に実装した上で，推論中の復号化を実行しない NN-Lock(BRAM 書込前復号化) 搭載回路と回路規模・推論処理時間を比較した．結果として回路量はおおよそ NN-Lock 回路増分の差があり，推論処理時間もおおよそ同一となっていることから，DNN 単体回路の FPGA 実装が十分に実装できていることを確認した．

第 3 章で提案した 2 つの NN-Lock 適用位置における，運転支援システムとしての運用を考慮した適性の評価については，次章 第 5 章で行う．

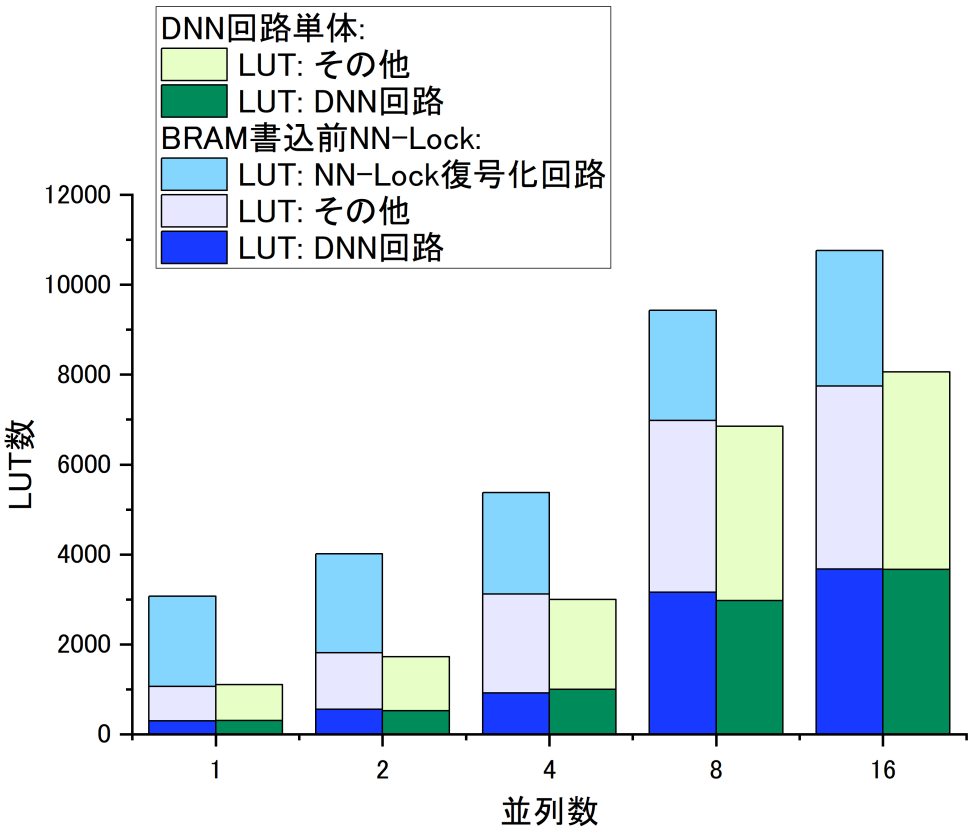

図 4.5: LUT 個数
(DNN FPGA 上単体実装/BRAM 書込前 NN-Lock 適用実装)

復号化箇所 / 並列数	LUT 数						
	DNN 単体			BRAM 書込前 NN-Lock			
	DNN 回路	その他	合計	DNN 回路	NNLock 復号化回路	その他	合計
1	306	799	1,105	294	2,013	766	3,073
2	523	667	1,190	552	2,201	1,261	4,014
4	996	2,001	2,997	921	2,253	2,198	5,372
8	2,972	3,875	6,847	3,156	2,449	3,827	9,432
16	3,669	4,388	8,057	3,676	3,014	4,067	10,757

