VAEを用いた画像圧縮と異常検知の実現,
及びエッジコンピューティングへの応用の検討

今村　優希[†a)]　　　川崎　大雅[†b)]

Yuki IMAMURA[†a)] and Taiga KAWASAKI[†b)]

あらまし　ああ（大体300文字）

キーワード　VAE, FPGA, エッジコンピューティング

1. はじめに

近年，無線通信技術は飛躍的に向上しており，5G通信の普及が進んでいる．5Gは従来の4Gなど通信規格と異なり，「高速大容量」「低遅延」「多数同時接続」の3つの特徴を備えており，その中でも「低遅延」と「多数同時接続」は新たな通信環境を構築する上で重要な軸となっている[1]．従来の4G通信は，人が使用するスマートフォンや携帯に焦点を当てていた．しかし，5Gでは車両，ドローン，センサなどのIoT機器が大量にネットワークに接続されることを前提としている．

また，今現在のIT業界ではクラウドが主流である．図1の左側の処理のように，エッジデバイスからの処理をクラウドで実施し，その結果をエッジデバイスに通知するという仕組みである．ただ，多くの端末から取得したデータをクラウドのみで処理を行うのはある程度限界があり，またトラヒック量が増加して，5Gのメリットを享受できないという問題が発生すると考えられる．

このような課題を解決するため，エッジコンピューティングという手法が近年注目され始めている．エッジコンピューティングとは，従来はクラウドで行っていた処理の一部を，ユーザ端末（スマートフォンやIoT機器）の近い位置である基地局やその至近に設置されているサーバなどでデータ処理を行う技術である[2]．この手法を用いることでクラウドにかかる処理をエッジコンピューティングで分散することが可能で，通信のトラヒック量，5Gの特徴のひとつである「低遅延」に貢献することも可能である．

そこで，エッジコンピューティングの実現をVAEとFPGAを用いて実現することを考えた．すでにVAEを用いて，道路上の異物を検知するシステムも作成されている．したがって，今回は図1の右側の処理の一部を作成することを目標とし，車載カメラからの情報に対して，画像圧縮や異常を検知するシステム構築を目指した．

† 九州工業大学情報工学部 情報・通信工学科3年
福岡県飯塚市川津 680-4
Kyushu Institute of Technology, School of Computer Science and System Engineering, Department of Computer Science and Networks
a) E-mail: imamura.yuuki475@mail.kyutech.jp
b) E-mail: kawasaki.taiga000@mail.kyutech.jp

2. 方　　法

2.1 実行環境

今回のシステム作成において使用したツール及びそのバージョンを表1で示す．また，FPGAの評価ボード

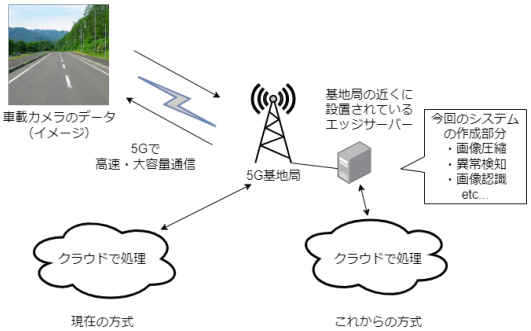


図 1: エッジコンピューティングのイメージと
今回のシステムの作成部分

表 1: 使用したツール

用途	使用ツール
VAE シミュレーション用	MATLAB 2024b
ハードウェアシミュレーション	MATLAB/Simulink 2022a
HDL コード生成	HDL Coder
FPGA 設計ソフトウェア	Vivado 2022.1
ハードウェアアクセラレーション	Vitis 2022.1
評価ボード	DIGILENT 製 ZYNQ-7010

として，DIGILENT 製の SoC を搭載した ZYNQ-7010 を使用する．

2.2 システム構造

今回の VAE を搭載したシステムでは，以下 2 つの機能を搭載した．

1　画像圧縮

VAE の特徴のひとつである次元圧縮能力を画像に応用する．

2　異常検知

もう一つの特徴である異常検知を，元画像と生成画像とを比較して行う．

全体の構想について解説する．今回使用する画像は，簡単化のためにグレースケール化したものを使用する．また，JPEG のようにブロックに分割して，それぞれのブロック毎に処理を行う．ブロックサイズは 16 × 16 に設定した．

