顕著領域検出における固有値計算のハードウェア化

徳永 颯太[1,a)]　中武 叶人[1,b)]　久我 守弘[2,c)]

概要：顕著領域検出は画像処理における後段処理の計算コスト削減に有効である．また，CIELUV 色空間を用いた SBD 法では特徴ベクトルの高次元化により検出精度を向上させることができる．しかしながら検出処理で行う固有値計算がボトルネックのひとつとなっている．本研究では，固有値計算の FPGA によるハードウェア加速処理において，ハウスホルダー変換と Implicit QR 法を用いるとともに，パッチサイズを縮小し，次元削減による計算量の削減を行った．評価の結果，検出精度である AUC は 0.949 であり約 3.6%の低下に抑えるとともに，先行研究に対し固有値計算を 19.5 倍，システム全体で 1.76 倍の高速化を達成した．

キーワード：顕著領域検出，FPGA，固有値計算，高位合成，ハードウェアアクセラレータ

Hardware Implementation of Eigenvalue Calculation for Salient Region Detection

Sota Tokunaga[1,a)]　Kanato Nakatake[1,b)]　Morihiro Kuga[2,c)]

Abstract: Detecting salient regions is effective for reducing computational costs in subsequent image processing stages. Furthermore, the SBD method using the CIELUV color space can improve detection accuracy by increasing the dimensionality of feature vectors. However, the eigenvalue calculations performed during detection processing remain one of the bottlenecks. This study employs the Householder transformation and the Implicit QR method for hardware acceleration of eigenvalue computation using an FPGA. It also reduces computational load through dimension reduction by decreasing the patch size. Evaluation results show that the detection accuracy (AUC) is 0.949, limiting the decrease to approximately 3.6%. Compared to previous studies, this approach achieves an 8.3-fold acceleration in eigenvalue computation and a 1.7-fold acceleration in overall system processing speed.

Keywords: Saliency region detection, FPGA, Eigenvalue calculation, High-level synthesis, Hardware accelerator

1. はじめに

画像処理技術は医療診断や製造検査，自動運転など幅広い分野で活用されており，処理の高速化は重要な課題である．顕著領域検出（Saliency Detection）は，膨大な画像データから人間が注目する領域を自動抽出する技術であり，後段の認識処理の計算コスト削減に寄与する．

SBD 法（Saliency Detection with Spaces of Background-based Distribution）[1] は，画像境界のパッチを背景と仮定し，マハラノビス距離に基づいて顕著性を算出する手法である．先行研究 [2] では，人間の視覚特性に近い CIELUV 色空間を導入することで検出精度を向上させたが，特徴ベクトルが 441 次元（9 色× 7 × 7 画素）に増大し，共分散行列の固有値計算において $O(N^3)$ の計算量を要するため，処理全体の約 32%を占めるボトルネックとなっていた．

[1] 熊本大学大学院 自然科学教育部 情報電気工学専攻
Department of Computer Science and Electrical Engineering, Graduate School of Science and Technology, Kumamoto University
[2] 熊本大学大学院 先端科学研究部
Faculty of Advanced Science and Technology, Kumamoto University
[a)] s_tokunaga@st.cs.kumamoto-u.ac.jp
[b)] nakatake@st.cs.kumamoto-u.ac.jp
[c)] kuga@cs.kumamoto-u.ac.jp

本研究では，CIELUV 色空間を用いた SBD 法の検出精度を維持しつつ，FPGA（Field Programmable Gate Array）を用いたハードウェア化によって固有値計算を高速化する手法を提案する．提案手法の主要な貢献は以下の 2 点である．

第 1 に，画像の局所的な空間相関に着目し，パッチサイズを 7×7 画素（441 次元）から 5×5 画素（225 次元）へ縮小することで，検出精度を低下させることなく計算負荷を大幅に削減できることを明らかにした．

第 2 に，顕著領域検出において情報は分散の大きい上位の固有値に集約されるという知見に基づき，固有値計算の反復回数を制限する打ち切り計算（Early Termination）を FPGA ハードウェアに実装し，計算量の調整を可能とするアーキテクチャを提案した．

2. 関連研究と課題

2.1 先行研究における計算ボトルネック

先行研究 [2] では，CIELUV 色空間の導入により検出精度が向上したが，以下の課題が生じた．

- 特徴ベクトル：9 次元× 7 × 7 画素 = 441 次元
- 共分散行列：441 × 441 の密行列
- 固有値計算：$O(N^3)$ の計算複雑度

プロファイリングの結果（**表 1**），処理時間の内訳はマハラノビス距離計算 56%，固有値計算 32%となり，これら 2 つが主要なボトルネックである．

表 1 顕著領域検出処理時間割合
Table 1 Processing Time Distribution in Saliency Detection.