表 4.2: DNN FPGA 上単体実装/BRAM 書込前 NN-Lock 適用実装
LUT 個数

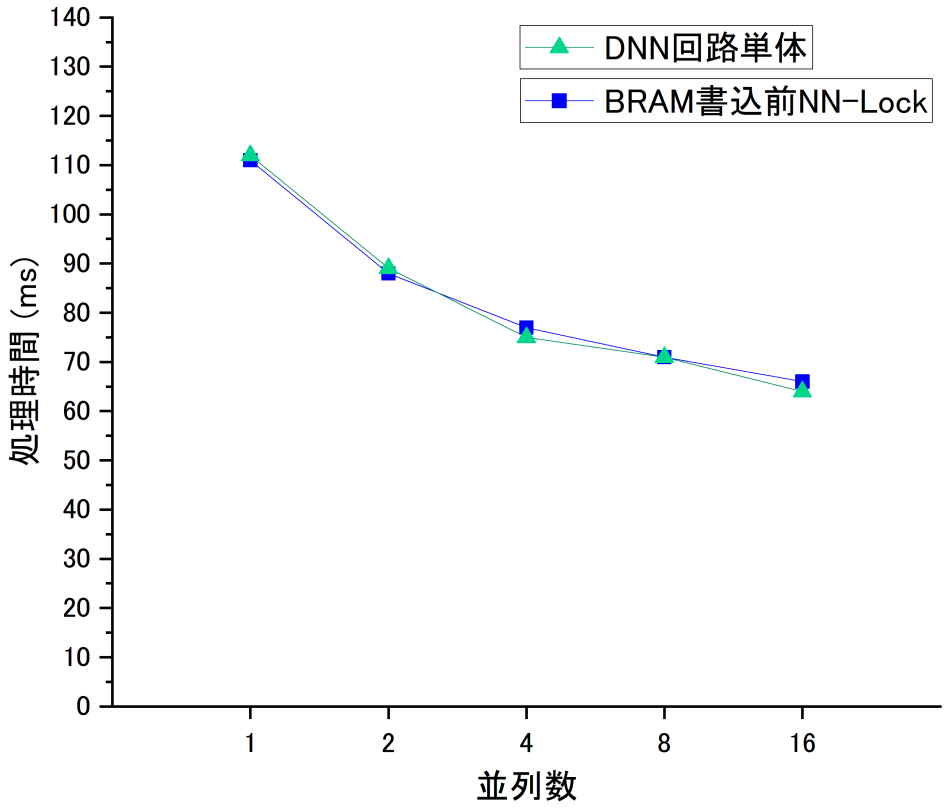

図 4.6: DNN FPGA 上単体実装/BRAM 書込前 NN-Lock 適用実装 推論処理時間の比較

並列数	推論時間 [ms]
1	112
2	89
4	75
8	71
16	64

表 4.3: 推論処理時間 (DNN FPGA 上単体実装)

並列数	推論時間 [ms]
1	111
2	88
4	77
8	71
16	66

表 4.4: 推論処理時間 (BRAM 書込前 NN-Lock 適用実装)

第5章　NN-Lockを適用したDNNの実験・考察

5.1　はじめに

この章では，回路規模と実存するFPGAボードでの動作要件，推論処理に要する時間と運転支援システム採用時の要件，NN-Lock適用箇所毎の重みパラメータの窃取耐性の3点について議論する．これにより，提案手法を運転支援システムとして採用する際の現状の適性と課題を考察する．

5.1.1　実験環境

実験環境は表5.1の環境を使用した．

5.2　回路規模の検討

はじめに，NN-Lock適用箇所・並列数毎のLUT使用数と，実際に現在流通しているFPGAの価格とLUT数を比較し，図5.1と表5.2に示す．

NN-Lock適用箇所毎でのLUTの差は，同じNN-Lockのモジュールを使用し，箇所のみ違うため，当然だがほとんど見られなかった．

続いて，現在流通しているFPGAボードのうち安価な順から5つ取り上げ，LUT数を比較し，表5.3に示す．小さい並列数の回路を採用することで，最も低価格帯のFPGAボードにもBRAM書込前NN-Lock/DNN実行前NN-Lock両方の適用実装によるDNNモデルの保護機構を提供することができる．

5.3　運転支援システム採用の想定と処理時間の検討

続いて，NN-LockのFPGA実装における処理時間について，提案手法の運転支援システムへの採用を想定しながら検討する．

Jonathan Horganらの論文[16]によれば，運転支援システムのビジョンアプリケーションにおいて，リアルタイム処理を実現するためには，カメラのキャプチャレートで全ての画像を処理する必要があると主張している．そこで，BRAM書込前

FPGA ボード	AMD Kria KR260
▼ PS 側 DNN モデル学習 · 提案手法実装環境	
CPU	AMD Zynq UltraScale+ EV (Quad Arm Cortex-A53 + Dual Arm Cortex-R5F)
DRAM	4,096 MB
使用モデル	MNIST 全結合層 2 層 int8 量子化モデル
▼ PL 側 提案手法実装環境	
LUT	256,000
BRAM	36Kb × 144
使用モデル	MNIST 全結合層 2 層 int8 量子化モデル

表 5.1: NN-Lock 実験環境 [8]

