アジャイルワーク 2
実験計画書

24G1089 武本 龍

2025 年 10 月 24 日 (金)

1　実験の概要および目的

以下に，本実験の基本的な概要を図 1に示す.

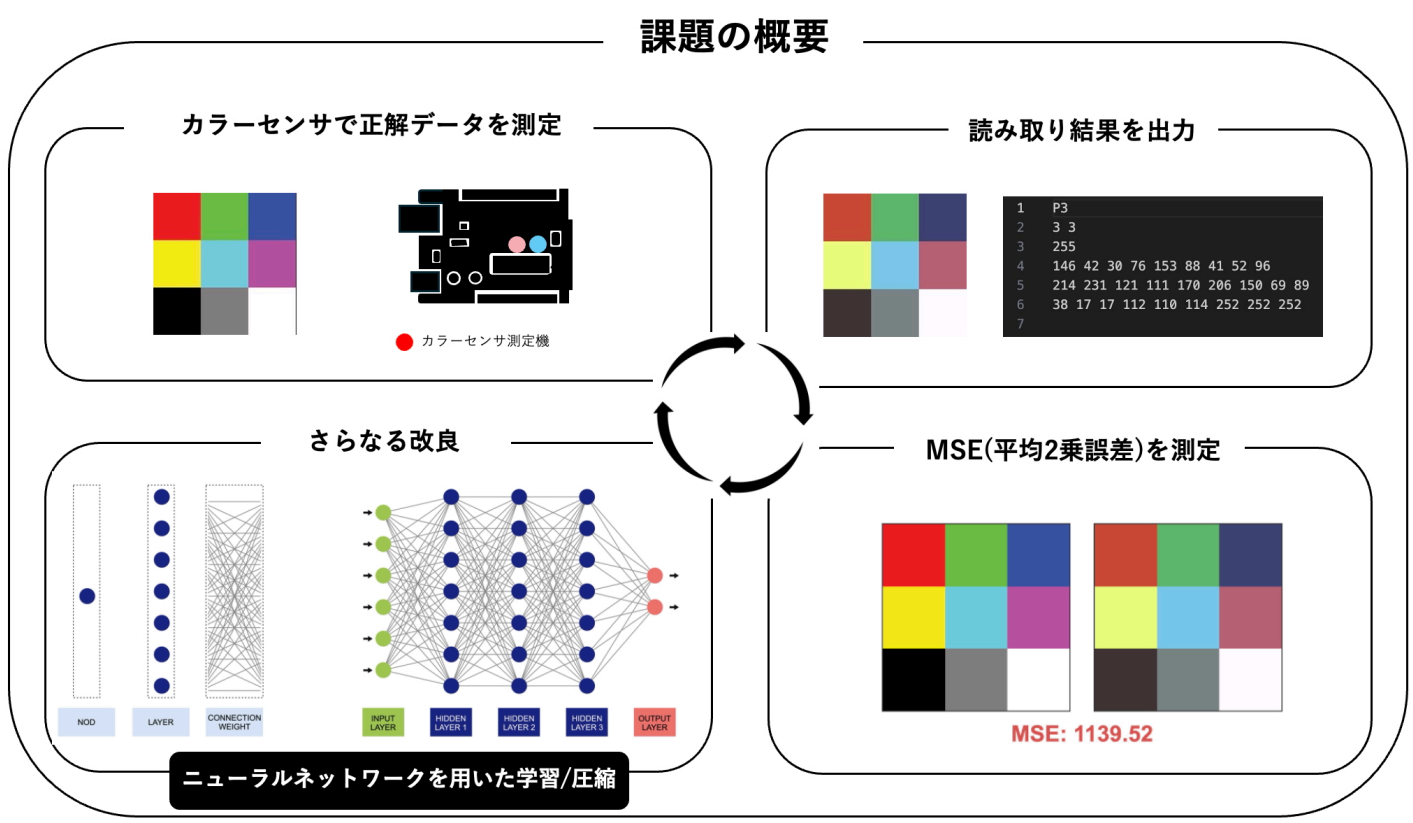

図 1: 基本的な実験概要図

本実験では，図 1に示すように，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センサで測定する．主な目的は，測定したカラーチャートと基準のカラーチャートの平均二乗誤差（MSE）を小さくすること，読み取り時間をできるだけ短くすること，の 2 点である．これらの目的を達成するため，特に MSE の低減にニューラルネットワークによる学習を活用し，ニューラルネットワークの隠れ層で用いるデータの圧縮手法を導入することで，ソフトウェアの品質を向上させることを目指す．以下では，読み込み品質の測定方法，およびソフトウェア的な改善手法について詳述する．

