目 次

第1章　まえがき

近年の IoT の普及拡大の伴い，現場で動作するエッジデバイスに機械学習モデルを実装し、その場で推論処理を行うエッジ AI が注目を集めている．エッジ AI はインターネットを介さずに予測を行うため，従来法であるクラウドを利用した推論処理と比較して，リアルタイム性の高さ，セキュリティの高さ，通信コストを節約できる点などがメリットとしてあげられる．

エッジ AI の普及には，処理能力の高いエッジデバイスが求められる．IoT 端末のようなエッジデバイスに用いられる機器は、CPU などのリソースが限られており，クラウドなどの GPU サーバーと比較すると処理能力が非常に低い．一方で，現場で行われる画像分類や物体検出といったタスクに用いられるディープラーニング精度と性能の最大化にニューラルネットワークの構造が複雑になりつつあり，膨大な量のメモリと計算資源が欠かせないため，限られたリソースの中でエッジデバイスを単独で学習・推論させるためには限界があるのが現状である．

この問題を解決するために，AI 処理に特化したハードウェアを迅速に実装できる FPGA（Field Programmable Gate Array）を用いた開発アプローチが注目されている．FPGA とは再構成可能な論理デバイスであり，AI 処理に必要とした膨大な行列演算をハードウェアで実装し，高速・低レイテンシーの AI 処理を実現することができる．また，複雑な機械学習モデルをハードウェアリソースの限られるエッジデバイスに適するために，精度を維持した上，モデルのサイズや必要な計算を削減する「モデル圧縮」技術が挙げられる．モデル圧縮は，メモリ使用量の削減によってエッジデバイス上での推論を実現したり，あるいはパラメータ数を少なくすることによって計算量を小さくさせる，すなわち学習や予測に要する時間を減らすことができるといったメリットが期待できる手法である [4]．

このようなモデル圧縮等のモデルの変換を行い，より少ないハードウェアリソースで AI 推論を実現するための開発プラットフォームとして Xilinx 社が提供しはじめる Vitis AI がある．Vitis AI には、AI の推論処理に関わる最適化された IP やモデル、およびモデル最適化とハードウェアコンパイラツールなどが含まれ，FPGA の設計経験がないユーザーでも高いパフォーマンスの深層学習推論アプリケーションを容易に開発することが可能となる．

本研究では，Vitis AI 開発環境に対する性能評価を目的とし，Vitis AI 開発プラットフォームの構成要素であるモデル圧縮ツール AI Optimizer と AI Quantizer の性能を評価する。具体的には，学習済みのサンプルモデルに対して，AI Optimizer と AI Quantizer を用いて、異なるパラメータでモデルの最適化と量子化を行う．量子化済みのモデルを Ultra 96 FPGA ボード上に実装し，モデル圧縮による正答率の推移など調査することによって，Vitis AI プラットフォームを評価する．

本論文の構成は以下の通りである．第 2 章では Vitis AI の概要と Vitis AI を用いたエッジ AI 開発フローや Vitis AI によるモデル変換，本論文で用いられる用語についての説明を述べる．第 3 章では，Vitis AI に含まれる開発ツールの機能と使用方法について説明する．第 4 章では Vitis AI を用いた AI 推論の実装方法と，実験結果を示し，それについての考察を述べる．第 5 章では本論文のまとめを述べる．

第2章　Vitits AI

本章では Vitis AI の概要と，Vitis AI を用いたエッジデバイスへ AI 推論を実装する際の全体の開発フロー，その中でも本研究で主に扱う部分である Vitis AI によるモデル圧縮について説明する．

2.1　Vitis AI とは [1]

Vitis AI 開発環境とは，エッジデバイスと Alveo アクセラレータカード (注1) の両方を含む，Xilinx 社のハードウェアプラットフォーム向け AI 推論開発キットであり，最適化された IP コア，開発ツール，ソフトウェアライブラリ，ネットワークモデル，サンプルデザインが含まれている．

Vitis AI は以下の要素から構成されている．Vitis AI の構成図を図 2.1 に示す．

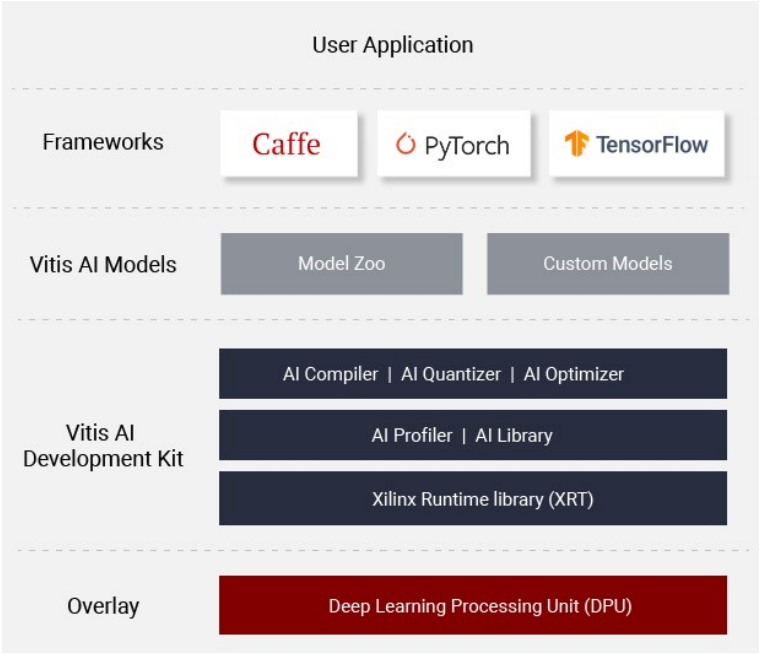

図 2.1　Vitis AI の構成 [3]

