

卒業論文

題目

ニューラルネットワークを用いた手術画像
セグメンテーションシステムのFPGA実装

指導教員

柴田裕一郎 教授

2019年度

長崎大学工学部工学科 情報工学コース

藤田光暉 (35316034)

論文内容の要旨

情報工学コース

履修番号	35316034	氏名	藤田光暉
研究室名	柴田研究室		
研究題名	ニューラルネットワークを用いた手術画像 セグメンテーションシステムの FPGA 実装		

論文内容の要旨

あぶすと

目 次

第 1 章 緒論	1
第 2 章 背景と目的	2
2.1 関連研究	2
2.2 プロジェクトの現状	3
2.3 研究目的	4
第 3 章 理論	5
3.1 ニューラルネットワーク	5
3.2 畳み込みニューラルネットワーク	7
3.3 セマンティックセグメンテーション	8
3.4 FPGA	9
第 4 章 設計と実装	10
4.1 アルゴリズム	10
4.1.1 ネットワーク構成	10
4.1.2 畳み込み層	11
4.1.3 pooling 層	12
4.1.4 unpooling 層	13
4.2 設計と実装	13
4.2.1 システム構成	14
4.2.2 stream_patch モジュール	15
4.2.3 ExtNet・RdcNet・ItgNet モジュール	16
4.2.4 pooling モジュール	18
4.2.5 unpooling モジュール	18
4.2.6 buf モジュール	20
4.3 学習	21
4.3.1 ツール	21
4.3.2 学習データ	21
4.3.3 量子化手法	22
第 5 章 評価と考察	23
5.1 評価	23
5.1.1 資源使用量	23
5.1.2 最大動作周波数	24

5.1.3	レイテンシ	24
5.2	考察	25
5.2.1	現在の実装に対する評価について	25
5.2.2	1台のFPGAによる実装に向けて	26
第6章 結論		28

図目次

2.1 U-Net ネットワーク構造 : [6] より引用	3
3.1 各ニューロンにおけるプロセスの模式図 ($n = 2$)	5
3.2 ニューラルネットワークの例	6
3.3 畳み込み層：畳み込み演算を「*」で表記	7
3.4 pooling (max pooling)	8
3.5 セグメンテーションの例 : [9] より引用	8
3.6 unpooling	9
4.1 ネットワーク全体図	10
4.2 3種の小規模ネットワーク	11
4.3 パディングの有無によるサイズ変化の違い	12
4.4 pooling 層の適用による縮小	13
4.5 unpooling 層の適用による拡大	13
4.6 システム構成概略図	14
4.7 LEVEL ごとの有効画素変化	14
4.8 パッチ切り出し	15
4.9 layer モジュールによるネットワーク作成イメージ	16
4.10 加算器ツリーによる畳み込み演算	17
4.11 ツリーによる pooling	18
4.12 LEVEL による unpooling モジュール動作イメージの変化	19
4.13 buf モジュール	21
4.14 実際の手術画像データ	21
4.15 教師データ	22
5.1 フィルタ適用に要するレイテンシの概算 (行)	25
5.2 シミュレーションによるレイテンシ解析	25
5.3 buf モジュールに必要な FIFO サイズ	26

表目次

4.1	出力座標値と LEVEL による有効画素判定 (layer モジュール)	17
4.2	出力座標値と LEVEL による有効画素判定 (pooling モジュール)	18
4.3	拡張方向選択方法 : LEVEL0	20
4.4	拡張方向選択方法 : LEVEL ≠0	20
5.1	資源使用量	23
5.2	動作周波数と fps	24
5.3	各 Net モジュールにおける資源使用量	26

第1章

緒論

近年、医療現場において腹腔鏡手術の需要が高まっている[1]。腹部に小さく開けた穴から内視鏡を挿入し対象部位の外科手術を行う腹腔鏡手術は、開腹手術に比べ出血量や術後の回復期間¹などにおいて患者への負担が少ない。しかし、腹腔鏡手術には通常の執刀医に加え、内視鏡操作を行う技師が必要となる。患者数の増加や医師の地域偏在に起因する医療従事者不足が医療業界の深刻な問題となっているが、内視鏡技師もその例外ではなく、拡大する腹腔鏡手術需要に対応しきれていないのが現状である。

一方で、医療分野におけるロボットによる支援の進歩は目覚ましく、その活躍は神経外科、腹腔外科、胸部外科など多岐に渡る。既に実績を残しているロボットとしては内視鏡下手術支援ロボットである「da Vinci(ダビンチ)[3]」が挙げられ、2018年にはダビンチを使って行われた手術が約100万件に達した。このような医療用ロボットは年々開発が進められている。

これらの状況を踏まえ、長崎大学工学部、長崎大学病院、中央大学工学部の共同研究として、胆囊摘出手術を想定した内視鏡操作ロボットの開発が行われている。このロボットは内視鏡のカメラワークを自動で調節し、執刀医のみによる腹腔鏡手術を可能にすることを目標としており、この実現によって拡大する手術需要に対応することが期待される。なお、胆囊摘出手術は腹腔鏡手術による施術が特に増加している手術の一種であり²、ダビンチの分野毎における利用数の推移[3]からもその需要の高まりが窺える。

胆囊周辺の特定部位へとカメラ画角を調整するには、前段階として内視鏡画面内の胆囊周辺部位座標を取得する必要がある。そこで本プロジェクトでは、畳み込みニューラルネットワークを用いた推論によってセグメンテーションを行い、リアルタイムでの判別が可能なシステムの設計を行っている[4]。本研究では、今後行われるであろうシステムの機能拡張に際してリアルタイム性を維持できるようにするために、ハードウェア化による高速化手法を提案し、既に実装されているシステムと、正答率・処理速度の観点から比較・評価を行う。

本論文の構成は以下の通りである。まず第2章において、本研究の背景と目的を述べる。第3章では、ニューラルネットワークを始めとした、本研究におけるシステムの基となる理論について説明を行う。第4章では、構築されるネットワークの特徴と、ハードウェアに実装するための手法について述べる。続く第5章では、FPGA上に実装されたシステムの評価・考察を行い、最後に第6章にて、本研究における結論を述べる。

¹良好ならば術後4日目に退院、仕事復帰は約1週間[2]

²29の病院を対象とした調査では胆囊摘出における腹腔鏡手術の割合は平均で93.3%[1]

第2章

背景と目的

内視鏡操作をロボットで行うにあたり、画角調節のため対象物体である胆嚢周辺部位の判別が必要となる。本プロジェクトでは、ニューラルネットワークを用いたセグメンテーションシステムによってこれを実現する。しかし、一般的にニューラルネットワークは学習に多量のデータセットを必要とするのに対し、医用画像はその特殊性¹からデータセットが少ないという問題点がある。

本章では、このような問題を抱える医用画像解析の分野において、機械学習を用いて一定の成果を納めた研究事例を挙げるとともに、実装済みのシステムとプロジェクトの現状、それらを踏まえた本論文における研究目的について述べる。

2.1 関連研究

U-Net[6] は、医用画像セグメンテーション用として提案され、2015 年の ISBI (IEEE International Symposium on Biomedical Imaging) で Dental X-Ray Image Segmentation Challenge と Cell Tracking Challenge の 2 部門で優勝するなど、高い評価を得ているセマンティックセグメンテーション手法である。画像のダウンサンプリング (低解像度化) に対して、アップサンプリング (高解像度化) を行うのは他のセグメンテーション手法でも見られるが、U-Net の特徴はアップサンプリング時にダウンサンプリング前のデータを連結することにある。この連結はスキップ接続と呼ばれ、データが伝搬する過程で詳細な情報が失われてしまい、セグメンテーション結果が粗くなる (粗大化する) のを防ぐ効果を持つ。図 2.1 にネットワーク構造を示す。ただし、U-Net はネットワークサイズが大きいことから、少ないデータセットでは訓練データを丸暗記してしまう過学習を引き起こす可能性が高い。

国立研究開発法人国立がん研究センターと日本電気株式会社は、深層学習を用い、大腸がん及び前がん病変を内視鏡検査時にリアルタイムに発見するシステム [7] の開発に成功した。このシステムは、内視鏡医師によってアノテーション (教師データとなるようにラベリング) された約 5000 例の内視鏡画像を教師データとして学習が行われる。約 5000 例という値は一般的な深層学習の教師データとしては少ないながらも、98 % という高い認識率と約 30fps (frames per second) の高速処理を実現している。

¹情報セキュリティやデータ提供に関する負担などの観点からガイドライン [5] に沿った適切な取得・運用が求められることに加え、教師データの作成は専門の医師による手作業で行われることが多い。

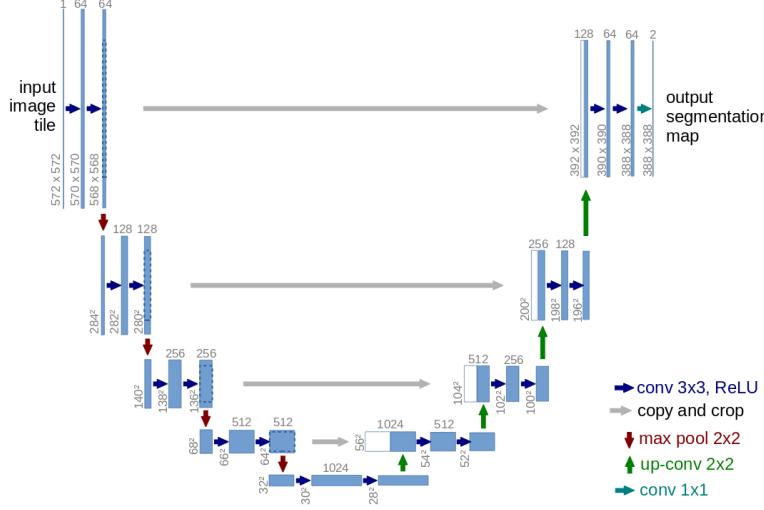


図 2.1: U-Net ネットワーク構造 : [6] より引用

2.2 プロジェクトの現状

本プロジェクトでは、GPU によるニューラルネットワークを用いたセグメンテーションシステムが実装されており、評価用データを用いた実験では正答率 61.2 %、約 17fps の性能を達成した [4]。