画像圧縮のアルゴリズムは，ブロック毎に元画像と圧縮後の画像との比較を PSNR を用いて行う．PSNR が設定した閾値以上だった場合は，圧縮した潜在空間を用いても問題がないので，潜在空間を画像データとして利用する．それ以下だった場合は，元の画像データを送信する．異常検知のアルゴリズムは，今から通過するであろうブロックの PSNR の値が閾値以下だった場合は道路以外の可能性が高いので異常と判定するよう設定する．

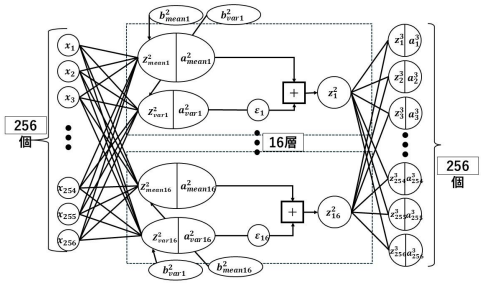


図 2: 256 × 16 × 256VAE の構造

上記の機能を実現するために，VAE の構造を 2.2.1 で説明する．また，FPGA の構造の詳細を 2.2.2 にて説明し，最後に SoC FPGA の構造を 2.2.3 にて説明する．

2.2.1　VAE 構 造

VAE の構造の概略を図 2 に示す．16 × 16 の画像を使用することから，入力 256 次元，出力 256 次元で設計を行った．潜在空間の次元は，圧縮空いたとしてもある程度判別がつくように 16 次元で作成を行った．エンコーダ部分の活性化関数に関しては，平均では ReLU 関数を使用し，分散ではソフトプラス関数を使用している．また，デコーダ部分ではシグモイド関数を利用している．

$$f(x) = x \qquad : ReLU \text{ 関数} \qquad (1)$$

$$f(x) = \log(1 + e^{x}) \qquad : Softplus \text{ 関数} \qquad (2)$$

$$f(x) = \frac{1}{1 + e^{-x}} \qquad : Sigmoid \text{ 関数} \qquad (3)$$

また，VAE の学習方法を図 3 に示す．まず，道路のみのブロックを大量に用意する．そのブロックを教師データとし，VAE を学習させる．学習の条件を表 2 にまとめた．今回は道路が映った写真を用意し 3，道路のみ映った部分をブロック化することで教師データの用意を行った．また，MATLAB で VAE の学習のみ行い，そこから出力された重みやパラメータを使用して LSI 設計を行った．

2.2.2　FPGA 構造

使用したボードは，DIGILENT 製の ZYNQ-7010 で

表 2: 学習条件

epoch	10000
eta	0.0005
Layer2(潜在空間の数)	16

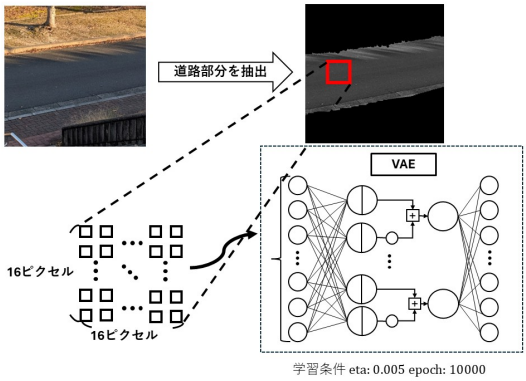


図 3: VAE 学習方法

ある．今回の設計では，図 4 に示すように，入力データとして，$X(16)$，$W(16\times2)$，$b(16)$，出力として $Z(16)$ を用意した（() 内は次元数）．赤色の枠で囲われているユニットは出力を $Output_1$ とすると，

$$Output_1 = X_1 \times W1_1 \tag{4}$$

のような，入力と重みのパラメータを乗算する演算を行っている．その演算ユニットを 16 個用意したものが，青色の枠線で囲われているユニットである．最終的な一つの Z の出力は，

$$Z_1 = \sum_{i=0}^{16} X_i \times W1_i + b1 \tag{5}$$

の計算を行っている．

当初は入力 256，出力 256 の演算を FPGA に載せたかったが，容量に限界があった．したがって，今回設計した $256\times16\times256$ の VAE を実現しやすい，X の入力が 16 になるように設計を行った．