(注1) FPGA を搭載したデータセンターのワークロードを高速化するアクセラレーションカード．CPU の高速 PCIe コネクタにそのまま刺して使用する．

2.1.1　DPU

DPU とは，DeepLearning Processor Unit の略で，Xilinx 社の FPGA に実装可能な Neural Network の演算専用の Processor である．DPU は Vitis AI でコンパイルしたビットストリームの処理を行う．DPU はすでにハードウェアに実装されたパラメーター指定可能な IP コアであり，デバイス上に実装するための配置，配線工程が不要のため，プロセッサ向けのソフトウェア開発と同じ感覚で開発を進めることができる．DPU には，Vitis AI 用の命令セットが含まれているため，多くのディープラーニングネットワークを効率的に実装することができる．

2.1.2　モデル・ライブラリ

Vitis AI Model Zoo

Vitis AI Model Zoo には，Xilinx プラットフォームで深層学習推論を素早く運用するための，最適化済み深層学習モデルが含まれ，GitHub 上で公開されている [2]．これらのモデルは，ADAS（先進運転支援システム），自動運転，ビデオ監視，ロボット工学，データセンターなど，様々なアプリケーションに対応している．これらの学習済みモデルを利用することで，深層学習推論の高速化が可能になる．

AI ライブラリ

Vitis AI ライブラリは，DPU を使用して効率的な AI 推論を実装するために構築された高レベルのライブラリと API で構成されている．統合 API を備えた Vitis AI ランタイム (後述) をベースに構築されている．Vitis AI ライブラリは，効率的かつ高品質のニューラルネットワークをカプセル化することで，統合された使いやすい AI アプリケーションインターフェイスを提供する．これにより，深層学習や FPGA について深い知識がなくても深層学習ニューラルネットワークを容易に使用することが可能である．

2.1.3　開発ツール

Vitis AI 開発ツールには AI Optimizer，AI Quantizer，AI Compiler 及び AI Profiler が含まれる．AI Optimizer，AI Quantizer，AI Compiler の 3 つについては，3 章で詳しい解説を行う．

AI プロファイラー

Vitis AI プロファイラーは，AI アプリケーションのプロファイリングと視覚化を行うツールである．ボトルネックの認識や多様なデバイス間における演算リソースの割り当てに役立つ．コードの変更や再コンパイルの必要はなく，関数呼び出しや実行時間，CPU，DPU，メモリなどの使用率を取得できる．

2.1.4　ランタイムソフトウェア

Vitis AI ランタイムにより，アプリケーションはクラウドとエッジの両方に対応する高レベルの統合されたランタイム API を使用することができる．これにより，クラウドとエッジ間の運用がシームレスかつ効率的になる．

Vitis AI ランタイム API の機能は以下の４つである．

・アクセラレータへジョブを非同期送信

・アクセラレータからジョブを非同期取得

・C++ および Python で実装

・マルチスレッド および マルチプロセスの実行に対応

2.2　Vitis AI を用いたエッジ AI 開発フロー

Xilinx 社のハードウェアプラットフォームで AI 推論を開発するときの開発フローを図 2.2 に示す．

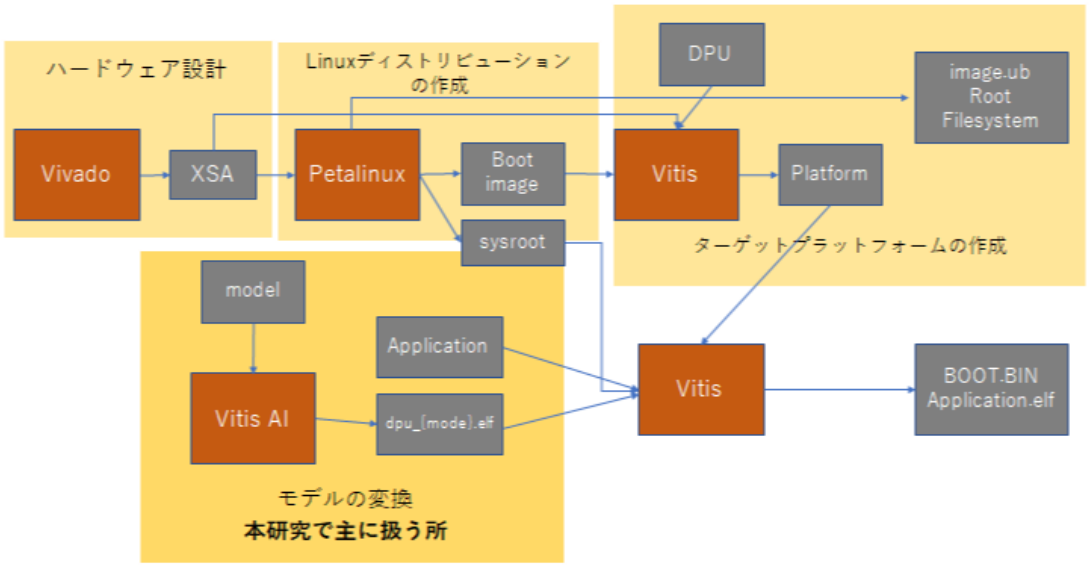

図 2.2　Vitis AI を用いたエッジ AI 開発フロー

図 2.2 に示した通り，開発フローは以下の 5 つの段階に分けられる．

(1) ハードウェア開発 : Vivado を使用して CPU の設定 FPGA の機能 Block の構築などを行う．

(2) Linux ディストリビューションの作成 : PetaLinux を使用して Linux Kernel の構築を行う．

(3) ターゲットプラットフォームの作成:(1) および (2) で生成した Data をもとに DPU(DeepLearning Processor Unit) を搭載した Vitis Target Platform の構築を行う．