このシステムの検証実験としてブタの生体を用いた動作テストを行った。ブタは形態学的、生理学的にヒトとの類似性が高いことが知られており、医学、免疫学、再生医療などの分野において広範囲に利用されている。検証実験の結果、セグメンテーションシステム自体の精度向上の他に、以下のような機能の必要性が見出された。

- ブタとヒトの差異にシステムを対応させるための前処理
- 手術補助としてのセグメンテーション結果のオーバーレイ

ブタはヒトとの類似性が比較的高いものの、臓器表面の色彩やテクスチャにおいて多少の差異が存在するため、ヒトのデータセットによって学習された当システムの検証対象としては適切ではない可能性がある。しかしながら、新たにブタのデータセットを十分に用意するのは難しく、たとえ用意できたとしてもそれはブタに適応したシステムとなってしまう。そこで、カメラからの動画像にブタの臓器画像をヒトのそれに近づける前処理を追加する予定である。これにより、ブタを用いた検証実験によって当システムのヒトへの有効性を正しく評価することを狙う。後者は執刀医に対する手術補助を意図して、内視鏡映像にセグメンテーション結果をオーバーレイするものである。

このような操作が現在のシステムに追加された場合、その分の処理時間が増加し、現在の実装ではリアルタイム性が失われる可能性がある。そこで、実装されているセグメンテーションシステムをより高速にし、かつ処理の追加にも適した実装とすることで、当システムのリアルタイム性を維持できるようにする必要がある。

2.3 研究目的

本研究は、現在 GPU を用いて実装されている胆嚢周辺画像のセマンティックセグメンテーションの高速化手法として、FPGA によるシステムのハードウェア化を提案し、それによる演算速度の変化を検証することを目的とする。

畳み込みニューラルネットワークは、推論、学習ともに並列性の高い大量の積和演算を行うことから、一般的には GPU による処理が行われることが多い。しかしながら、動画像を対象とした畳み込み処理においては、単純な並列化に加えてパイプライン化による恩恵が大きく、計算分野によっては、FPGA 等によるハードウェア処理のほうが GPU よりも高速に動作することが知られている [8]。処理の追加に関しても、パイプラインで処理できるものならば、レイテンシの増加のみでスループットを維持した自然な実装が可能であり、画像処理の追加が予定される当システムに適した手法であるといえる。

また、FPGA は電力効率の観点から組み込み系機器での運用に適しており、持ち運びのできる組み込み系機器での運用が想定される当システムにおいては大きな利点となる。そこで本研究では、広範な機器で利用でき、電力効率にも優れたシステムを構築可能でありながら、より高速な動作が見込める FPGA を利用する。

第3章

理論

本章では、システムの実装に用いる理論についての説明を行う。

3.1 ニューラルネットワーク

ニューラルネットワーク (Neural Network) は、生物の脳内におけるニューロン (神経細胞) の結びつき方をモデルにした情報処理システムである。学習能力を持つため、サンプルとなるデータに基づき必要とされる機能を自動形成することができる。

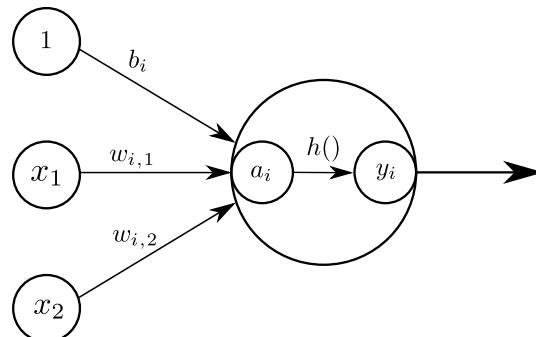


図 3.1: 各ニューロンにおけるプロセスの模式図 ($n = 2$)

ニューラルネットワークを構成するニューロンの模式図を、前段のニューロン数 n が 2 のときを例として図 3.1 に示す。ニューロン i の出力 y_i は、入力 x_j ($1 \leq j \leq n$) を基に、3.1 式により決定される。

$$y_i = h(b_i + \sum_{j=1}^n x_j w_{i,j}) \quad (3.1)$$

各入力 x_j には固有の値である重み $w_{i,j}$ が設定され、同じく固有の値であるバイアス b_i と共に計算に用いられる。この固有の値がニューラルネットワークの機能を決定づける要素であり、適切な値に設定することで必要とする機能を形成する。また、 h は活性化関数と呼ばれる関数であり、以下のような非線形関数が用いられることが多い。

$$h(x) = (1 + e^{-x})^{-1} \quad (\text{標準シグモイド関数}) \quad (3.2)$$

$$h(x) = \max(0, x) \quad (\text{ReLU}) \quad (3.3)$$

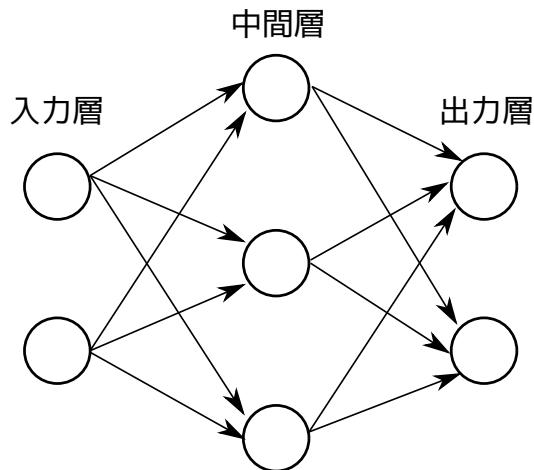


図 3.2: ニューラルネットワークの例

ニューラルネットワークの代表的な構造として、順伝播型ニューラルネットワークの1種である多層パーセプトロンを例に挙げる。中間層が1層以上存在する多層パーセプトロンでは、任意の連続関数を近似可能であることが知られている。ただし、線形な関数のみで構成されるネットワークは、どんなに層を増やしたとしてもそれと等価な单層ネットワークが存在するため、層を増やす恩恵を得るには非線形な関数が各層で用いられる必要がある。そのため、一般的に活性化関数には非線形関数が用いられている。

先述した通り、ニューラルネットワークで期待される機能を実現するには、各ニューロンの重みとバイアスを適切に設定しなければならない。この調整をデータに基づいて自動で行う仕組みが学習である。学習には大きく分けて教師あり学習と教師なし学習があるが、ここでは教師あり学習について説明を行う。

教師あり学習では、入力データと教師データが対になって与えられたデータセットを利用し、ネットワークに入力データを渡した際の出力が教師データと一致するように重みやバイアスを変化させていく。一般的に、出力精度の指標としては2乗和誤差や交差エントロピー誤差などの損失関数が、学習手法としては誤差逆伝播法が用いられる。入力データに基づく出力を y_k 、それと対応する教師データを t_k として以下に損失関数の例を示す。

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2 \quad (\text{2乗和誤差}) \quad (3.4)$$

$$E = - \sum_k t_k \log y_k \quad (\text{交差エントロピー誤差}) \quad (3.5)$$

損失関数とは、出力精度の低さを示す指標であり、これによって得られた誤差を最小とするように重みおよびバイアスを変更する。また、重みに基づいて誤差を逆伝播させ、中間層の重みおよびバイアスを順次更新していく。損失関数は逆伝播可能な関数ではなくてはならず、活性化関数との組み合わせによっては学習が遅くなることもある。上記の損失関数は逆伝播可能で、一般低な活性化関数との利用に適しており、最尤推定に則った損失関数であるため用いられることが多い。

3.2 署み込みニューラルネットワーク

署み込みニューラルネットワーク (Convolutional Neural Network : CNN) は、図 3.2 に示したような、隣接する層の全ニューロン間で結合がある (全結合) ニューラルネットワークとは異なり、入力と設定されたフィルタの署み込み演算に基づいて出力の決定を行うニューラルネットワークの一種である。画像認識や音声認識など CNN の活用形態は多岐にわたり、本研究においても画像認識を目的として CNN を利用している。本項では、本研究で行われる 2 次元画像に CNN を適用する場合を例として CNN について説明を行う。

画像認識における全結合ネットワークには、空間的情報が失われるという問題点がある。入力が 2 次元画像の場合、空間的に近いピクセルは似たような値である、距離の離れたピクセルは互いに影響を及ぼさない、画素の RGB 値には密接な関連があるなどといった、空間的形状には汲み取るべき本質的なパターンが含まれていると考えられる。しかし全結合層はこのような形状を無視し、すべて同等のニューロンとして処理を行うため、空間的情報を生かすことができない。

そこで CNN は形状を維持するために署み込み層を利用する。署み込み層で行われる署み込み演算は $n \times n$ サイズの入力画像を $X(i, j)$ 、 $m \times m$ サイズのフィルタを $F(i, j)$ として、式 3.6 のように定義できる。

$$(X * F)(i, j) = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} X(i+k, j+l)F(k, l) \quad (3.6)$$

また、 $n = 4, m = 3$ として、署み込み層全体の処理を図 3.3 に示す。

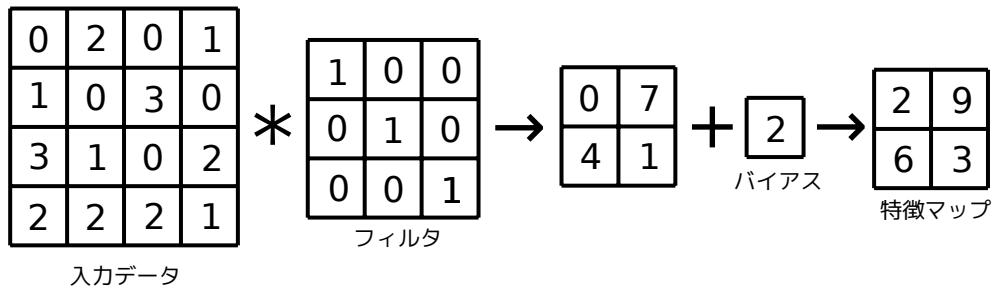


図 3.3: 署み込み層：署み込み演算を「*」で表記

以上のように、フィルタと入力データの対応する要素の積和演算を行い、対応する場所へ格納していく。このフィルタこそが重みに対応するパラメータであり、フィルタの値が CNN の動作を決定付ける。バイアスは全結合ニューラルネットワークと同様に、フィルタ (重み) の適用後に加算される。

