2025 年度　アジャイルワーク 報告書

24G1089　武本 龍

2025 年 12 月 12 日

1　はじめに

近年、ディープラーニングをはじめとする AI 技術は急速に発展しており、その学習や推論には大規模な計算資源や電力を必要とすることが一般的となっている。しかし、組込み機器や IoT デバイスのような小型環境では、CPU 性能やメモリ容量、電力供給といったリソースが大幅に制限されるため、従来の手法をそのまま適用することは困難である。そこで本研究では、既存の Arduino Uno R4 WiFi 環境において可能な限り高性能なニューラルネットワーク推論を実現することを目的とし、学習データの軽量化やモデルの圧縮を含む最適化手法を検討し、特に、学習データを圧縮した上で推論に必要な情報を保持できるか、また限られた計算能力の中で最大限の推論精度を引き出せるかを検証し、その実験手法および得られた知見について報告する。

2　実験の概要

図 1に，本実験の全体構成を示す．

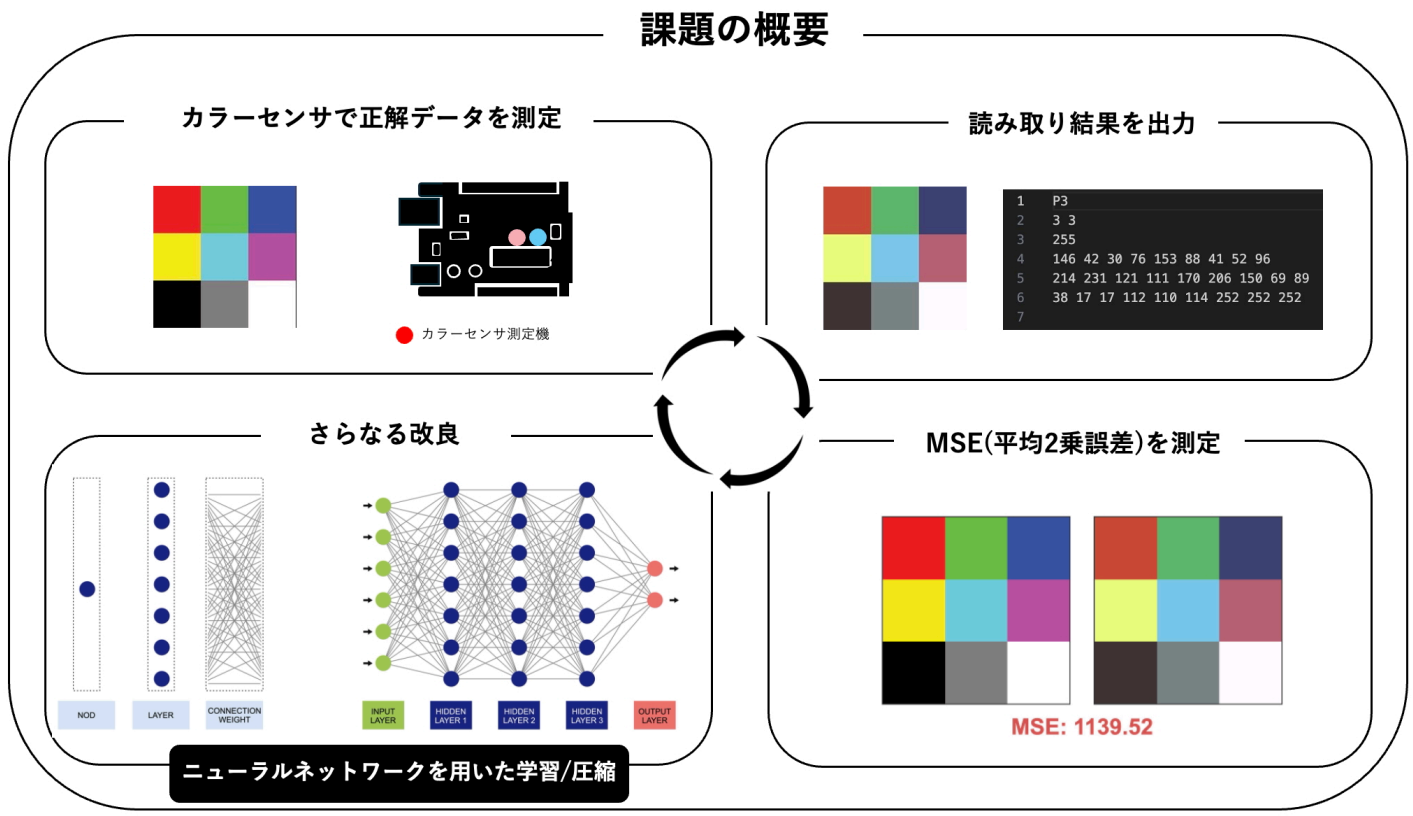


図 1: 実験の全体構成

本実験では，Arduino Uno R4 WiFi を用いてカラーセンサーを構築し，授業内で配布された 3 × 3 のカラーチャートを測定する．測定後，ニューラルネットワークを用いた補正処理により，平均二乗誤差（MSE）の低減を目指す．前章で述べた 2 つの検証項目に対応し，本実験では以下の観点から評価を行う．

第一の検証として，ニューラルネットワークのパラメータ圧縮によるメモリ効率の改善を検討する．具体的には，隠れ層のデータ圧縮手法を導入し，推論に必要な情報を保持しながらメモリ使用量をどの程度削減できるかを評価する．

第二の検証として，限られた計算資源の中での推論性能の最大化を検討する．測定したカラーチャートと基準カラーチャート間の MSE を評価指標とし，ニューラルネットワークによる学習を活用して推論精度の向上を図る．また，predict 関数の処理時間を計測し，実行速度についても評価する．

2.1 平均二乗誤差（MSE）

平均二乗誤差（MSE: Mean Squared Error）は，2 枚の画像の対応するピクセル位置における輝度差の 2 乗の平均値として定義され，以下の式で表される．

$$\mathrm{MSE} = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(I(i,j) - K(i,j) \right)^2 \tag{1}$$

ここで，I と K は比較対象となる 2 枚の画像，$m \times n$ は画像サイズを表す．

本実験では RGB 画像を扱うため，各ピクセルについて R，G，B チャネルごとの差の 2 乗を加算し，チャネル数で除した値を MSE として算出する．具体的な計算式を以下に示す．

$$\mathrm{MSE} = \frac{(r_{\mathrm{diff}})^2 + (g_{\mathrm{diff}})^2 + (b_{\mathrm{diff}})^2}{3} \tag{2}$$

MSE の値が小さいほど，2 枚の画像間の差異が少なく，色再現性が高いことを示す．本研究では，この MSE を推論精度の評価指標として用いる．

3　実験理論

本章では，ニューラルネットワークを用いた読み込み品質の改善手法について述べる．

3.1　ニューラルネットワークによる品質改善

サンプル画像と測定画像との誤差を最小化するため，本研究ではニューラルネットワークを用いた学習手法を導入する．特に，低リソース環境である Arduino 上での推論実行を前提とし，モデルの圧縮および疎化手法を併用しながら性能の最適化を図る．

3.1.1　ネットワーク構造

本実験で採用したモデルは，全結合型（Fully Connected）の 4 層ニューラルネットワークである．入力層は RGB の 3 次元，2 つの隠れ層はそれぞれ可変次元（実験では 40〜600 次元），出力層は再び RGB の 3 次元とした．各層の構成を表**??**に示す．

表 1: ニューラルネットワークの構造

層	次元数	活性化関数
入力層	3	–
隠れ層 1	40〜600（可変）	ReLU
隠れ層 2	40〜600（可変）	ReLU
出力層	3	Sigmoid

各ニューロンにおける推論は以下の式で表される．

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_{n-1} w_{n-1} + b \tag{3}$$

ここで，w_i は重み，b はバイアスを表す．算出された値は活性化関数を通過し，次層へ伝搬される．

モデルは PyTorch により定義・訓練し，L1 正則化を導入して重みの疎化を促進した．訓練後の重みおよびバイアスは C++ 配列としてエクスポートし，Arduino 上で実行可能な形式に変換した．

本研究で適用する圧縮手法（CSR 形式，量子化）は，主にパラメータ数の多い隠れ層 1 および隠れ層 2 の重み行列に対して適用する．隠れ層の次元数を拡張するとパラメータ数が増加し，メモリ使用量が増大するため，圧縮手法の適用が不可欠となる．

3.2　隠れ層の圧縮手法

本研究では，ニューラルネットワークの隠れ層パラメータに対して以下の圧縮手法を適用する．

3.2.1　圧縮の必要性

Arduino Uno R4 WiFi のようなマイクロコントローラでは，利用可能なメモリ（SRAM 約 32 KB，Flash 256 KB）が極めて限定的である．隠れ層の次元を増加させると，重みおよびバイアスの格納や推論演算に必要なメモリが不足する可能性が高い．

