2025 年度　アジャイル レポート課題

24G1133　安尾 晃貴

はじめに

0.1　実験の目的

現代社会において IoT 機器の普及は著しく，様々なセンサーから大量のデータが生成されている [1]．これらのデータは記録や解析，遠隔伝送といった用途で広く活用される一方で，ストレージ容量の制約が現実上の課題となっている [2]．特に Arduino R4 WiFi（SRAM 32KB，フラッシュメモリ 256KB）のような組み込みマイコンボードでは，大規模なデータやディープランニングを扱う上で容量が制約となる．

本研究において，カラーセンサーの補正を目的としたニューラルネットワークを Arduino R4 WiFi に実装を試みた際に，メモリ不足の問題が発生した．具体的には，入力層 3 ノードは，隠れ層 70 ノード× 2 層，出力層 3 ノードの 3 層ニューラルネットワークの全結合層を実装しようとしたところ，重み行列だけで $(3 \times 70 + 70 \times 70 + 70 \times 3) \times 4$ バイト $= 21,280$ バイトを消費する．さらにバイアス項 $(70 + 70 + 3) \times 4$ バイト $= 572$ バイト，活性化関数の中間出力やその他の変数を含めると，総メモリ使用量は約 22KB 以上となり，SRAM 容量（32KB）の約 7 割を占有してしまう．加えて，プログラム本体や WiFi 通信用のライブラリ，スタック領域なども考慮すると，実際に利用可能なメモリは更に限られ，プログラムが正常に動作しない事態に陥った．

したがって，限られたメモリ資源の中でニューラルネットワークモデルを実装するためには，重み行列を効率的に圧縮する技術が不可欠である．特にニューラルネットワークでは，学習の過程で多くの重みがほぼ 0 となり，重み行列が疎行列として表されることが多い [3]．疎行列とは大部分の要素が 0 である行列であるため，0 以外の値とその位置情報のみを保持することで保存すべきデータ量を大幅に削減でき，メモリ効率の向上が期待できる [3].

本実験では代表的なデータ圧縮方式を調査し，その性能を比較・評価することで，削減率と復元精度の両立を目指す．具体的には，疎データに適した行列表現（CSR 形式）や値の量子化を組合せて実装し，圧縮後のデータサイズと復元時の MSE（平均二乗誤差），さらに圧縮によるメモリ削減率を定量的に測定する．これにより，Arduino R4 WiFi 等の組み込み環境においても実用可能なニューラルネットワークモデルの実装に必要なデータ圧縮手法の有効性を明らかにし，IoT デバイス上でより大きな AI の実装に向けた知見を得ることを目的とする．

0.2　実験の理論

本実験で検討する 3 つの圧縮方式について理論的な背景と選定理由を述べる．

まず，本研究ではスパース化の導入について検討する．スパース化とは，データ中の重要度の低い要素を選択的に 0 にして疎な表現へ変換する処理を指す．ニューラルネットワークにおいては，学習により得られた重み行列の中には出力への寄与が小さい要素が多数存在することが知られており [3]，これらを 0 に設定することでモデルの表現能力を大きく損なうことなく疎行列化が可能である．

スパース化の利点は，0 でない要素の割合を減らすことで，保存すべきデータ量を大きく削減できる点にある．密行列では多くの 0 を含んでいても全ての要素を保持する必要があるが，疎行列であれば 0 でない要素とその位置だけを記録すればよく，メモリ使用量を効率的に抑えられる．また，CSR 形式などを用いることで，計算面でも高速化が期待できる．しかし，スパース化によって一部の情報が失われるため，過度に 0 でない要素を減らすと推論精度が低下する可能性がある．

本研究では，疎行列の格納方式として CSR 形式の導入を検討する．CSR は，各行における 0 でない要素の値とその列インデックスを配列として保持し，さらに行の先頭位置を示すポインタ配列を併せて管理する表現方式である．この形式を用いることで，実際に計算に関与する 0 でない要素のみを格納でき，行列をそのまま保存する場合と比較してメモリ使用量を大幅に削減できる．特にニューラルネットワークの重み行列は，スパース化によって行方向に多数の 0 の要素が発生することが多い [3]．そのため，本研究で扱う重み行列に CSR 形式を適用することで，メモリ削減効果が期待できる．

本研究では，値領域の削減手法として量子化を導入を検討する．量子化は，浮動小数点値を低ビット幅の整数にし，1 つの重みを表現するために必要なビット数を削減する手法である．これにより，重み行列全体の保存容量を効率的に抑えることができる．本研究では，スパース化に加えて量子化を併用することで，重み行列の表現効率をより一層高め，高い圧縮効果を実現することを目的とする．

0.3 実験方法

本実験では，複数の圧縮手法をまとめて適用できるように実装する．全体の手順は，スパース化，CSR 形式への変換，量子化の 3 段階から構成される．各段階の実装方法を以下に示す．

0.3.1 スパース化処理

入力データである浮動小数点の重み行列 W に対してスパース化処理を行う．絶対値が閾値 τ 未満の要素を 0 に置換し，0 でない要素数を削減する．

$$W_{ij} = \begin{cases} W_{ij}, & \text{if } |W_{ij}| \geq \tau \\ 0, & \text{if } |W_{ij}| < \tau \end{cases} \tag{1}$$

0.3.2 CSR 形式への変換

CSR 方式の実装は以下の手順で行う．まず、圧縮対象となる学習済みの重みを取得し，疎行列化のしきい値を設定する．取得したテンソルは NumPy 配列に変換し，配列内の絶対値がしきい値未満の要素は 0 に置換する．これにより，計算上無視できる微小な値を除去し、疎行列化の効果を高めることができる．

次に，しきい値処理後の配列を CSR 形式に変換する．CSR 形式は，0 でない要素を格納する data 配列、列インデックスを示す indices 配列，行ごとの先頭位置を示す indptr 配列，及び行列サイズを示す shape 配列で構成する．