図 3.3 でも示されるように、フィルタの適用により出力 (特徴マップ) は入力データに比べて一回り小さくなる。これを回避するために、入力データの端に 0 を追加する (ゼロパディング) 操作を加える場合がある。これにより、図 3.3 のように、フィルタの大きさが 3×3 かつ、フィルタの適用範囲を動かす量 (ストライド) が 1 ならば、入力と出力の大きさを一致させることができる。フィルタやストライドの大きさが変化する場合は、それを考慮してパディングの大きさも変化させる必要がある。

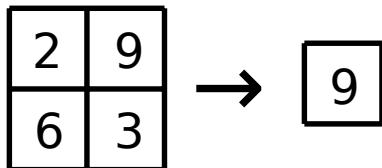


図 3.4: pooling (max pooling)

CNN を利用したアプリケーションでは、位置不变性 (パターンの位置がずれても影響を受け辛い性質) が求められることが多く、その際には pooling 層と呼ばれる新たな層がネットワークに追加される。画像を一定領域に区切り、各領域を最大値や平均値を用いて 1 画素に置き換える処理 (pooling) により、パターンのずれを吸収させ、位置不变性を高める。その処理内容からわかる通り、pooling 層において学習するパラメータはなく、層間における画素ごとのニューロン数 (チャネル) の変化もない。ただし、縦横方向の空間は小さくなるため、画素ごとの位置情報は失われる。よって、元々の位置情報が必要となるアプリケーションなどでは注意が必要となる。

3.3 セマンティックセグメンテーション

一般的な CNN による物体認識は、画像全体に対して何らかの判定・分類を行うものが多い。例としては、写真群から猫が写っている画像だけを抽出する機能などが挙げられる。一方で、セマンティックセグメンテーションは画素ごとに判定・分類を行う。例えば、複数の物体が写っている写真に対して、猫が写っている部分だけ塗りつぶしを行う機能が挙げられる。図 3.5 に示すセグメンテーション例では、猫とソファが区別されて塗り分けられている様子が確認できる。



図 3.5: セグメンテーションの例 : [9] より引用

1 画素だけを見てその画素が何にあたるか判定するのは通常不可能であるため、周辺画素ひいては画像全体を見て判定する必要がある。そこで、CNN による空間的情報を利用した判定を行うのだが、pooling 層を適用する場合、画素の位置情報が破棄されるという問題が発生する。

医用画像セグメンテーションを目的としたネットワークである U-Net[6] では、pooling によって縮小された特徴マップを後段で拡張することによって位置情報の復元を行う。この操作は pooling に対して unpooling と呼称される。U-Net における unpooling 処理を図 3.6 に示す。pooling の際に最大値だった画素の位置を保持し、unpooling では保持された

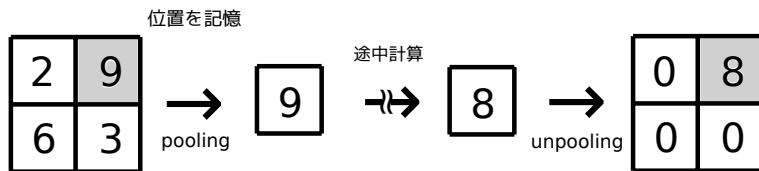


図 3.6: unpooling

位置を利用して拡張を行う。また、U-Net は pooling 層に通す前の特徴マップを後段に直接連結させることで、粗大化を避けられない unpooling 層の出力を補佐し、セグメンテーションの精度を高めている。

3.4 FPGA

Field Programmable Gate Array (FPGA) は、製造後に設計者が構成を設定できる集積回路である。FPGA は、複数入力のルックアップテーブル (LUT) 等で構成された論理ブロックを多数搭載し、LUT を書き換えることによって論理積や論理和といった様々な論理を表現できる。また、論理ブロック間を結ぶ内部配線についても構成を変化させることができるために、設計者は論理を表現したブロックを適切に組み合わせることによって任意の論理回路を実装する。

FPGA の設計は、一般的に Verilog HDL や VHDL といったハードウェア記述言語で行われる。用途に合わせて設計される集積回路である ASIC (Application Specific Integrated Circuit) に比べ、集積密度や電力効率、動作速度では劣る一方、開発・製造期間は短く、設計の変更も容易である。また、コスト面に関しても、製造のための初期コストは不要であり、FPGA 自体は汎用品であることから、少量生産においては生産コストでも有利である。

一般的に、ソフトウェアで動画像処理をするときは、動作クロック周波数の高い高性能 CPU が必要となる。一方ハードウェアは、回路を並列化やパイプライン化することで処理性能を上げることができ、ソフトウェアと比較して低い動作クロックで同等の処理を実現できる。そのため、画像処理システムには FPGA を始めとするハードウェアが用いられることが多い。また、計算分野によっては処理の並列化特性から高性能な CPU・GPU よりも高速に動作する可能性がある [8]。CNN もチャネル方向の並列化に加え、画素情報を順次走査で受け取りながらのストリーム処理が可能な点からハードウェアに適した計算処理であるといえる。

第4章

設計と実装

本章では、当プロジェクトにおいてソフトウェア実装されているシステムのネットワーク構造について説明し、そのハードウェア実装に際しての設計・実装方法について述べる。

4.1 アルゴリズム

本システムは、入力された手術画像に対して、ニューラルネットワークを用いたセグメンテーションを行い、ラベル画像を出力する。入力はサイズ 640×512 の RGB 画像であり、出力として 4 種類のクラスラベルを返す。クラスは、背景・胆嚢・胆嚢管・総胆管とする。

4.1.1 ネットワーク構成

本プロジェクトで実装されているネットワークの全体図を図 4.1 に示す。

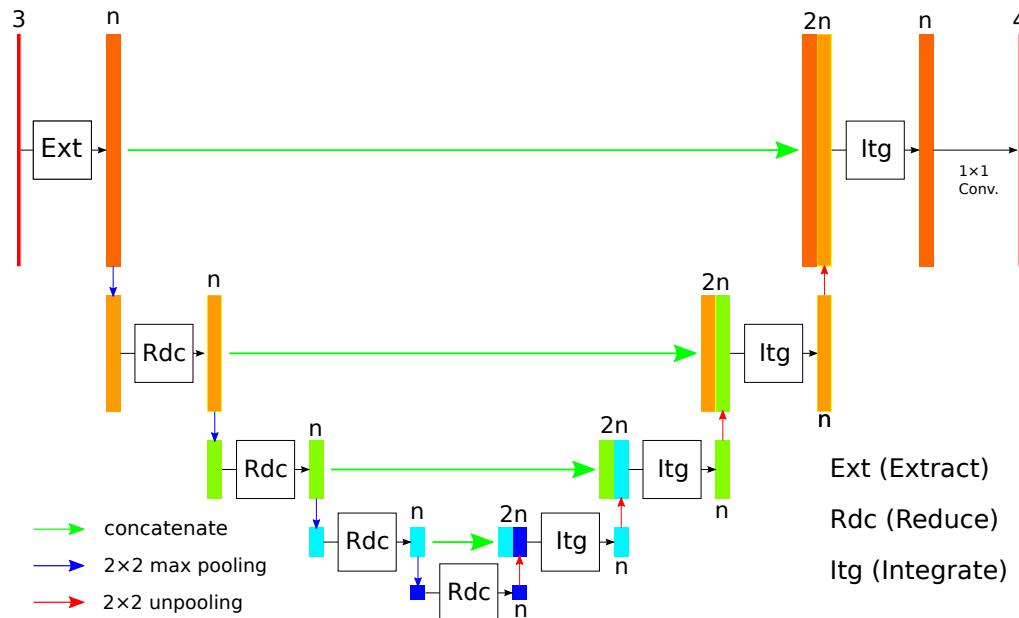


図 4.1: ネットワーク全体図

図 4.1 に則して説明を行う。まず入力画像は Ext という小規模ネットワークに与えられ、特徴マップが生成される。Ext は入力が RGB の 3 チャネル、出力が n チャネルとなっており、 n はパラメータで任意に設定できる。続いて、特徴マップは 2×2 max pooling で縮小された後、Rdc という小規模ネットワークに渡される。Rdc は入出力ともに n チャネルである。pooling による縮小と Rdc ネットワークの適用を再帰的に繰り返すことで、低次元への特徴マップの集約を行い、大域の情報を利用できるようにする。最下層に到達した後、unpooling による拡張が行われ、縮小処理を経ていない特徴マップと共に Itg という小規模ネットワークに渡される。よって Itg の入力は $2n$ チャネル、出力は n チャネルである。unpooling による拡張と前段の特徴マップの統合によって、pooling 層で失われた位置情報の復元を狙う。この拡張と Itg による統合は、縮小と Rdc の適用と同回数行われる。最後に、元画像と同じ大きさとなった特徴マップに 1×1 畳み込み演算を行い、指定の 4 クラスに対する尤度マップを生成する。