例えば，隠れ層次元を 40 から 70 以上に拡張するとパラメータ総数は急増する．

- 40 次元：約 1,923 パラメータ
- 70 次元：約 3,000 パラメータ以上

float32 形式では 10 KB を超える場合，Arduino 上での動作が困難となる．したがって，隠れ層の次元を拡張しながら高い推論性能を維持するには，重みデータの圧縮が不可欠である．

3.2.2　プルーニング（Pruning）

プルーニングとは，閾値以下の重みを 0 とみなし接続を削除することで，モデルを疎行列化する手法である．Han ら [2] は，重要接続の学習，閾値による削減，再訓練を繰り返すことで，大規模モデルを 10 倍以上圧縮できることを示した．

本研究では，L1 正則化により重みを 0 に近づけた後，閾値 0.01 でプルーニングを実施する．この処理により多くの重みが 0 となり，後述する疎行列表現による効率的な格納が可能となる．

■メリット

- パラメータ削減：非ゼロ率を大幅に低下可能（40 次元モデルで 1,923 から 200 以下も実現可能）
- 計算量削減：不要な演算が減少し推論が高速化
- 精度維持：軽微な削減であれば再訓練により精度回復が可能

■デメリット

- 再訓練なしでは精度低下が生じやすい
- インデックス保存など疎行列特有のオーバーヘッドが存在
- プルーニング量の決定に追加の探索が必要

3.2.3　CSR（Compressed Sparse Row）形式

プルーニングにより疎化された重み行列は，そのまま密行列として保持すると 0 要素にもメモリを消費するため非効率である．CSR 形式は，非ゼロ要素のみを行単位でまとめて保存する疎行列表現であり，値配列（data），列インデックス（indices），行ポインタ（indptr）で構成される．行方向のアクセスが高速であり，行列ベクトル積を多用するニューラルネットワークの推論に適している．

■メリット

- 行方向のアクセスが高速で推論が効率的
- インデックス管理が最適化されメモリ効率が高い
- 固定長配列で Arduino 実装に適する
- 非ゼロ率に応じて 3〜10 倍のメモリ削減が可能

■デメリット

- 変換時にソートが必要
- 列方向アクセスは非効率

なお，疎行列表現としては COO（Coordinate）形式も存在し，実装が容易という利点があるが，行列ベクトル積の計算効率が CSR 形式より劣るため，本研究では CSR 形式を採用した．

3.2.4　量子化（Quantization）

量子化とは，ニューラルネットワークの重みや活性化値を 32 ビット浮動小数点から 8 ビット整数へ変換し，メモリ削減と計算高速化を図る手法である．Jacob ら [1] が提案したアフィン変換

$$r = S(q - Z) \tag{4}$$

を用いることで，推論時の整数演算が可能となる．ここで，S はスケール，Z はゼロポイントを表す．

量子化は，プルーニングや CSR 形式とは独立に適用可能であり，CSR 形式で格納された重みに対しても追加適用できる．float32 から int8 への変換により，さらに約 4 倍のメモリ削減が期待できる．

■メリット

- メモリ削減：32 ビットから 8 ビットへの変換により約 4 倍削減
- 計算高速化：整数演算により浮動小数点演算より低レイテンシ
- 精度維持：量子化感知訓練（QAT）を用いることで MSE をほぼ維持可能

■デメリット

- モデル規模が小さい場合，チャネルごとの差により誤差が生じやすい
- スケール・ゼロポイントの計算や int32 蓄積処理など実装がやや複雑
- 訓練時に fake quantization ノードの挿入が必要

3.2.5 本研究における適用方針

表 1に，各圧縮手法の特徴を示す．

表 2: 重み圧縮手法の比較

手法	メモリ削減率	計算効率	実装難易度	Arduino 適合性
CSR 形式	3–10 倍	高	中	高
量子化	約 4 倍	高	中	高
CSR+ 量子化	12–40 倍	高	高	高

3.2.6 段階的圧縮アプローチ

本研究では，段階的な圧縮アプローチを採用する．その理由は以下のとおりである．

第一に，圧縮処理は推論精度に影響を与える可能性があるため，必要最小限の圧縮にとどめることが望ましい．過度な圧縮は精度劣化を招くリスクがあり，圧縮手法を一度に複数適用するよりも，段階的に適用して各段階で精度を確認する方が安全である．

第二に，プルーニングと CSR 形式は本質的に連続した処理である．L1 正則化によって重みを疎化し，その結果生じた 0 要素を効率的に格納するのが CSR 形式の役割である．したがって，この二つを組み合わせた「CSR 圧縮」を第一段階として適用するのが自然な流れとなる．

第三に，量子化は CSR 形式とは独立に適用可能であり，追加的な圧縮手段として位置づけられる．CSR 圧縮のみでメモリ制約を満たせる場合は量子化を省略でき，さらなる圧縮が必要な場合にのみ追加適用すればよい．

以上の考察に基づき，本研究では次の方針を採る．まず，L1 正則化およびプルーニング後の重み行列を CSR 形式に変換し，推論品質への影響を評価する．CSR 圧縮により MSE が許容範囲内に収まり，かつメモリ制約を満たす場合はそのまま採用する．一方，さらなるメモリ削減が必要な場合や隠れ層次元の拡張を行う場合には，CSR 圧縮モデルに対して量子化を追加適用する．この段階的なアプローチにより，推論精度とメモリ効率のバランスを最適化することを目指す．

4　システムの構成

本章では，実験に用いたシステムのハードウェア構成およびソフトウェア構成について述べる．

4.1　ハードウェア構成

本節では，実験で使用した配線および回路構成について述べる．表 2に Arduino のピン割り当て，表 3に使用機材一覧，図 2に回路図を示す．

表 3: Arduino のピン割り当て

ピン	機能	説明
D2	タクトスイッチ (赤)	最大値・最小値の切り替え
D3	タクトスイッチ (青)	RGB データの読み取りトリガー
A0	照度センサー	フォトトランジスタからのアナログ入力

表 4: 使用機材一覧

機材名	型番	個数
炭素皮膜抵抗 330Ω	-	3
炭素皮膜抵抗 3.3$k\Omega$	-	1
炭素皮膜抵抗 10$k\Omega$	-	2
RGB フルカラー LED	OSTA5131A	1
照度センサー (フォトトランジスタ)	NJL7302L-F3	1
タクトスイッチ (赤・青)	1273HIM-160G-G	2
ジャンパーワイヤ	BBJ-65	13
マイコンボード	Arduino UNO R4 WiFi	1
ブレッドボード	-	1