最後に，これらの配列を C++ で利用可能な形に出力することで，組み込みシステムに組み込むことが可能となる．上記の手順をフローチャートで示すと以下のようになる．

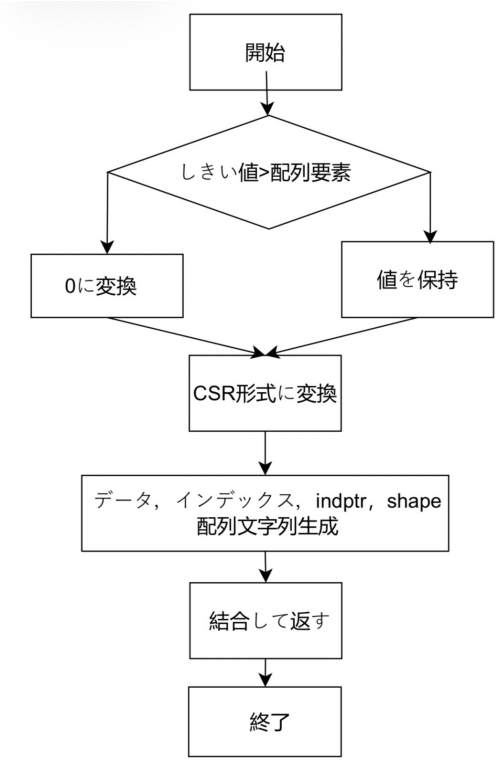

図 1　CSR 形式変換のフローチャート

0.3.3 量子化

量子化は，まず層内の重みの中で絶対値が最大の値を求め，これを用いてスケールを計算する．スケールは int8 の最大値 127 に対応させるように設定し，これにより元の float 値の比率を保持したまま整数化できる．次に各重みをこのスケールで割り，小数点以下を四捨五入することで int8 に変換する．こうして得られた int8 の値は C++ 側でスケールを掛けることで元の float 値に近い形に復元できる．

0.4 評価指標

本実験では，圧縮手法の性能を

- MSE 精度
- モデルサイズの削減効果
- 推論時間

の 3 つの観点から評価する．

まず，MSE 精度の評価には MSE を用いる．元データを x_i，ニューラルネットワークの出力を $\hat{x}_i$，データの全要素数を N とすると，MSE は次式で定義される．

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$

各条件について複数回の推論を行い，MSE の平均値を用いることで，乱数初期値などによるばらつきの影響を低減した．

次に，モデルサイズの評価には，元のモデルサイズ S_{orig} と圧縮後のモデルサイズ S_{comp}（いずれも生成されたヘッダファイルのバイト数）から計算される「削減率」を用いる．削減率は

$$\text{削減率} = 1 - \frac{S_{\mathrm{comp}}}{S_{\mathrm{orig}}}$$

と定義し，元のモデルからどの程度サイズを削減できたかを表す．削減率が 1 に近いほど，モデルサイズを大きく削減できていることを意味する．

さらに，組み込み環境での実用性を考慮して，推論時間も重要な評価指標とする．推論時間 T_{infer} は，各モデルに対して同一条件のもとで一定回数の推論を行ったときの処理時間（ミリ秒）を測定し，モデルサイズとの関係を比較することで，どの程度小さいモデルで，どの程度の誤差と推論時間を実現できるかを評価する．

以上のように，本実験では MSE 精度，削減率およびモデルサイズによるメモリ効率，推論時間による実行性能を総合的に評価することで，Arduino R4 WiFi のようなメモリ制約の厳しい環境における圧縮手法の有効性を検証する．

0.5 実装方法

計画書と異なる実装とした場合は、その理由（単に「実装が難しかったから」ではなく、第三者を納得させられる論理的な説明とすること）。

実装方法（アルゴリズム）の詳細、ソースコードの説明を以下に示す。

0.5.1 スパース化の実装

スパース化（sparsification）とは，配列中の小さな値を 0 に置き換える操作であり，データの疎性を高めることで計算効率やメモリ使用量を削減する手法である．本研究では，ある閾値（threshold）以下の値を 0 に置き換える簡単な方式を採用した．具体的な実装をリスト 1 に示す．

ソースコード 1　スパース化の実装

```
1 def sparsify_array(arr: np.ndarray, threshold: float = 0.01):
2     arr_copy = arr.copy()
3     arr_copy[np.abs(arr_copy) < threshold] = 0
4     return arr_copy
```

まず入力配列 `arr` をコピーし，変更可能な配列 `arr_copy` を作成する．続いて，絶対値が閾値 `threshold` 未満の要素を `0` に置き換える．最終的に，スパース化された配列を返す．

この実装により，微小な値を 0 にすることでデータの疎性を高め，後続の計算処理の効率化やメモリ削減を実現できる．また，閾値を調整することで，疎性の度合いを柔軟に制御可能である．

さらに，関数 `sparsify_array` の処理の流れを図 2 に示す．

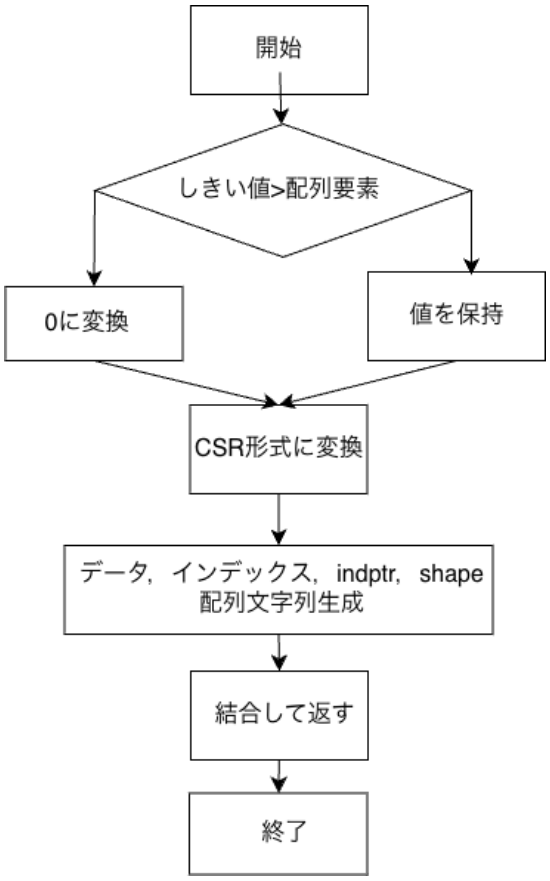

図 2　スパース化のフローチャート

0.5.2　量子化の実装

量子化とは，実数値を有限の整数値範囲に写像する操作であり，ここでは代表的な 8 ビット量子化（int8 quantization）を採用した．実装には NumPy を用い，入力配列をスケーリングして 8 ビット整数範囲 $[-128, 127]$ に丸め込む方式を採用した．スケーリング係数（scale）は，入力配列の絶対値最大値が 127 に対応するように自動計算する．具体的な実装をリスト 2 に示す．

ソースコード 2　量子化の実装

```
def int8_quantize(arr: np.ndarray, scale=None):
    if scale is None:
        max_abs = float(np.max(np.abs(arr)))
        scale = max_abs / 127.0 if max_abs != 0 else 1.0
    q = np.round(arr / scale).astype(np.int32)
    q = np.clip(q, -128, 127).astype(np.int8)
    return q, scale
```

まず入力配列 `arr` の絶対値最大値 `max_abs` を求め，それを 127 で割ることでスケーリング係数 `scale` を算出する．続いて，配列を `scale` で割り整数に丸めた後，範囲外の値を `np.clip` により $[-128, 127]$ に制限し，最終的に 8 ビット整数型（`np.int8`）として返す．また，同時にスケーリング係数を返すことで，後段の復元処理（dequantization）にも利用可能である．