以下の項目では、ここに挙げた各処理について詳細を述べる。

4.1.2 畳み込み層

本ネットワークの畳み込み層は 3 つの小規模ネットワークから構成されている。図 4.2 にそれぞれの形状を示す。

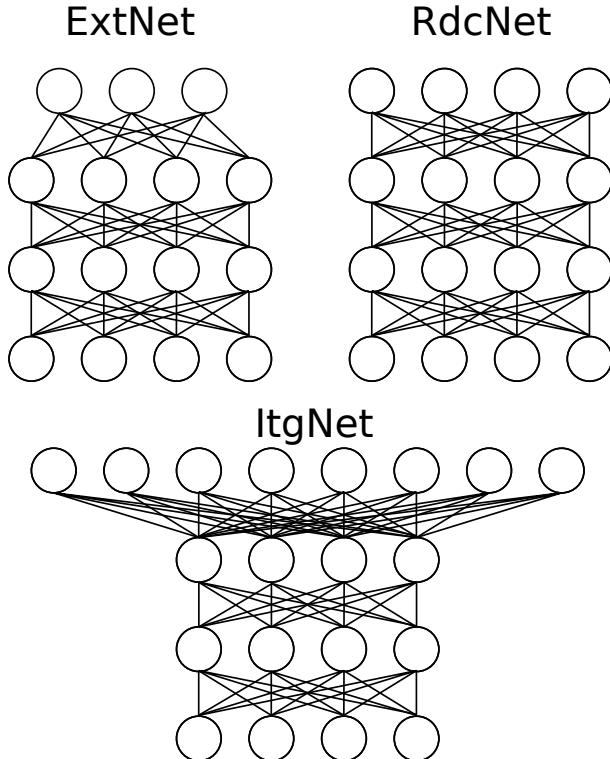


図 4.2: 3 種の小規模ネットワーク

フィルタサイズはいずれも 3×3 であり、入力と出力のサイズを同じにするため各層でパディングが行われている。活性化関数には、式 4.1 に示す Leaky ReLU[10] を用いる。

Leaky ReLU を用いる理由だが、式 3.3 に示した ReLU はいかなる負の入力に対しても 0 を出力するため、学習が進むにつれて絶対値の大きい負数が出力されることがある。本研究ではハードウェア化にあたり固定小数点演算を採用していることから、この現象は演算に必要なビット幅の増大や演算精度の低下につながる。一方 Leaky ReLU は、負領域においても 0 でない重み a を持つため、この問題を緩和できる。 a の値については、ビットシフト演算による単純な実装が可能な $a = 0.25 = 2^{-2}$ を採用した。

$$f(x) = \max(ax, x), a = 0.25 \quad (\text{Leaky ReLU}) \quad (4.1)$$

本ネットワークと U-Net の差異として、小規模ネットワークの再帰的な適用が挙げられる。医用画像はデータセットが乏しいため、訓練データを丸暗記してしまう過学習を引き起こす可能性が高い。そこで、小規模ネットワークの組み合わせによってネットワークを構築することで、記憶できるパラメータ数を意図的に減らし過学習を抑制する。加えて、異なる解像度のデータに対するネットワークの再帰的な適用により、サイズに左右されない、より普遍的な特徴抽出が期待できる点も、過学習抑制に繋がる。

また、各層でパディングが行われるのも U-Net との差異として挙げられる。これは、入力と出力のサイズを変化させないことで、設計を単純化できるためである。パディングを行わない場合、図 4.3 に示すように pooling・unpooling における動作が複雑化することが予測できる。各層でのパディングにより、小規模ネットワーク適用後、pooling では入力を半分に縮小し、unpooling では入力を 2 倍に拡張するという単純な動作となるため、設計を単純化できるという利点がある。

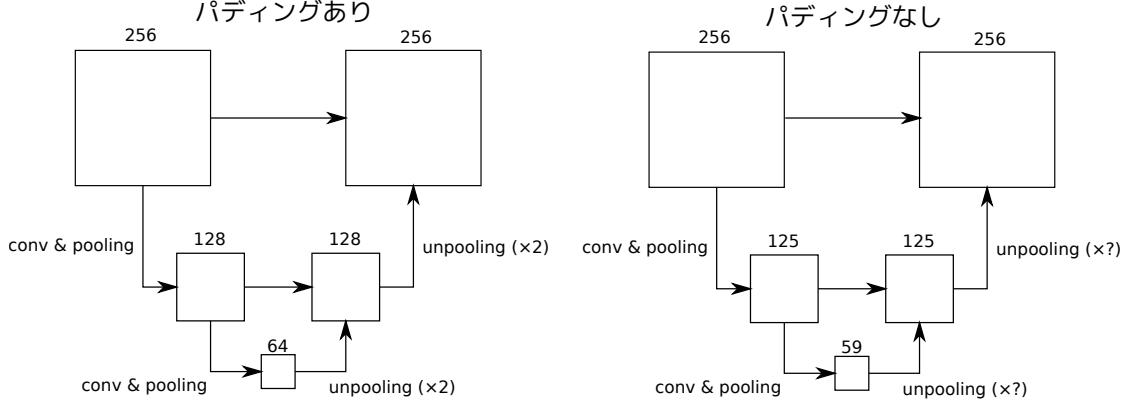


図 4.3: パディングの有無によるサイズ変化の違い

4.1.3 pooling 層

本システムの pooling 層では、一般的な CNN と同様に 2×2 max pooling を適用する。入力は 2×2 の領域に区切られ、各領域内の最大値 1 画素が出力される。よって、pooling 層の出力は前段からの入力に対し、縦横ともに半分のサイズとなる。図 4.4 にその様子を示す。4.1.2 節で述べたように、pooling 層の前後で実行される畳み込み層においてサイズは変化しないため、pooling 層でのサイズ変更は単純かつ規則的であることが確認できる。

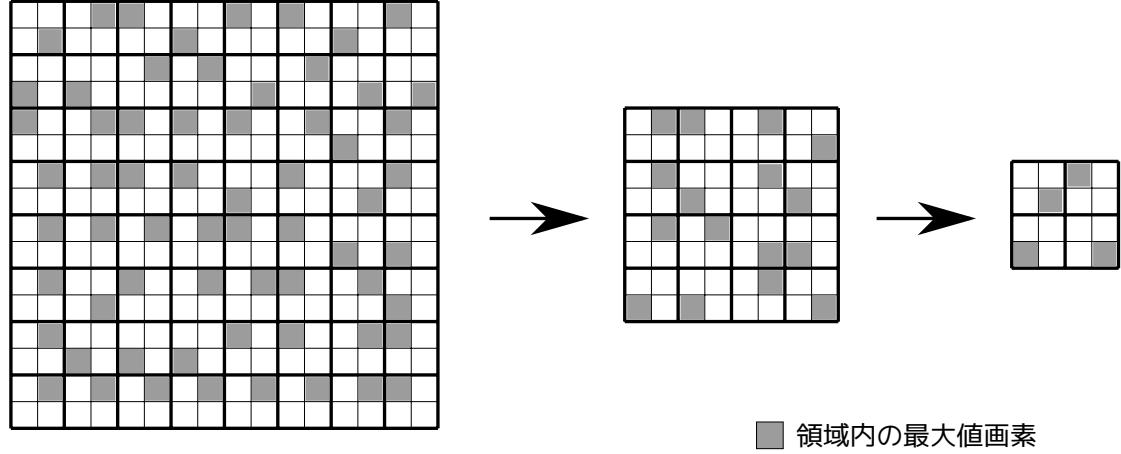


図 4.4: pooling 層の適用による縮小

4.1.4 unpooling 層

本システムの unpooling 層で行われている処理は図 3.6 とは異なり、値を周辺画素に転写する処理となっているため、位置情報を記憶する必要はない。pooling 層を経ていない特徴マップとサイズを一致させるには、4.1.2 節で述べたように縦横のサイズを 2 倍にすればよい。また、4.1.1 節で述べたように、unpooling 層の適用は pooling 層の適用と同回数行われ、最終的な出力のサイズは最初の入力と一致する。unpooling 層の動作を図 4.5 に示す。

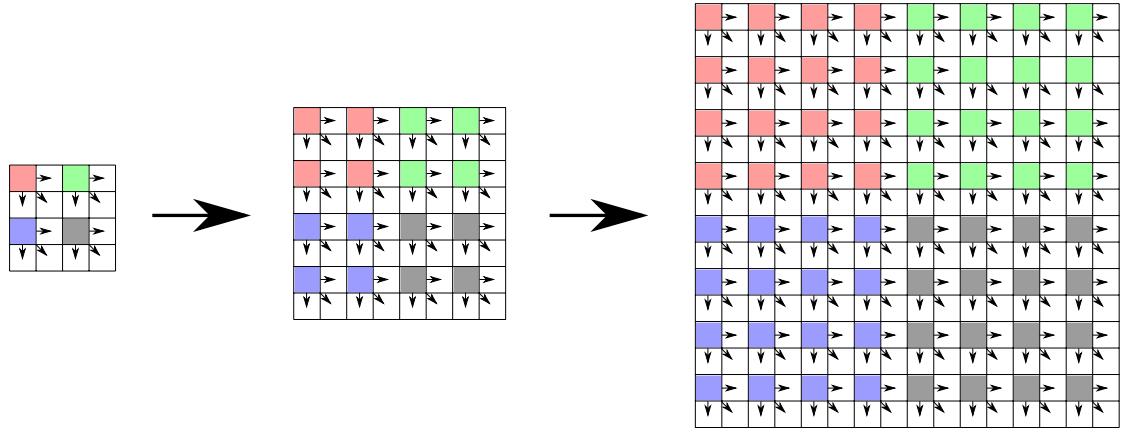


図 4.5: unpooling 層の適用による拡大

4.2 設計と実装

4.1 節で説明したシステムのハードウェア化にあたり、外部からの入力画像は、順次走査により水平および垂直座標 (h_cnt, v_cnt) と共に画素値 (in_pixel) のストリームとして与えられることを想定する。それに伴い、出力は座標と共に 4 種類のクラスラベルが 1 画素ずつ返される。

FPGAへの実装は、Verilog HDLを用いたRTL記述にて行った。論理合成にはVivado 2018.3を用い、ターゲットFPGAはVirtex UltraScale xcvu095-ffva2104-2-e2とした。