機能	割合 [%]
マハラノビス距離計算	56
固有値計算	32
パッチ作成	1
その他	11

本研究では，以下の理由から固有値計算をハードウェア化の対象とした．

(1) マハラノビス距離計算は固有値計算の結果を入力とするため，固有値計算の高速化が処理フロー全体に与える影響が大きい
(2) 固有値計算は $O(N^3)$ の反復的密行列演算であり，FPGA の並列処理能力を活かした高速化が効果的
(3) マハラノビス距離計算は画素ごとの独立した計算であり，将来的に GPU や SIMD 命令による並列化が容易

2.2 既存の FPGA 実装研究と本研究の位置づけ

FPGA を用いた固有値計算の高速化については，Bravo ら [3] が Jacobi 法の HLS（High Level Synthesis）実装を，Yan ら [4] が並列 Jacobi 法と CORDIC を組み合わせた手法を報告している．しかし，これらは数値計算としての精度維持を目的としており，アプリケーション特性に応じた近似計算は検討されていない．

本研究は，上記の既存研究とは異なり，顕著領域検出というアプリケーションの特性に着目した点に新規性がある．具体的には，SBD 法において検出精度に寄与する情報が上位の固有値に集約されるという知見に基づき，計算の早期打ち切りによる近似手法を提案する．既存の FPGA による固有値計算研究が数値的精度の維持を目的としているのに対し，本研究はアプリケーションの要求精度に応じた柔軟な計算量調整を可能とする．

対称行列の固有値計算に関する数値解析手法としては，QR 法 [5] や Jacobi 法 [6] が広く知られている．QR 法は安定性に優れるが，密行列への直接適用は計算コストが高い．本研究では，ハウスホルダー変換による前処理と Implicit QR 法の組み合わせにより，1 反復あたりの計算量を $O(N^2)$ に削減する．

3. 提案手法

3.1 システム全体構成

提案システムの全体構成を**図 1** に示す．Zynq-7000 SoC の PS 部（ARM Cortex-A9）で画像入力，色空間変換，パッチ分割，共分散行列生成を行い，PL 部（FPGA）で固有値計算を実行する．PS 部と PL 部は AXI4-Master（データ転送）および AXI4-Lite（制御信号）インタフェース [7] で接続される．

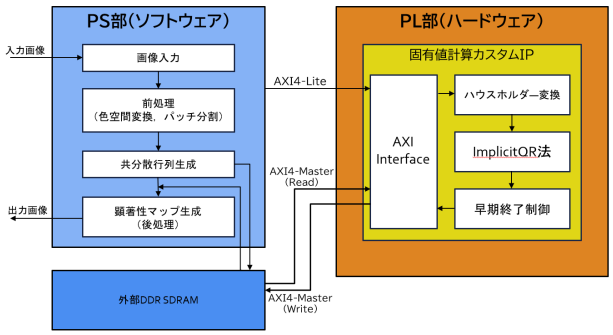

図 1 提案システムの全体構成
Fig. 1 Overview of the proposed system.

3.2 次元削減による計算負荷の低減

本研究で採用した Implicit QR 法の計算量は $O(N^2)$ で増加する．先行研究では，パッチサイズを 7×7 画素とし，各画素 9 次元の特徴量を用いていたため，特徴ベクトルの次元数は $N = 9 \times 7 \times 7 = 441$ であった．本研究では，パッチサイズ 5×5 の場合，次元数 $N_{5\times5}$ は式（1）のようになる．

$$N_{5\times5} = 9 \times 5 \times 5 = 225 \quad (1)$$

この場合，理論演算量は $(225/441)^3 \approx 0.133$ 倍（約 87%削減）となる．

さらに，パッチサイズ 4×4 の場合，次元数 $N_{4\times4}$ は式（2）のようになる．

$$N_{4\times4} = 9 \times 4 \times 4 = 144 \tag{2}$$

この場合，理論演算量は $(144/441)^3 \approx 0.035$ 倍（約 97%削減）という劇的な削減が見込まれる．

3.3 固有値計算ハードウェアの実装

本研究では，対称行列の固有値計算アルゴリズムとして，ハウスホルダー変換と Implicit QR 法を採用した．

3.3.1 アルゴリズム選定理由

固有値計算アルゴリズムとして，Jacobi 法，べき乗法，および QR 法のハードウェア化を検討した．

- Jacobi 法：並列性は高いが，高次元行列では収束が遅い
- べき乗法：単純だが，デフレーション時に誤差が累積
- QR 法：安定だが，密行列への直接適用は 1 反復 $\mathrm{O}(N^3)$