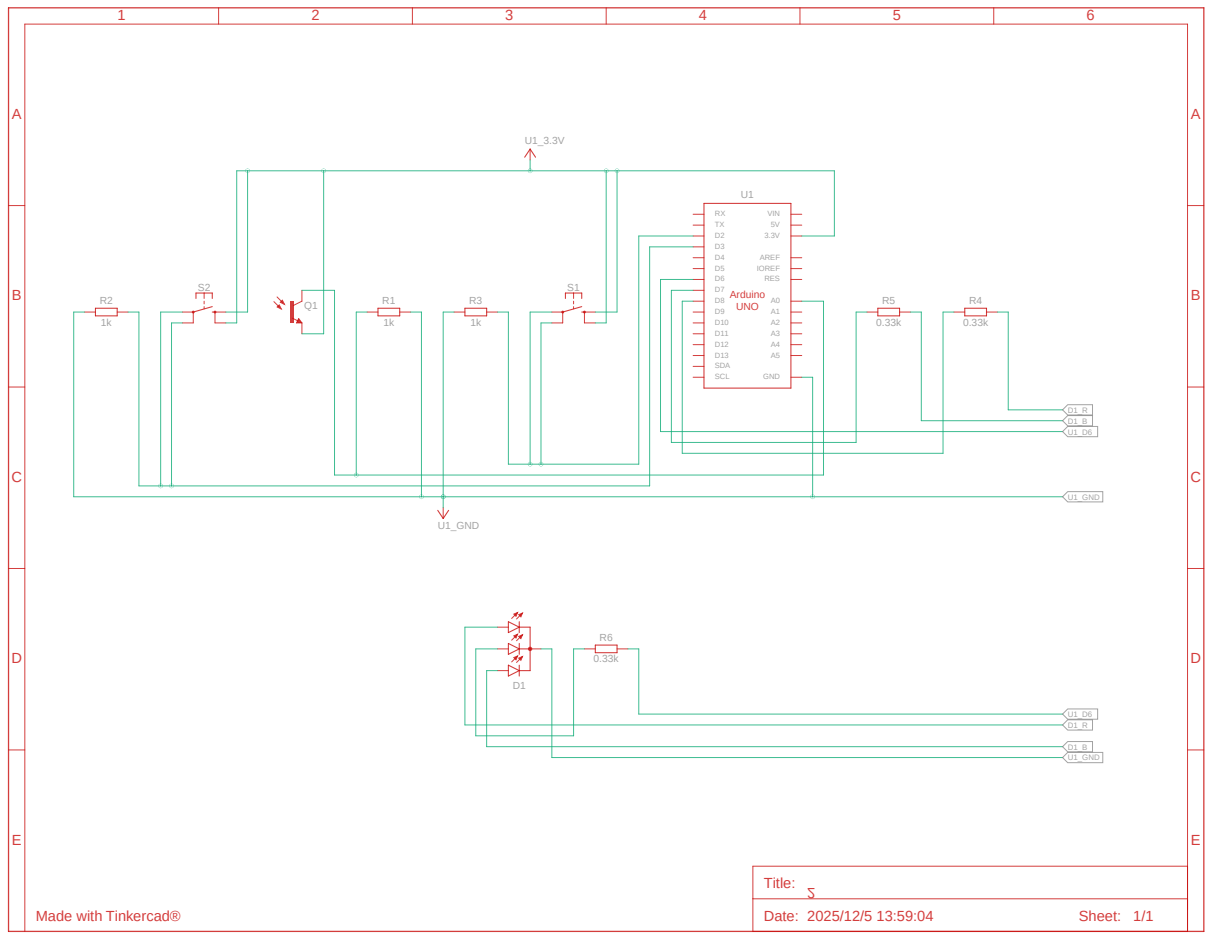

1
2
3
4
5
6
A
B
C
D
E
U1_3.3V
U1
RX
TX
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
SDA
SCL
VIN
5V
3.3V
AREF
IOREF
RES
A0
A1
A2
A3
A4
A5
GND
Arduino
UNO
R2
1k
S2
Q1
R1
1k
R3
1k
S1
R5
0.33k
R4
0.33k
D1_G
D1_B
U1_D6
U1_GND
U1_GND
R6
0.33k
D1
U1_D6
D1_R
D1_B
U1_GND
Made with Tinkercad®
Title:
Date: 2025/12/5 13:59:04
Sheet: 1/1

図 2: 回路図

4.2 ソフトウェア構成

本システムのソフトウェアは，PC 上で実行する学習フェーズと，Arduino 上で実行するデプロイフェーズの 2 段階で構成される．本節では，まずニューラルネットワークの構造を述べた後，各フェーズの処理内容について説明する．

4.2.1 ニューラルネットワークの構造

本システムで使用するニューラルネットワークは，RGB 色補正を目的とした 4 層の全結合ネットワークである．ネットワーク構造の詳細（各層の次元数，活性化関数，推論の数式）については，第 3 章の表**??**を参照されたい．

学習フェーズでは，隠れ層 1 および隠れ層 2 の次元数を 40〜600 の範囲で変化させて複数のモデルを訓練した．デプロイフェーズでは，訓練済みモデルのパラメータを Arduino に組み込み，実機上で推論を実行する．

4.2.2 学習フェーズ

学習フェーズでは，PC 上で Python および PyTorch を用いてニューラルネットワークの学習を行う．学習には Jupyter Notebook（`train_L1_normalization.ipynb`）を使用し，以下の処理を実行する．

1. データの読み込みと前処理
2. L1 正則化を適用したニューラルネットワークの学習
3. プルーニングおよび CSR 形式への変換（隠れ層 1，隠れ層 2 の重み行列に適用）
4. Arduino で読み込み可能なヘッダファイル（`.h`）の生成

L1 正則化により隠れ層 1 および隠れ層 2 の重みが疎化され，プルーニング後に CSR 形式へ変換することで，メモリ使用量を大幅に削減できる．

4.2.3 圧縮手法の実装

第**??**節で述べた圧縮手法に基づき，本システムでは 3 種類の実装を用意した．

1. **密行列形式（AI_Model）**：圧縮なし．2 次元配列で重み行列を格納し，標準的な行列ベクトル積で順伝播を行う．
2. **CSR 形式（AI_CSR）**：プルーニング後に CSR 形式へ変換．`data`，`indices`，`indptr` 配列を用いて非ゼロ要素のみを計算する．
3. **量子化 CSR 形式（AI_CSR_Quantized）**：CSR 形式に加えて int8 量子化を適用．推論時にスケールファクタで逆量子化を行う．

各実装は Arduino スケッチとして独立したディレクトリ（`AI_Model/`，`AI_CSR/`，`AI_CSR_Quantized/`）に配置される．

4.2.4 デプロイフェーズ

デプロイフェーズでは，学習フェーズで生成したパラメータファイルを Arduino に組み込み，実機上で推論を実行する．Arduino コードは，重み行列の格納形式に応じて以下の 3 種類を用意した．

- `AI_Model`：密行列形式（非圧縮）のニューラルネットワーク．隠れ層 1 および隠れ層 2 の重み行列を通常の 2 次元配列として格納する．
- `AI_CSR`：CSR 形式で圧縮したニューラルネットワーク．隠れ層 1 および隠れ層 2 の重み行列を CSR

形式で格納し，メモリ使用量を削減する．

- `AI_CSR_Quantized`：CSR 形式に量子化を追加適用したニューラルネットワーク．隠れ層 1 および隠れ層 2 の重みを int8 形式に量子化し，さらなるメモリ削減を実現する．

次節では，これらのニューラルネットワークを組み込んだ RGB 画像読み取りシステムの処理フローについて述べる．

4.3 RGB 画像読み取りシステムの処理フロー

本節では，Arduino で動作する RGB 画像読み取りシステムの処理フローについて述べる．本システムでは，RGB センサーから画素値を逐次読み取り，前節で述べたニューラルネットワークによって RGB 値を補正した上で，PPM 形式の画像として出力する．

処理は大きく，センサー入出力や画像バッファ管理を行う「共通部分」と，RGB 補正を行う「NN 部分」の 2 つに分けられる．共通部分はニューラルネットワークの格納形式（密行列形式，CSR 形式，量子化 CSR 形式）に依存しない処理であり，NN 部分は格納形式に応じた推論処理を実装する．

4.3.1 共通部分（センサー読み取りと画像バッファ管理）

共通部分では，RGB センサーの読み取り，画素値の正規化，NN 出力の保存，PPM 形式での画像出力などを実装する．主に以下の 5 つの関数から構成される．

■`allocateArray` 関数　画像の高さ N と幅方向の要素数 M を引数として受け取り，2 次元配列（$N \times M$ の画素バッファ）をヒープ領域に動的確保する（図 4）．行ポインタの確保に失敗した場合や，途中の行の確保に失敗した場合には，確保済みメモリを解放して `NULL` を返すことで，メモリ不足に対処する．

■`setup` 関数　シリアル通信の初期化，RGB LED ピンの出力設定および初期消灯，ボタン入力ピンの `INPUT_PULLUP` 設定，割り込みハンドラの登録を行う（図 5）．続いてシリアルモニタから画像の高さ（Height）N を読み取り，幅方向の要素数 $M = 3N$ を計算し，`allocateArray` 関数により画像バッファを動的確保する．確保に成功した場合は，全要素を 0 で初期化する．

■`read` 関数　ループ変数 `count` に応じて R，G，B の順に LED を点灯しながらアナログ入力を行う（図 6）．すなわち，`count==0` のときは R のみ点灯して `rgb[0]` を読み取り，`count==1` では G のみ点灯して `rgb[1]` を読み取り，それ以外（`count==2`）では B のみ点灯して `rgb[2]` を読み取る．3 色の読み取り後にはすべての LED を消灯する．

■`readAndProcess` 関数　まず `read` 関数を呼び出し `rgb[]` に格納された生のセンサー値を取得し，事前にキャリブレーションされた最小値（`r_min`，`g_min`，`b_min`）と最大値（`r_max`，`g_max`，`b_max`）を用いて 0〜1 に正規化した入力ベクトル `RGBInput[3]` を生成する（図 7）．続いて `forward_rgb` 関数を呼び出してニューラルネットワーク推論を行い，補正後の RGB 値である `RGBOutput[3]` を得る．ここで呼び出される `forward_rgb` 関数が，入力層から隠れ層 1，隠れ層 2 を経て出力層に至る順伝播計算を実行する．最後に，出力値を 0〜255 にクリッピングし，B，R，G の順でシリアル出力する．

■`loop` 関数　処理を 2 つに分割して実行する．1 つ目は画素読み取りと保存を行う処理（図 8），2 つ目は Min/Max キャリブレーションを行う処理（図 9）である．

画素保存処理では，`buttonInputPressed` が真のとき，最新の `RGBOutput` を `array[n][m..m+2]` に書き込み，列インデックス `m` を 3 進める．行が埋まれば `n` をインクリメントし，全行の読み取りが完了した場合には PPM 形式で全画素を出力し，`NVIC_SystemReset()` によりリセットする．

キャリブレーション処理では，`buttonMinMaxPressed` が真のとき，変数 `pushed` の偶奇に応じて最小値または最大値を更新する．最小値は `rgb[]` そのものを保存し，最大値は `rgb[]` から最小値を引いた差分として設定する．