(4) モデルの圧縮と変換：VitisAI を用いてモデルの最適化および量子化を行い，DPU 専用のビットストリームに変換する．

(5) 統合：(3)〜(4) を統合,Vitis Target Platform 上で動作する Application を構築し，SD カードイメージとして書き出す．

本研究では，(4) の Vitis AI によるモデルの圧縮に着目し，Vitis AI の評価を行う．

2.3　Vitis AI によるモデル圧縮

Vitis AI を用いたモデル変換では，図 2.3，2.4 に示したように Vitis AI に含まれる開発ツール，AI Optimizer，AI Quantizer，AI Compiler を用いて機械学モデルの最適化，量子化，及び DPU 専用のビットストリームへのコンパイルを行う．

開発ツールの機能と使用方法については次章で詳しい説明を行う.

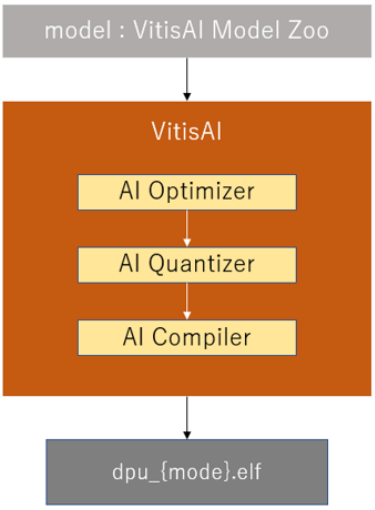

図 2.3　Vitis AI によるモデル圧縮 (1)

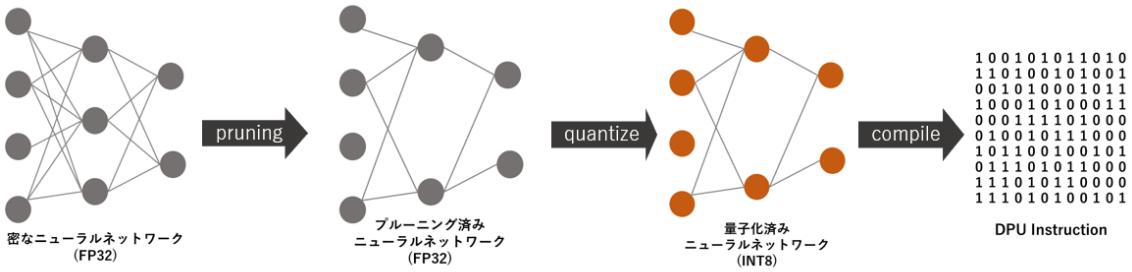

図 2.4　Vitis AI によるモデル圧縮 (2)

第3章　Vitis AI 開発ツール

本章では Vitis AI に含まれる開発ツール (AI Optimizer，AI Quantizer，AI Compiler) の機能と実行手順について説明する．

3.1　AI Optimizer [5]

AI Optimizer はニューラルネットワークモデルの最適化を行う．現在，AI Optimizer に含まれるのはプルーナーと呼ばれるツールのみである．Vitis AI プルーナー (VAI プルーナー) は，冗長な接続を刈り込み (プルーニング)，必要な演算数を全体的に削減する．VAI プルーナーで生成したプルーニング済みモデルは，AI Quantizer で量子化を行い，AI Compiler でコンパイルして FPGA 上で運用することができる．

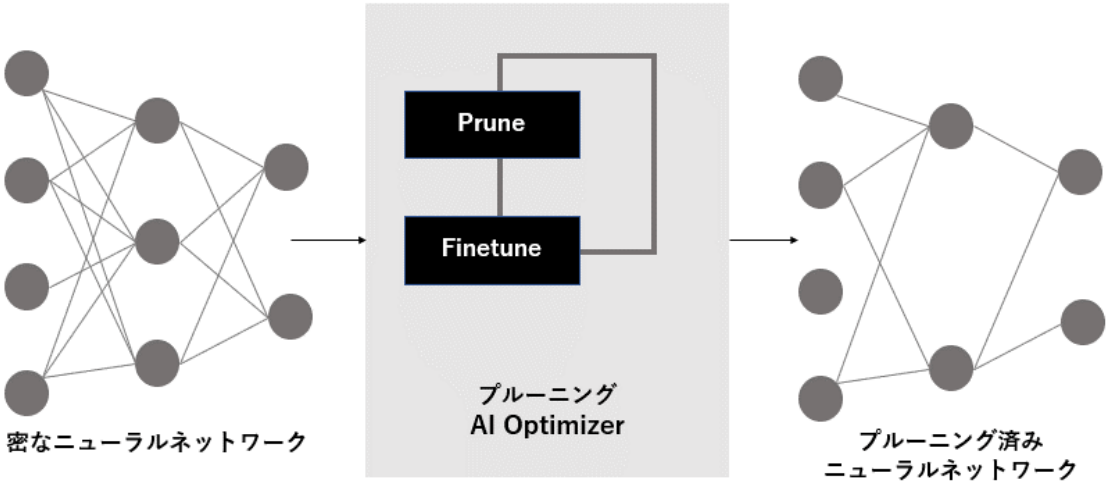

図 3.1　AI Optimizer [6]

3.1.1　プルーニング

プルーニングの概要

ほとんどのニューラルネットワークは，特定の精度を達成するために，冗長性が高く，過剰にパラメータ化されている.「プルーニング」は．精度の低下をできるだけ低く抑えながら，過剰な重みを排除するプロセスである．もっとも単純なプルーニングは,「細粒度プルーニング」と呼ばれ，結果としてスパース型重み行列になる．VAI ブルーナーでは「粗粒度プルーニング」が採用されている．これは，ネットワークの精度に大きく寄与しないニューロンを排除する手法である．畳み込み層では，粗粒度プルーニングによって 3D カーネル全体がプルーニングされるため，チャンネルプルーニングとも呼ばれる．