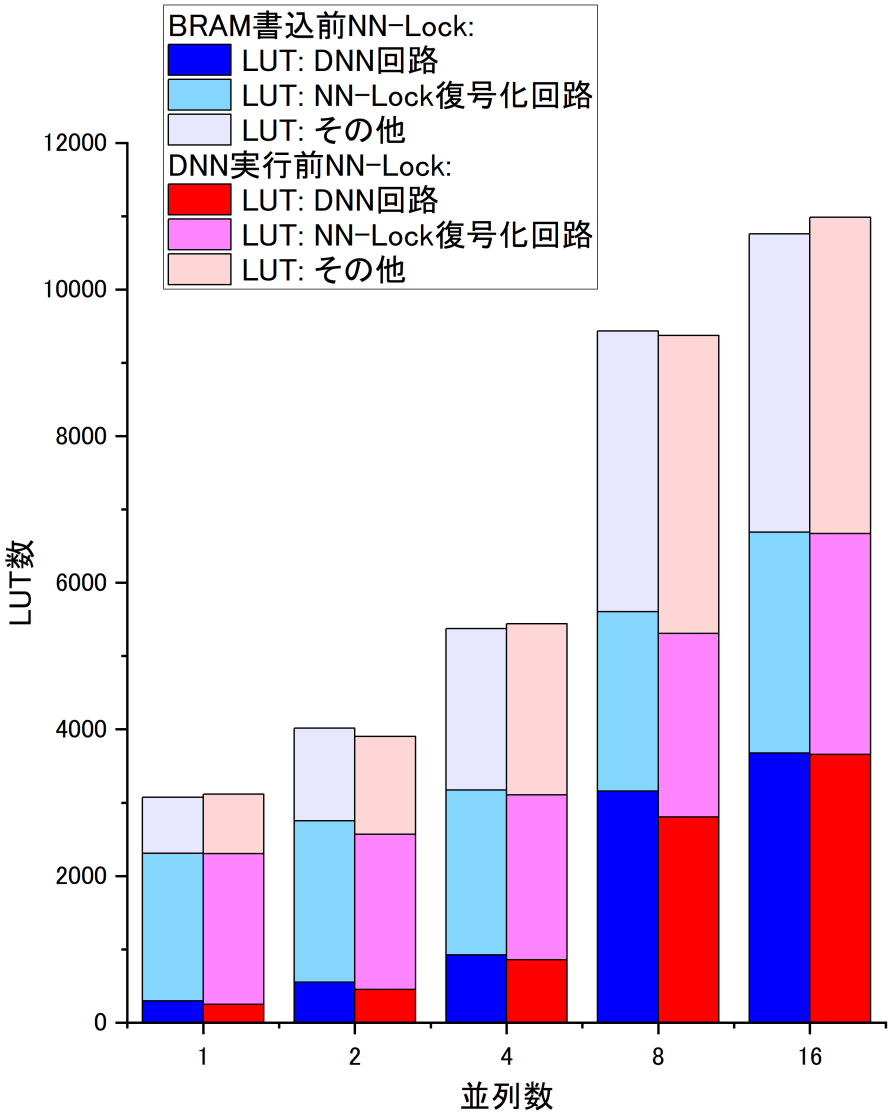

図 5.1: BRAM 書込前 NN-Lock/DNN 実行前 NN-Lock LUT 数の比較

復号化 箇所 / 並列数	LUT 数							
	BRAM 書込前				DNN 実行前			
	DNN 回路	NNLock 復号化 回路	その他	合計	DNN 回路	NNLock 復号化 回路	その他	合計
1	294	2,013	766	3,073	247	2,057	812	3,116
2	552	2,201	1,261	4,014	453	2,117	1,334	3,904
4	921	2,253	2,198	5,372	859	2,248	2,332	5,439
8	3,156	2,449	3,827	9,432	2,802	2,504	4,065	9,371
16	3,676	3,014	4,067	10,757	3,658	3,012	4,314	10,984

表 5.2: BRAM 書込前 NN-Lock/DNN 実行前 NN-Lock
LUT 数

製品名	参考価格 ($)	LUT 数	実装可能並列数
Sipeed Tang Nano 4K[11]	18.50[11]	4,608	1, 2
Sipeed Tang Nano 9K[12]	25.99[12]	8,640	1, 2, 4
Sipeed Tang Nano 20K[13]	31.99[13]	20,736	1, 2, 4, 8, 16
AMD Kria KV260[14]	249.00[14]	256,000	1, 2, 4, 8, 16
AMD Kria KR260[15]	349.00[15]	256,000	1, 2, 4, 8, 16