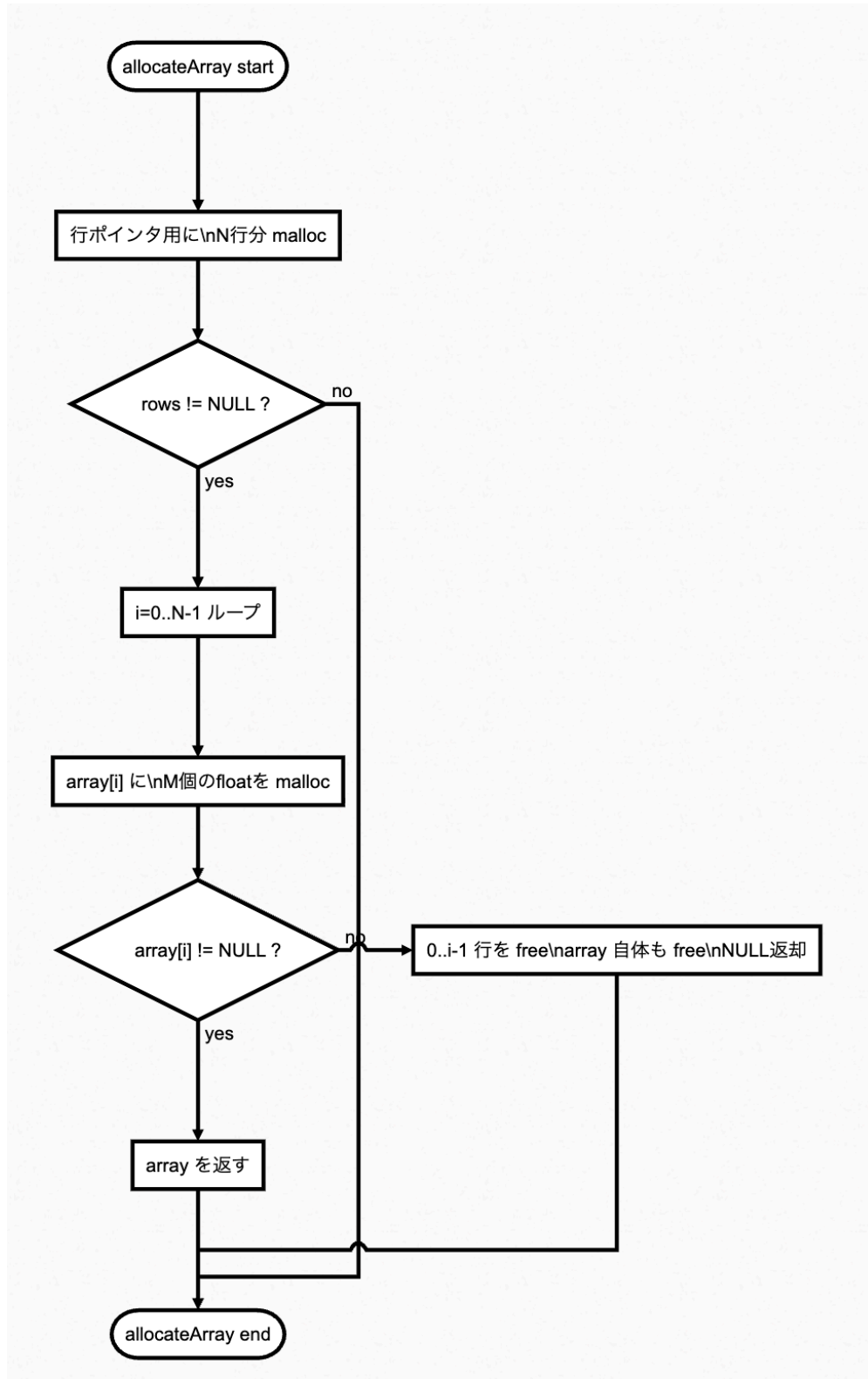
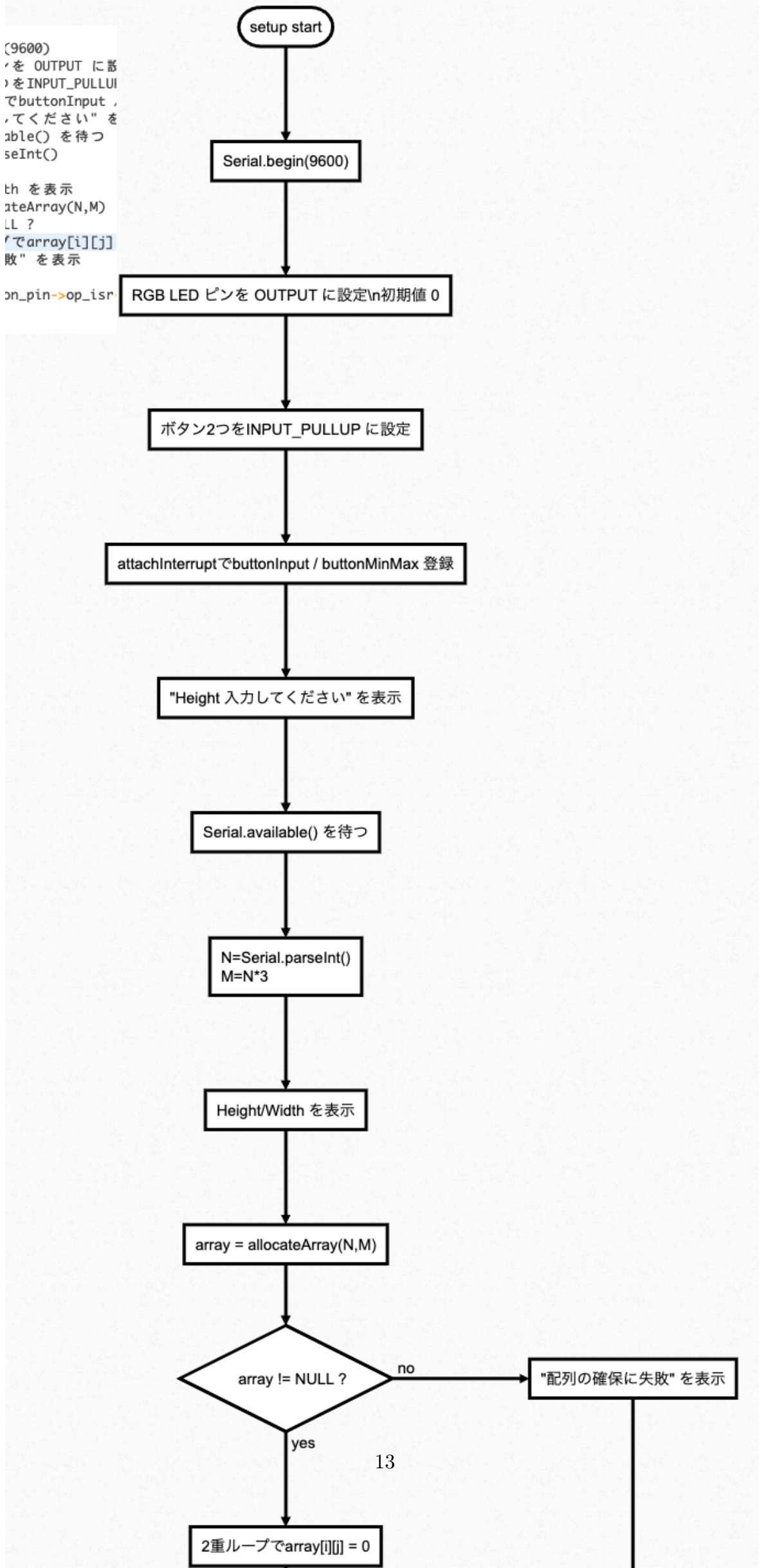


図 3: RGB 画像バッファを動的確保する `allocateArray` 関数のフローチャート

```
(9600)
を OUTPUT に設
をINPUT_PULLUI
でbuttonInput ,
,てください" を
able() を待つ
seInt()

th を表示
ateArray(N,M)
LL ?
でarray[i][j]
敗" を表示

on_pin->op_isr
```