さらに，関数 `int8_quantize` の処理の流れを図 3 に示す．

0.5.3　CSR 配列生成の実装

本研究では，疎行列を効率的に格納するため，Compressed Sparse Row（CSR）形式への変換を行う関数 `to_csr` を実装した．CSR 形式は，非ゼロ要素のみをデータ配列に保持し，行インデックスや列インデックスを別途管理することで，メモリ効率の高い行列表現を可能にする方式である．本関数では入力テンソルを NumPy 配列に変換した後，量子化を行い，さらに CSR 形式へ変換し，C 言語で利用可能な配列文字列として出力する．具体的な実装をリスト 3 に示す．

ソースコード 3　CSR 配列生成の実装

```
def to_csr(tensor: torch.Tensor, name: str, threshold: float = 0.01):
    arr = tensor.detach().numpy()
    arr_quantized, scale = quantize_array(arr, threshold)
    csr = csr_matrix(arr_quantized)
    data_str = f"int8_t␣{name}_data[]␣=␣{{␣" + ",␣".join(map(str, csr.data))
        + "␣};"
    indices_str = f"int␣{name}_indices[]␣=␣{{␣" + ",␣".join(map(str, csr.
        indices)) + "␣};"
    indptr_str = f"int␣{name}_indptr[]␣=␣{{␣" + ",␣".join(map(str, csr.
        indptr)) + "␣};"
    shape_str = f"int␣{name}_shape[2]␣=␣{{{csr.shape[0]},␣{csr.shape[1]}}};"
    scale_str = f"float␣{name}_scale␣=␣{scale};"
    return "\n".join([data_str, indices_str, indptr_str, shape_str,
        scale_str])
```

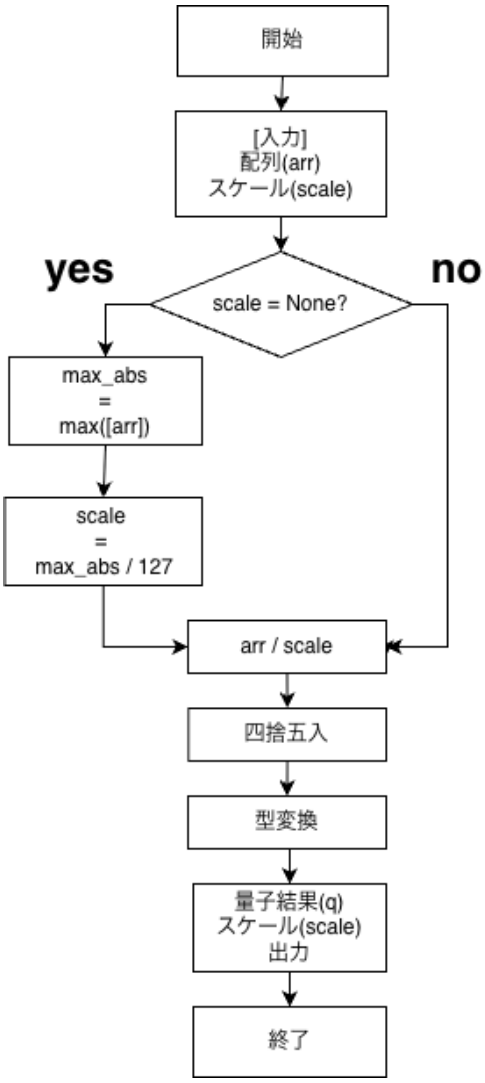

図 3　量子化のフローチャート

関数 `to_csr` は以下の処理手順で動作する．本実装により，量子化と疎行列化を組み合わせつつ，C 言語で直接利用可能な配列形式としてエクスポートすることが可能になる．処理の概要を図 4 に示す．