表 5.3: FPGA ボードの参考価格と搭載 LUT 数，実装可能並列数

NN-Lock/DNN 実行前 NN-Lock 適用時 それぞれの推論処理時間を基に，組み込み機器向けのカメラモジュールとして代表的なものである”Raspberry Pi CAMERA MODULE V2”[17] を使用し議論する．

Raspberry Pi CAMERA MODULE V2 は 1080p で 30fps，つまり 1 フレーム当たり 33ms で動作する．そのため，推論時間 (=画像認識にかかる時間) は 33ms 以内に完了することが望ましい．

しかし，図 5.2，表 5.4, 表 5.5 に示す，BRAM 書込前 NN-Lock/DNN 実行前 NN-Lock 適用時の推論処理時間の比較結果によれば，本研究で行えた評価範囲では，最も処理時間が短い “ BRAM 書込前 NN-Lock: 16 並列 ” で 1 枚当たり 66ms を要しているため，運転支援システムの要求性能を満たすためには更なるモジュール単位での高速化，より大きなサイズでの並列化を実現する必要がある．

5.4 NN-Lock 適用箇所における重みパラメータの窃取耐性の考察

最後に，NN-Lock の適用箇所について，BRAM への重みパラメータ書込直前で NN-Lock を適用する場合と，DNN 演算直前で NN-Lock を適用する場合，両者の窃取耐性について，FPGA を対象とした攻撃手法から考察する．

Stephen M. Trimberger らは，産業用制御等のクリティカルな局面での FPGA の活用が増えていることから，それに伴うデータ保護の必要性を主張し，FPGA を対象とした攻撃手法を提示している [18]．Stephen らが提示した攻撃手法のうち，Bitstream の上書きや改ざんと，電力計測を使用したサイドチャネル攻撃から複数取り上げ考察する．その理由としては，前者においては提案手法の実装・評価に使用している FPGA ボード [8] が Processing System(SoC) を搭載し，Processing System の Programmable Logic 制御機構が攻撃経路となり得るからである．後者においては，サイドチャネル攻撃のうち，電力を対象としたものが最も実行・考察が容易であると考えたためである．

5.4.1 BRAM 書込前 NN-Lock

図 5.3 に回路の概要図と攻撃懸念箇所を示す．

はじめに，自身の提案手法で，Processing System(PS) 側から Programmable Logic(PL) 側間への重みパラメータや画像パラメータの転送は，AXI バスを経由して行っている．但し，そのようなアクセスは，AXI バスを設計し，実装しない限り不可能である．また，この手法でも PS 側から PL 側への一方的なアクセスを行う AXI バスしか実装していないため，原則，PS 側から BRAM 内の値を参照することは不可能である．