以上より，本研究では，QR 法の中で計算量の低い Implicit QR 法を採用した．

3.3.2 提案手法の 2 段階処理

計算負荷の高い行列演算を効率化するために，処理を以下の 2 段階に分割する．**Step 1: ハウスホルダー変換による前処理**

まず，密行列である共分散行列 C を，固有値を保存したままヘッセンベルグ行列（Hessenberg Matrix）T に変換する．

$$T = P^T C P \tag{3}$$

ここで，P は直交行列である．この前処理により，後続の反復計算のコストを大幅に引き下げることができる．

Step 2: Implicit QR 法による反復計算 Implicit QR 法の計算フローを**図 2** に示す．まず入力行列をヘッセンベルグ行列に変換し，ヘッセンベルグ行列 T に対して QR 法を適用する．ここで，通常の QR 分解を行うのではなく，Implicit QR 法（ウィルキンソンシフト付き）を用いる．通常の QR 法の場合，1 反復あたりの計算量は $O(N^3)$ となるが，Implicit QR の場合，ギブンス回転を用いた Bulge Chasing により，1 反復あたりの計算量は $O(N^2)$ まで削減される．

3.4 打ち切り計算の導入

3.4.1 顕著性と固有値の関係

SBD 法において，共分散行列の固有値計算は，背景分布の空間的な広がり（主成分）を評価するために用いられる．しかし，最終的な出力である顕著性マップに求められるのは，画素ごとの相対的な重要度（Saliency）であり，数学的に厳密な固有値の解そのものではない．画像統計学の観点からは，分散の大きい上位の固有値情報さえ概ね正しく抽出できていれば，下位の微小な固有値に多少の誤差が含まれていても，顕著領域の検出結果には大きな影響を与えないと考えられる．この知見に基づき，反復計算を途中で終了させる「打ち切り計算（Early Termination）」を導入した．

3.4.2 反復回数指定による計算打ち切り

本研究では，C++（Vitis HLS）コードレベルにおいて，ハウスホルダー変換および QR 法のループ回数に上限回数 k を設定し，指定回数に達した時点で無条件に処理を終了する制御方式を採用した．

- ハウスホルダー変換：$N-2$ 回 $\rightarrow k$ 回で打ち切り
- Implicit QR 法：収束まで $\rightarrow k$ 回で打ち切り

このアプローチには以下の利点がある．

- **処理時間の決定性:** 反復回数が固定されるため，画像の内容に依存せず，常に一定のクロックサイクル数で処理が完了する．
- **回路規模の削減:** 複雑な収束判定ロジックや分岐処理が不要となり，パイプライン化の効率が向上する．

3.5 高位合成による最適化

高位合成ツールである AMD 社の Vitis HLS[8] を用いて C++コードから回路を生成し，以下の最適化を適用した．

3.5.1 パイプライン最適化

固有値計算の主要ループに対して，以下のプラグマを適用した．

```
#pragma HLS PIPELINE II=1
```

これにより，ループの各反復が 1 クロックサイクルごとに開始され，スループットが最大化される．特に，行列要

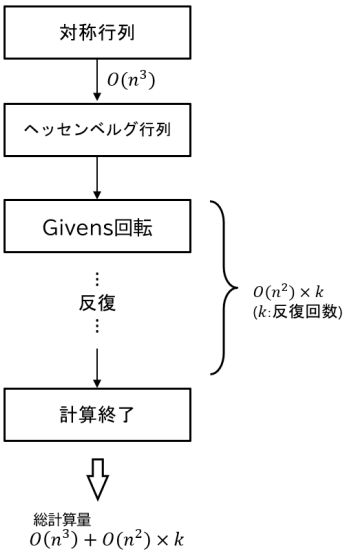

図 2 Implicit QR 法の計算フロー
Fig. 2 Computational flow of implicit QR method.

素の積和演算において，DSP ブロックを効率的に活用できる．

3.5.2 配列分割とメモリアクセス最適化

行列データを格納する配列に対して，以下の最適化を行った．

```
#pragma HLS ARRAY_PARTITION variable=matrix
            type=block factor=8 dim=2
```

これにより，行列の列方向に 8 つのメモリバンクに分割され，1 クロックサイクルで 8 つの要素に並列アクセスできる．ただし，BRAM 使用量が増加するため，行列サイズとのトレードオフを考慮する必要がある．

3.5.3 AXI インタフェースの設定

共分散行列データの転送には, AXI4-Master インタフェースを使用した．

```
#pragma HLS INTERFACE m_axi port=matrix
            offset=slave bundle=gmem
```

また，制御パラメータ（反復回数など）には，AXI4-Lite インタフェースを使用した．

```
#pragma HLS INTERFACE s_axilite port=num_iter
#pragma HLS INTERFACE s_axilite port=return
```

これにより，PS 部の Python スクリプトから，PL 部の IP コアを柔軟に制御できる．

4. 実験および評価

本章では，前章で提案した「パッチサイズ縮小による次元削減」および「打ち切り計算を導入した FPGA による固有値計算」の有効性を検証するために行った実装実験の結果について述べる．