プルーニングを実行すると，元のモデルの精度が低下するため，再学習（微調整）することで，残りの重みを調整して精度を回復する，

反復プルーニング

VAI ブルーナーは，図 3.2 に示すような反復的プロセスによって，精度の低下を最小限に抑えながらモデルのパラメーター数を削減する．

VAI ブルーナーの主なタスクは，次の 4 つである．

(1) 解析 ： モデルの感度分析を実行して，最適なプルーニング手法を決定する．

(2) プルーニング ： 入力モデルの演算数を削減する．

(3) 微調整 ： 再学習により，精度を回復させる．

(4) 変換 ： 重みが削減されたデンス (密) モデルを生成する．

次の手順に従ってモデルをプルーニングする．

(1) 元のベースライン モデルを解析する．

(2) モデルをプルーニングする．

(3) プルーニング済みモデルを微調整する．

(4) 手順 2 と 3 を数回繰り返す．

(5) プルーニング済みのスパース (疎) モデルを最終的なデンス (密) モデルに変換する.

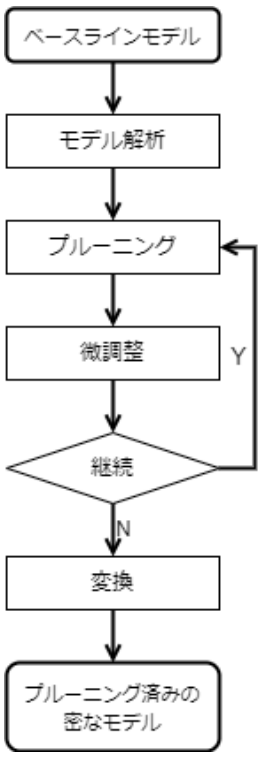

図 3.2　プルーニングワークフロー

3.1.2　VAI ブルーナーの使用方法

VAI プルーナーは，TensorFlow，PyTorch，Caffe，および Darknet の 4 つの深層学習フレームワークがサポートされており，それぞれに対応するツール名は，vai_p_tensorflow，vai_p_pytorch，vai_p_caffe，および vai_p_darknet である.「p」はプルーニングを表している.

本研究は，深層学習フレームワークに Tensorflow を使用したため，vai_p_tensorflow について説明する.

vai_p_tensorflow の実行には，次の表 3.1 に示す引数が使用できる.

表 3.1　vai_p_tensorflow の引数

名前	タイプ	説明
action	文字列	実行するアクション. 有効なアクションは,「ana」,「prune」,「transform」,「flops」
workspace	文字列	出力ファイルを格納するディレクトリ.
input_graph	文字列	ネットワークのアーキテクチャを定義した GraphDef protobuf ファイルのパス
input_ckpt	文字列	チェックポイント ファイルのパス.
eval_fn_path	文字列	モデルの評価に使用される Python ファイルのパス.
target	文字列	モデルの精度を示す出力ノード名.
max_num_batches	int	評価するバッチの最大数 (デフォルトではすべて使用).
output_graph	文字列	プルーニング済みネットワークを格納する GraphDef protobuf ファイルのパス
output_ckpt	文字列	重みを格納するチェックポイント ファイルのパス.
gpu	文字列	使用する GPU のデバイス ID (カンマ区切り).
sparsity	float	プルーニング後のネットワークの目標スパース度
exclude	リピート	プルーニングから除外されるたたみ込みノード
input_nodes	リピート	推論グラフの入力ノード.
input_node_shapes	リピート	入力ノードの形状.
output_nodes	リピート	推論グラフの出力ノード.
channel_batch	int	プルーニング後、出力チャネルの数はこの値の倍数値となる.

モデルの解析

モデルのプルーニングを実行する前に，まずモデルを解析する必要がある．このプロセスの主な目的は，モデルをプルーニングする際の適切なプルーニング手法を見つけることである．以下に示したようにオプションを指定し，vai_p_tensorflow を呼び出し，モデル解析を実行する．

```
vai_p_tensorflow \
 --action=ana \
 --input_graph=inference_graph.pbtxt \
 --input_ckpt=model.ckpt \
 --eval_fn_path=eval_model.py \
 --target="recall_5" \
 --max_num_batches=500 \
 --workspace:/tmp \
 --exclude="conv node names that excluded from pruning" \
 --output_nodes="output node names of the network"
```

プルーニングループ

ana コマンドが完了後，モデルのプルーニングを開始できる．prune コマンドは ana コマンドとよく似ており，同じコンフィギュレーションファイルを使用できる．

```
vai_p_tensorflow \
 --action=prune \
 --input_graph=inference_graph.pbtxt \
 --input_ckpt=model.ckpt \
 --output_graph=sparse_graph.pbtxt \
 --output_ckpt=sparse.ckpt \
 --workspace=/home/deephi/tf_models/research/slim \
 --sparsity=0.1 \
 --exclude="conv node names that excluded from pruning" \
 --output_nodes="output node names of the network"
```

prune コマンドが完了すると，モデルを新規に学習させる手順と同じ手順で，プルーニン

グ済みモデルの再学習による微調整を行う．プルーニングと微調整が完了すると，プルーニングの 1 回の反復が完了したことになる．一般に，精度を大幅に低下させることなくプルーニング率を向上させるには，モデルを数回プルーニングする必要がある．1 回の「プルーニング/微調整」の反復が終わるたびに，コマンドに次の 2 つの変更を加えてから次のプルーニングを実行する．