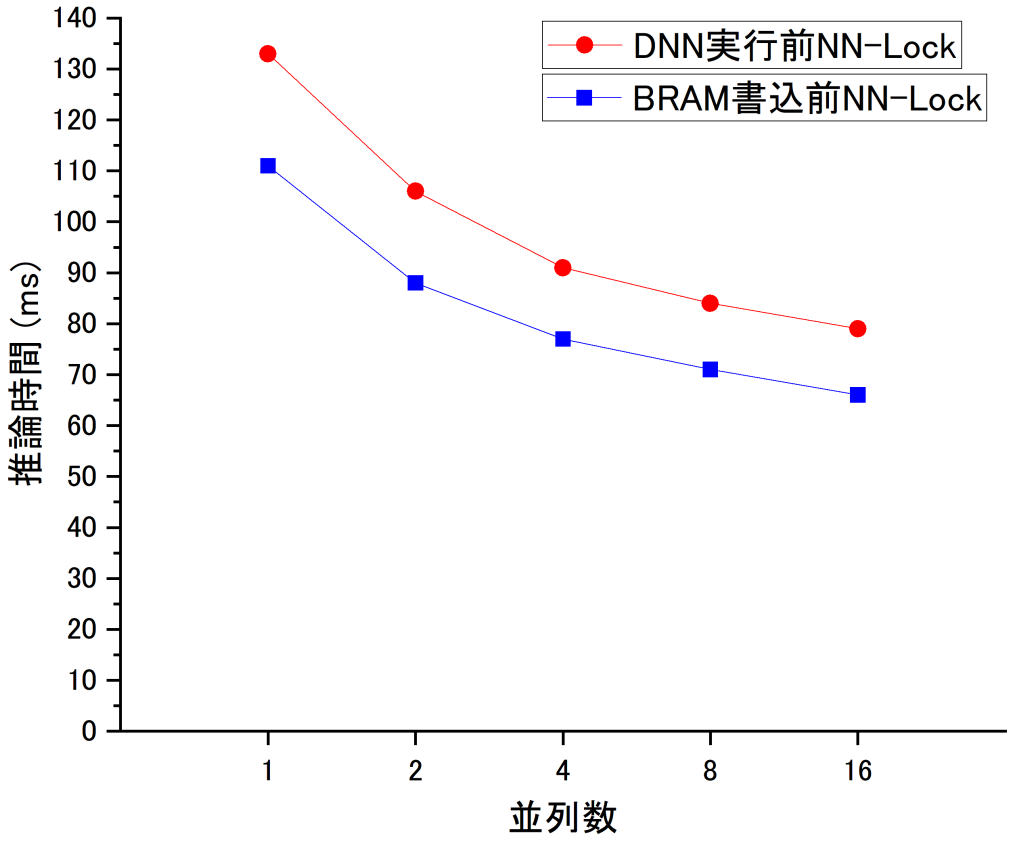

図 5.2: BRAM 書込前 NN-Lock/DNN 実行前 NN-Lock 推論処理時間の比較

並列数	推論時間 [ms]
1	111
2	88
4	77
8	71
16	66

表 5.4: 推論処理時間 (BRAM 書込前 NN-Lock)

並列数	推論時間 [ms]
1	133
2	106
4	91
8	84
16	79

表 5.5: 推論処理時間 (DNN 実行前 NN-Lock)

但し，FPGA 側の制御を担う PS 側で動作する OS の権限が奪取された場合，Bitstream の上書き (図 5.3 中 1.) や改ざん (図 5.3 中 2.) といった攻撃が可能になる．その場合，BRAM 上の復号状態の重みパラメータへのアクセスが可能となってしまうと考えられる．

続いて，電力計測を使用したサイドチャネル攻撃 (図 5.4 中 3.) からの窃取耐性について考える．DNN の推論演算中は，回路上に復号化された重みパラメータが流れ続けることになるため，電力計測等を使用したサイドチャネル攻撃 (図 5.3 中 3.) による推論処理内容の傍受に対しても脆弱であるように考えられる．しかし，表 5.6 に示すように，最大でも回路全体で 2.508W の電力消費に対し，NN-Lock Module は 0.011W と，1%未満の電力消費である．さらに重みパラメータは 8bit=256 段階のため，µW 単位での電力計測が必要であり，攻撃の実行は困難であると考えられる．

5.4.2 DNN 実行前 NN-Lock

図 5.4 に回路の概要図と攻撃懸念箇所を示す．

はじめに，PS 側 OS の権限奪取による Bitstream の上書き (図 5.4 中 1.) や改ざん (図 5.4 中 2.) からの窃取耐性について考える．この攻撃手法によって DNN 回路や NN-Lock 回路は上書きされ使用不能になり，BRAM の値が読出可能になった場合でも，読み出せるのは暗号化済の重みパラメータのみとなる．

続いて，電力計測を使用したサイドチャネル攻撃 (図 5.4 中 3.) からの窃取耐性について考える．ここで，並列数にもよるが，推論処理毎に復号化される重みパラメータは全体の 0.01%程度となる上，1~2 クロック以内に破棄されるため，サイドチャネル攻撃等を利用してレジスタを傍受しようとしても，オリジナルの重みパラメータの全貌の把握が非常に困難であると考えられる．

さらに，前節と同様，表 5.6 に示すように，最大でも回路全体で 2.508W の電力消費に対し，NN-Lock Module は 0.011W と，1%未満の電力消費のため，こちらも µW 単位での電力計測が必要であり，より電力計測を使用したサイドチャネル攻撃は現実的ではないと考えられる．

5.5 まとめ

推論実行前に復号化を行う場合，復号化された重みパラメータは数サイクルしかレジスタに乗らないので奪取される可能性が低くなる．しかし NN-Lock によるオーバーヘッドが発生し，推論時間が長くなることが分かった．

対して BRAM 書込前に復号化を行う場合は，推論時間を維持することができるが，Processing System 側の制御を奪取された際の BRAM への攻撃に著しく弱く，サイドチャネル攻撃等も推論実行前復号化に比べて実現可能性が高い．そのため，セキュリティの観点から見れば，推論実行前復号化を行うことが望ましい．