4.1 実験

4.1.1 評価内容

実験は FPGA ボードを用いた実機評価により行い，以下の 4 つの観点から提案手法の性能を評価する．

(1) ハードウェアリソース使用量: 提案手法を論理合成した際のリソース消費量を確認し，ターゲットデバイスへの実装可能性およびリソース効率を評価する．

(2) 実行時間: CPU によるソフトウェア処理と FPGA によるハードウェア処理の実行時間を比較し，提案手法による高速化の効果を検証する．

(3) 検出精度: PR 曲線（Precision-Recall Curve）および AUC（Area Under the Curve）を指標として用い，次元削減と近似計算が検出精度に与える影響を定量的に評価する．

(4) 視覚的評価: 生成された顕著性マップを正解画像（Ground-Truth）と比較し，数値的な指標では捉えきれない画像品質や実用性を定性的に評価する．

4.1.2 実験環境

ハードウェアとして PYNQ-Z2（Zynq-7000 XC7Z020）ボードを使用した．ソフトウェア環境は Python 3.10.4, NumPy 1.21.5, PYNQ 3.0.1 である．開発環境は Vitis HLS 2023.2，Vivado 2023.2 を用い，データ型は float32，動作周波数 100MHz である．データセットは Jian Li 提供画像（480 × 640）[9] を使用した．

4.1.3 評価モデルの命名規則

本項では，提案手法の評価基準となるベースライン，およびハードウェア化を行わずに次元削減のみを適用した場合の効果を検証するために，以下の 3 つの CPU 実装モデルを定義する．これらは，アルゴリズムレベルでの計算量削減効果と，FPGA 実装による加速効果を分離して評価するための比較対象である．本論文では，以下の命名規則で各評価手法を表記する．

（1）CPU ベースライン

- **CPU 1**：先行研究 [2] の条件．パッチサイズ 7 × 7（441 次元），ARM Cortex-A9 上で NumPy による全固有値計算（225 個）
- **CPU 2**：次元削減のみ適用．パッチサイズ 5 × 5（225 次元），ARM Cortex-A9 上で NumPy による全固有値計算（225 個）
- **CPU 3**：次元削減のみ適用．パッチサイズ 4 × 4（144 次元），ARM Cortex-A9 上で NumPy による全固有値計算（144 個）

（2）FPGA での設計パラメータ

FPGA 実装においては，提案手法である「打ち切り計算（Early Termination）」の有効性を検証するため，算出する固有値の個数（反復回数）をパラメータとして変化させる．顕著領域検出において情報は分散の大きい上位の固有値に集約されるという仮説に基づき，算出数を段階的に減らすことで，計算精度（算出数）と処理速度，および検出精度のトレードオフを明らかにする．FPGA による実装では，「FPGA XX」の形式で表記する．ここで，XX は算出する固有値の個数を示す．

- **FPGA full**：全固有値を算出（打ち切りなし）．行列サイズ 225 × 225 の場合は 225 個，144 × 144 の場合は 144 個すべてを計算
- **FPGA 100**：上位 100 個の固有値のみ算出（行列サイズ 225 × 225 の場合のみ）
- **FPGA 70**：上位 70 個の固有値のみ算出（行列サイズ 144 × 144 の場合のみ）
- **FPGA 50**：上位 50 個の固有値のみ算出
- **FPGA 40**：上位 40 個の固有値のみ算出（行列サイズ 225 × 225 の場合のみ）
- **FPGA 30**：上位 30 個の固有値のみ算出
- **FPGA 20**：上位 20 個の固有値のみ算出
- **FPGA 10**：上位 10 個の固有値のみ算出
- **FPGA 5**：上位 5 個の固有値のみ算出（行列サイズ 144 × 144 の場合のみ）

4.2 評価

4.2.1 リソース使用量

表 2 および**表 3** にリソース使用量を示す．BRAM 使用率が 51.1%（225 × 225）と最も高く，配列分割による並列アクセス性能とのトレードオフを示している．144 × 144 では 28.2%に減少し，次元削減の効果が確認できる．DSP 使用率は 15.5%以下であり，拡張性に十分な余裕がある．

表 2 行列サイズ 225 × 225 のリソース使用量
Table 2 Resource Utilization for Matrix Size 225 × 225

リソース	使用量	使用可能量	使用率 [%]
LUT	12,552	53,200	23.6
FF	8,311	106,400	7.8
DSP	34	220	15.5
BRAM	143	280	51.1

表 3 行列サイズ 144 × 144 のリソース使用量
Table 3 Resource Utilization for Matrix Size 144 × 144

リソース	使用量	使用可能量	使用率 [%]
LUT	12,319	53,200	23.2
FF	7,778	106,400	7.3
DSP	27	220	12.3
BRAM	79	280	28.2

4.2.2 実行時間

本節では，提案手法による高速化の効果を検証するため，実行時間の計測結果について述べる． 計測項目は以下の 3 点である．

単体時間: 1 回の固有値計算関数（IP コアまたは NumPy 関数）の実行にかかる時間．

固有値計: 画像 1 枚の処理において行われる計 12 回の固有値計算の合計時間．

全体時間: 画像の読み込みから顕著性マップの出力までにかかるシステム全体の処理時間．

ARM プロセッサ（CPU）のみを用いて処理を行った場合の実行時間を**表 4** に示す．表 4 よりパッチサイズの縮小による次元削減のみで，固有値計算の実行時間は 5 × 5 では 4.9 倍，4 × 4 では 13.7 倍の高速化という結果となった．次に，FPGA 実装の実行時間を**表 5** および**表 6** に示す．