2.2.3 SoC FPGA の構造

SoC FPGA のシステム構造の概要を図 5 にて示す．`processing_system7_0` がバスを通って様々な処理を行っおり，CPU の役割を担っている．また，`dut_forwa_ip_0` の部分が今回作成した FPGA の部分である．実装した後のリソース利用評価を表 3 に示す．

次に，SoC FPGA の処理の概要を図 6 にて示す．SD カードに格納されている VAE の学習重みや画像データを CPU が読み取る．その後，作成した FPGA に従って入力データの処理を行い，FPGA にそのデータを送信する．FPGA はデータが格納されると同時に実行し，出力結果を保存する．その出力結果を CPU が読み取

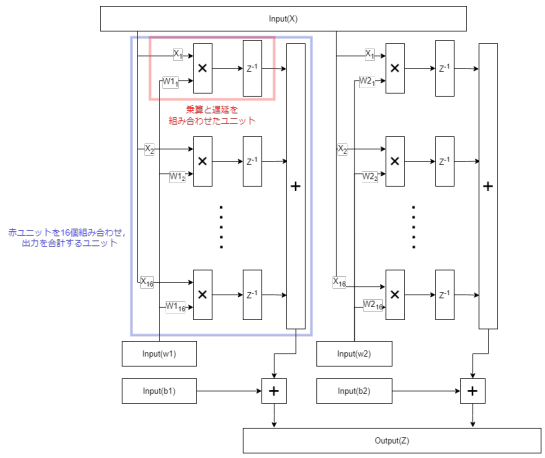


図 4: FPGA の構造

表 3: FPGA のリソース利用率

Resorce	Utilization	Available	Utilization[%]
LUT	3301	17600	18.76
LUTRAM	62	6000	1.03
FF	3297	35200	9.37
DSP	64	80	80.0
IO	12	100	12.0
BUFG	1	32	3.13

りに行き，出力データを処理する．それらの処理を繰り返し行う．

今回の VAE が $256\times16\times256$ であり，2.2.2 で作成した FPGA の構造が 16×2 である．設計した FPGA でエンコーダを計算する際は，図のような動作を行う必要がある．一つの出力で z^2_{mean} を，もう一つの出力で z^2_{var} を計算させる．1 つの潜在空間を計算させるためには，FPGA を 16 回使用する必要がある．その潜在空間が 16 個あるので，FPGA は合計で $16\times16=256$ 回稼働する．デコーダ部分を計算する際は，FPGA を一回稼働させるだけで，3 層目 Z^3_i の出力を 2 つ得られる．したがって，デコーダでは 128 回 FPGA を稼働させる．このような制御を CPU で作成した．

出力で得た z^2_{mean}，z^2_{var}，z^3 は，ソフトウェア上で活性関数を用いて a^2_{mean}，a^2_{var}，a^3 の計算を行う．さらに，a^2_{mean} と a^2_{var} を用いて z^2 の計算もソフトウェア上で行っている．最後に，入力画像と出力の a^3 の値を比較し，PSNR を計測を行っている．

2.3 実験方法

今回は，図 8 に示す 3 種類の画像を用意した．図 8a は，何も異常がない画像で，図 8b と図 8c は，道路上に異物があるのをイメージしたものである．画像のサ

