

# 2025 年度 アジャイルワーク 報告書

24G1089 武本 龍

2025 年 12 月 11 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。

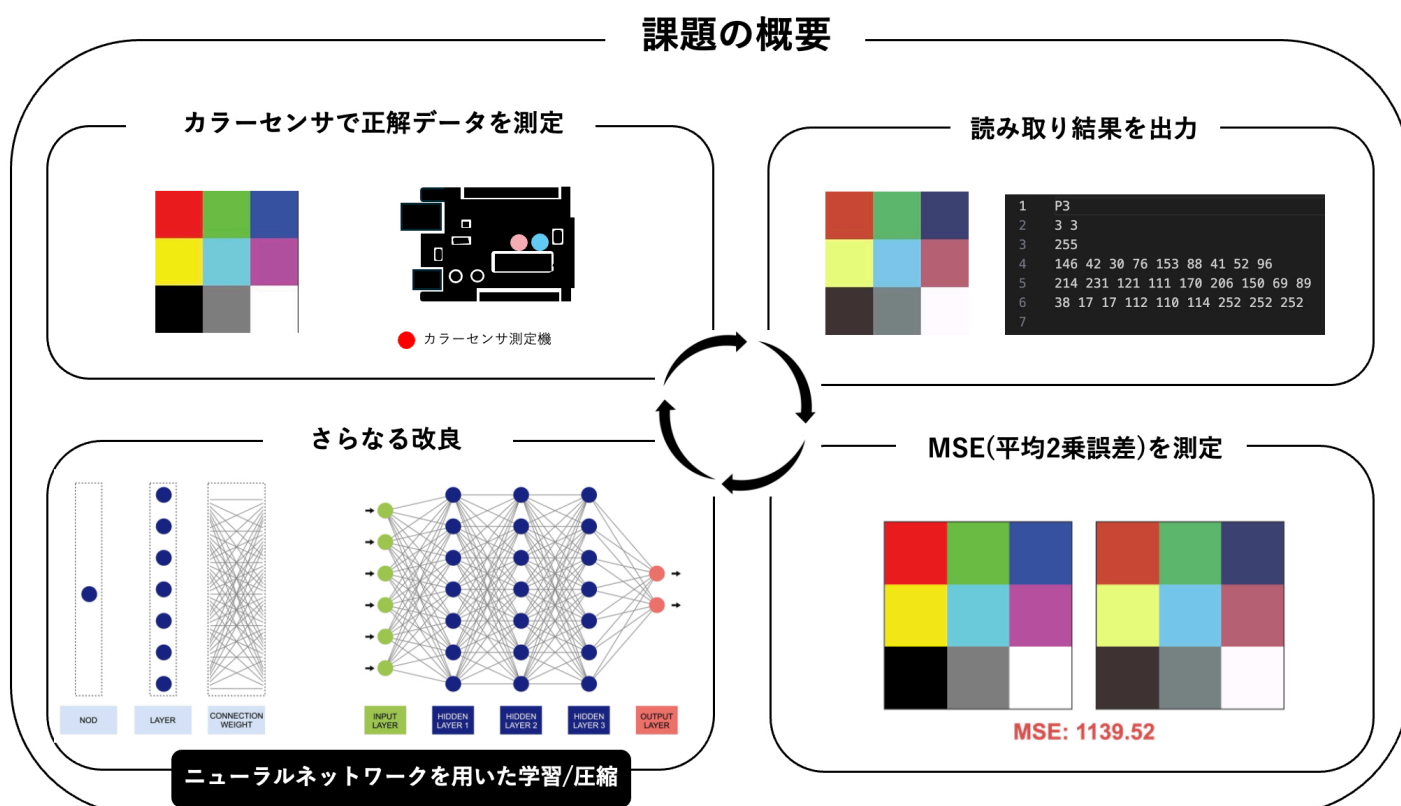


図 1: 実験の全体構成

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  $3 \times 3$  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) 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。

$$\text{MSE} = \frac{1}{m \times n} \sum_{i=1}^m \sum_{j=1}^n (I(i, j) - K(i, j))^2 \quad (1)$$

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 チャンネルごとの差の 2 乗を加算し、チャン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：

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

## 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 40 次元（40 個のニューロンが全結合），出力層は再び RGB の 3 次元とした．各ニューロンにおける推論は以下の式で表される．

$$y = x_1w_1 + x_2w_2 + \cdots + x_{n-1}w_{n-1} + b \quad (3)$$

ここで， $w_i$  は重み， $b$  はバイアスを表す．算出された値は ReLU 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．さらに，本研究では隠れ層の次元拡張性を検証するため，40 次元に加えて 80 次元，120 次元のモデルについても実験を行った．