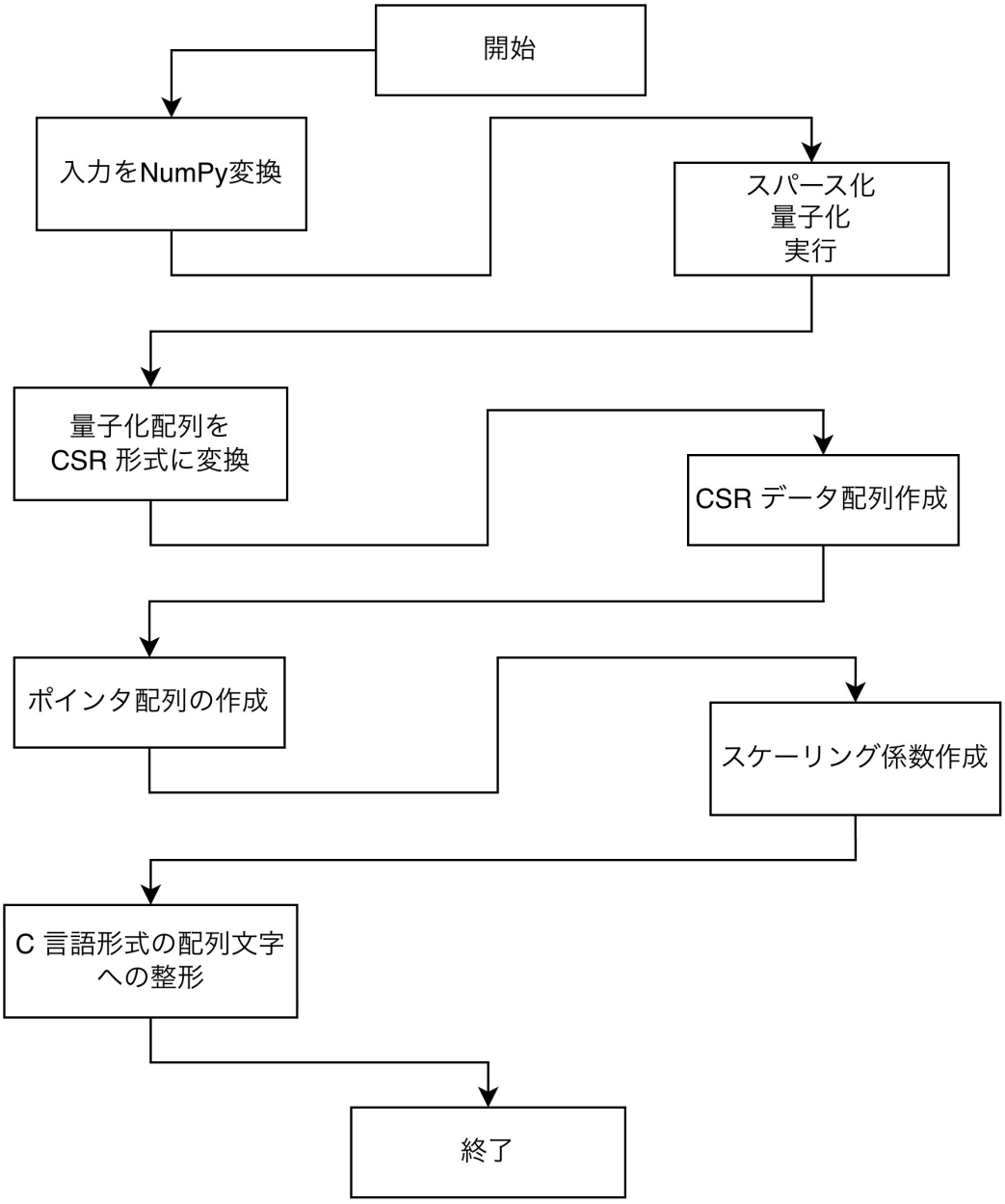

図 4　CSR 形式生成のフローチャート

1　実験結果

本節では，カラーコード読み取りハードウェア向けに設計した深層学習モデルについて，スパース化・量子化・CSR 化による圧縮が MSE とモデルサイズに与える影響を整理して述べる．MSE は 複数回の実験の平均値 mean を用い，モデルサイズは 作成される.h ファイルの大きさを指標とした．

1.1　ベースライン非圧縮モデルの性能

圧縮を行っていないベースラインモデル nomal_model の性能を確認する．表 1 は，パラメータ数を変化させたつのベースラインモデルの結果を表にまとめたものである．nomal_model_70 は，モデルサイズが大きく測定不能であった．

また，図 5 はパラメータと MSE（平均）の関係を示したものである．図 5 より，MSE がパラメータ数に依存せず分布している様子が確認できた．

表 1　ベースモデルの計測結果（model$_7$0 は測定不能）

ファイル名	容量（B）	時間（ms）	1 回目	2 回目	3 回目	4 回目	5 回目	平均
nomal_model_30	16458	716	391.52	608.52	240.52	319.37	528.70	417.726
nomal_model_40	27885	1177	441.00	462.70	516.30	421.78	431.93	454.742
nomal_model_50	42273	1655	195.37	366.37	371.07	224.67	87.22	248.94
nomal_model_60	59560	2399	311.93	582.30	320.04	593.63	424.52	446.484
nomal_model_70	79630	—	—	—	—	—	—	—

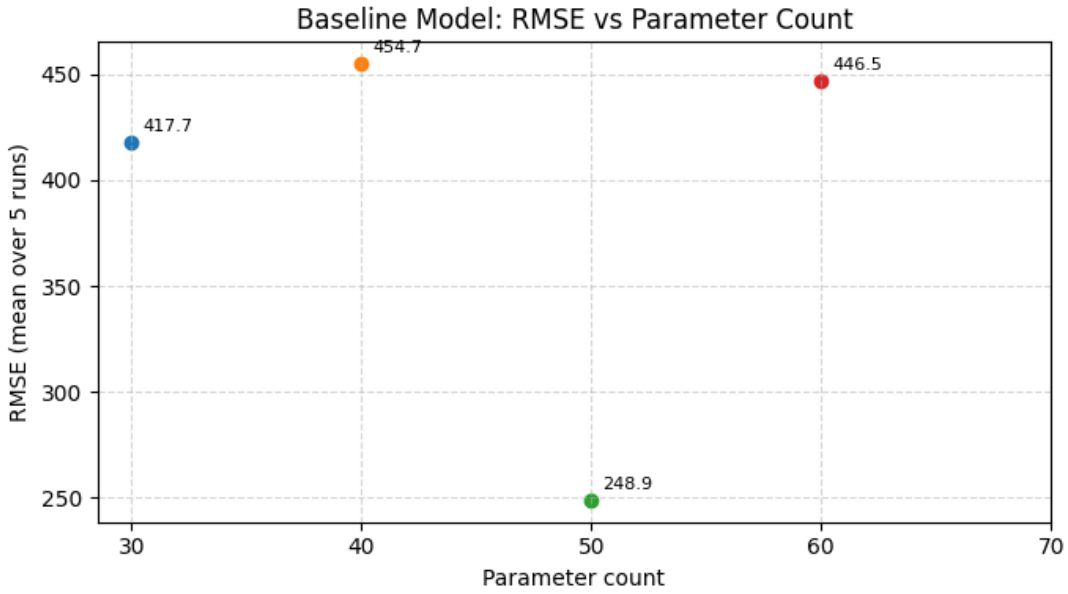

図 5　パラメータと MSE の関係

1.2　スパース化のみ適応したモデルの性能

ベースラインモデルに対してスパース化を適用したモデルの結果を表 2 に示す．比較の公平性を保つため，モデルのパラメータ数を 30 に固定し，スパース化の閾値 a のみを変化させて性能への影響を評価した．スパース化では，重みが閾値 a 以下である要素を 0 に置き換えることでモデルを軽量化し，計算コストの削減を図る．本実験では a=0.05，0.01，0.005 の 3 種類を設定し，それぞれのモデルで MSE を 5 回測定し，その平均値を算出した．さらに，スパース化による モデルサイズの減少率 と 精度（MSE）の変化量 を併せて評価し，削減率と精度のトレードオフについても検討した．

また，図 6 は閾値と MSE（平均）の関係を示したものである．図より，閾値が小さくなるにつれて MSE が低下する傾向が確認できる．

表 2　スパース化モデル（パラメータ=30）の測定結果

ファイル名	実装容量（B）	ベース容量（B）	削減率	時間	1 回目	2 回目	3 回目	4 回目	5 回目	平均
sparse_p30_0.005	7550	16458	0.5413	11424	286.30	163.37	209.30	295.67	483.70	287.668
sparse_p30_0.01	7540	16458	0.5419	11424	124.11	868.44	487.89	605.22	127.52	442.636
sparse_p30_0.05	7440	16458	0.5479	11424	719.74	916.56	470.22	755.52	117.74	595.956