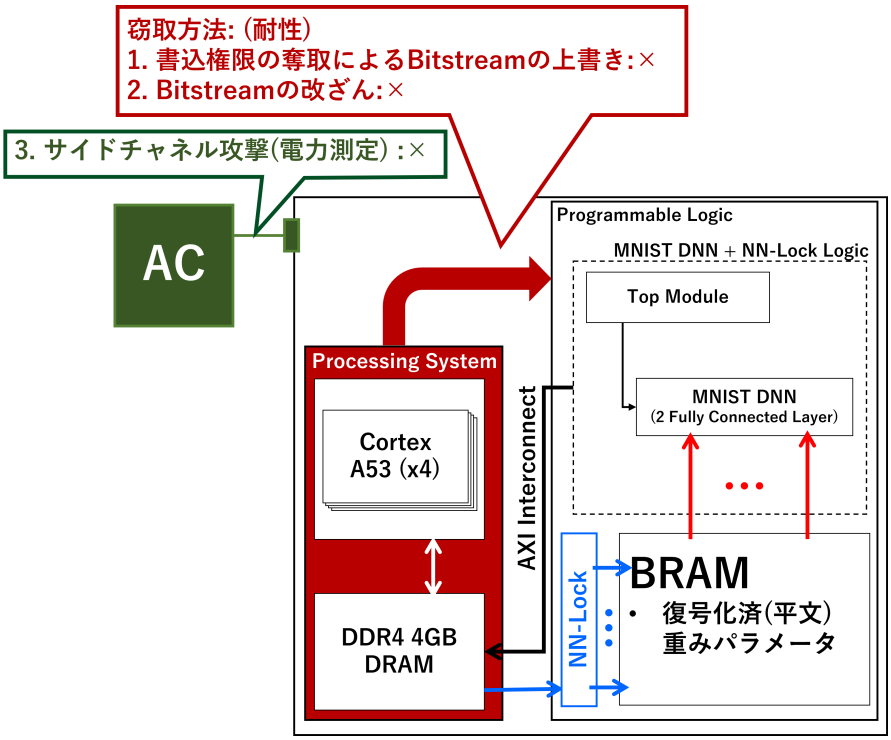

図 5.3: BRAM 書込前 NN-Lock
攻撃懸念箇所と窃取耐性

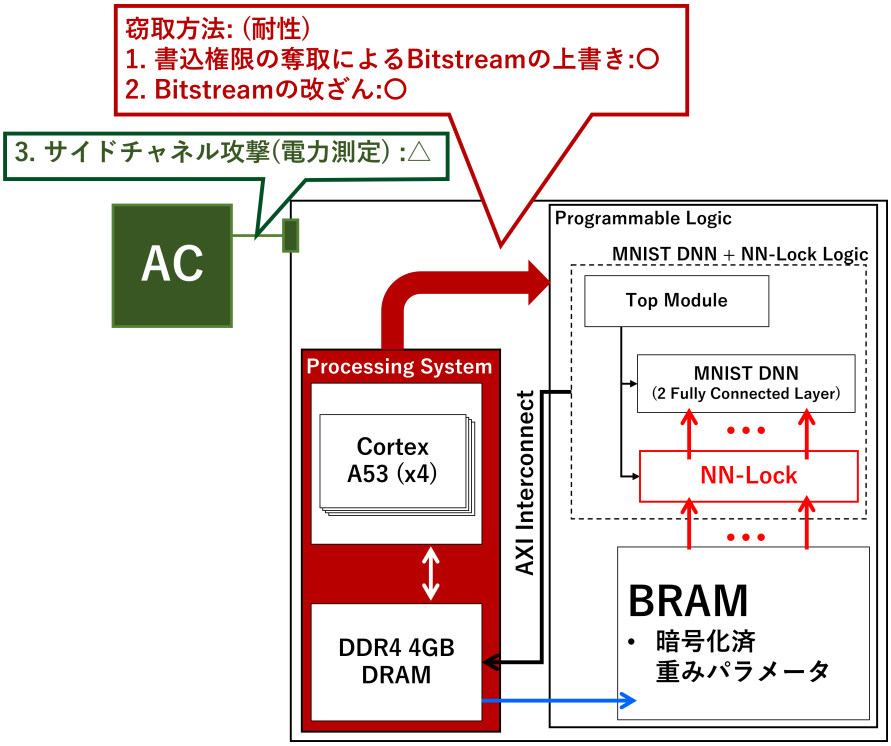

図 5.4: DNN 実行前 NN-Lock
攻撃懸念箇所と窃取耐性

消費電力 / 並列数	消費電力 [W]			
	BRAM 書込前		DNN 実行前	
	全体	NN-Lock Module	全体	NN-Lock Module
1	2.466	0.01	2.465	0.011
2	2.468	0.011	2.468	0.011
4	2.47	0.01	2.47	0.01
8	2.484	0.01	2.496	0.01
16	2.508	0.011	2.508	0.01