(1) –input_ckpt フラグを，直前の微調整プロセスで生成したチェックポイントファイルに変更する．

(2) –sparsity フラグの値を大きくして、次の反復のプルーニング量を増やす．

密なモデルへの変換

プルーニングの反復がすべて終了したあと，transform コマンドで最終的な密なモデルへの変換を行う．

```
vai_p_tensorflow \
 --action=transform \
 --input_ckpt=model.ckpt-10000 \
 --output_ckpt=dense.ckpt
```

これにより，プルーニング済みモデルのアーキテクチャを格納した GraphDef ファイルと，学習済みの重みを保存したチェックポイント ファイルが生成される．

3.2　AI Quantizer [6]

Vitis AI Quantizer(VAI クオンタイザー) は，32 ビット浮動小数点の重みやアクティベーションコードを INT8 などの固定小数点に変換 (量子化) することで，予測精度を損なうことなく計算の複雑さを軽減させるためのツールである．固定小数点ネットワークモデルの方が，浮動小数点モデルよりも必要なメモリ帯域幅が狭く，速度と電力効率が向上する．

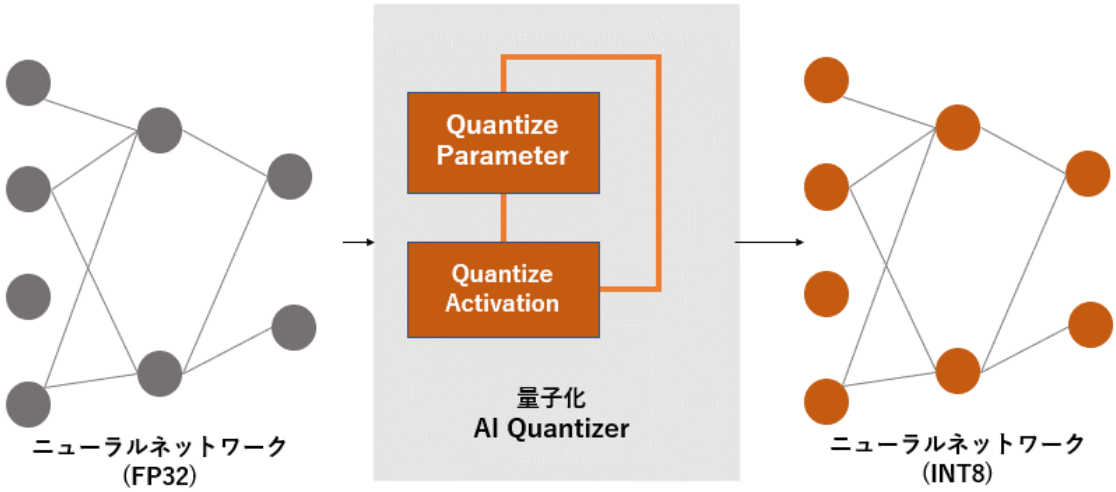

図 3.3　AI Quantizer [6]

3.2.1　量子化

一般的に，ニューラル ネットワークの学習では 32 ビット浮動小数点の重みと活性化値を使用する．VAI クオンタイザーは，32 ビット浮動小数点の重みやアクティベーションコードを 8 ビットの整数 (INT8) に変換することで，予測精度を損なうことなく計算の複雑さを軽減できる．固定小数点ネットワーク モデルの方が，浮動小数点モデルよりも必要なメモリ帯域幅が狭く，速度と電力効率が向上する．Vitis AI クオンタイザーは，たたみ込み，プーリング，完全接続、バッチ正規化など，ニューラル ネットワークの一般的なレイヤーをサポートしている．

3.2.2　VAI クオンタイザーの使用方法

VAI クオンタイザーは，現在 TensorFlow，PyTorch，および Caffe の 3 つの深層学習フレームワークがサポートされており，ツールの名前は、それぞれ vai_q_tensorflow、vai_q_pytorch、および vai_q_caffe となっている．

本研究は，深層学習フレームワークに Tensorflow を使用したため，vai_q_tensorflow について説明する．

vai_q_tensorflow の実行には，次の表 3.2 に示す引数が使用できる．

表 3.2　vai_q_tensorflow の引数

名前	タイプ	説明
–input_frozen_graph	文字列	量子化キャリブレーションに使用される， 浮動小数点モデル用の TensorFlow の 凍結された推論 GraphDef ファイル.
–input_nodes	文字列	量子化グラフの入力ノードネームリストであり， 「–output_nodes」と共に使用される.
–output_nodes	文字列	量子化グラフの出力ノードネーム リストであり， 「–input_nodes」と共に使用される.
–input_shapes	文字列	input_nodes の形状リスト.
–input_fn	文字列	この関数はグラフの入力データを提供し， キャリブレーションデータセットで使用される.
–output_dir	文字列	量子化結果を保存するディレクトリ.
–weight_bit	Int32	量子化後の重みおよびバイアスのビット幅．デフォルト: 8
–activation_bit	Int32	量子化後の活性化値のビット幅．デフォルト: 8
–method	Int32	量子化の方法. 0: Non-overflow 量子化プロセスで値がオーバーフローしない. 外れ値の影響を大きく受ける. 1: Min-diffs 量子化プロセスでオーバーフローを許可し， 量子化誤差を極力抑える．外れ値に対する耐性がある. 選択肢: [0, 1] デフォルト: 1
calib_iter	Int32	キャリブレーションの反復数．デフォルト: 100
–skip_check	Int32	1 に設定されている場合， 浮動小数点モデルのチェックはスキップされる. 選択肢: [0, 1] デフォルト: 0

vai_q_tensorflow を使用したモデルの量子化

次のコマンドを実行してモデルを量子化する.

```
vai_q_tensorflow quantize \
    --input_frozen_graph frozen_graph.pb \
    --input_fn           image_input_fn.calib_input \
    --output_dir          ${QUANT_DIR} \
    --input_nodes        ${INPUT_NODES} \
    --output_nodes       ${OUTPUT_NODES} \
    --input_shapes       ${INPUT_SHAPE_Q} \
    --calib_iter         100 \
    --weight_bit         8 \
    --activation_bit     8 \
    --method             1
```

量子化されたモデルの出力

vai_q_tensorflow コマンドが正常に実行されると、 –output_dir で指定したディレクトリに 2 つのファイルが生成される.