4.2.1 システム構成

図4.1のネットワーク構成を基に、図4.6のような設計を行った。

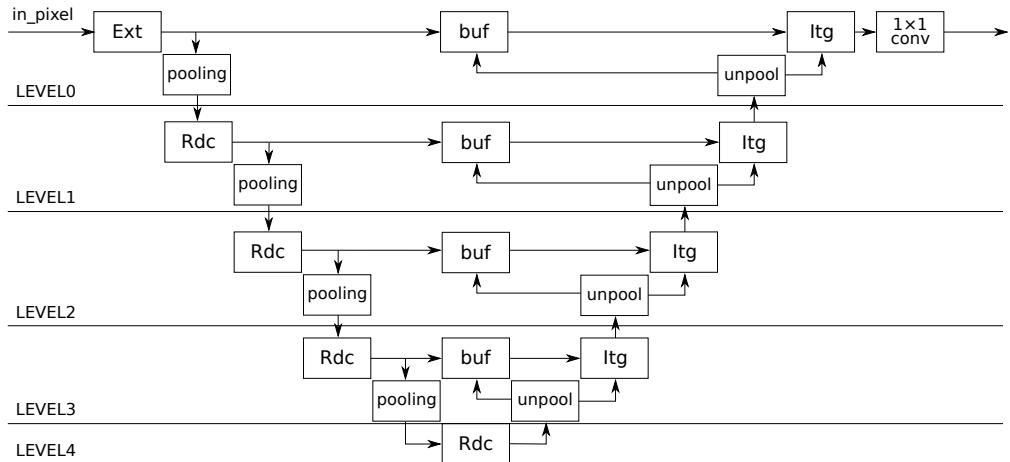


図4.6: システム構成概略図

システム全体は、順次走査により与えられる画像情報に対して、完全ストリーム処理を行うことができる設計となっている。畳み込み等のフィルタ演算は、適切なバッファリングにより切り出された画素の周囲画像(パッチ)とフィルタを用いて、ツリー構造の演算器で処理される。ただし、pooling・unpoolingにより有効画素の出力タイミングが変化するため、有効画素のタイミングを表すパラメータ(LEVEL)とイネーブル信号によってストリーム処理を維持する。LEVELは、図4.7に示す通り、毎クロック有効画素を出力するタイミングをLEVEL0とし、タイミングが縦横それぞれ半分になる度に1,2...と設定した。なお、図4.6に示したLEVELの区分については、poolingモジュールは入力に対して、unpoolモジュールは出力に対して、それ以外のモジュールでは入出力に対して図4.7に示したようにタイミングをとる。

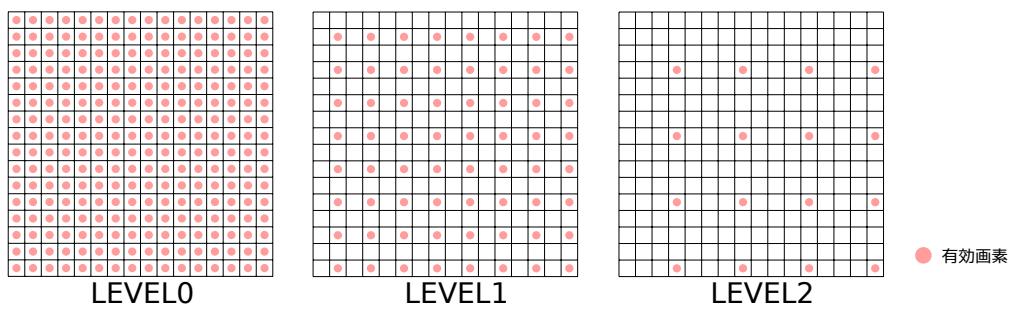


図4.7: LEVELごとの有効画素変化

以下の項目では、ここに挙げた各モジュールについて、LEVEL とイネーブル信号による制御方法に触れながら、処理の設計と実装について詳細を述べる。

4.2.2 stream_patch モジュール

フィルタ演算が行われる各モジュール内では、パッチ切り出しのためのモジュール (stream_patch) が用いられている。有効画素の入力ごとに、フィルタに対応した大きさのパッチが切り出される様子を図 4.8 に示す。これにより、有効画素が入力されるクロックごとのフィルタ演算が可能となる。

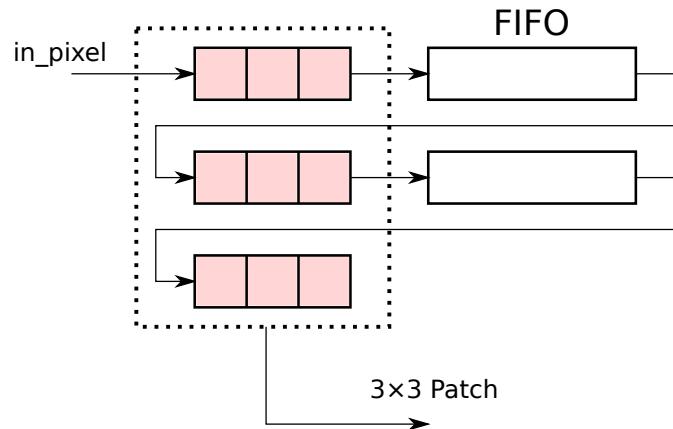


図 4.8: パッチ切り出し

4.2.1 節で述べた通り、本システムでは pooling・unpooling によって有効画素のタイミングが変化する。そこで、stream_patch モジュールにイネーブル信号を追加し、有効画素が入力されないクロックでは FIFO を停止させることで、全体のストリーム処理を維持する設計とした。stream_patch モジュール内で用いられているバッファ用のメモリは、Vivado に組み込まれた IP カタログにより生成されたシンプルデュアルポートメモリである。このメモリは書き込みイネーブルを入力に持つため、0 を入力することで図 4.8 に示した FIFO を停止させることができる。よって、必要となる FIFO サイズは 1 行あたりの有効画素数となり、LEVEL を用いた式 4.2 によって決定される。有効画素以外をメモリに格納する必要がないため、イネーブル制御を用いずに FIFO を停止させない実装に比べて、資源の消費量も少ない。

$$\text{FIFO サイズ} = \text{WIDTH(画像横幅)} \div 2^{\text{LEVEL}} \quad (4.2)$$

また、stream_patch が出力すべき座標値は、そのクロックで入力された座標値と stream_patch モジュールのレイテンシを利用した計算によって求められる。有効画素の入力タイミングの違いによりモジュールのレイテンシは変化するが、必要とする有効画素の入力数は変化しないため、基準となる LEVEL0 のレイテンシ (LATENCY) と、有効画素が入力される間隔を用いて、式 4.3 のように LEVEL ごとのレイテンシを求めることができる。

$$\text{patch_stream モジュールのレイテンシ} = \text{LATENCY} \times 2^{\text{LEVEL}} \quad (4.3)$$

`stream_patch` の変更により、フィルタ演算を行うモジュールは適切なイネーブル信号を前段から受けとることで、有効画素のタイミング変化に対応できるようになった。各モジュール内での、その他の停止操作やイネーブル信号の出力方法については以下の項目内にて詳細を述べる。

4.2.3 ExtNet・RdcNet・ItgNet モジュール

4.1.2 節で説明した通り、本システムの畳み込み層は 3 種類の小規模ネットワークから構成されるため、それぞれ 3 種類のモジュールとして実装を行うが、3 種類のネットワークには積和演算や活性化関数の適用など共通する部分も多い。そこで、図 3.2 に表したような、層間におけるニューロンの計算を共通モジュール (layer モジュール) として実装することで設計の単純化を狙う。図 4.9 に layer モジュールによるネットワーク作成のイメージ図を示す。layer モジュールはパラメータによって、内部のニューロンの数を変更でき、

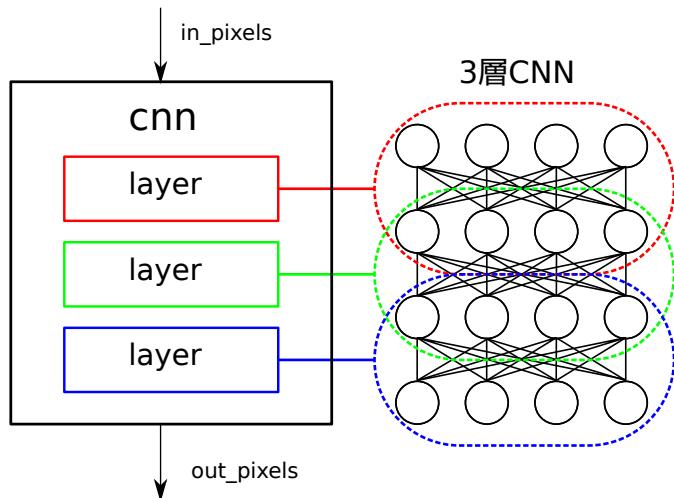


図 4.9: layer モジュールによるネットワーク作成イメージ

`layer` を重ねることで、多段ネットワークを実装する。よって、ネットワーク全体の入出力、`layer` モジュールを呼び出す段数、各 `layer` モジュールのパラメータの変更を行うことで、今回実装する ExtNet, RdcNet, ItgNet, 1×1 _conv の各 Net モジュールが実装可能となる。

`layer` モジュールについて説明する。`layer` モジュールは n チャネルの画素をそれぞれ 1 画素ずつ受け取り、`stream_patch` モジュールによってパッチに切り出した後、畳み込み演算、バイアス加算、活性化関数の適用を行う。畳み込みにおける加算器ツリーを、フィルタの一辺の長さ (`FLT_SIZE`) を 3、前段のニューロン数 (`PREV_NEURONS`) を 3、後段のニューロン数 (`NEW_NEURONS`) を 2 としたときの例を図 4.10 に示す。`layer` モジュールが output する座標値は、`stream_patch` モジュールが output する座標値と、畳み込み演算と活性化関数の適用に必要なレイテンシによって求められる。畳み込み演算のレイテンシは、パッチあたりの画素数 (`PATCH_NUM = FLT_SIZE2`) と、前段のニューロン数 (`PREV_NEURONS`)

に依存し、式 4.4 のように求められる。

$$\text{畳み込み演算のレイテンシ} = \log 2(\text{PATCH_NUM} \times \text{PREV_NEURONS}) \quad (4.4)$$