表 5.6: BRAM 書込前 NN-Lock/DNN 実行前 NN-Lock
回路全体と NN-Lock Module の消費電力 (Vivado 2023.2 の計測値)

第6章　おわりに

6.1　本研究の概要と成果

本研究では，DNN モデルの実行基盤としての FPGA に着目し，モデル内の重みパラメータを暗号化して保護するソフトウェア上の手法 NN-Lock を，FPGA に複数方針で DNN モデルと共に実装し，複数の条件を考案した上で評価した．

第 2 章では，画像分類における DNN モデルの構造とパラメータの量子化，先行研究である NN-Lock とそのアルゴリズムの中核である AES Key Scheduling について説明した．推論処理において，入力画像の特徴を抽出する重みパラメータを NN-Lock で暗号化することで，不正な利用者が誤った鍵でそのモデルを使用しようとした場合，正しくないパラメータが伝播するため推論精度が大幅に低下し使用できなくなる．実際に DNN モデルの作成と NN-Lock アルゴリズムの再現実装を行い，正しい暗号化鍵と間違った暗号鍵を用いた場合の推論精度がほぼ一致することを示した．

第 3 章では，FPGA 上での NN-Lock の複数の適用手法を提案し，NN-Lock による FPGA 上での重みパラメータ復号化のタイミングを検討した．重みパラメータは既に NN-Lock で暗号化されており，FPGA には NN-Lock 復号化処理を実装する．提案した適用箇所は，第一に DRAM から BRAM への転送時点 (BRAM 書込前 NN-Lock)，第二に DNN 直前での重みパラメータの復号化 (DNN 実行前 NN-Lock) である．加えて，両者とも 1~16 の間で並列数を変更しての適用が可能である．

第 4 章では，NN-Lock に適した FPGA 上での DNN の検討・実装手法を提案し，並列度毎における回路規模 (LUT 数) と 1 枚毎の推論処理時間を計測した．

第 5 章では，第 4 章の実装に対し第 3 章で考案した NN-Lock の複数の適用手法を実装し，並列度毎における回路規模 (LUT 数) と 1 枚毎の推論処理時間を計測し，実際に市場に流通している FPGA ボードやカメラモジュールの性能を参考に，回路規模と推論処理時間の要求要件を考察した．回路規模に関しては，参考価格 $100 を下回る安価な FPGA に対しても，並列数を小さくすることで DNN 処理とその保護機構を実装できることを明らかにした．一方で，推論処理時間に関しては，全ての提案適用手法で一般的なカメラモジュールの性能である 30fps に到達できていないことが明らかになった．

最後に，NN-Lock の適用箇所について，BRAM 書込前 NN-Lock と DNN 実行前 NN-Lock，両者の窃取耐性の議論を行った．前者は推論時間を維持することができるが，PS 側の制御権奪取による PL 側への Bitestream の上書きや改ざん，サ

イドチャネル攻撃等による BRAM への攻撃に比較的弱いと考えられる．後者は NN-Lock によるオーバーヘッドが発生するが，復号化される重みパラメータは毎回パラメータ全体の極一部であり，数サイクルしかレジスタに乗らないので奪取される可能性がより低いと考えられる．そのため，セキュリティの観点から見れば，推論実行前復号化を行うことが望ましいという結論に達した．

6.2 今後の課題

本研究では．FPGA DNN 推論回路上での NN-Lock の適用箇所とその適性についての評価を中心に行った．またそれに伴い，DNN モデルも全結合層のみの単純な構成を使用し，AES 暗号鍵の格納場所についても考慮していなかった．

特に今回使用した全結合層 2 層 MNIST モデルは，DNN モデルの基本単位となるモデルのためその他の大規模なモデルでの応用実装の基礎となると考えていたが，MNIST モデル自体が数字文字の認識に使用されるモデルであることから，運転支援システムを想定した実装・評価に使用するには十分に適切ではなかったと考えられる．

また，AES 暗号鍵についても，5.4 章では重みパラメータを窃取対象として窃取耐性評価を行っており，AES 暗号鍵の窃取耐性の考察を十分に行っていなかった．AES 暗号鍵が窃取できれば暗号化された重みパラメータは全て復号化できるため，窃取耐性を評価するのであれば重みパラメータの一方だけでなく，AES 暗号鍵についても同様に保護手法を考案する必要がある．

そのため，今後の課題としては，CNN を含むより複雑な画像分類モデルに対しての評価検証と，AES 暗号鍵の TPM 等を使用したセキュアな格納方式の考案が必要であると考えている．