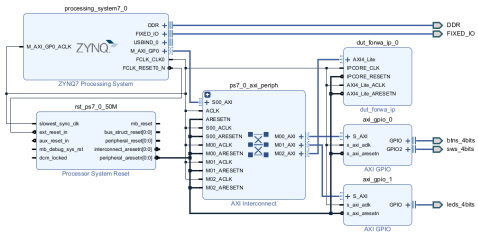


図 5: SoC FPGA の構成図

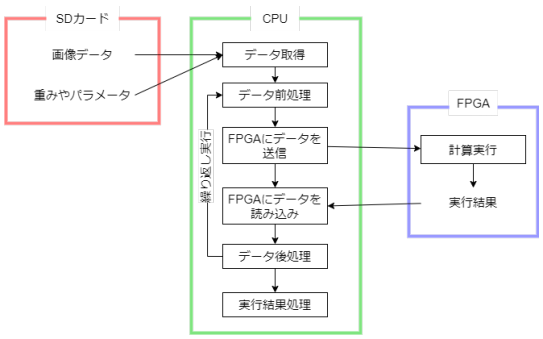


図 6: SoC FPGA の処理概要

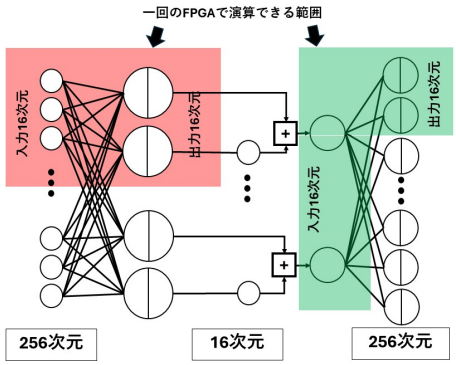


図 7: FPGA の利用イメージ

イズは 512×512 で，VAE に通す際は，画像をグレー画像にしている．16×16 のブロックで分割をしながら処理を行い，それぞれ PSNR を計測する．その PSNR が MATLAB の出力と FPGA を用いた出力でどの程度の差があるのかを比較する．また，実行時間に関しても可能な限り計測を行う．

3. 実験と考察

3.1 実験結果

まず，SoC FPGA の実行画面を図 9 に示す．ボタン 0 を押すことで SD カードに保存されている csv ファイルを読み込んでいる．ボタン 1 から 3 で画像を VAE に通過させる処理を行っている．図 9 はボタン 0 を押し，パラメータを取得後の状態である．

(a) もともとの画像 画像 1　(b) 異物設置した画像 画像 2　(c) 異物設置した画像 画像 3

図 8: 今回テストする評価画像

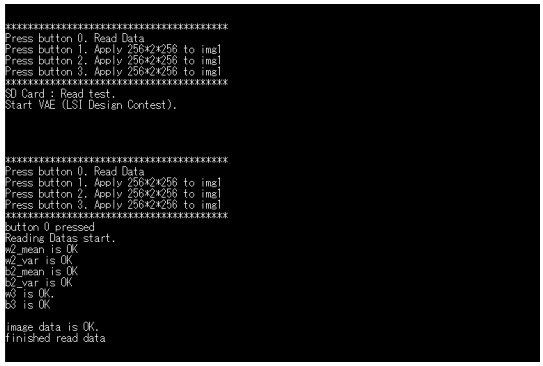
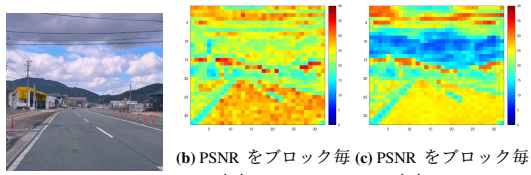


図 9: SoC FPGA の実行画面



(a) テストする画像 1　(b) PSNR をブロック毎に出力 MATLAB での出力　(c) PSNR をブロック毎に出力 MATLAB での出力

図 10: 今回テストする評価画像

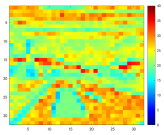
3.1.1 画像 1（図 8a）の出力比較

図 10b に MATLAB での出力，図 10c に FPGA での出力を示す．MATLAB では道路の部分付近が PSNR25 以上の値を示していることがわかるが，空や山のいち部分も PSNR が高くなっている．FPGA の出力では，MATLAB と比較して全体的に PSNR が低くなっていることがわかる．道路部分に関しては PSNR25 程度と，結果としては悪くない値を出している．

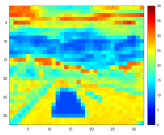
3.1.2 画像 2（図 8b）の出力比較

図 11b に MATLAB での出力，図 11c に FPGA での出力を示す．MATLAB．FPGA の出力どちらも異物が置かれている部分の PSNR が悪化していることから，