1. quantize_eval_model.pb ： CPU/GPU で評価するために使用され，ハードウェアで結果をシミュレーションするために使用できる.
2. deploy_model.pb ： このファイルから DPU コードをコンパイルし，DPU 上でコードを運用する．これは Vitis AI コンパイラ用の入力ファイルとして使用できる.

3.3　AI Compiler [6]

Vitis AI コンパイラ (VAI コンパイラ) は，各種の DPU に対するニューラル ネットワーク計算の最適化に使用されるコンパイラファミリへの統合インターフェイスである．各コンパイラは，ネットワーク モデルを，高度に最適化された DPU 命令シーケンスにマップする．

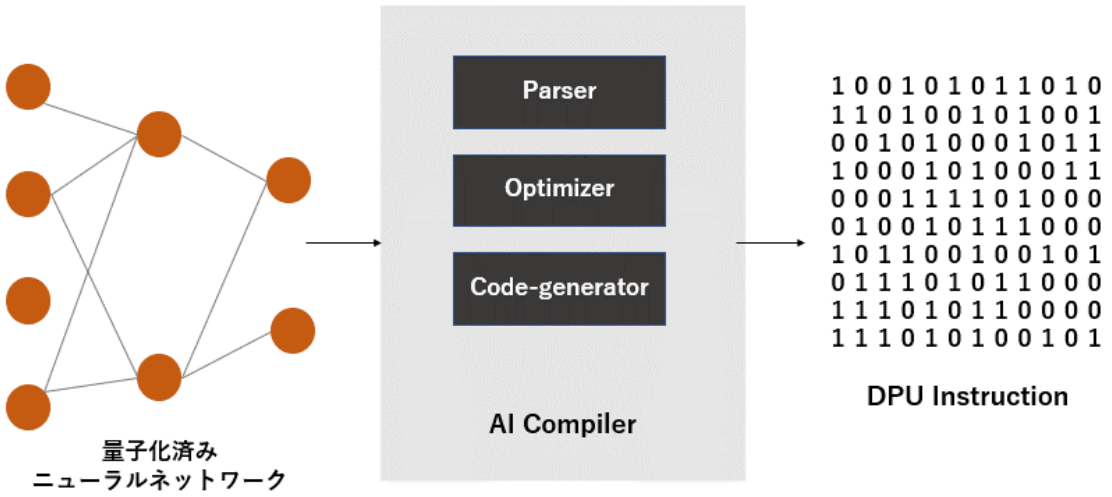

図 3.4　AI Compiler [6]

3.3.1　VAI コンパイラの使用方法

Caffe および TensorFlow フレームワークに対応する Vitis AI コンパイラは，それぞれ vai_c_caffe と vai_c_tensorflow であり，クラウドおよびエッジ DPU で利用可能である．vai_c_tensorflow のオプションを次の表 3.3 に示す．

表 3.3　vai_c_tensorflow のオプション

名前	説明
–arch	JSON 形式の VAI_C コンパイラの DPU アーキテクチャコンフィギュレーションファイル．
frozen_pb	コンパイラ vai_c_tensorflow 用の TensorFlow の凍結された protobuf ファイルのパス．
–output_dir	コンパイル プロセス後の出力ディレクトリのパス．
–net_name	VAI コンパイラでコンパイルされた後のネットワーク モデルの DPU カーネルの名前．

次のコマンドを実行してコンパイルを行う.

```
vai_c_tensorflow \
    --frozen_pb  deploy_model.pb \
    --arch       arch.json \
    --output_dir ${COMPILE_ZCU102} \
    --net_name   CNN
```

第4章　Vitis AI を用いた AI 推論の実装 [7]

本章では本研究で行った実験の内容，及び結果と考察について述べる．実験環境を表 4.1 に示す．

表 4.1　実験環境

OS	Ubuntu 18.04.5
CPU	Intel(R) Xeon(R) W-2225 CPU @ 4.10GHz
	Python 3.6.13 Tensorflow 1.15
	Vitis AI ver.1.2
FPGA	Ultra96v2

4.1　実験内容

本研究では Vitis AI を用いて FPGA(Ultra96v2) 上で MINIST(手書き数字) の判別を行う CNN(畳み込みニューラルネットワーク) を実装する．

Vitis AI に含まれる開発ツールの内，AI Optimizer のみは別途で商用ライセンスの取得が必要だが，研究目的のみでのライセンスの販売は行っておらず取得することができなかったため，AI Optimizer は使用していない．

Tensorflow を用いて作成した CNN を VAI クオンタイザーで量子化し，VAI コンパイラでコンパイルして Ultra96v2 上で動作確認を行う．量子化を行う際に，vai_q_tensorflow のオプションを変更し正答率等の変化を調べる．