また，FPGA 上で運転支援システムへの NN-Lock の実装を検討する場合，一般的なカメラモジュールのフレームレート (30fps) に対して全ての適用手法で推論処理時間が追いついていないため，DNN，NN-Lock 両方の実装の更なるモジュール単位での高速化，複数モジュールの配置によるより大きなサイズでの並列化を実現する必要がある．

最後に，NN-Lock 適用箇所における重みパラメータの窃取耐性については議論に留まったため，今後の展望として，FPGA 本体に対して実際にサイドチャネル等の攻撃を実施し，攻撃可能性を検証する必要がある．

参考文献

[1] Manaar Alam, Sayandeep Saha, Debdeep Mukhopadhyay, and Sandip Kundu. NN-Lock: A Lightweight Authorization to Prevent IP Threats of Deep Learning Models, ACM Journal on Emerging Technologies in Computing Systems, Volume 18, Issue 3, Article 51 (2022)

[2] 斎藤 康毅，”ゼロから作る Deep Learning—Python で学ぶディープラーニングの理論と実装”，オライリー・ジャパン，pp. 39–44 (2016)

[3] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines, In Proceedings of the 27th International Conference on International Conference on Machine Learning, pp.807–814 (2010)

[4] Benoit Jacob, Skirmantas Kligys, and Bo Chen et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 2704–2713 (2018)

[5] Deng, L. The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine, 29(6), pp. 141–142 (2012)

[6] Joan Daemen, and Vincent Rijmen. AES Proposal: Rijndael, NIST AES Algorithm Submission (1999)

[7] 結城 浩，”暗号技術入門 秘密の国のアリス 第 3 版”，SB クリエイティブ，pp. 71–75 (2015)

[8] ”Kria KR260 Robotics Starter Kit Data Sheet (DS988) - AMD” https://docs.amd.com/r/en-US/ds988-kr260-starter-kit/Revision-History，閲覧日 (2025 年 1 月 30 日)

[9] ”LogiCORE IP AXI Interconnect (v1.06.a) - Xilinx” https://docs.amd.com/v/u/en-US/ds768_axi_interconnect，閲覧日 (2025 年 1 月 30 日)

[10] ”PyTorch - Linux Foundation” https://pytorch.org/，閲覧日 (2025 年 1 月 30 日)

[11] "Tang Nano 4K - Sipeed" https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-4K/Nano-4K.html，閲覧日 (2025 年 1 月 30 日)

[12] "Tang Nano 9K - Sipeed" https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-9K/Nano-9K.html，閲覧日 (2025 年 1 月 30 日)

[13] "Tang Nano 20K - Sipeed" https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-20K/Nano-20K.html，閲覧日 (2025 年 1 月 30 日)

[14] "Kria KV260 Vision AI Starter Kit - AMD" https://www.amd.com/ja/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html，閲覧日 (2025 年 1 月 30 日)

[15] "Kria KR260 Robotics Starter Kit - AMD" https://www.amd.com/ja/products/system-on-modules/kria/k26/kr260-robotics-starter-kit.html，閲覧日 (2025 年 1 月 30 日)

[16] Jonathan Horgan, Ciarán Hughes, John McDonald, and Senthil Yogamani. Vision-Based Driver Assistance Systems: Survey, Taxonomy and Advances. In Proceedings of the 2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC '15), IEEE Computer Society, USA, pp. 2032—2039 (2015)

[17] "Raspberry Pi Camera Module 2 - Raspberry Pi Foundation" https://www.raspberrypi.org/products/camera-module-v2/，閲覧日 (2025 年 1 月 30 日)

[18] Stephen M. Trimberger and Jason J. Moore. FPGA Security: Motivations, Features, and Applications, in Proceedings of the IEEE, vol. 102, no. 8, pp.1248–1265 (2014)

研究業績

1. 小俣 直史, 井口 寧, ”MobileNetV2 における機械学習モデル保護手法 (LEWIP) の評価”, 2024 年度 電気・情報関係学会 北陸支部連合大会, JHES24G1_1, 1 page, 金沢工業大学 (Online), Sep. 14, 2024

謝辞

本研究を進めるにあたり，多大な助言やご指導をいただいた井口 寧教授に深謝いたします． 加えて，中間審査・最終審査で有益なご助言をいただきました田中 清史教授，青木 利晃教授，冨田 尭准教授に感謝いたします． 最後に井口研究室の皆様には，ゼミでの議論や修士生活において大きな刺激をいただけたことに感謝の意を表します．