(a) テストする画像 2



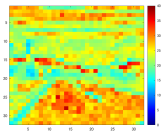
(b) PSNR をブロック毎に出力 MATLAB での出力



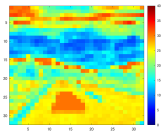
(c) PSNR をブロック毎に出力 FPGA での出力

図 11: 今回テストする評価画像

(a) テストする画像 3



(b) PSNR をブロック毎に出力 MATLAB での出力



(c) PSNR をブロック毎に出力 FPGA での出力

図 12: 今回テストする評価画像

異物を判定することができる．

3.1.3 画像 3（図 8c）の出力比較

図 12b に MATLAB での出力，図 12c に FPGA での出力を示す．画像 2 の出力と異なり，逆に異物がある部分の PSNR が向上している．

3.1.4 結果まとめ

それぞれの出力画像を比較して，MATLAB と FPGA の出力結果には違いが見られた．しかし，FPGA でも道路の部分が他の部分と比較して PSNR が高いため，失敗はしていないと考える．

また，異常検知に関しては，異常物体の色に影響を受けることが発見された．異常物体が白であると，正しく判定できるが，赤色では判定ができなかった．

3.1.5 エッジコンピューティングでの採用の評価

最後に，エッジコンピューティングとして採用できるかを評価する．

まずは，画像圧縮について評価する．今回は PSNR の閾値を 25 とし，閾値以上を圧縮可能，それ未満を圧縮不可能として判定を行う．閾値が 25 以上のブロック数，圧縮前の容量，圧縮後の容量，圧縮率を表に示す．容量の計算を (6) に示す．元画像が 0 から 255 の値を取ることから 1 ピクセル 8bit である．また，潜在空間を圧縮データを用いるため，8bit の固定小数点数を使用するとする．

$$size = B \cdot 16 \cdot 8[bit] + (32 \cdot 32 - B) \cdot 16 \cdot 16 \cdot 8[bit] \tag{6}$$

3.2 考　　察

第一に，MATLAB でシミュレーションした結果と，FPGA を用いて出力した値が異なっていること点である．これに関しては，使用変数を見直したり，固定小数点数を変更することで解決が可能化を検証していく．課題の 2 つ目としては，実行時間が長すぎることである．実行時間に関しては，評価ボードの制約やバスを用いた通信によって遅延が生じていると思われる．したがって，実際に産業応用を考える際は，より高性能な FPGA 等を用いることで解決を図ることが可能だと思う．ただ，VAE を FPGA を用いて実現し，画像圧縮，異常検知を行えることが示すことができたため，これからの研究・開発に活用できるのではないかと考える．

4. 結論と今後の展望

今回は，VAE を FPGA を上に実装し，道路画像の圧縮及び異常検知を行うシステムを構築した．その結果，FPGA を用いたエッジコンピューティングにおける画像処理が実現可能であることを示した．

本システムにはいくつかの課題があるものの，適切な改善により解決可能であると考える．したがって，将来的な産業応用の可能性も秘めていると思う．

今回のコンテストで FPGA を用いた LSI 開発を体験することができた．FPGA の将来性を感じたし，AI が日常的に使用されている現在に計算を効率よく行うために必要なデバイスであることを認知した．これからも FPGA の能力を発揮できるような開発を行っていきたいし，高位合成等の FPGA を開発するための技術も日々進化しているので，様々なことに挑戦していきたい．

謝辞　今回のコンテスト開催にあたり，主催して頂いた委員会の皆さん，協賛して頂いた企業・自治体の皆さん，本当にありがとうございました．また，研究室にも配属されていないのに，様々な支援をして頂いた先生方，ありがとうございました．

文　　献

[1] 森川博之, 5G 次世代移動通信規格の可能性, 岩波書店,
[2] 田中裕也, 高橋紀之, 河村龍太郎, ”IoT 時代を拓くエッジコンピューティングの研究開発”, NTT 技法ジャーナル, vol.27, no.8, pp.59-63, 2015.