1.1 平均二乗誤差（MSE）とは

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 乗の平均値で定義され，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 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 チャネルの差の 2 乗を加算し，3 で割った値を MSE として計算する．また，ソフトウェアの実装例は以下の通りである．

$$\mathrm{mse} = \frac{r_{\mathrm{diff}} \times r_{\mathrm{diff}} + g_{\mathrm{diff}} \times g_{\mathrm{diff}} + b_{\mathrm{diff}} \times b_{\mathrm{diff}}}{3} \tag{2}$$

1.2 読み込み品質の測定方法

読み込み品質の評価は，以下の手順で実施する．

1. **2 つのボタンプログラムによるカラーチャートの読み取り**
   事前に黒と白のサンプルを測定し，最小値・最大値を基準として読み込み精度をキャリブレーションする．これにより，センサの感度を調整し，安定した RGB 値を取得する．
2. **PPM 画像出力**
   授業で配布された 3×3 のカラーチャートをセンサで読み取り，取得した RGB データを PPM 形式の画像ファイルとして出力する．この画像は，後続の品質評価に用いる．
3. **平均二乗誤差（MSE）の計算**
   出力した PPM 画像と基準画像の品質を，専用の MSE 測定ソフトウェアで評価する．

2　実験の目的

以下に実験の目的を示す.

2.1　読み込み品質のソフトウェア的改良 (ニューラルネットワークを用いた学習)

サンプル画像と測定画像の MSE を最小限に抑えるために，ニューラルネットワークを用いた学習方法およびニューラルネットワークの圧縮方法を以下に示す.

2.1.1　ニューラルネットワークの基本構造

まず，ニューラルネットワークの基本構造について説明する．ニューラルネットワークの最小単位は「ユニット」と呼ばれ，複数の入力を受け取り，1 つの出力を計算する．各ユニットは，入力値にそれぞれ異なる重み（weight: $w_1, w_2, w_3, \ldots$）を掛けて加算し，さらにバイアス（b）を加えた総入力 u を計算する．具体的には，以下の式で表される.

$$u = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + b \tag{3}$$

この総入力 u は，活性化関数 f に入力され，出力 $z = f(u)$ が生成される.

2.1.2　順伝搬型ネットワーク

次に，ニューラルネットワークの層構造について考える．層は $l = 1, 2, 3, \ldots$ で表され，$l = 1$ を入力層，$l = 2$ を中間層，$l = 3$ を出力層と呼ぶ．各層の計算は以下のようになる．例えば，層 $l = 2$ では,

$$u^{(2)} = W^{(2)} x^{(2)} + b^{(2)}, \quad z^{(2)} = f(u^{(2)}) \tag{4}$$

また，層 $l = 3$ では,

$$u^{(3)} = W^{(3)} x^{(3)} + b^{(3)}, \quad z^{(3)} = f(u^{(3)}) \tag{5}$$

これを任意の層数 L に一般化すると，層 $l + 1$ のユニットの出力 $z^{(l+1)}$ は，1 つ前の層 l の出力 $z^{(l)}$ を用いて以下のように計算される.

$$u^{(l+1)} = W^{(l+1)} x^{(l)} + b^{(l+1)}, \quad z^{(l+1)} = f(u^{(l+1)}) \tag{6}$$

ここで，入力層の出力は $z^{(1)} = x$ とし，$l = 1, 2, 3, \ldots, L - 1$ の順に計算を進めることで，各層の出力 $z^{(2)}, z^{(3)}, \ldots, z^{(L)}$ を順次決定できる.

入力 x を受け取り，各層の計算を順番に実行して最終的に出力 $y = z^{(L)}$ を得るネットワークを，順伝搬型ネットワークと呼ぶ．この入力 x から出力 y を得る計算は，各層間の結合の重みパラメータ $W^{(l)}$（$l = 2, \ldots, L$）およびユニットのバイアスパラメータ $b^{(l)}$（$l = 2, \ldots, L$）によって決定される．これらすべてのパラメータをまとめて表現するため，$W^{(2)}, \ldots, W^{(L)}, b^{(2)}, \ldots, b^{(L)}$ を成分とするベクトル w を定義し，出力は $y(x; w)$ と表記する.