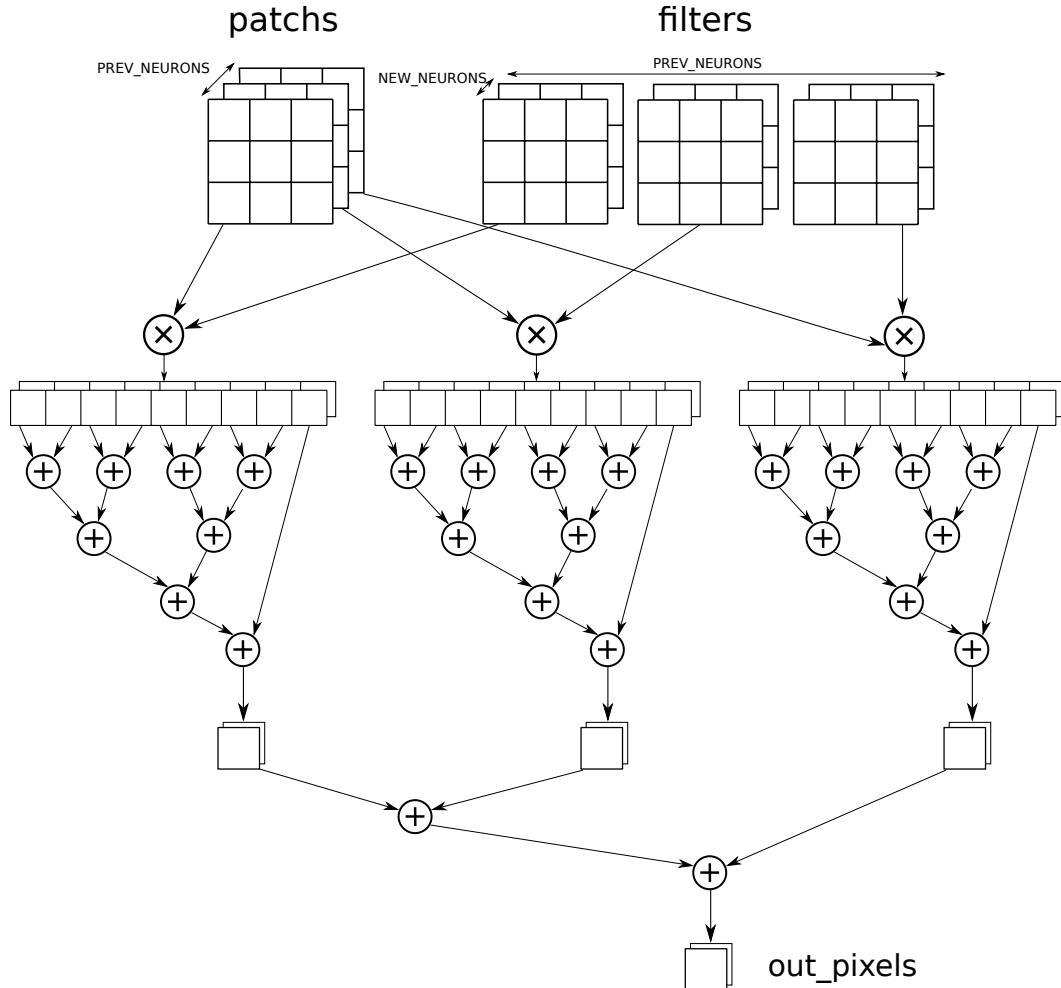


図 4.10: 加算器ツリーによる畳み込み演算

また、layer モジュールが出力するイネーブル信号については、自身の LEVEL と自らが output する座標値を利用した有効画素判定によって信号の値を決定できる。図 4.7 を基にして設定された、LEVEL によって注目すべき座標値のビット幅を決定する判定方法を表 4.1 に示す。

表 4.1: 出力座標値と LEVEL による有効画素判定 (layer モジュール)

	イネーブル信号値
LEVEL0	1'b1
LEVELN($N \neq 0$)	$(\&hcnt[N - 1 : 0]) \&& (\&vcnt[N - 1 : 0])$

4.2.4 pooling モジュール

pooling モジュールも layer モジュールと同じく、stream_patch モジュールを用いて対象画素の入力ごとに動作できる。ここで用いる対象画素とは、今回 2×2 max pooling を適用するため、画像全体を 2×2 の領域で区切った際の各領域の右下画素のことである。右下画素が入ったタイミングで pooling を行い、イネーブル信号と合わせて出力を行う。切り出されたパッチ内の最大値を求めるツリーを、ニューロンの数が 3 のときを例として図 4.11 に示す。図 4.11 からわかる通り、pooling に必要なレイテンシは layer モジュールのようにニューロンの数で変化することはない。

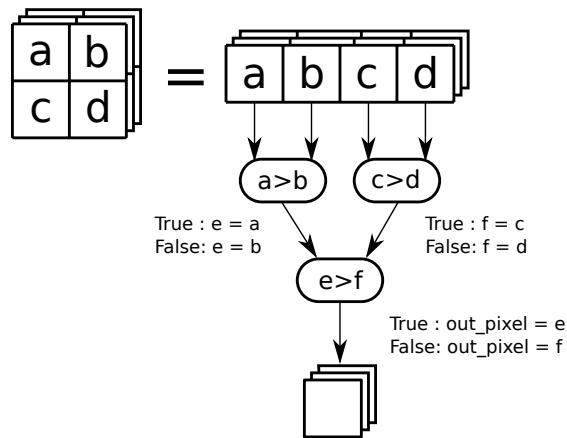


図 4.11: ツリーによる pooling

layer モジュールと同じく、出力するイネーブル信号の値は、自身の LEVEL と出力する座標値を用いた判定によって決定することができる。ここでは、自身の LEVEL という表現に注意しておきたい。4.2.1 節で述べた通り、pooling モジュール内で扱われる LEVEL の値は入力を基準に与えられているが、pooling モジュールは、入力に対して出力を縦横それぞれ半分にするため、いわば LEVEL を下げる処理を行っているといえる。そのため、出力するイネーブル信号の判定は LEVEL が変化するという前提で行う必要があり、同一 LEVEL の layer モジュールの判定とは違ったものとなる。表 4.2 にその判定方法を示す。表 4.1 と見比べると、LEVEL の使い方に違いがあることが確認できる。

表 4.2: 出力座標値と LEVEL による有効画素判定 (pooling モジュール)

	イネーブル信号値
LEVELN	$(\&hcnt[N : 0]) \&& (\&vcnt[N : 0])$

4.2.5 unpooling モジュール

4.1.4 節で述べたように、本システムにおける unpooling 動作は単純な画素拡張操作であるため、入力画素のバッファリングによって実装する。前段からの有効画素に対して、図 4.5 のように拡張を行うが、実際に入力される有効画素の座標値は pooling モジュール

における操作から、区切られた領域内での右下画素である。つまり、unpooling は区切られた領域内の右下の画素を受け取り、新たに設定された領域の右下へと拡張する操作といえる。図 4.12 に unpooling モジュールで行われる操作のイメージ図を示す。これを基に、必要となるバッファリングの大きさ、出力する座標値、イネーブル信号を適切に設定する。

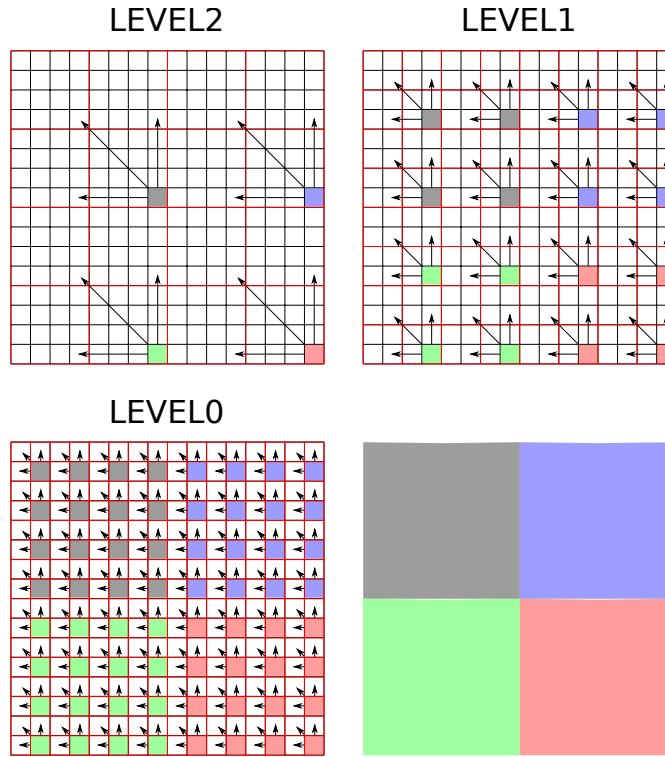


図 4.12: LEVEL による unpooling モジュール動作イメージの変化

まず、座標値についてだが、左上方向への拡張は有効画素が入力されたときに出力することが可能である。したがって、入力される有効画素の座標値に対し、式 4.5 で表されるだけのレイティシが存在すると仮定した。このレイテンシを基に出力する座標値を計算することで、入力される右下の座標値から出力する左上の座標値を求めることができる。

$$\text{LATENCY} = \text{WIDTH} \times (2^{\text{LEVEL}}) + 1 \quad (4.5)$$

この事から、unpooling モジュールは、有効画素が入力されたタイミングで左上方向への拡張を行う動作になったといえる。これを基準として、他方向への拡張に必要なバッファの大きさを決定する。以下に残りの 3 方向の拡張に必要なバッファの大きさを LEVEL を用いた式によって示す。

- 右上方向 : $\text{UppR_BUF} = 2^{\text{LEVEL}}$
- 左下方向 : $\text{LowL_BUF} = 2^{\text{LEVEL}} \times \text{WIDTH}$ (入力画像横幅)
- 右下方向 : $\text{LowR_BUF} = \text{UppR_BUF} + \text{LowL_BUF}$

これにより、すべての拡張方向に対する適切なバッファリングを設定することができた。このバッファリングされた画素値と、出力する座標値による拡張方向選択を組み合わせることによって、現在のタイミングで出力すべき拡張方向の画素値を出力する。表 4.3 に LEVEL0 のとき、表 4.4 にそれ以外の拡張方向の選択方法をそれぞれ示す。