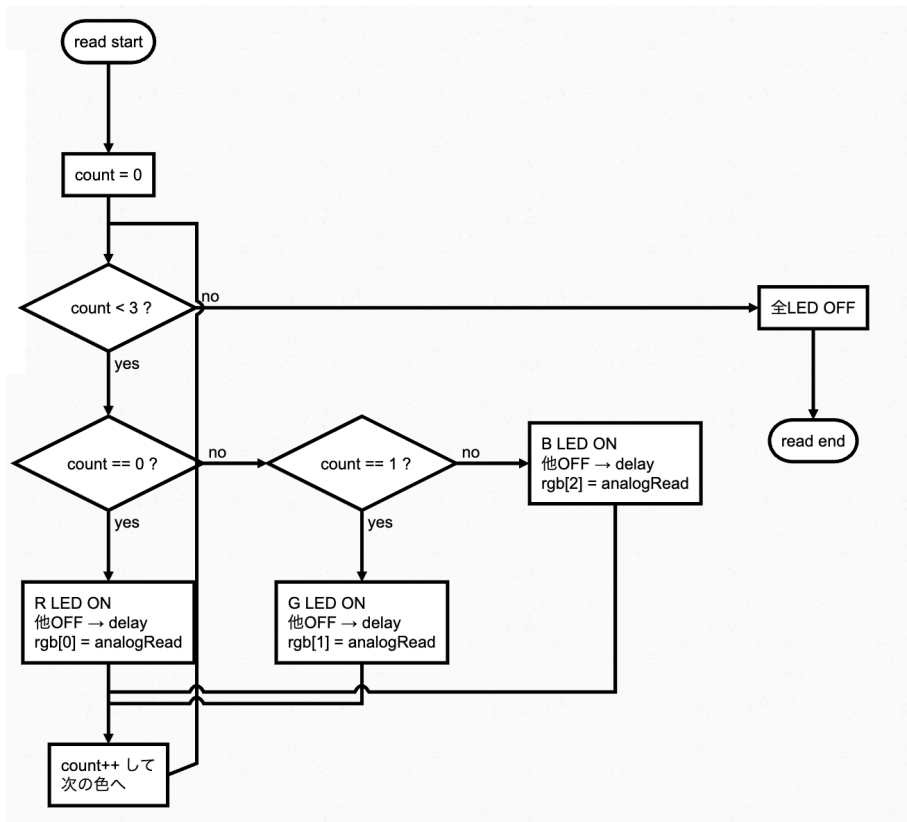


図 5: `read` 関数（R/G/B 順に LED 点灯と読み取りを行う）のフローチャート

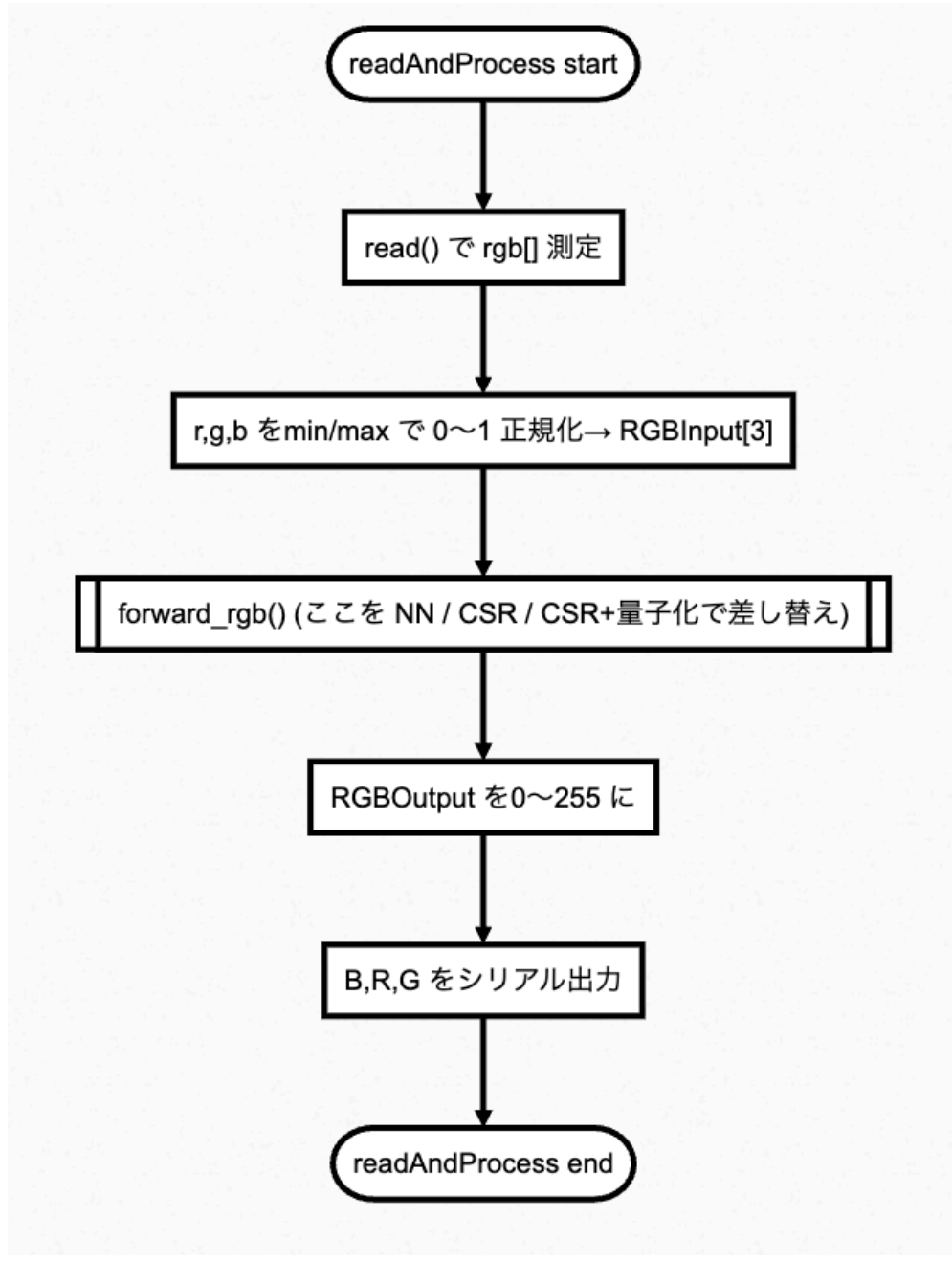
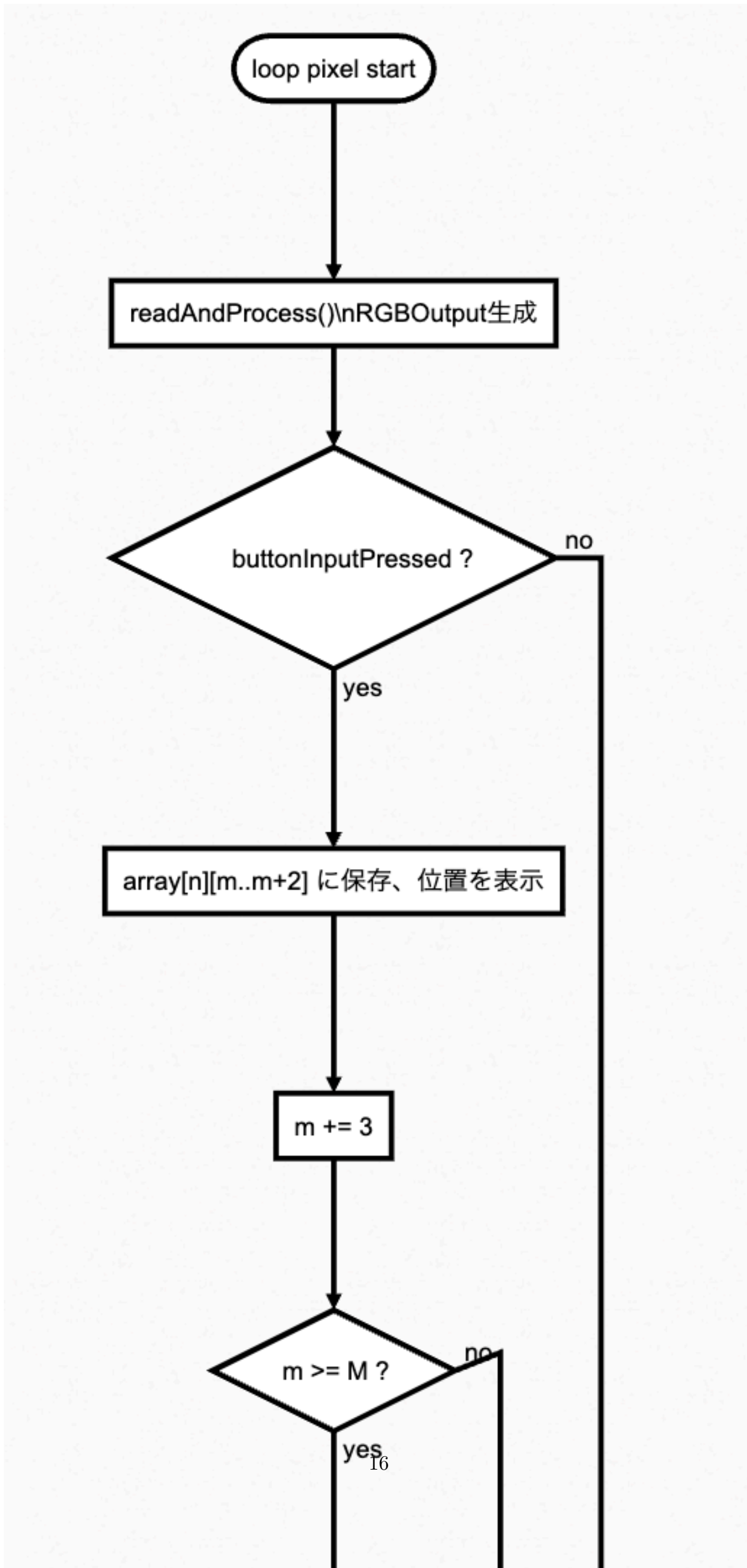