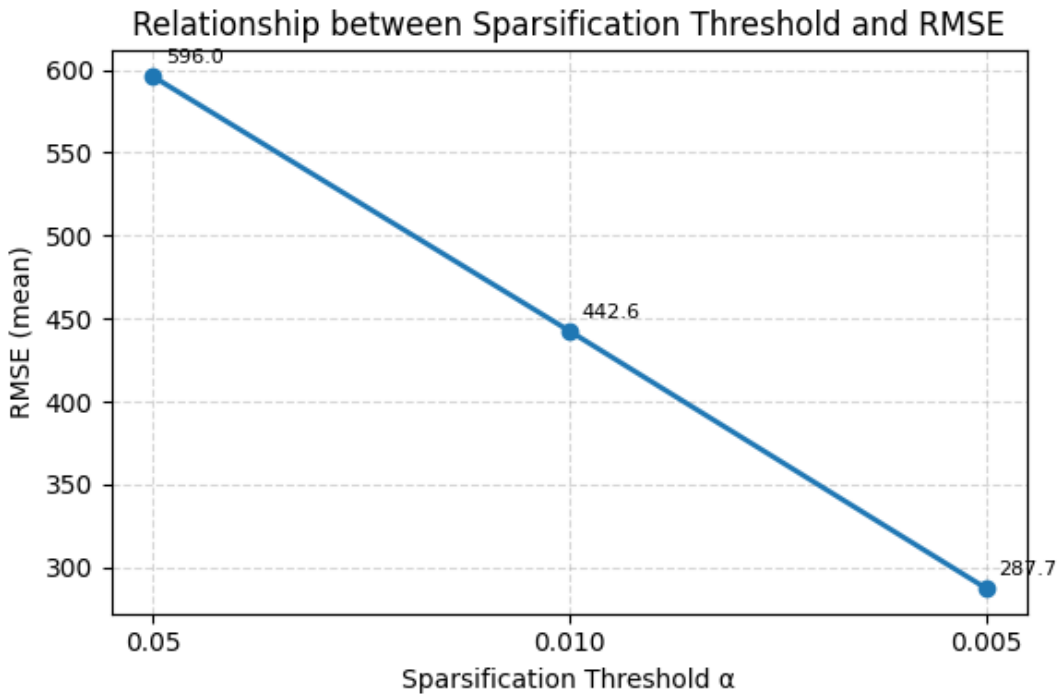

図 6　閾値と MSE の関係

1.3　CSR 形式と int8 量子化を適応したモデルの性能

ベースラインモデルに対してスパース化および CSR 形式と int8 量子化を適用したモデルの結果を表 3 に示す．ここでは，前節の検討で最も良好であった閾値 ($\alpha = 0.005$) を用い，モデルのパラメータ数だけを 30〜1200 の範囲で変化させて評価を行った．各条件について最大 5 回推論を行い，そのときの MSE の平均値を MSE として算出した．なお，一部の条件（パラメータ 80〜700, 1100）は 3 回分の結果から平均値を求めた．1200 で容量釣果をしたため MSE は測定不能であった．

圧縮後のモデルサイズは 2368〜20119 バイト となり，元のモデルサイズと比較して削減率（compression ratio）はおよそ 85.6〜99.9% であった．MSE平均 は パラメータ 30〜1100 の範囲で 216.73〜526.49 の値を取り，最小値は パラメータ 70 のときの 216.73 であった．また，パラメータ 1100 でも 216.87 と同程度の MSE を示した．

図 7 はパラメータと削減率の関係を示したものである．図 7 より，パラメータ増加するにつれ削減率が上昇していることが確認できた．

また，図 8 はパラメータと MSE（平均）の関係を示したものである図 8 より，MSE がパラメータ数に依存せず分布している様子が確認できた．

表 3　Sparse + CSR + int8（閾値 $\alpha = 0.005$）モデルの測定結果

ファイル名	実装容量（B）	ベース容量（B）	削減率	時間	1 回目	2 回目	3 回目	4 回目	5 回目	平均
model_csr_int8_p30	2368	16458	0.8561	7924	191.44	347.63	563.44	437.93	357.93	379.674
model_csr_int8_p40	2578	27885	0.9075	8032	512.44	542.33	325.44	454.26	424.15	451.724
model_csr_int8_p50	2815	42273	0.9334	8188	491.26	389.89	421.78	247.89	534.93	417.150
model_csr_int8_p60	2918	59560	0.9510	8272	457.56	205.37	326.19	249.30	437.33	335.150
model_csr_int8_p70	3053	79630	0.9617	8324	173.00	466.00	278.52	23.19	142.96	216.734
model_csr_int8_p80	3097	103073	0.9700	8384	247.56	400.52	363.74	—	—	337.273
model_csr_int8_p90	3482	129080	0.9730	8616	540.48	318.33	581.96	—	—	480.257
model_csr_int8_p100	3650	158034	0.9769	8696	469.59	585.78	524.11	—	—	526.493
model_csr_int8_p200	5278	609431	0.9913	9608	250.67	281.56	361.74	—	—	297.990
model_csr_int8_p500	5278	3720238	0.9986	12660	246.19	338.70	320.48	—	—	301.790
model_csr_int8_p700	10290	7255329	0.9986	14600	301.11	386.22	222.15	—	—	303.160
model_csr_int8_p1000	13566	14759027	0.9991	17632	274.74	261.85	381.93	—	—	306.173
model_csr_int8_p1100	20119	17842882	0.9994	18616	150.26	210.41	289.93	—	—	216.867
model_csr_int8_p1200	20119	—	—	—	—	—	—	—	—	—