表 4.3: 拡張方向選択方法 : LEVEL0

h_cnt[0]==0 v_cnt[0]==0	h_cnt[0]==1 v_cnt[0]==0	h_cnt[0]==0 v_cnt[0]==1	h_cnt[0]==1 v_cnt[0]==1
左上出力	右上出力	左下出力	右下出力

表 4.4: 拡張方向選択方法 : LEVEL ≠ 0

	h_cnt[LEVEL]==0 v_cnt[LEVEL]==0	h_cnt[LEVEL]==1 v_cnt[LEVEL]==0	h_cnt[LEVEL]==0 v_cnt[LEVEL]==1	h_cnt[LEVEL]==1 v_cnt[LEVEL]==1
((&h_cnt[LEVEL-1:0]) && (&v_cnt[LEVEL-1:0]))==1	左上出力	右上出力	左下出力	右下出力
otherwise	出力なし			

4.2.1 節で述べた通り、unpooling の LEVEL は出力を基準に設定されている。そのため、出力イネーブル信号の出力は layer モジュールで用いた表 4.1 による判定によって同様に決定される。

4.2.6 buf モジュール

buf モジュールは、ItgNet モジュールが受け取る unpooling モジュールからの出力と pooling 層適用前の特徴マップを対応付ける機能を持つ。通常ならば、それぞれの特徴マップ出力に必要なレイテンシの差分を求め、差分だけバッファリングさせることで出力のタイミングを合わせるのが一般的である。

一方で、本研究における実装はニューロン数やネットワーク段数など、パラメータによるネットワーク変更が可能な設計がされている。これはネットワークの改良に対して柔軟に対応するための設計であるが、レイテンシの差分だけバッファリングする方法では、パラメータによるネットワーク変更を行う度にレイテンシを正確に求める必要があり、煩雑な設計が求められる。そこで、画像 1 枚分のメモリを確保し、座標値 (h_cnt, v_cnt) に基づくアドレスによって対応付けることとした。図 4.13 に buf モジュールの設計を示す。

out_hcnt, out_vcnt は各 ItgNet モジュール前段の unpooling モジュールの出力が渡される。これにより、unpooling モジュールからの画素値と同じ座標値を持つ pooling 前の画素値を対応付けることができた。ただし、この実装方法は一般的なバッファリングによる実装方法に比べて資源使用量が増加することが予想される。??節でバッファリングに必要な資源量と実際の資源使用量を比較した考察を行うこととする。

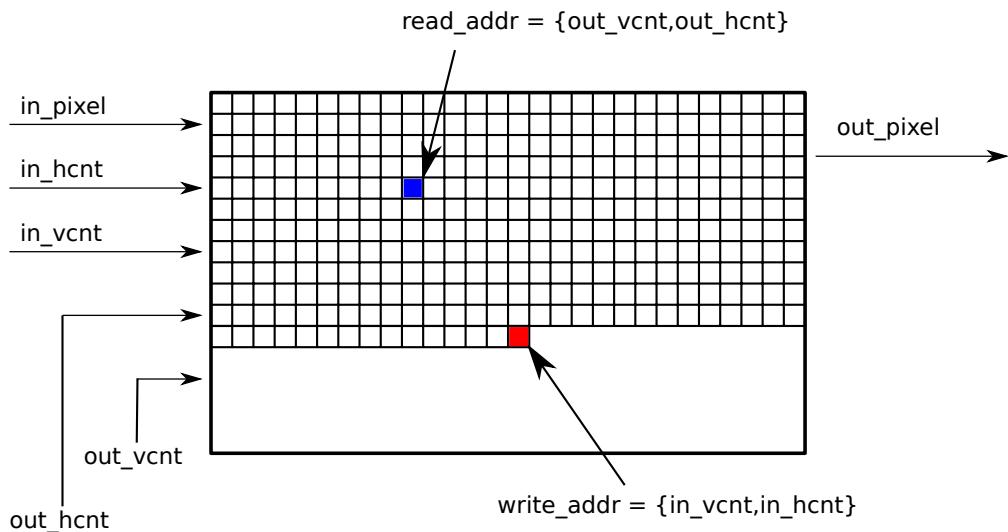


図 4.13: buf モジュール

4.3 学習

4.3.1 ツール

学習には、Python 上で動作するニューラルネットワーク向けフレームワークである Chainer5.3.0 [11] を用いる。Chainer では、複雑なデータ構造を簡潔な記述で構築でき、CUDA による GPU を用いた高速な学習も可能である。また、学習および評価における各処理には、数値計算ライブラリの Numpy や、画像処理ライブラリの OpenCV を用いる。

4.3.2 学習データ

今回ニューラルネットワークの学習に用いる画像データと教師ラベルからなるデータセットは 183 組であり、このうち 138 組を訓練用データセット、45 組を評価用データセットとして学習を行う。画像データは長崎大学病院から提供された実際の手術画像であり、画像サイズは 640×512 となっている。また、それらの画像を医師が目視で判断し、手動でラベリングを行い作成された画像を教師データとした。これらの画像例を以下の図 4.14、図 4.15 に示す。



図 4.14: 実際の手術画像データ

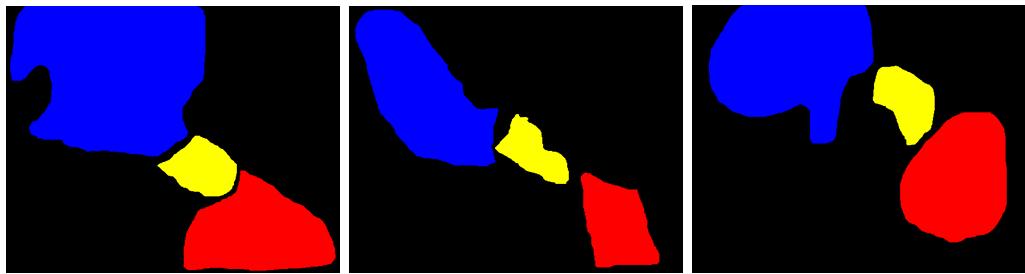


図 4.15: 教師データ

4.3.3 量子化手法

ソフトウェアでの実装においては、各演算は 32 ビットの浮動小数点型で行われ、学習の結果得られる重みやバイアス等も同じ型となっている。しかしながら、ハードウェア実装においてパラメータに浮動小数点型を用いるのは資源容量的に厳しく、固定小数点型を利用するのが望ましい。

そのため、本来ならば学習においても固定小数点型を利用して学習を進め、パラメータを決定付けるべきだが、Chainer では浮動小数点型を用いることが前提となっており、固定小数点型で動作させるのは困難である。そこで、学習は浮動小数点型で行い、得られたパラメータを固定小数点型に変換して実装することとした。

第5章

評価と考察

本章では、設計・実装したセマンティックセグメンテーションシステムの性能を、資源使用量、最大動作周波数、レイテンシの観点から評価するとともに、それらに関する考察を行う。

5.1 評価

評価には vivado による論理合成結果と、計算によって求められるレイテンシを用いた。また、レイテンシ計算結果の正確度を検証するため verilog シミュレーションである xmverilog を利用する。ネットワーク構成は図 4.1 に示す通りであり、 $n = 12$ とした。 n の値はソフトウェアによる実装において良好な結果が得られたときの値を採用している。

5.1.1 資源使用量

論理合成によって得られた資源使用量を表 5.1 に示す。なお、DSP・BRAM については最大限利用して合成を行うよう設定し、溢れた分は他の資源を用いて合成される。DSP は主に固定小数の乗算に、、BRAM は主に FIFO を始めとするメモリに用いられる。

表 5.1: 資源使用量

資源	使用量	使用可能	割合
LUT	2302985	537600	428.58%
LUT-RAM	140369	76800	182.77%
FF	1732778	1075200	161.16%
BRAM	1715	1728	99.25%
DSP	752	768	97.92%
CARRYs	306272	67200	455.76%

5.1.2 最大動作周波数

最大動作周波数は、vivado による論理合成を行った際に生成される Report Timing Summary を利用する。論理合成段階での判定であるため、配置配線の結果によって変動する可能性のある値ではあるが、動作可能周波数の目安としては十分であると判断し、評価に利用した。

入力されるクロックを 10ns (100MHz) としたときの、Report Timing Summary から得られた Worst Negative Slack (WNS) は 5.111ns であった。よって、論理合成時点での最大動作周波数は 204.54MHz となる。100MHz、最大動作周波数でそれぞれ駆動させたときの fps を表 5.2 に示す。なお、入出力画像のサイズは 4.1 節と同じく 640×512 を想定する。

表 5.2: 動作周波数と fps

動作周波数	100.00MHz	204.54MHz
fps	305.17fps	597.21fps

5.1.3 レイテンシ

システム全体のレイテンシ、つまり画素の入力から対応した出力が行われるまでの経過クロック数は、完全ストリーム処理である本実装では各モジュールのレイテンシの総和で求めることができ、実行中に変化することもない。

本システムにおいて最もレイテンシを増加させる原因となりうるのはフィルタ適用である。注目画素をフィルタの中心画素とした場合、結果が出力されるには右下画素の入力を待つ必要があり、フィルタ適用 1 回ごとに少なくとも画像横幅分のレイテンシが必要となる。さらに、本実装では有効画素の入力タイミングが変化することで、右下画素の入力を待つのに必要なクロックは指数関数的に増加する。図 5.1 に各フィルタ適用に要するクロックを行単位で示す。実際にはニューロン数の違いなどで各フィルタ処理に必要なレイテンシは細かく変化する。

図 5.1 より、全体のレイテンシにおいて大きな割合を占めるであろうフィルタ処理に必要なレイテンシは、約 139 行分であることが概算的に求められた。これを踏まえ、verilog シミュレーションである xmverilog によって、全体のレイテンシを検証する。simvision による結果は、図 5.2 の通りとなった。また、波形表示には simvision を利用している。end_hcnt=0, end_vcnt=0 地点が最初の出力である。入力の座表値に対し 154 行 + 57 クロック分のレイテンシを経て出力されていることが読み取れる。概算で求めた行数に対し約 15 行分の差異が存在するが、フィルタ処理以外の操作に必要なレイテンシを鑑みた場合約 15 行分の差異は妥当であり、シミュレーションによって解析されたレイテンシの信頼性は高いと言える。