(1) 225 × 225 の場合

表 7 に先行研究（CPU 1: 7 × 7）との比較を示す．

表 4 CPU における固有値計算の実行時間
Table 4 Execution time for eigenvalue calculation on the CPU

手法	パッチサイズ	単体時間 [s]	固有値計 [s]	全体時間 [s]
CPU 1	7 × 7	1.78	21.4	65.6
CPU 2	5 × 5	0.36	4.34	40.0
CPU 3	4 × 4	0.13	1.59	21.3

表 5 行列サイズ 225 × 225 における固有値計算の実行時間
Table 5 Execution Time of Eigenvalue Calculation for Matrix Size 225 × 225

手法	算出数	単体時間 [s]	固有値計 [s]	全体時間 [s]
FPGA full	225	0.53	6.35	45.1
FPGA 100	100	0.47	5.66	41.7
FPGA 50	50	0.32	3.80	39.8
FPGA 40	40	0.27	3.22	39.2
FPGA 30	30	0.21	2.57	38.7
FPGA 20	20	0.16	1.89	37.7
FPGA 10	10	0.09	1.10	37.3

表 6 行列サイズ 144 × 144 における固有値計算の実行時間
Table 6 Execution Time of Eigenvalue Calculation for Matrix Size 144 × 144

手法	算出数	単体時間 [s]	固有値計 [s]	全体時間 [s]
FPGA full	144	0.15	1.77	21.7
FPGA 70	70	0.14	1.70	21.6
FPGA 50	50	0.12	1.48	21.3
FPGA 30	30	0.09	1.11	21.1
FPGA 20	20	0.08	0.91	20.8
FPGA 10	10	0.04	0.53	20.5
FPGA 5	5	0.04	0.52	19.8

表 7 CPU 1 と提案手法の実行時間比較（225 × 225）
Table 7 Execution Time Comparison between CPU 1 and Proposed Method (225 × 225)

手法	固有値計算 高速化率	全体時間 高速化率
FPGA full	3.36	1.46
FPGA 100	3.77	1.57
FPGA 50	5.62	1.65
FPGA 40	6.63	1.67
FPGA 30	8.32	1.69
FPGA 20	11.3	1.74
FPGA 10	19.5	1.76
CPU 1: 7 × 7 パッチ， 固有値計算 21.4s， 全体 65.6s		

- FPGA full：固有値計算 3.36 倍，全体 1.46 倍
- FPGA 30：固有値計算 8.32 倍，全体 1.69 倍
- FPGA 10：固有値計算 19.5 倍，全体 1.76 倍

FPGA full の段階でパッチサイズの縮小効果により，3.36 倍高速化し，そこから固有値の算出数を絞るとさらに高速化し，最終的に FPGA 10 では，固有値計算 19.5 倍，顕著領域検出全体 1.76 倍の高速化という結果となった．

(2) 144 × 144 の場合

表 8 に先行研究との比較を示す．

- FPGA full：固有値計算 12.1 倍，全体 3.02 倍
- FPGA 30：固有値計算 19.2 倍，全体 3.11 倍
- FPGA 5：固有値計算 41.2 倍，全体 3.32 倍

行列サイズ 144 × 144 のときも 225 × 225 のときと同

表 8　CPU 1 と提案手法の実行時間比較（144 × 144）
Table 8　Execution Time Comparison between CPU 1 and Proposed Method (144 × 144)

手法	固有値計算 高速化率	全体時間 高速化率
FPGA full	12.1	3.02
FPGA 70	12.6	3.04
FPGA 50	14.5	3.08
FPGA 30	19.2	3.11
FPGA 20	23.5	3.15
FPGA 10	40.3	3.19
FPGA 5	41.2	3.32
CPU 1: 7 × 7 パッチ， 固有値計算 21.4s， 全体 65.6s		

様に，FPGA full の段階で更なるパッチサイズの縮小効果により，12.1 倍高速化し，そこから固有値の算出数を絞るとさらに高速化し，最終的に FPGA 5 では，固有値計算 41.2 倍，顕著領域検出全体 3.32 倍の高速化という結果となった．

4.2.3　検出精度評価

評価指標として，PR 曲線および AUC を採用した．PR 曲線は横軸に再現率，縦軸に適合率をとったものであり，曲線が右上に位置するほど検出精度が高いことを示し，その曲線下面積である AUC が 1 に近いほど，検出精度が高いことを示す．**図 3** および**図 4** に PR 曲線を示す．