図 6: `readAndProcess` 関数（正規化および NN 推論処理）のフローチャート


loop pixel start
readAndProcess()\nRGBOutput生成
buttonInputPressed ?
no
yes
array[n][m..m+2] に保存、位置を表示
m += 3
m >= M ?
no
yes

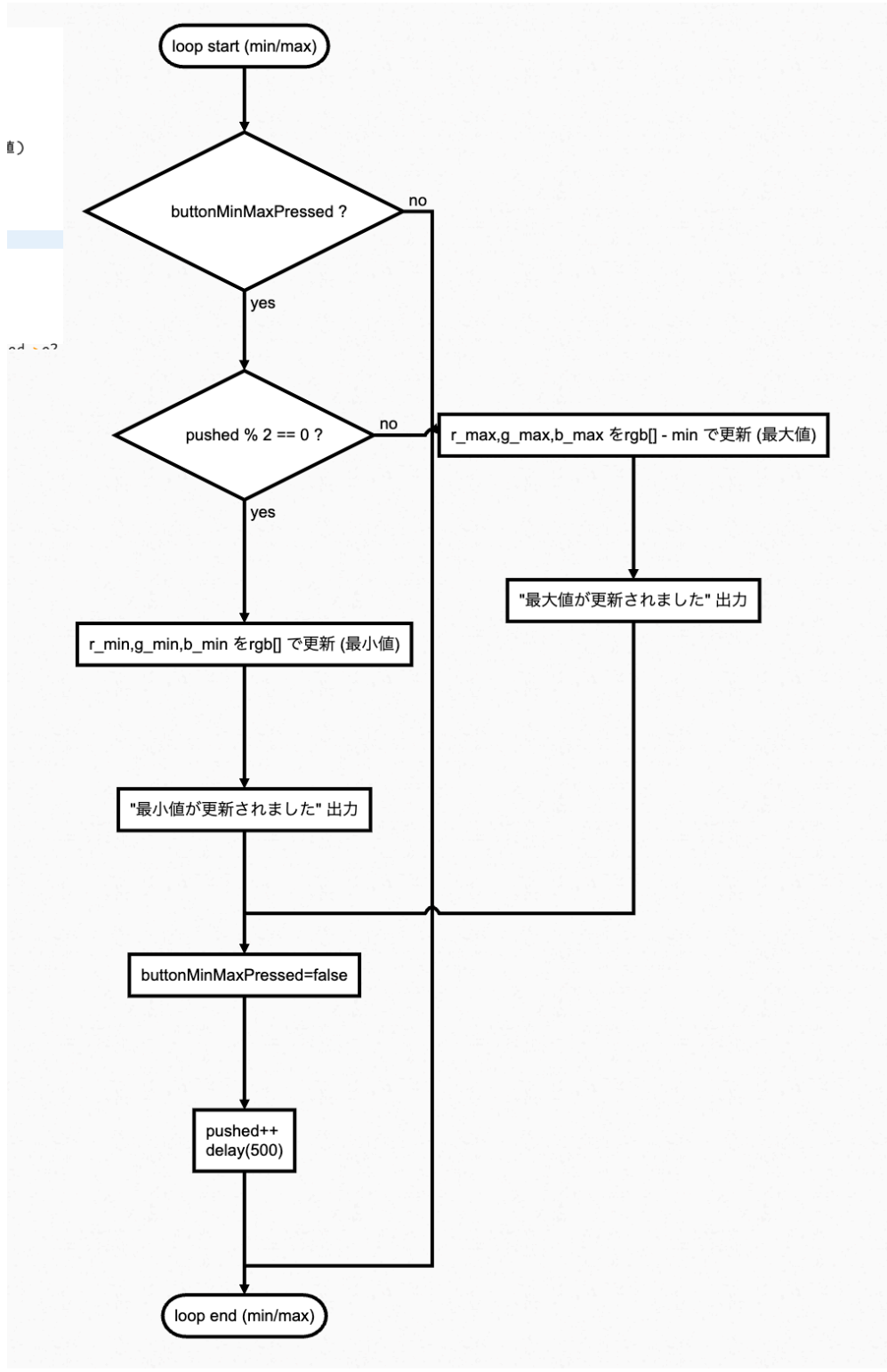


図 8: `loop` 関数における Min/Max キャリブレーション処理のフローチャート

4.3.2　NN 部分（ニューラルネットワーク推論）

NN 部分では，共通部分の `readAndProcess` 関数から呼び出される `forward_rgb` 関数を実装する．この関数は，正規化された RGB 入力（3 次元）を入力層で受け取り，隠れ層 1，隠れ層 2 を経て，出力層から補正された RGB 出力（3 次元）を返す．

`forward_rgb` 関数における順伝播計算の流れは以下のとおりである．

1. 入力層（3 次元）から隠れ層 1 への変換：重み行列 W_1 との積とバイアス b_1 の加算，ReLU 活性化
2. 隠れ層 1 から隠れ層 2 への変換：重み行列 W_2 との積とバイアス b_2 の加算，ReLU 活性化
3. 隠れ層 2 から出力層（3 次元）への変換：重み行列 W_3 との積とバイアス b_3 の加算，Sigmoid 活性化

`forward_rgb` 関数の実装は，重み行列の格納形式に応じて以下の 3 種類が存在する．

■密行列形式（`AI_Model`）　隠れ層 1 および隠れ層 2 の重み行列（W_1，W_2，W_3）を通常の 2 次元配列として格納し，標準的な行列ベクトル積により順伝播計算を行う（図**??**）．実装が単純であるが，メモリ使用量が大きいため，隠れ層の次元拡張に制約がある．

■CSR 形式（`AI_CSR`）　プルーニングにより疎化された隠れ層 1 および隠れ層 2 の重み行列を CSR 形式で格納し，非ゼロ要素のみを用いて行列ベクトル積を計算する（図**??**）．CSR 形式では，値配列（data），列インデックス（indices），行ポインタ（indptr）を用いて疎行列を表現する．メモリ使用量を削減できるため，より大きな隠れ層次元を扱うことが可能である．

■量子化 CSR 形式（`AI_CSR_Quantized`）　CSR 形式に加えて，隠れ層 1 および隠れ層 2 の重みを int8 形式に量子化することで，さらなるメモリ削減を実現する（図**??**）．推論時には，各層ごとに保存されたスケール S とゼロポイント Z を用いて逆量子化を行い，浮動小数点演算に変換して計算する．量子化による精度劣化を最小限に抑えるため，量子化感知訓練（QAT）を適用することも可能である．

5　実験方法

本章では，前章で述べた段階的圧縮アプローチの有効性を検証するために実施した実験について述べる．

5.1　実験の目的と方針

本実験の目的は，Arduino Uno R4 WiFi のメモリ制約下において，ニューラルネットワークの隠れ層次元を拡張しつつ，推論精度を維持することである．具体的には，隠れ層次元を現行の 40 から 600 以上へ拡張可能とし，画像復元品質の向上を実現することを目指す．

この目的を達成するため，重み圧縮手法として CSR（Compressed Sparse Row）形式を第一優先とし，CSR 圧縮のみでは不十分な場合には CSR 圧縮モデルに対して量子化を追加適用する方針とした．L1 正則化による重みの疎化を活かすことで，メモリ効率と推論精度のバランスを最適化する．

実験では，隠れ層次元 40 から開始し，「非圧縮（密行列形式）」「CSR 圧縮」「CSR ＋量子化」の 3 種類のモデルについて MSE を比較し，各圧縮手法が推論品質に与える影響を評価する．

5.2　実験の全体フロー

本実験は，以下の 3 つのフェーズから構成される．

1. **訓練・圧縮フェーズ**：PC 上で PyTorch を用いてニューラルネットワークを訓練し，CSR 形式への変換および量子化を適用してパラメータファイルを生成する
2. **デプロイフェーズ**：生成したパラメータファイルを Arduino Uno R4 WiFi に組み込み，実機へ書き込む
3. **評価フェーズ**：センサーデータを用いて推論を実行し，出力画像と参照画像の MSE を計算して品質を評価する

5.3　実験条件

本実験では，隠れ層次元を変化させながら，各圧縮手法の効果を評価した．表 5に実験条件を示す．

表 5: 実験条件

項目	**設定値**
隠れ層次元	40, 50, 60, 70, 80, 120, 160, 200, 400, 600
プルーニング閾値	0.01（固定）
正則化	L1 正則化（係数: 1×10^{-6}）
学習エポック数	50000
最適化手法	Adam（学習率: 0.01）
疎行列形式	CSR 形式
量子化	int8（必要に応じて適用）

プルーニング閾値を 0.01 に固定した理由は，予備実験において精度と圧縮率のバランスが良好であったためである．また，L1 正則化係数を 1×10^{-6} としたのは，重みの疎化を促進しつつ学習の収束性を維持するためである．