実装した CNN モデルの構造を図 4.1 に示す．

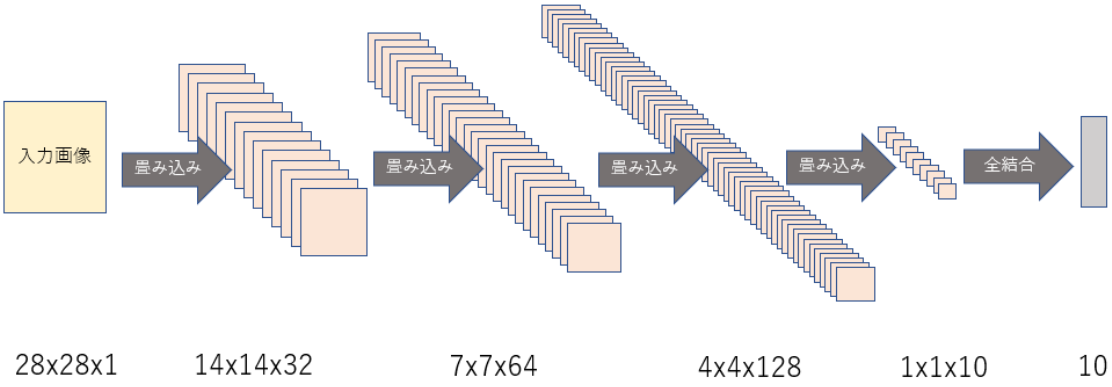

図 4.1　CNN モデルの構造

変更するオプションは表 3.2 にある，–weight_bit，–activation_bit，–method の 3 つで，重みと活性化値のビット幅が 16 ビット，8 ビット，4 ビットの場合にオーバフローを許可する量子化と，許可しない量子化をそれぞれ行い，2 ビットと 1 ビットの場合でオーバフローを許可する量子化を行う．計 8 種類のパターンで CPU で動作させた場合の正答率，Ultra96v2 のＤＰＵで動作させた場合の正答率，計算時間を計測し比較を行う．

また，Vitis AI Model Zoo で公開されている学習済みモデルにも Vitis AI によるモデル変換を行い Ultra96v2 上で動作確認を行う．Vitis AI Model Zoo からは，AlexNet と ResNet を使用し，AlexNet では Dogs vs. Cats データセットで犬猫判別，ResNet では IMAGENET(ILSVRC2012) データセットで視覚認識を実行した．

4.2　結果と考察

4.2.1　結果

Vitis AI を用いて FPGA(Ultra96v2) 上で MINIST(手書き数字) の判別を行う CNN を実装した結果の量子化による識別精度と画像 1000 枚に対して推論処理を行った際の計算時間の変化を表 4.2 に，実行時の DPU 使用率を図 4.2 に示す (注1)．

(注1) DPU 使用率はすべてのパラメータ設定で変化がなかった．

表 4.2　量子化とコンパイルによる識別精度の変化

量子化ビット幅	オーバフロー	正答率 (CPU)	計算時間 (s)	正答率 (DPU)	計算時間 (s)
量子化前 (32)	—	0.9766	2.746	—	—
16	あり	0.9766	3.321	0.0980	4.2469
16	なし	0.9766	3.331	0.0980	4.2469
8	あり	0.9766	3.343	0.9831	4.2152
8	なし	0.9764	3.321	0.9823	4.2032
4	あり	0.9594	3.264	0.9507	4.1943
4	なし	0.9194	3.241	0.8033	4.1972
2	あり	0.3460	3.121	0.1017	4.2041
1	あり	0.0978	3.197	—	—