### 3.2 隠れ層の圧縮手法

#### 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 圧縮手法の概観

ニューラルネットワークの圧縮手法は多岐にわたるが，本研究のようなマイクロコントローラ環境では，実装の複雑さと圧縮効果のバランスが重要となる．代表的な手法として，プルーニング（枝刈り），疎行列表現（CSR 形式など），量子化が挙げられる．

これらの手法は独立に適用することも，組み合わせて適用することも可能である．以下では，各手法の原理と特徴を述べた上で，本研究における適用方針を示す．

### 3.2.3 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

### 3.2.4 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低い
- 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。

### 3.2.5 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

$$r = S(q - Z) \quad (4)$$

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、チャン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

### 3.2.6 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 1: 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           |

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，実験で使っ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 2: 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 3: 使用機材一覧

| 機材名                 | 型番                  | 個数 |
|---------------------|---------------------|----|
| 炭素皮膜抵抗 $330\Omega$  | -                   | 3  |
| 炭素皮膜抵抗 $3.3k\Omega$ | -                   | 1  |
| 炭素皮膜抵抗 $10k\Omega$  | -                   | 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  |

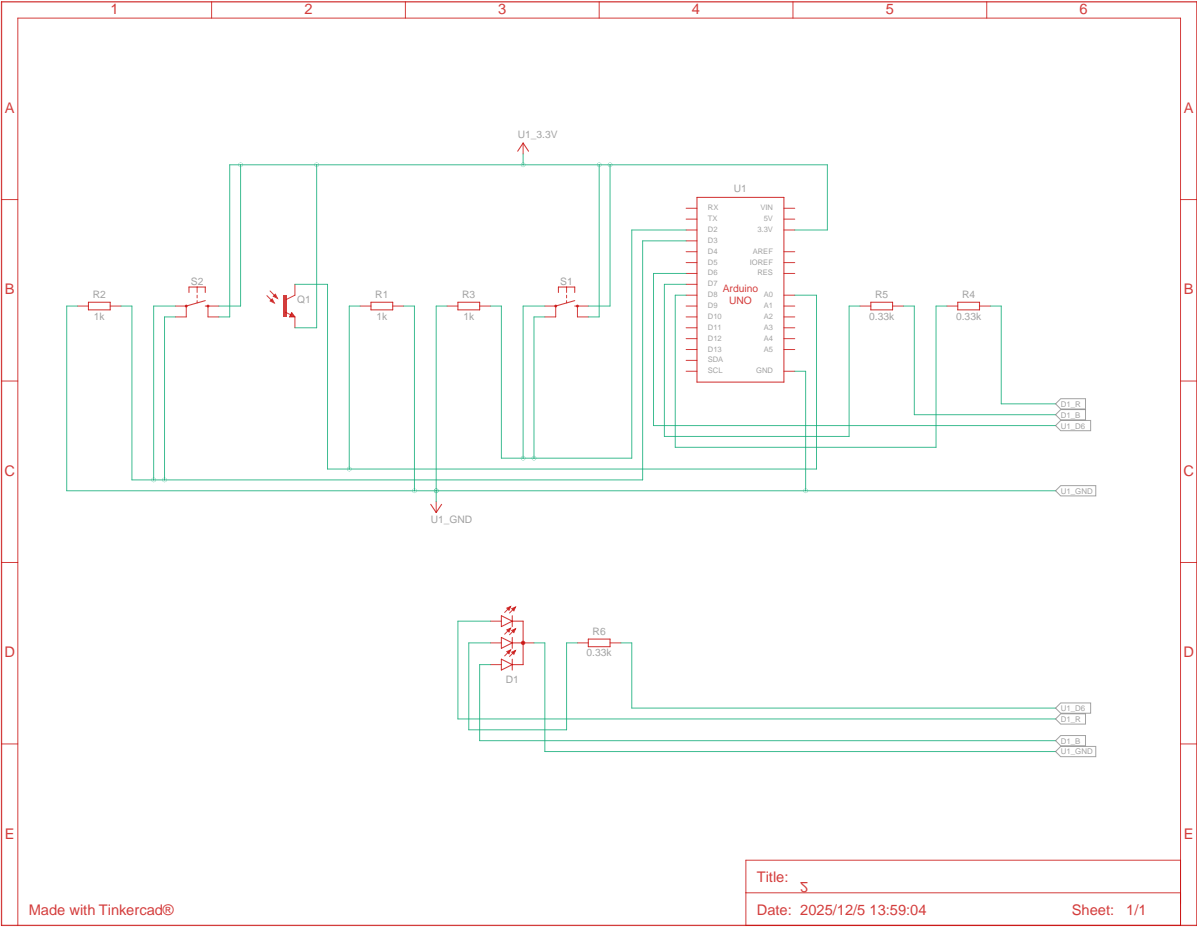


图 2: 回路图

## 4.2 ソフトウェア構成

本システムのソフトウェアは，学習フェーズとデプロイフェーズの2段階で構成される．

### 4.2.1 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
4. Arduino で読み込み可能なヘッダファイル (`.h`) の生成

### 4.2.2 デプロイフェーズ

デプロイフェーズでは，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 コードは用途に応じて以下の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
- `AI_CSR`：CSR 形式で圧縮したニューラルネットワーク
- `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

```
1 % TODO: コードを記載
```

Listing 1: Arduino メインプログラム（抜粋）

```
1 % TODO: コードを記載
```

Listing 2: CSR 形式での行列ベクトル積演算



## 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 100～200 へ拡張可能とし、画像復元品質の向上と読み取り時間の短縮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 4 に実験条件を示す。

表 4: 実験条件

| 項目       | 設定値             |
|----------|-----------------|
| 隠れ層次元    | 40, 80, 120     |
| プルーニング閾値 | 0.01 (固定)       |
| 正則化      | L1 正則化          |
| 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。