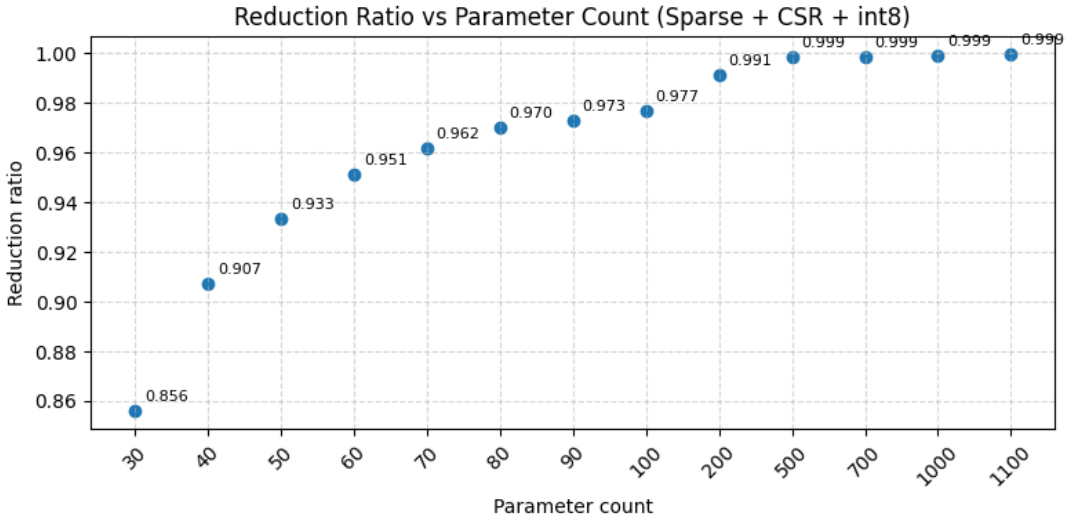

図 7　パラメータ と 削減率 の関係

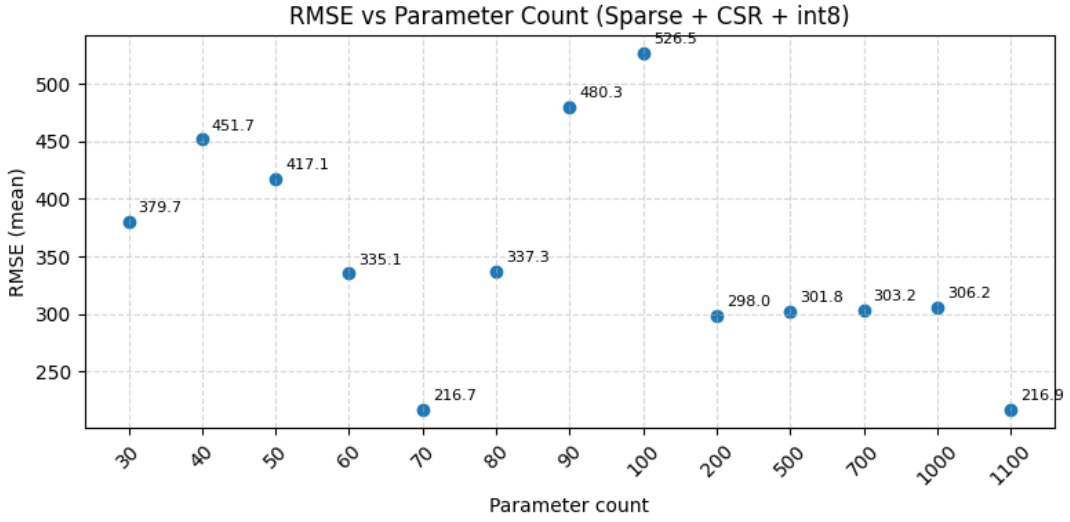

図 8　パラメータ と MSE の関係

1.4　同一パラメータ数における比較

ベースラインモデルと Sparse + CSR + int8（$\alpha = 0.005$）を，同一パラメータ数で比較した結果を表 4 に示す．本比較では，パラメータ数を 30〜70 に固定し，各モデルの実装容量と MSE（5 回測定の平均値）を整理した．

モデルサイズについては，すべてのパラメータ設定で圧縮後の容量が大幅に削減され，削減率は 0.8561〜0.9617 の範囲となった．特に パラメータ = 70 では，79630 バイトから 3053 バイトへと最も大きな削減が確認された．

MSE の結果は，パラメータ = 30，40，60 において圧縮モデルとベースラインが同程度の値を示し，パラメータ = 50 では圧縮モデルが高い MSE を示した．また，パラメータ = 70 についてはベースラインの MSE が測定不可であるが，圧縮モデルの MSE は 216.734 であった．

図 9 は，表 4 の各モデルの MSE とパラメータのデータをグラフ化したものである．図からは，MSE がパラメータ数に依存せず分布している様子が確認できる．また，2 つのモデル間で MSE に大きな差異は見られない．

表 4　通常モデルと Sparse + CSR + int8（$\alpha = 0.005$）の比較（パラメータ = 30〜70）

パラメータ	ベース（B）	CSR+int8（B）	削減率	ベース（平均）	CSR+int8（平均）
30	16458	2368	0.8561	417.726	379.674
40	27885	2578	0.9075	454.742	451.724
50	42273	2815	0.9334	248.940	417.150
60	59560	2918	0.9510	446.484	335.150
70	79630	3053	0.9617	—	216.734

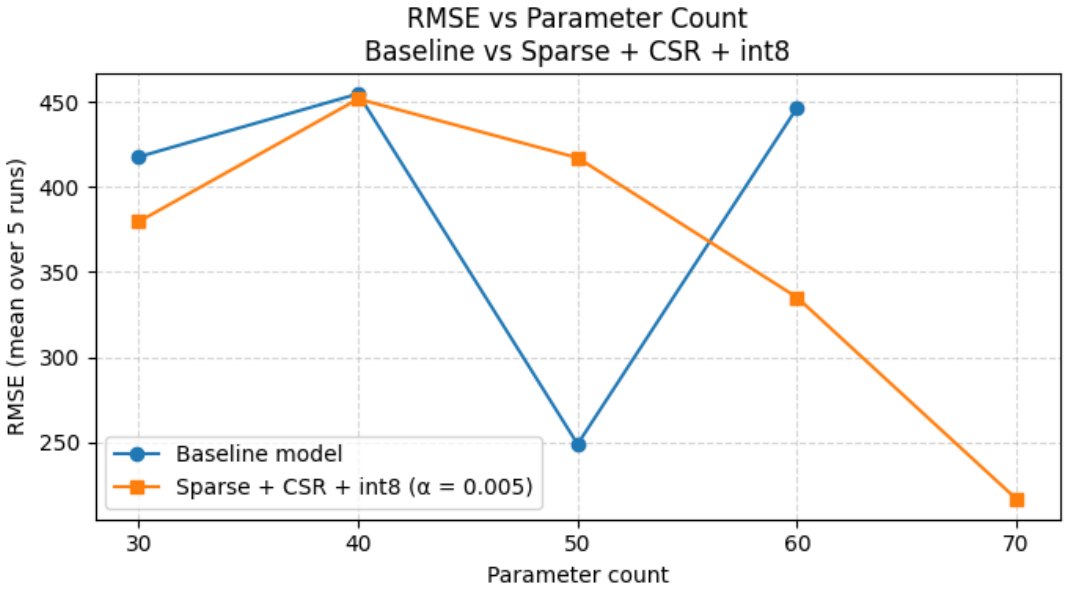

図 9　各モデルのモデルサイズと MSE の関係

1.5　モデルサイズと MSE の関係

図 10 は，ベースラインモデル，Sparse + CSR + int8 を適用したモデルについて，モデルサイズと MSE（平均値）の関係を示したものである．

本図より，各モデルにおいてパラメータ数の増加に伴いモデルサイズが大きくなり，それに応じて MSE も変化する様子が確認できる．特に Sparse + CSR + int8 モデルは，ベースラインに比べて大幅にモデルサイズが小さくなる一方で，MSE の分布に相関関係は見られなかった．