（1）行列サイズ 225 × 225 における評価

パッチサイズを 5 × 5 とし，行列サイズを 225 × 225 とした場合，全ての固有値を計算する FPGA full の AUC は 0.942 であり，CPU1 での浮動小数点演算結果と比較するとわずかな低下が見られる．しかしながら，打ち切り計算を適用した場合，FPGA 100 から FPGA 10 までのすべての条件において，AUC は 0.946〜0.949 の範囲で安定しており，FPGA full よりもむしろ良好な数値を示している．特に，計算時間を大幅に短縮可能な FPGA 10（上位 10 個のみ算出）においても，AUC は 0.949 という高い値を維持している．この結果は，全固有値の約 4.4%のみを算出するという極端な近似計算であっても，実用上十分な検出精度が得られることを示している．

（2）行列サイズ 144 × 144 における評価

パッチサイズを 4 × 4 とし，行列サイズを 144 × 144 まで縮小した場合，FPGA 実装において，FPGA full の AUC が 0.918 であるのに対し，打ち切りを行った FPGA 70 から FPGA 5 までの手法では AUC が 0.930〜0.934 となった．この結果は，行列サイズ 225 × 225 の場合と同様に，打ち切り計算を適用した方がより高い検出精度を達成できることを示している．

4.2.4　視覚的評価

図 5〜**図 10** に 225 × 225 における視覚的評価結果を示す．FPGA 10（上位約 4.4%の固有値のみ算出）においても，顕著領域であるボートの外郭は明瞭に抽出されており，正解画像と比較して主要な形状は正しく捉えられている．極端な近似計算でも実用上十分な品質が得られることを視覚的に確認できる．

5.　考察

5.1　次元削減と精度のトレードオフ

5 × 5 画素では検出精度が維持された（AUC 0.985）．これは，画像の局所領域における空間的冗長性により，5 × 5 の範囲で背景と顕著領域を識別する十分な情報が含まれることを示唆している．

一方，4 × 4 画素では AUC が 0.979 に低下した．これはパッチサイズが小さすぎることで，物体の境界やテクスチャといった空間構造の情報が失われたためと考えられる．SBD 法では，境界付近のパッチを背景と仮定して分布を推定するため，ある程度の空間的広がりを持つ領域の情報が必要である．

したがって，本実験条件（画像サイズ 640 × 480）においては，5 × 5 パッチサイズが最適なトレードオフ点であ

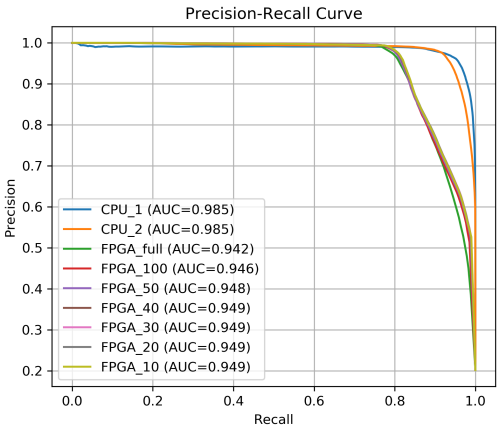

図 3　行列サイズ 225 × 225 における PR 曲線
Fig. 3　PR Curves for Matrix Size 225 × 225.

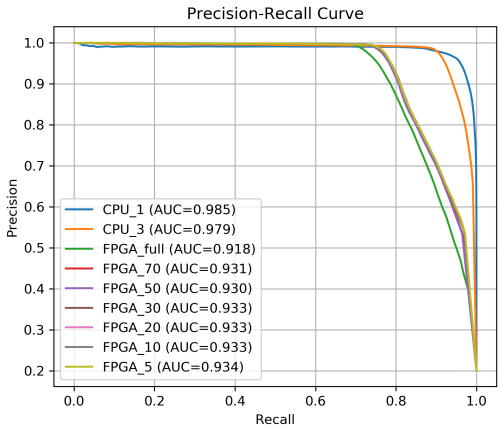

図 4　行列サイズ 144 × 144 における PR 曲線
Fig. 4　PR Curves for Matrix Size 144 × 144.

図 5 入力画像
Fig. 5 Input Image.

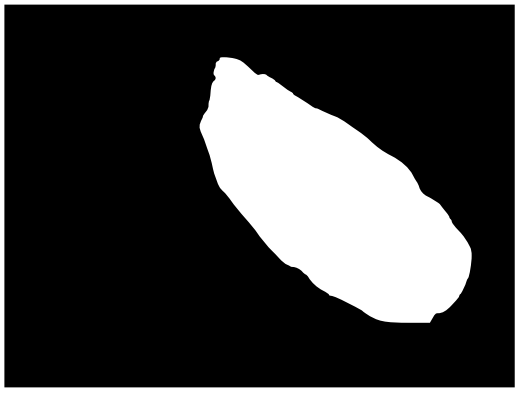

図 6 正解画像
Fig. 6 Ground-Truth Image.