5.4　実験手順

5.4.1　モデルの学習とパラメータ生成

モデルの学習およびパラメータ生成は，Jupyter Notebook（`train_L1_normalization.ipynb`）を用いて実施した．以下に各ステップの詳細を示す．

■ステップ 1：環境セットアップとデータ準備　まず，PyTorch，NumPy，Matplotlib 等の必要なライブラリをインストールし，`cv2`，`torch`，`numpy` 等をインポートした．

次に，PPM 形式のカラーチャート画像を読み込み，参照画像（3×3 および 6×6 のカラーチャート）を準備した．推論用データ（X）と正解データ（Y）は 0〜1 の範囲に正規化し，`formatted_arrays.txt` として保存した．

■ステップ 2：ニューラルネットワークの学習　第 3 章で述べた 4 層全結合ニューラルネットワークを，L1 正則化を適用して学習した．

学習パラメータは表 6のとおりである．

表 6: 学習パラメータ

項目	**設定値**
損失関数	MSE + L1 正則化
L1 正則化係数	1×10^{-6}
最適化手法	Adam
学習率	0.01
エポック数	50000
進捗表示間隔	500 エポックごと

本実験では，隠れ層次元（`HIDDEN_DIM`）を 40, 50, 60, 70, 80, 120, 160, 200, 400, 600 と変化させて複数のモデルを学習した．学習完了後，密行列形式のパラメータファイル（`model_parameters.h`）が自動生成される．このファイルには全層の重みとバイアスが C++ 配列として出力され，圧縮なしのベースラインモデルとして使用する．

■ステップ 3：CSR 形式への変換　学習済みモデルに対し，プルーニング閾値 0.01 でプルーニングを実施した後，重み行列を CSR 形式に変換した．CSR 形式への変換は，隠れ層 1 の重み行列（入力層 → 隠れ層 1）および隠れ層 2 の重み行列（隠れ層 1→ 隠れ層 2）に対して適用した．

変換処理は以下の手順で実施した．

1. 重み行列を取得し，絶対値が 0.01 未満の要素を 0 に設定（プルーニング）
2. 非ゼロ要素を抽出し，CSR 形式の 3 つの配列に変換：
   - `data`：非ゼロ要素の値配列（float32）
   - `indices`：列インデックス配列（uint16_t）
   - `indptr`：行ポインタ配列（uint16_t）
3. スパース性（非ゼロ率）と圧縮率を算出

変換後のパラメータは, 各隠れ層次元に応じて `model_parameters_csr40.h`, `model_parameters_csr80.h` 等のファイル名で出力した．

■ステップ 4：量子化の適用（オプション）　CSR 圧縮のみではメモリ制約を満たせない場合，CSR 形式のパラメータに対して int8 量子化を追加適用した．

量子化処理は以下の手順で実施した．

1. CSR 形式の値配列（`data`）に対してスケールファクタを計算：

$$\text{scale} = \frac{127.0}{\max(|\texttt{data}|)} \tag{5}$$

2. 各要素を量子化し，−128〜127 の範囲にクリッピング：

$$\texttt{quantized} = \text{clip}(\text{round}(\texttt{data} \times \text{scale}), -128, 127) \tag{6}$$

3. 逆量子化誤差を算出して精度劣化を確認

量子化は隠れ層 1 および隠れ層 2 の重み行列に対して，各層ごとに独立したスケールファクタを用いて適用した．量子化後のパラメータは，`model_parameters_csr_quantized40.h` 等のファイル名で出力した．

■次元変更時の手順　異なる次元で実験する場合は，以下の手順を実行した．

1. Jupyter Notebook で `HIDDEN_DIM` を変更
2. 学習を実行し，密行列形式パラメータを生成
3. CSR 変換を実行し，CSR 形式パラメータを生成
4. 必要に応じて量子化を適用
5. 生成ファイルを Arduino プロジェクトにコピー
6. Arduino コードの `HIDDEN_DIM` を同じ値に設定

5.4.2 Arduino へのデプロイ

生成したパラメータファイルを Arduino プロジェクトに組み込み，実機へ書き込みを行った．

■ステップ 1：パラメータファイルの配置　生成されたパラメータファイルを，表 7に示す対応する Arduino プロジェクトフォルダにコピーした．

表 7: パラメータファイルの配置

生成ファイル	コピー先	用途
`model_parameters*.h`	`Arduino/AI_Model/`	密行列形式
`model_parameters_csr*.h`	`Arduino/AI_CSR/`	CSR 形式
`model_parameters_csr_quantized*.h`	`Arduino/AI_CSR_Quantized/`	CSR+ 量子化

■ステップ 2：Arduino コードの設定　各 Arduino プロジェクト内の `.ino` ファイルにおいて，`HIDDEN_DIM` マクロを学習時と同一の値に設定し，対応するパラメータファイルをインクルードした．

```
#define HIDDEN_DIM 80   // 学習時と同じ値に設定
#include "model_parameters_csr80.h"   // 対応するファイルをインクルード
```

Listing 1: Arduino コードの設定例（80 次元の場合）

`HIDDEN_DIM` の値とパラメータファイルの次元が一致していない場合，正常に動作しないため注意が必要である．

■ステップ 3：コンパイルとアップロード　Arduino IDE を用いて，ボード設定を「Arduino UNO R4 WiFi」に設定し，コンパイル・アップロードを行った．コンパイル時に表示される SRAM 使用量が 32 KB を超える場合はアップロードに失敗するため，その場合はより高い圧縮率の手法を適用する必要がある．

5.4.3　スキャン実行とデータ取得

Arduino に接続した RGB カラーセンサを用いて参照画像をスキャンし，推論結果を PPM 形式の画像ファイルとして出力した．

■ステップ 1：キャリブレーション　センサの感度は環境光や個体差により変動するため，測定前にキャリブレーションを行った．2 つのタクトスイッチを用いて，黒色サンプルで最小値を，白色サンプルで最大値を記録し，これらを基準として読み取り範囲を正規化した．

■ステップ 2：カラーチャートの読み取り　キャリブレーション完了後，3×3 のカラーチャートを 1 セルずつセンサで読み取った．取得した RGB 値はニューラルネットワークによる推論を経て補正され，全 9 セルの読み取りが完了するまで繰り返した．

■ステップ 3：PPM 画像の出力　取得した RGB データは，シリアル通信を介して PC 上のスキャナプログラム（`scanner.c`）に送信され，PPM 形式の画像ファイルとして出力された．出力ファイルは実験条件ごとにフォルダを分けて保存し，各条件につき 10 回程度のスキャンを実施した．

5.5　評価方法

推論精度の評価指標として，第 2 章で定義した平均二乗誤差（MSE）を用いた（式**??**，式**??**参照）．MSE の値が小さいほど，参照画像との差異が少なく，色補正の精度が高いことを示す．

評価には自作の MSE 計算ツール（`mse_calculator.html`）を使用した．このツールはブラウザ上で動作し，参照画像とスキャン結果の PPM ファイルをドラッグ&ドロップすることで MSE を算出できる．各実験条件のすべてのスキャン結果について MSE を算出し，平均値・最小値・最大値を求めた．

5.6　比較対象

本実験では，以下の 3 種類のモデルを比較対象とした．

1. **密行列形式（非圧縮）**：L1 正則化のみを適用し，圧縮処理を行わないベースラインモデル
2. **CSR 圧縮**：プルーニング後に CSR 形式へ変換したモデル（隠れ層 1 および隠れ層 2 に適用）
3. **CSR ＋量子化**：CSR 圧縮に加えて int8 量子化を適用したモデル（隠れ層 1 および隠れ層 2 に適用）

これらのモデルについて，各隠れ層次元における MSE とメモリ使用量を比較し，段階的圧縮アプローチの有効性を検証した．最終的な目標は，MSE を許容範囲内に抑えつつ，隠れ層次元を 600 以上へ拡張可能とすることである．

6　実験結果

本章では，前章で述べた実験方法に基づいて実施した実験の結果を示す．

6.1　MSE 測定結果

各圧縮手法および隠れ層次元における MSE 測定結果を表 8に示す．

表 8: 圧縮手法別 MSE 比較

圧縮手法	隠れ層次元	MSE 最小値	MSE 最大値	MSE 平均値
非圧縮（密行列形式）				
密行列形式	40	277.67	792.85	595.95
密行列形式	50	231.11	535.59	406.93
密行列形式	60	138.00	423.22	305.48
密行列形式	70	コンパイルエラー	コンパイルエラー	コンパイルエラー
CSR 圧縮				
CSR 圧縮	40	324.00	692.37	518.58
CSR 圧縮	80	225.19	729.26	464.86
CSR 圧縮	120	295.37	814.33	620.49
CSR 圧縮	160	295.52	1179.59	589.08
CSR 圧縮	200	291.41	979.85	493.35
CSR 圧縮	400	390.89	1978.81	864.40
CSR 圧縮	600	253.44	2143.81	670.69
CSR 圧縮	800	コンパイルエラー	コンパイルエラー	コンパイルエラー
CSR ＋量子化				
CSR ＋量子化	40	278.78	588.15	408.97
CSR ＋量子化	80	200.22	323.96	270.68
CSR ＋量子化	120	169.37	310.63	244.83
CSR ＋量子化	160	–	–	–
CSR ＋量子化	200	–	–	–
CSR ＋量子化	400	–	–	–
CSR ＋量子化	600	–	–	–
CSR ＋量子化	800	コンパイルエラー	コンパイルエラー	コンパイルエラー