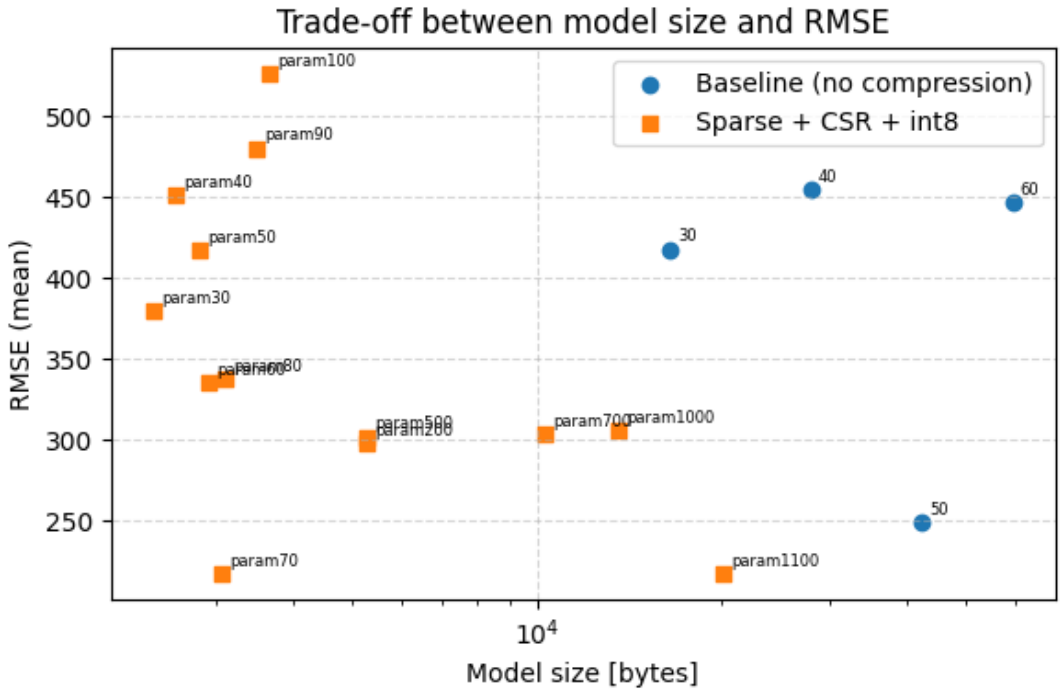

図 10　モデルサイズと MSE の関係

1.6　モデルサイズと推論時間の関係

図 11 は，ベースラインモデル，Sparse のみ，Sparse + CSR + int8 の 3 種類のモデルについて，モデルサイズと推論時間（ms）の関係を示したものである．

図より，ベースラインモデルではモデルサイズの増加に伴い推論時間も増加する相関傾向が見られる．また，Sparse + CSR + int8 モデルでもパラメータが 500 時以外は，同様の傾向が確認できる．

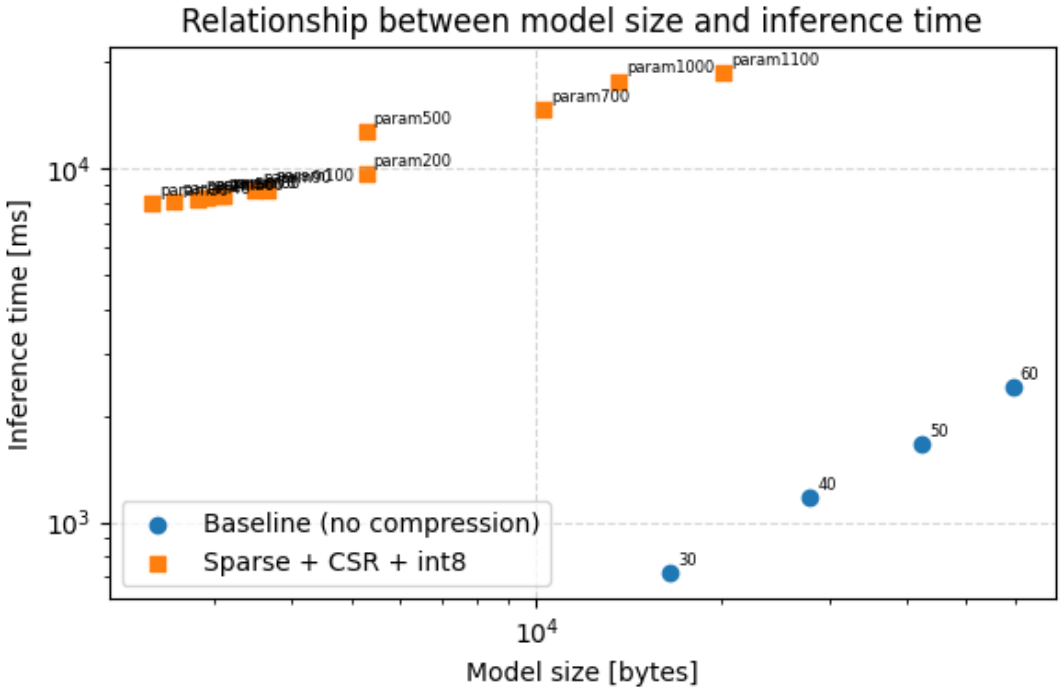

図 11　モデルサイズと推論時間の関係

2　考察

2.1　MSE とモデルサイズの関係

本実験では，ベースラインモデルおよび Sparse + CSR + int8 モデルのいずれにおいても，パラメータ数やモデルサイズと MSE の間に明確な相関関係は見られなかった（図 5，図 8，図 10）．ベースラインモデルでは，パラメータ数 30〜60 に対して MSE の平均値は約 250〜450 の範囲でばらついており，パラメータ数 50 のときに最小 MSE（約 249）が得られている一方で，それより大きなパラメータ数 60 では MSE が再び増加している．同様に，Sparse + CSR + int8 モデルでも，パラメータ数を 30 から 1100 まで大きく変化させても MSE は 200〜500 程度の範囲に散らばっており，モデルサイズの増加が必ずしも MSE の改善にはつながっていない．

この結果は，本研究で扱ったカラーセンサー補正タスクが比較的単純であり，小さなモデルでも必要な表現能力をすでに満たしている可能性を示唆している．すなわち，一定以上パラメータ数を増やしても表現能力の上限に達しており，モデルの自由度の増加が MSE の低減として現れなかったと解釈できる．加えて，学習データ数に対してパラメータ数が大きくなり過ぎると，学習過程での初期値や最適化経路の違いによるばらつきや，軽度の過学習の影響が MSE に表れている可能性もある．

また，量子化およびスパース化を施したモデルでは，重みが int8 に丸められ，一部の小さな重みが 0 に置き換えられているにもかかわらず，MSE が大きく悪化していない点も考察する．これは，本タスクでは高精度の浮動小数点表現が必須ではなく，8 ビット整数および疎構造に変換しても，色補正の性能には大きな影響を与えないことを意味している．一部の条件ではむしろ圧縮モデルの MSE がベースラインと同等かそれ以下

になっており，量子化やスパース化による「ゆるい正則化効果」が過学習の抑制に寄与した可能性も考えられる [3].