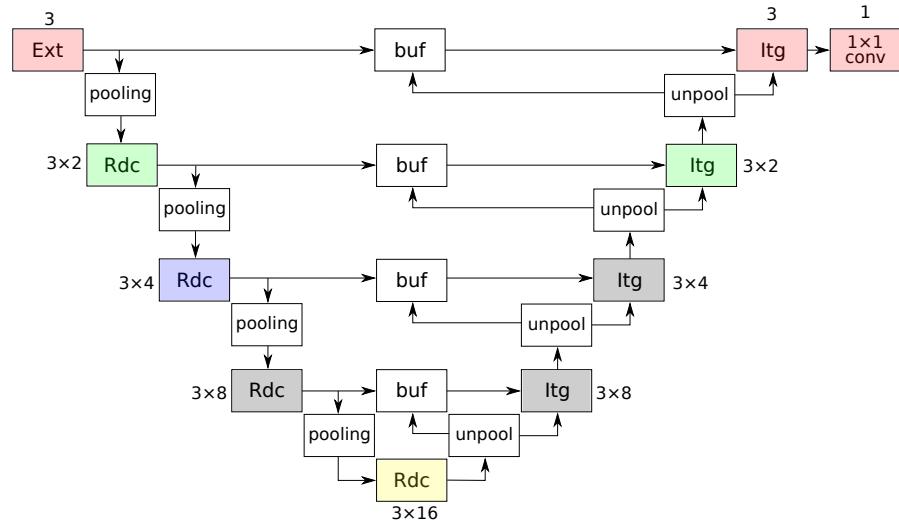


図 5.1: フィルタ適用に要するレイテンシの概算(行)



図 5.2: シミュレーションによるレイテンシ解析

5.2 考察

5.2.1 現在の実装に対する評価について

表 5.1 から分かる通り、ほとんどの資源において資源使用量が使用可能量を超えていたため、現在の実装では 1 台の FPGA に配置配線することが不可能である。特に、LUT と CARRYs の使用量の割合が高く 400% を超えているが、これは固定小数乗算のために合成される大量の乗算器によるものが大半である。表 5.3 に、各 Net モジュールの各資源使用量を示す。なお、RdcNet・ItgNet に関しては、FIFO に用いるための LUT_RAM 以外において、4 つのネットワークでの資源使用量の差はあまり見られないため、それぞれ LEVEL1 における使用量を示している。

しかし、本システムは完全ストリーム処理が可能な実装となっており、モジュール間で渡される情報は、1 クロックにつき 1 画素分の画素値・座標値・1bit の信号のみである。そのため、複数台の FPGA を接続することによる実装は理論上可能であり、FPGA 間でのデータの受け渡しにかかるレイテンシも、そのデータ量から性能を格段に低下させるほどではないと予想される。複数台での実装が可能となれば、本システムは仮に 100MHz で

表 5.3: 各 Net モジュールにおける資源使用量

資源	ExtNet	RdcNet	ItgNet	1×1 conv
LUT	21497	256177	296603	1738
DSP	161	36	106	21
CARRYs	25932	32091	37773	226

駆動したとしても、305.17fps を達成し、かつ約 9.8617×10^{-4} という低レイテンシでの動作が可能である。

また、4.2.6 節で述べた、buf モジュールの実装方法によるメモリの増加についてだが、シミュレーションの結果、単純なバッファリングによる実装に必要なレイテンシは LEVEL0 における buf モジュールで約 148 行分であった。よって、必要な FIFO のメモリサイズは、画像 1 枚分のメモリを確保している現在の実装に比べて 3 分の 1 以下に抑えることができる。BRAM の使用量は buf モジュールで占められているため、現在オーバーしている BRAM・LUT-RAM の使用量に関しては、バッファリングによる実装に変更することによって 1 台分の資源量に抑えられる可能性がある。

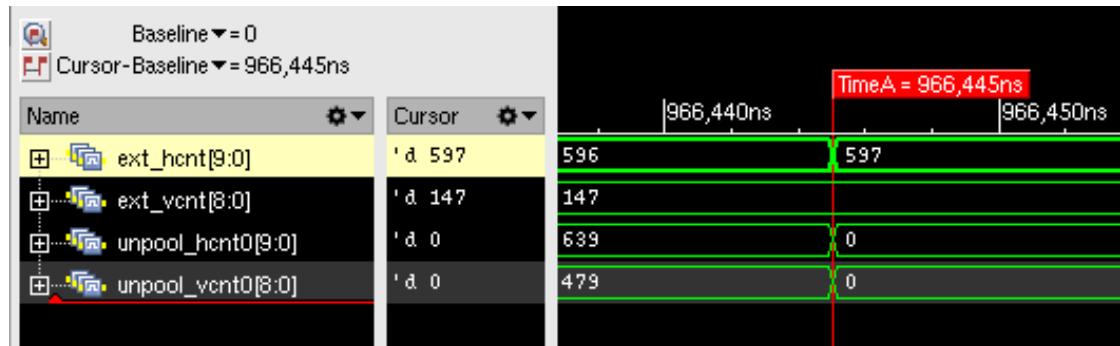


図 5.3: buf モジュールに必要な FIFO サイズ

5.2.2 1 台の FPGA による実装に向けて

本項では、1 台の FPGA による実装を目的として資源使用量の削減を行う場合、どのような軽量化手法が効果的なのか考察を行う。

まず考えられるのはネットワーク規模の削減である。特に、各層におけるニューロンの数と乗算数は指數関数の関係にあり、ニューロン数を減少させることで資源量の大幅な削減が見込める。しかし、ネットワーク構成の大規模な変更はネットワークの出力精度に大きく影響し、目的とする機能を構築できなくなる可能性が高い。そのため、ネットワーク構成の変更による資源使用量削減は、出力精度の観点から現実的でないといえる。

次に考えられるのは、本プロジェクトにおけるネットワークの特徴ともいえる、小規模ネットワークの再帰的利用に着目した削減方法である。LEVEL が低くなるにつれ、有効画素が入力されるタイミングは減少し、それに伴って乗算器が動作しなければならないク

ロックも減少する。そこで、乗算器が動作しないタイミングでは後段のネットワークとして動作することで、ItgNetにおいては2つ、RdcNetにおいては1つのネットワークのみで全体の処理が可能となる。これは、表5.3に示した通り各Netモジュールが全体の資源使用量を占める本実装において、効果的な削減手法といえる。

資源使用量増加の原因である乗算の削減方法としては、重みパラメータの量子化が挙げられる。ここで用いる量子化とは、一般的に用いられることの多い連続的な値を離散的な値に変換するという意味ではなく、値の表現bit数の削減、特に1bitでの表現に変換することを指す。重みを1bitで表した場合、パラメータに用いるメモリ使用量が削減されるうえ、積和演算が単純なビット演算で行えるようになり、乗算器の大幅な削減となる。量子化ニューラルネットワーク手法は、2015年のBinaryConnect[12]を皮切りとして様々な方法が提案されているが、小さいネットワークでは良好な結果が得られるものの、一般的なネットワークでは精度低下を引き起こしてしまうのが現状である。そのため、本システムを量子化ニューラルネットワークとして実装する場合、効果的な量子化手法の検証が重要になると考えられる。

第6章

結論

本研究では、畳み込みニューラルネットワークを用いた、手術画像向けセマンティックセグメンテーションシステムをハードウェア記述言語を用いて実装し、その合成結果から性能検証を行った。

本システムはそのネットワーク規模から、現在の実装では資源使用量が1台のFPGAにおける資源量を超えていたため、1台のFPGAによる動作は望めない。しかし、完全ストリーム処理であることと1クロックあたりに受け渡す必要のある信号線の少なさから、複数台のFPGAによる実装が見込まれる。その場合、本システムは高速なフレームレートと低レイテンシによる動作が可能であり、本研究の目的である、セマンティックセグメンテーションシステムの高速化は達成した。

一方で、組み込み系機器での運用が想定される本システムでは、消費電力の観点から1台のFPGAによる実装が望ましく、資源使用量の削減を目的として様々な手法を提案した。特に、本ネットワークの特徴であるネットワークの再帰的な利用に着目した削減方法は、出力精度に影響を与えることなく大きな削減が見込める。

謝 辞

Thank you

参考文献

- [1] 公益社団法人全日本病院協会. 胆囊切除術患者に対する腹腔鏡下手術施行率, <https://www.ajha.or.jp/hms/qualityhealthcare/indicator/21/>.
- [2] 医療法人光生会. 腹腔鏡手術について, http://www.koseikai-hp.or.jp/kouseikai_index/kouseikai_clinicindex/01_koseikai_h_26/01_koseikai_h_261.html.
- [3] Intuitive — da vinci robotic assisted surgical systems : <https://www.intuitive.com>.
- [4] Taito Manabe, Koki Tomonaga, and Yuichiro Shibata. CNN Architecture for Surgical Image Segmentation Systems with Recursive Network Structure to Mitigate Overfitting. International Symposium on Computing and Networking (CAN-DAR'19). 2019.
- [5] 一般社団法人日本医療情報学会. 医療画像データ収集事業に用いる情報システム構築ガイドライン, http://jami.jp/about/documents/amed_report.pdf.
- [6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net:Convolutional Networks for Biomedical Image Segmentation. In Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention, pp.234-241. 2015.
- [7] AI を活用したリアルタイム内視鏡診断サポートシステム開発 : https://jpn.nec.com/press/201707/20170710_01.html.
- [8] 佐野 健太郎, 河野郁也, 中里直人, Alexander Vazhenin, Stanislav Sedukhin. FPGAによる津波シミュレーションの専用ストリーム計算ハードウェアと性能評価. 2015-HPC-149-5 p1-7. 2015.
- [9] PASCAL VOC2011 Example Segmentations : <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/segeexamples/>.
- [10] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. in proceedings of the 30th international conference on machine learning. 2013.
- [11] Preferred networks, inc. chainer : A exible framework of neural networks : <http://chainer.org/>.

- [12] Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. NIPS. 2015.