## 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：データ準備

まず、学習に必要なライブラリ (PyTorch, NumPy, SciPy 等) をインポートした。次に、参照画像として PPM 形式のカラーチャート画像を読み込み、推論用データを準備した。

#### ■ステップ 2：ニューラルネットワークの学習

L1 正則化を適用したニューラルネットワークを学習した。学習時には、隠れ層次元を指定するパラメータ `HIDDEN_DIM` を設定した。本実験では、`HIDDEN_DIM` を 40, 80, 120 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。このファイルは、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。変換処理では、閾値以下の重みを 0 として扱い、非ゼロ要素のみを値配列 (`data`)、列インデックス (`indices`)、行ポインタ (`indptr`) として抽出した。変換後のパラメータは `model_parameters_csr.h` として出力される。

各隠れ層次元に対してこの処理を行い、生成されたファイルは以下のようにリネームして管理した。

- 40 次元：`model_parameters_csr40.h`
- 80 次元：`model_parameters_csr80.h`
- 120 次元：`model_parameters_csr120.h`

#### ■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、またはさらなる圧縮が必要な場合には、CSR 形式のパラメータに対して `int8` 量子化を追加適用した。量子化処理では、`float32` 形式の重みを `int8` 形式に変換し、スケールおよびゼロポイントを算出した。量子化後のパラメータは `model_parameters_csr_quantized.h` として出力される。

### 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。以下に詳細な手順を示す。

#### ■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、対応する Arduino プロジェクトフォルダにコピーした。表 5 にファイルの配置先を示す。

表 5: 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/         | 密行列形式モデル    |
| model_parameters_csr*.h          | Arduino/AI_CSR/           | CSR 形式モデル   |
| model_parameters_csr_quantized.h | Arduino/AI_CSR_Quantized/ | CSR+ 量子化モデル |

#### ■ステップ 2：Arduino コードの設定

各 Arduino プロジェクト内の .ino ファイルを開き、以下の設定を行った。

まず、HIDDEN\_DIM マクロを学習時と同一の値に設定した。

```
1 #define HIDDEN_DIM 80 // 学習時と同じ値に設定
```

Listing 3: 隠れ層次元の設定

次に、使用するパラメータファイルをインクルードした。

```
1 #include "model_parameters_csr80.h" // 対応するファイルを指定
```

Listing 4: パラメータファイルのインクルード

なお、HIDDEN\_DIM の値と読み込む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、以下の手順でコンパイルおよびアップロードを行った。

1. Arduino IDE で対象の .ino ファイルを開く
2. ボード設定を「Arduino UNO R4 WiFi」に設定する
3. 接続されているシリアルポートを選択する
4. 「検証」ボタンでコンパイルエラーがないことを確認する
5. 「アップロード」ボタンで Arduino に書き込む

コンパイル時には、メモリ使用量が表示される。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。スキャンは以下の手順で実施した。

#### ■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。具体的には、2 つのタクトスイッチを用いたプログラムにより、黒と白のサンプルを事前に測定し、センサ出力の最小値および最大値を取得した。

キャリブレーション手順は以下のとおりである。

1. 赤色タクトスイッチ（D2 ピン）を押し、キャリブレーションモードに入る
2. 黒色サンプルをセンサにかざし、青色タクトスイッチ（D3 ピン）を押して最小値を記録する
3. 白色サンプルをセンサにかざし、青色タクトスイッチを押して最大値を記録する
4. 赤色タクトスイッチを押し、キャリブレーションモードを終了する