2.2 削減率の挙動に関する考察

本実験では，モデルサイズの評価指標として「削減率」を用い，これは以下のように定義される．

$$\text{削減率} = 1 - \frac{S_{\text{comp}}}{S_{\text{orig}}}$$

そのため，表 3 における 0.8561 や 0.9994 といった値は，「元のモデルサイズから何割削減できたか」を表している．

Sparse + CSR + int8 モデルでは，パラメータ数 30 のときの削減率は約 85.6% であり，元のモデルの約 8 程度の容量にまで削減できている．パラメータ数を増やしていくと削減率はさらに増加し，パラメータ数 200 以上では 99% を超え，1100 では約 99.94% に達している．これは，ベースラインモデルのサイズがパラメータ数にほぼ比例して増加する一方で，圧縮後のモデルサイズはパラメータ数に対して緩やかにしか増加していないことを意味する．

この挙動の背景には，以下の 2 点があると考えられる．1 つ目は，スパース化により多くの重みが 0 に置き換えられていることである．パラメータ数を増やすほど，学習後に実質的に寄与しない小さな重みも増加しやすく，閾値以下の要素が大量に 0 となる [3]．その結果，CSR 形式で保持すべき非ゼロ要素の個数はパラメータ数ほどは増えず，容量の増加が抑制される．2 つ目は，量子化による 1 要素あたりの表現ビット数の削減である．ベースラインでは float32（4 バイト）で重みを表現しているのに対し，圧縮モデルでは int8（1 バイト）を用いており，非ゼロ要素に対しては理論上 1/4 まで縮小できる．これに CSR 形式による疎構造の活用が加わることで，総体として 1% 以下まで容量を抑えられている．

スパース化のみを適用した表 2 では，閾値 (α) を 0.05 から 0.005 に変化させても削減率は 0.54 台からほとんど変化していない．これは，元の重み分布において ($|w| < 0.005$) のような「非常に小さい値」がそもそも少なく，0.005 以下の範囲で閾値を変えても 0 になる要素の数が大きく変化しなかった可能性を示唆している．つまり，学習済みモデルはもともとある程度スパースな性質を持っており，今回の閾値設定の範囲では削減効果に大きな差が出なかったと考えられる．

2.3 モデルサイズと推論時間の関係

図 11 に示されるように，ベースラインモデルではモデルサイズの増加に伴い推論時間も増加しており，両者の間に正の相関傾向が確認できる．これは，全結合層の計算量がパラメータ数にほぼ比例するためであり，より大きなモデルほど行列積演算の回数が増え，推論時間が長くなることを反映している [4].

また，Sparse + CSR + int8 モデルにおいても，パラメータ数が大きくなるほど推論時間が増加する傾向が見らた．CSR や int8 から浮動小数点への復元が推論時に必要となるためベースラインモデルよりも推論時間が増加している．

このことから，圧縮モデルはサイズ削減によるメモリ効率の利点を持つ一方で，推論処理においては復元処理が無視できない時間コストとなり，その分だけベースラインモデルより遅くなるケースが存在する．これは，組み込みデバイス上で圧縮モデルを運用する際に，メモリ削減と計算コストの両面から最適な圧縮パラメータを選択する必要があることを示唆している．

2.4　圧縮モデルとベースラインモデルの比較

同一パラメータ数におけるベースラインモデルと Sparse + CSR + int8 モデルの比較（表 4, 図 9）からは，圧縮によってモデルサイズを大幅に削減しつつ，MSE をほぼ維持できていることが確認できる．例えば param = 30 では，ベースラインの MSE が 417.726 に対して圧縮モデルは 379.674 であり，MSE をほとんど悪化させることなく容量を約 85% 削減できている．param = 40 と 60 でも，両モデルの MSE は同程度であり，いずれも圧縮による精度劣化は限定的である．

一方で，param = 50 では圧縮モデルの MSE がベースラインよりも大きくなっており，圧縮処理による情報損失の影響が現れた例と考えられる．ただし，各条件で MSE は 5 回測定の平均値であり，ばらつきも存在することから，単一条件の逆転のみで圧縮手法の有効性を否定することはできない．全体として見れば多くの条件で容量は削減しながら，MSE は同程度または許容範囲内の増加に収まっていると評価できる．

さらに，param = 70 に関しては，ベースラインモデルはモデルサイズが大きく測定不能であった一方，Sparse + CSR + int8 モデルでは約 3KB まで圧縮され，しかも MSE は 216.734 と，本実験中でも最小クラスの値を示した．これは，圧縮技術を用いることで，ベースラインでは実装困難であったパラメータ数のモデルを，実用的なサイズで扱えるようになることを示している．メモリ制約の厳しい Arduino R4 WiFi のような環境では，大きなモデルを圧縮して載せる戦略が特に有効であると考えられる．

3　まとめ

本実験では，スパース化・CSR 形式・int8 量子化を組み合わせることで，ニューラルネットワークのモデルサイズを最大で元の 1/100 程度まで削減しつつ，MSE の劣化を小さく抑えられることを示した．しかし，学習条件やデータセットは各パラメータ数で共通の設定とし，個別にハイパーパラメータ探索を行っていないため，MSE とモデルサイズの関係について十分に最適化された結果とはいえない．さらに，本研究で検討した量子化は 8 ビット固定であり，4 ビット以下のより高圧縮な方式や層ごとのスケール最適化までは扱えていない．

今後は，各パラメータ数に対する学習条件のチューニングや複数回学習による統計的評価を行い，より厳密に精度と削減率の関係を分析する必要がある．また，Arduino 側の推論時間・メモリ使用量・消費電力の最適化を行うことが必要である．さらに，低ビット量子化やハードウェアに最適化した演算実装を検討することで，より大規模なモデルを動作させるための実用的な指針を得ることが今後の課題である．

参考文献

[1] 総務省，『令和 3 年版情報通信白書』，「第 1 部 特集 デジタルで支える暮らしと経済」，2021 年．https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/html/nd105220.html，（参照 2025 年 10 月 24 日）．

[2] IEEE Access，“A Survey on Resource Management in IoT Operating Systems”，2018 年．https://ieeexplore.ieee.org/document/8300305/，（参照 2025 年 10 月 24 日）．

[3] 東芝レビュー，“IoT 時代のエッジコンピューティング技術”，Vol.74 No.4，pp.4–7，2019 年 7 月．https://www.global.toshiba/content/dam/toshiba/migration/corp/techReviewAssets/

tech/review/2019/04/74_04pdf/f01.pdf, (参照 2025 年 10 月 24 日).

[4] Ngo, D., et al., "Edge Intelligence: A Review of Deep Neural Network Inference on Edge Platforms," Electronics, Vol. 14, No. 12, 2025 年. https://www.mdpi.com/2079-9292/14/12/2495, (参照 2025 年 12 月 11 日).