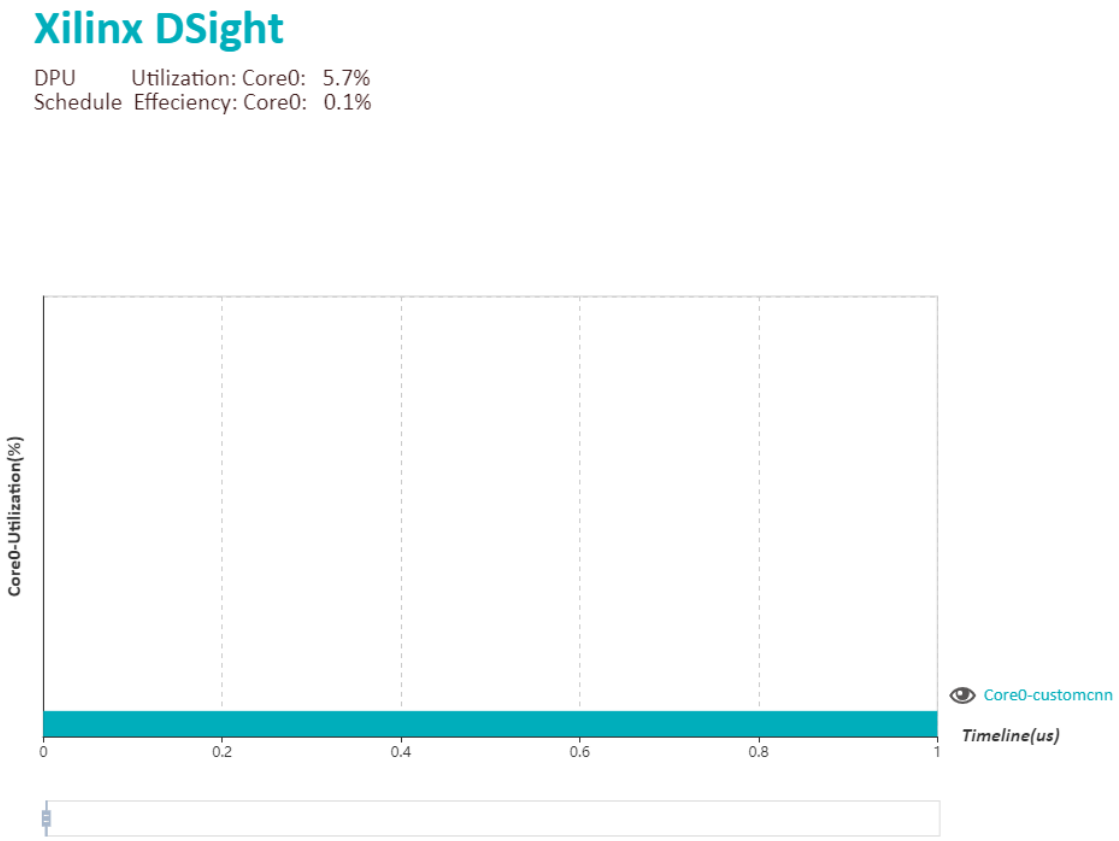

図 4.2　Ultra96v2 上での DPU 使用率

また，Vitis AI Model Zoo で公開されている学習済みモデルのモデル変換を行ったが，Vitis AI Model Zoo のモデルは Ultra96v2 をサポートしておらず，コンパイルすることができな

かったため FPGA 上での動作確認を行うことができなかった．

AlexNet と ResNet に対して vai_q_tensorflow で量子化を行い，CPU で動作させた際の正答率をそれぞれ表 4.3，表 4.4 に示す．

表 4.3　量子化による識別精度の変化 AlexNet

量子化ビット幅	オーバフロー	正答率 (CPU)
量子化前 (32)	—	0.9326
8	あり	0.9314
8	なし	0.9304
4	あり	0.8294
4	なし	0.8156
2	あり	0.5
1	あり	0.5

表 4.4　量子化による識別精度の変化 ResNet

量子化ビット幅	オーバフロー	正答率 (CPU)
量子化前 (32)	—	0.99555
8	あり	0.99445
8	なし	0.99435
4	あり	0.00105
4	なし	0.00235
2	あり	0.0

4.2.2　考察

CPU で実行した場合は 16 ビット量子化から，4 ビット量子化までの間は実用的な正答率で動作しており，オーバフローを許可しない場合外れ値の影響を受けやすく通常のデータセットの場合正答率が低下することも確認できた．どの場合においても実行時間と DPU 使用率にほとんど差が生まれなかったことと，16 ビット量子化が FPGA 上で動作させた場合に極端に正答率が低下したことの原因として，ネットワークの規模が小さかったことと，コンパイルで生成されるビットストリームを処理する DPU が 8 ビット量子化用に開発されたものであることが考えられる．

また，元のモデルが複雑になるほど 4 ビット以下での量子化の際の正答率の低下が大きくなることも確認でき，比較的単純なネットワークであれば 4 ビット量子化も現実的であるが，基本的には 8 ビットでオーバーフローありの量子化が良いと考えられる．

第5章　あとがき

近年の IoT の普及拡大の伴い，現場で動作するエッジデバイスに機械学習モデルを実装し、その場で推論処理を行うエッジ AI が注目を集めている．エッジ AI の普及には，処理能力の高いエッジデバイスが求められるが，IoT 端末のようなエッジデバイスに用いられる機器は、CPU などのリソースが限られており，その中でエッジデバイスを単独で学習・推論させるためには限界があるのが現状である．

この問題を解決する手法として，機械学習モデルの精度を維持した上，モデルのサイズや必要な計算を削減する「モデル圧縮」技術が挙げられ，このモデル圧縮等の技術を用いてモデルの変換を行い，より少ないハードウェアリソースで AI 推論を実現するための開発プラットフォームとして Xilinx 社が提供しはじめる Vitis AI がある．

本研究では，Vitis AI 開発環境に対する性能評価を目的とし，Vitis AI 開発ツールの核であるモデル圧縮ソフト AI Quantizer の性能評価を行った．評価実験の結果により，1〜4 ビットの量子化はあまり現実的ではないこと，オーバフローを許可した量子化の方が精度を維持しやすいことが確認できた.

今後の課題として，量子化による影響を受けやすいモデルに対する量子化後のモデルの再学習による精度回復プログラムの実装や，ZCU102 等の FPGA 使用した AI Model Zoo の学習済みモデルの評価などがあげられる．

謝辞

本研究を進めるにあたり，懇篤な御指導，御鞭撻を賜りました本学高橋寛教授に深く御礼申し上げます.

本論文の作成に関し，詳細なる御検討，貴重な御教授を頂きました本学甲斐博准教授、王森レイ講師に深く御礼申し上げます.

また，審査頂いた本学高橋寛教授，遠藤慶一准教授ならびに宇戸寿幸准教授に深く御礼申し上げます.

最後に，多大な御協力と貴重な御助言を頂いた本学工学部情報工学科情報システム工学講座高橋研究室の諸氏に厚く御礼申し上げます.

参考文献

［1］ ACRi，‘広がり続ける FPGA の応用と人工知能への活用 (4)，” https://www.acri.c.titech.ac.jp/wordpress/archives/7041#toc8，2022/1/30.

［2］ Xilinx，‘Xilinx/Vitis AI Model Zoo，” https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo，2022/1/30.

［3］ Xilinx，‘Vitis AI - Xilinx，” https://japan.xilinx.com/products/design-tools/vitis/vitis-ai.html，2022/1/30.

［4］ Laboro.AI, 内木 賢吾，‘ディープラーニングを軽量化する「モデル圧縮」３手法，” https://laboro.ai/activity/column/engineer/ディープラーニングを軽量化するモデル圧縮, 2022/1/30.

［5］ Xilinx，“ug1333-ai-optimizer，” Xilinx，2020.

［6］ Xilinx，“ug1414-vitis-ai，” Xilinx，2021.

［7］ Xilinx，‘Xilinx/Vitis-AI-Tutorial，” https://github.com/Xilinx/Vitis-AI-Tutorials, 20221/30.