順伝搬型ネットワークは，1 つの関数 $y(x; w)$ を表現し，この関数の形状はネットワークのパラメータ w に依存して変化する.

2.1.3 使用するニューラルネットワーク

本実験では，全結合型の 4 層ニューラルネットワークを使用する．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 RGB の 3 次元である．各ニューロンで行われる推論演算は，以下の通りである入力 $x_1 \times w_1 + x_2 \times w_2 + \cdots + x_{n-1} \times w_{n-1} + b$ を計算し，ReLU 活性化関数を通した値を次の層へ出力する．ここで，w_i は重み，b はバイアスを表す．モデルは PyTorch を用いて定義・訓練し，L1 正則化を導入することで重みを疎化させる．訓練後，重みとバイアスを C++ 配列としてエクスポートし，Arduino 上で推論を実行する．

2.1.4 なぜ隠れ層の圧縮が必要なのか

Arduino Uno R4 WiFi のようなマイクロコントローラーでは，メモリ容量が限定的である（SRAM: 約 32KB，Flash: 256KB）ため，ニューラルネットワークの隠れ層次元を増大させると，重み・バイアスデータの保存および推論演算に必要なメモリが不足する可能性がある．

例えば，隠れ層次元を 40 から 70 以上に拡大した場合，パラメータ総数が急増（例: 40 次元で約 1,923 パラメータ，70 次元で約 3,000 パラメータ以上）し，float32 形式で 10KB を超えると動作不能となる．この制約を克服するため，重みデータの圧縮が不可欠である．

主な手法として，(1) 量子化（float32 → int8 でメモリ 1/4 化），(2) プルーニング（L1 正則化により 0 に近い重みを 0 化し，非ゼロ率を低減），(3) 疎行列表現（CSR 形式で非ゼロ要素のみ保存）が有効である．これにより，次元を 100-200 まで拡張しつつ，MSE 誤差をできるだけ抑え，画像復元の品質を向上させることができる．最終的に，読み取り時間の短縮（演算量低減）と MSE の低減を両立させる．

3　実験理論

以下に実験理論を示す.

3.1　圧縮方法 1: 量子化とは

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高精度の浮動小数点数（例: 32-bit float）から低精度の整数（例: 8-bit uint8）へ変換する手法である．これにより，整数演算のみで推論が可能となり，組み込みデバイスのメモリ使用量と計算コストを大幅に削減する．論文では，アフィン変換スキーム $r = S(q - Z)$（S: scale, Z: zero-point）を提案し，畳み込み層の入力，出力，重みを 8-bit 整数で表現し，蓄積器を 32-bit 整数で扱うことで，浮動小数点演算を回避した [1]．本実験では，このスキームを RGB 画像復元ニューラルネットワーク（入力 3→ 隠れ 40→40→ 出力 3）に適用し，重み・活性化を uint8 へ量子化，バイアスを int32 で保持する予定である．

量子化の主なメリットとして，メモリ使用量の約 4 倍削減と，整数演算（uint8 × uint8 → int32 蓄積）による計算高速化が挙げられる．これにより，Arduino Uno R4 WiFi のようなリソース制約デバイスでの効率的なデプロイが可能となり，画像復元タスクのリアルタイム処理を促進する．

一方で，デメリットとしては，小規模モデルにおける精度低下のリスクと実装の複雑さが指摘される．特に，チャネル間の範囲差が大きい場合に相対誤差が増大しやすく，スケーリングやゼロポイントのオフライン事前計算，Arduino 上での 32-bit 整数蓄積および固定小数点乗算の処理が追加の負担となる可能性がある．

3.2　圧縮方法 2: プルーニングとは

プルーニングとは，ニューラルネットワークの重みを閾値以下で 0 にし，不要な接続を削除して疎行列化する手法である．これにより，過剰パラメータを削減し，メモリ・計算コストを低減できる．論文では，3 ステッププロセス（初期訓練で重要接続学習 → 低重み接続削除 → 残存接続再訓練）を提案し，複数回適用することで畳み込みニューラルネットワークモデルである AlexNet を精度損失なしに 9 倍圧縮している [2].