これらの値を基準として読み取り範囲を正規化することで、安定した RGB 値の取得を可能とした。

#### ■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センサで読み取った。読み取り手順は以下のとおりである。

1. カラーチャートの各セルにセンサを順番にかざす
2. 青色タクトスイッチ（D3 ピン）を押して RGB 値を取得する
3. 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る
4. 全 9 セルの読み取りが完了するまで繰り返す

#### ■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（Portable Pixel Map）形式の画像ファイルとして出力された。PPM 形式は非圧縮のラスタ画像フォーマットであり、画素値を直接比較できるため、後続の MSE 評価に適している。

出力された PPM ファイルは、実験条件ごとに以下のフォルダに整理して保存した。

- 密行列形式（40 次元）：`ppm/AI_Model.data/`
- CSR 形式（40 次元）：`ppm/csr_40_0.01/`
- CSR 形式（80 次元）：`ppm/csr_80_0.01/`
- CSR 形式（120 次元）：`ppm/csr_120_0.01/`

各実験条件につき複数回（10 回程度）のスキャンを実施し、結果のばらつきを考慮した。

## 5.5 評価方法

推論精度の評価指標として、平均二乗誤差（MSE: Mean Squared Error）を用いた。MSE は以下の式で定義される。

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (5)$$

ここで、 $N$  は画素数、 $y_i$  は参照画像の画素値、 $\hat{y}_i$  はスキャン結果の画素値である。

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、2つの PPM 画像をドラッグ&ドロップするだけで MSE を算出できる。

評価手順は以下のとおりである。

1. ブラウザで `mse_calculator.html` を開く
2. 参照画像 (`ppm/reference_image1.ppm`) をドラッグ&ドロップする
3. スキャン結果の PPM ファイルをドラッグ&ドロップする
4. 算出された MSE 値を記録する
5. 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・最大値を求める

## 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
3. **CSR + 量子化**：CSR 圧縮に加えて int8 量子化を適用したモデル

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 100~200 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 6 に示す．

表 6: 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    | 484.04   | 1559.96  | 730.13   |
| 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   | —        | —        | —        |
| CSR 圧縮     | 160   | —        | —        | —        |
| CSR 圧縮     | 200   | —        | —        | —        |
| CSR 圧縮     | 400   | —        | —        | —        |
| CSR 圧縮     | 500   | —        | —        | —        |
| CSR 圧縮     | 600   | —        | —        | —        |
| CSR 圧縮     | 800   | オーバーフロー  | オーバーフロー  | オーバーフロー  |
| CSR + 量子化  |       |          |          |          |
| CSR + 量子化  | 40    | —        | —        | —        |
| CSR + 量子化  | 80    | —        | —        | —        |
| CSR + 量子化  | 120   | —        | —        | —        |
| 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，同次元における圧縮手法間の比較を容易にするため，表 7 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 ?? に示す．

表 7: 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    | 730.13   | 518.58  | —        |
| 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  | —        |
| 120   | コンパイルエラー | —       | —        |
| 160   | コンパイルエラー | —       | —        |
| 200   | コンパイルエラー | —       | —        |
| 400   | コンパイルエラー | —       | —        |
| 500   | コンパイルエラー | —       | —        |
| 600   | コンパイルエラー | —       | —        |
| 800   | コンパイルエラー | オーバーフロー | コンパイルエラー |

## 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。

### 6.3.1 非圧縮 (密行列形式)

表 9に、密行列形式 (40 次元) の詳細な測定結果を示す。

表 10に、密行列形式 (50 次元) の詳細な測定結果を示す。

表 11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 12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

CSR +量子化モデルについては、現時点で MSE 測定は未実施である。各次元におけるメモリ使用量とコンパイル時間は表??に示したとおりである。

## 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

表 8: 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     | 500   | 4.548s   | 19556 バイト（59%） |
| CSR 圧縮     | 600   | 4.802s   | 21976 バイト（67%） |
| CSR 圧縮     | 800   | オーバーフロー  | オーバーフロー        |
| 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 9: 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 | 1559.96 | nn9.ppm  | 905.85 |
| nn5.ppm | 782.33  | nn10.ppm | 882.15 |

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: 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 11: 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 12: 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 | —         | —      | 測定未実施   |
| 160 | —         | —      | 測定未実施   |
| 200 | —         | —      | 測定未実施   |
| 400 | —         | —      | 測定未実施   |
| 500 | —         | —      | 測定未実施   |
| 600 | —         | —      | 測定未実施   |
| 800 | —         | —      | オーバーフロー |

## 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.