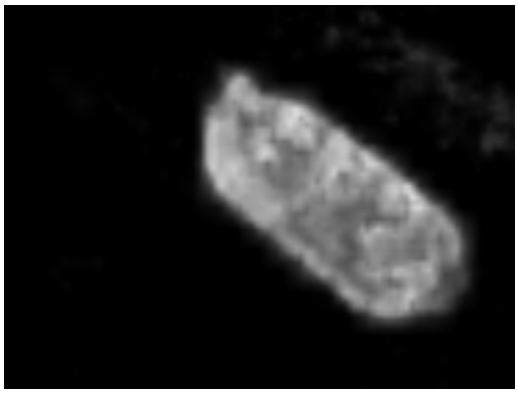

図 7 CPU 1 の出力画像
Fig. 7 Output Image of CPU 1.

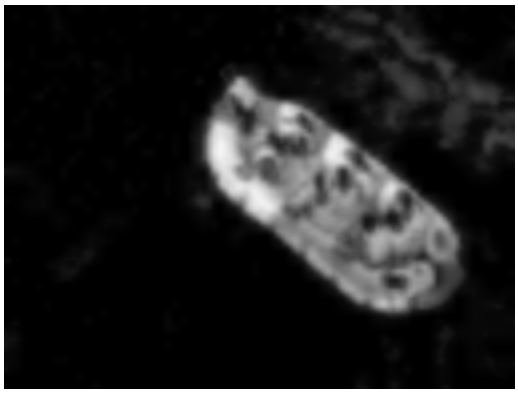

図 8 FPGA full の出力画像
Fig. 8 Output Image of FPGA full.

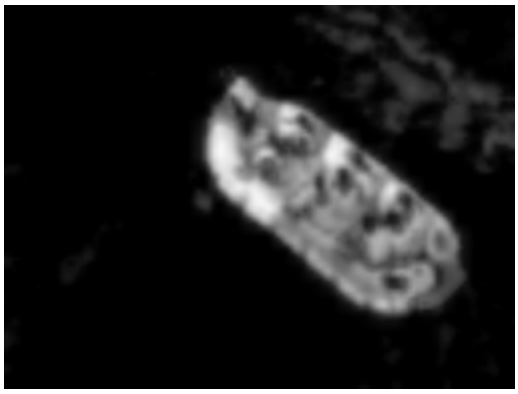

図 9 FPGA 40 の出力画像
Fig. 9 Output Image of FPGA 40.

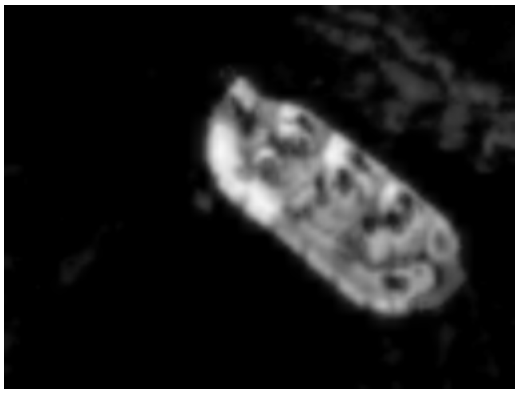

図 10 FPGA 10 の出力画像
Fig. 10 Output Image of FPGA 10.

るといえる．ただし，より高解像度の画像や，微細なテクスチャを持つ対象を扱う場合には，パッチサイズの再検討が必要となる可能性がある．

5.2 打ち切り計算とノイズ除去効果

上位固有値はデータの主要構造（信号）を表し，下位固有値は微小な変動やノイズを表す．打ち切り計算は上位固有値のみを選択的に抽出する操作と等価であり，結果として一種のノイズフィルタリングとして機能した．

この現象は，主成分分析におけるノイズ除去手法 [10] と本質的に同じメカニズムである．共分散行列の固有値分解は，データの分散を最大化する方向（主成分）を見つける操作であり，分散の小さい成分は「重要でない情報」または「ノイズ」として解釈できる．これらのノイズは空間的に相関が低いため，固有値分解において下位の固有値に対応することが多い．したがって，下位固有値を切り捨てることは，自動的にノイズ成分の除去につながる．

実験結果において，FPGA 30 や FPGA 10 が FPGA full よりも高い AUC を達成したことは，この理論的予測と一致している．全体の 5〜13%の上位固有値で十分な精度が

得られたことは，顕著性情報が非常に低次元の部分空間に集約されることを示している．

ただし，打ち切りを過度に強くすると（例：FPGA 5 で 144 次元中 5 個のみ），必要な情報まで失われる可能性がある．実際，表 8 において，FPGA 5 の AUC は 0.934 であり，FPGA 30 の 0.934 と同等であるが，これ以上打ち切りを強めると精度低下が予想される．したがって，アプリケーションの要求精度に応じて，適切な打ち切りレベルを選択することが重要である．