また，同一次元における圧縮手法間の比較を容易にするため，表 9に隠れ層次元別の結果を示す．

6.2　コンパイル時間とメモリ使用量

各条件における Arduino のコンパイル時間とメモリ使用量を表 10に示す．

6.3　詳細データ

以下に，各実験条件における個別のスキャン結果を示す．

表 9: 隠れ層次元別 MSE 比較（平均値）

隠れ層次元	非圧縮	CSR 圧縮	CSR ＋量子化
40	595.95	518.58	408.97
50	406.93	–	–
60	305.48	–	–
70	コンパイルエラー	–	–
80	コンパイルエラー	464.86	270.68
120	コンパイルエラー	620.49	244.83
160	コンパイルエラー	589.08	–
200	コンパイルエラー	493.35	–
400	コンパイルエラー	864.40	–
600	コンパイルエラー	670.69	–
800	コンパイルエラー	コンパイルエラー	コンパイルエラー

6.3.1　非圧縮（密行列形式）

表 11に，密行列形式（40 次元）の詳細な測定結果を示す．

表 12に，密行列形式（50 次元）の詳細な測定結果を示す．

表 13に，密行列形式（60 次元）の詳細な測定結果を示す．

6.3.2　CSR 圧縮

表 14に，CSR 圧縮モデルの詳細な測定結果を示す．

6.3.3　CSR ＋量子化

表 15に，CSR ＋量子化モデルの詳細な測定結果を示す．

6.4　結果のまとめ

現時点で得られた実験結果をまとめると，以下のとおりである．

1. 非圧縮（密行列形式）では，40 次元で MSE 平均値 730.13，50 次元で 406.93，60 次元で 305.48 と，次元を増やすほど精度が向上したが，70 次元ではメモリ不足によりコンパイルエラーが発生した．
2. CSR 圧縮（40 次元）の MSE 平均値は 518.58 であり，非圧縮（40 次元）の 730.13 と比較して約 29% 改善した．
3. CSR 圧縮（80 次元）の MSE 平均値は 464.86 であり，非圧縮（40 次元）と比較して約 36% 改善した．
4. CSR 圧縮により，隠れ層次元を 600 次元まで拡張しても Arduino Uno R4 WiFi のメモリ制約内で動作可能であった（メモリ使用量 67%）．
5. CSR ＋量子化により，さらにメモリ効率が向上し，600 次元でもメモリ使用量 66% で動作可能であった．
6. CSR 圧縮（120 次元以上）および CSR ＋量子化モデルについては，MSE 測定を実施予定である．

これらの結果から，CSR 形式による圧縮は推論精度を維持しつつ隠れ層次元の拡張を可能にする有効な手法であることが示唆される．詳細な考察は次章で述べる．

表 10: コンパイル時間とメモリ使用量

圧縮手法	隠れ層次元	コンパイル時間	メモリ使用量
非圧縮（密行列形式）			
密行列形式	40	4.303s	14524 バイト（44%）
密行列形式	50	4.589s	18444 バイト（56%）
密行列形式	60	4.831s	23164 バイト（70%）
密行列形式	70	コンパイルエラー	コンパイルエラー
CSR 圧縮			
CSR 圧縮	40	3.783s	8488 バイト（25%）
CSR 圧縮	80	4.037s	9508 バイト（29%）
CSR 圧縮	120	4.062s	10520 バイト（32%）
CSR 圧縮	160	4.042s	11332 バイト（34%）
CSR 圧縮	200	4.027s	12332 バイト（37%）
CSR 圧縮	400	4.316s	17204 バイト（52%）
CSR 圧縮	600	4.802s	21976 バイト（67%）
CSR 圧縮	800	コンパイルエラー	コンパイルエラー
CSR ＋量子化			
CSR ＋量子化	40	3.772s	8160 バイト（24%）
CSR ＋量子化	80	3.809s	9148 バイト（27%）
CSR ＋量子化	120	4.028s	10132 バイト（30%）
CSR ＋量子化	160	4.033s	12000 バイト（36%）
CSR ＋量子化	200	4.033s	12000 バイト（36%）
CSR ＋量子化	400	4.307s	16836 バイト（51%）
CSR ＋量子化	600	4.801s	21684 バイト（66%）
CSR ＋量子化	800	コンパイルエラー	コンパイルエラー

表 11: 密行列形式（40 次元）の詳細測定結果

ファイル	MSE	ファイル	MSE
nn1.ppm	484.04	nn6.ppm	715.63
nn2.ppm	713.74	nn7.ppm	523.63
nn3.ppm	733.93	nn8.ppm	792.85
nn4.ppm	277.67	nn9.ppm	284.15
nn5.ppm	782.33	nn10.ppm	651.52

表 12: 密行列形式（50 次元）の詳細測定結果

ファイル	MSE	ファイル	MSE
nn50_1.ppm	369.30	nn50_6.ppm	231.11
nn50_2.ppm	486.26	nn50_7.ppm	463.85
nn50_3.ppm	444.11	nn50_8.ppm	307.26
nn50_4.ppm	472.78	nn50_9.ppm	342.67
nn50_5.ppm	416.04	nn50_10.ppm	535.59

表 13: 密行列形式（60 次元）の詳細測定結果

ファイル	MSE	ファイル	MSE
nn60_1.ppm	193.07	nn60_6.ppm	332.41
nn60_2.ppm	138.00	nn60_7.ppm	297.22
nn60_3.ppm	376.22	nn60_8.ppm	417.44
nn60_4.ppm	227.26	nn60_9.ppm	240.33
nn60_5.ppm	423.22	nn60.ppm	409.30

表 14: CSR 圧縮の詳細測定結果

次元	ファイル	MSE	備考
40	csr1.ppm	600.00	測定完了
	csr2.ppm	517.48	
	csr3.ppm	569.78	
	csr4.ppm	659.04	
	csr5.ppm	324.00	
	csr6.ppm	365.07	
	csr7.ppm	522.59	
	csr8.ppm	495.04	
	csr9.ppm	692.37	
	csr10.ppm	440.41	
80	1.ppm	644.59	測定完了
	2.ppm	459.37	
	3.ppm	361.33	
	4.ppm	225.19	
	5.ppm	729.26	
	6.ppm	408.15	
	7.ppm	380.59	
	8.ppm	499.85	
	9.ppm	437.44	
	10.ppm	472.85	
120	1.ppm	724.67	測定完了
	2.ppm	295.37	
	3.ppm	663.78	
	4.ppm	514.59	
	5.ppm	694.22	
	6.ppm	710.96	
	7.ppm	656.44	
	8.ppm	584.59	
	9.ppm	814.33	
	10.ppm	545.93	
160	1.ppm	394.30	測定完了
	2.ppm	516.96	
	3.ppm	663.78	
	4.ppm	668.63	
	5.ppm	1179.59	
	6.ppm	353.07	
	7.ppm	686.56	
	8.ppm	295.52	
	9.ppm	434.04	
	10.ppm	698.33	
200	1.ppm	563.52	
	2.ppm	292.41	
	3.ppm	235.52	
	4.ppm	675.11	
	5.ppm	662.07	

表 15: CSR ＋量子化の詳細測定結果

次元	ファイル	MSE	備考
40	q1.ppm	289.30	測定完了
	q2.ppm	304.44	
	q3.ppm	381.19	
	q4.ppm	588.15	
	q5.ppm	278.78	
	q6.ppm	471.30	
	q7.ppm	582.63	
	q8.ppm	324.44	
	q9.ppm	321.37	
	q10.ppm	548.11	
80	q1.ppm	235.74	測定完了
	q2.ppm	237.63	
	q3.ppm	200.22	
	q4.ppm	292.22	
	q5.ppm	282.70	
	q6.ppm	268.26	
	q7.ppm	323.96	
	q8.ppm	270.19	
	q9.ppm	313.37	
	q10.ppm	282.48	
120	q1.ppm	297.70	測定完了
	q2.ppm	180.11	
	q3.ppm	239.15	
	q4.ppm	310.63	
	q5.ppm	309.41	
	q6.ppm	169.37	
	q7.ppm	228.59	
	q8.ppm	257.85	
	q9.ppm	196.19	
	q10.ppm	259.30	
160	–	–	測定未実施
200	–	–	測定未実施
400	–	–	測定未実施
600	–	–	測定未実施
800	–	–	コンパイルエラー

参考文献

[1] B. Jacob et al., “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” CVPR, 2018.

[2] Song Han et al., “Learning both Weights and Connections for Efficient Neural Networks,” NeurIPS, 2015.