5.3 ソフトウェア・ハードウェア協調設計の有効性

FPGA full（全固有値算出）の結果は CPU2，CPU3 による実行時間よりも劣る場合があった. これは，FPGA（PL 部）の動作周波数が 100 MHz であり，CPU（PS 部，650 MHz）と比較して低速であることや，メモリ転送のオーバーヘッドが影響しているためである. しかし，アルゴリズムレベルでの工夫（打ち切り計算）を導入した FPGA 30 や FPGA 10 においては，CPU を大きく上回る処理速度を達成した. これは，単に既存のアルゴリズムをそのままハードウェアに置き換えるだけでは性能向上に限界があることを示している. 本研究のように，ソフトウェア処理で柔軟な次元削減を行い，ハードウェア処理で近似計算による高速化を行うという協調設計のアプローチをとることで初めて，リソースの制約がある組込み FPGA 上でも，精度を犠牲にすることなく実用的な高速化が実現可能であることが確認できた.

本研究では，以下の 3 層の最適化を行った.

(1) **アルゴリズムレベル**：パッチサイズ縮小による次元削減
(2) **数値計算レベル**：Implicit QR 法による反復あたり計算量削減
(3) **ハードウェアレベル**：打ち切り計算とパイプライン化

これらの最適化が相乗効果を発揮し，最大 41 倍という劇的な高速化を実現した.

さらに，本手法はエッジデバイス向けの設計として，以下の利点を持つ.

- 低消費電力（100MHz 動作，DSP 使用率 15%）
- 決定的な実行時間（リアルタイム処理に適合）
- 柔軟なパラメータ調整（精度と速度のトレードオフ）

これらの特性は，製造ラインなどの実用アプリケーションにおいて重要である.

5.4 今後の展望

本研究では，処理速度と検出精度のトレードオフについて一定の成果を得たが，実用化に向けてはいくつかの課題が残されている． 第 1 に，リアルタイム処理の実現である．今回は Zynq-7000 という小規模な FPGA を用いたが，より大規模なデバイスを用いて並列実行数を増やすことや，現在ソフトウェア（PS 部）で行っている処理をハードウェア化することで，さらなる高速化が期待できる． 第 2 に，消費電力の評価である．エッジデバイスへの実装においては，処理速度だけでなく消費電力も重要な指標となる．今後は電力解析を行い，エネルギー消費の観点からも評価を行う必要がある． 第 3 に，適応的な打ち切り制御の導入である．入力画像の複雑さに応じて動的に打ち切り回数を制御することで，精度と速度のバランスをより最適化できる可能性がある．

6. おわりに

本研究では，CIELUV 色空間を用いた顕著領域検出の高速化を目指し，次元削減と固有値計算の打ち切りを導入した FPGA アクセラレータを開発した．評価の結果，実用的な精度（AUC 0.949）を維持しつつ，固有値計算を最大約 19.5 倍，システム全体を約 1.76 倍高速化することに成功した．これにより，エッジデバイスにおける提案手法の有効性を示した.

参考文献

[1] Tong Zhao, Lin Li, Xinghao Ding, Yue Huang and Delu Zeng, “Saliency Detection with Spaces of Background-based Distribution,” IEEE Signal Processing Letters in March 2016. DOI: 10.1109/LSP.2016.2544781
[2] Hashimoto, R. and Kuga, M.: A Case Study Implementation of Saliency Detection Based on SBD Method -Improving the Detection Accuracy CIELUV Color Space-, Joint Conf. of Electrical, Electronics and Information Engineers in Kyusyu, 2022.
[3] Bravo, I., Vazquez, C., Gardel, A., Lazaro, J.L. and Palomar, E.: High level synthesis FPGA implementation of the Jacobi algorithm to solve the eigen problem, Math. Probl. Eng., Vol.2015, Article ID 870569, pp.1–11, 2015.
[4] Yan, D., Wang, W.X. and Zhang, X.W.: High-performance matrix eigenvalue decomposition using the parallel Jacobi algorithm on FPGA, Circuits Syst. Signal Process., Vol.42, No.3, pp.1573–1592, Mar. 2023.
[5] 森正武：数値解析 第 2 版，共立出版, 2002.
[6] 櫻井鉄也：固有値問題の数値計算，朝倉書店, 2015.
[7] ARM: AMBA AXI and ACE Protocol Specification, ARM IHI 0022E, 2011.
[8] AMD Xilinx: Vitis High-Level Synthesis User Guide (UG1399), available from ⟨https://docs.xilinx.com⟩ (accessed 2026-02-13).
[9] Jian Li, Martin Levine, Xiangjing An, Hangen He, “Saliency Detection Based on Frequency and Spatial Domain Analysis,” BMVC 2011, The 22nd British Machine Vision Conference, University of Dundee, 2011.
[10] Jolliffe, I.T. and Cadima, J.: Principal component analysis: a review and recent developments, Philosophical Transactions of the Royal Society A, Vol.374, No.2065, 20150202, 2016.