プルーニングの主なメリットとして，重みの非ゼロ要素を 8-11 %に減らすことでメモリを大幅に節約し，疎行列演算により不要部分をスキップして計算量を 30-50 %低減できる点が挙げられる．これにより，Arduino Uno R4 WiFi のような組み込みデバイスでのリアルタイム画像復元が効率化される．

一方で，デメリットとしては，再訓練なしでは精度が急落するリスクや，疎表現のオーバーヘッド（非ゼロ位置のインデックス保存でメモリが 15.6 %増加し，疎演算が非効率となる可能性），さらに複数回のイテレーションが必要な訓練コスト増が指摘される．

3.3 圧縮方法 3: COO（COOrdinate）形式

COO（Coordinate List）形式は，疎行列を非ゼロ要素の座標（行番号，列番号）と値のリスト（triples: (row, col, value)）として表現するシンプルな圧縮手法である．授業のヒント通り，非ゼロ成分の値・列番号・行番号を列挙し，L1 正則化により重みが 0 に近い成分が多くなる本研究のニューラルネットワークでは，0 に近い成分（例: ± 0.01 未満）を 0 としてプルーニングした後，非ゼロ要素のみをリスト化することで，重み行列（例: 40 × 40）を効率的に保存する．重複エントリ（同一位置の複数値）は加算して扱うことが可能で，中間表現として CSR 形式への変換に適している．

COO 形式の主なメリットとして，非ゼロ要素を直接リスト化するため構築・変換が容易で，メモリ削減効果が高い点が挙げられる．特に，非ゼロ率が 50 %の場合，データ量を約 1/2 に抑えられるため，Arduino Uno R4 WiFi のメモリ制約下で軽量な重み保存が可能となる．

一方で，デメリットとしては，アクセス効率の低さが指摘される．行列-ベクトル乗算時に全非ゼロ要素を走査する必要があり，CSR 形式に比べて高速化しにくいため，推論時間の最適化が限定的となる可能性がある．

3.4 圧縮方法 4: CRS/CSR（Compressed Row Storage/Compressed Sparse Row）形式とは

CRS/CSR（Compressed Sparse Row）形式は，疎行列を値配列（data），列インデックス配列（indices），行ポインタ配列（indptr）で表現する行指向の圧縮手法である．非ゼロ要素を列順にソートし，各行の非ゼロ開始位置を indptr で記録するため，行アクセスが高速である．授業の参考通り，L1 正則化で生じた 0 成分をプルーニング後，40 × 40 重み行列を CSR で保存し，Arduino のメモリ（SRAM 32KB）制約をクリアする．

CSR 形式の主なメリットとして，行列-ベクトル乗算が O(NNZ) で効率的である点が挙げられ，推論速度が COO 形式の 1.5-2 倍向上する．また，インデックス共有による優れた圧縮率（非ゼロ率 30 %でデータ量 1/10，40 次元モデルで 1KB 未満可能）と，Arduino 適合性（固定サイズ配列（indptr: 41 要素）で実装しやすく，Flash 保存（PROGMEM）で RAM 節約，複数層対応）も大きな利点である．これにより，メモリ制約下での高次元隠れ層の実現が容易となる．

一方で，デメリットとしては，列アクセスの遅さ（列方向スキャンが必要で，転置行列使用時非効率，CNN のような畳み込み層には不向き），変換コスト（COO から CSR へのソートが必要（SciPy の tocsr()），動的 NNZ 変更で再構築），およびデッドコードリスク（完全 0 行が発生すると indptr が無駄を生むが，プルーニングで稀）が指摘される．これらを考慮し，本実験では全結合層中心に適用する．

3.5　重み圧縮手法の比較

以下に，量子化，プルーニング +COO，CSR の手法を比較した表を示す．本研究の文脈（Arduino 上 RGB 復元ニューラルネットワーク，L1 疎化）を考慮し，定性的基準で評価する．この比較から，CSR の行アクセス効率と圧縮率の高さから基盤採用を決定し，追加で量子化を組み合わせることでメモリ/速度の両立を図る．

表 1: 重み圧縮手法の比較（CSR 基盤 + 量子化追加で最適化）

手法	メモリ削減	計算効率	精度影響
量子化 (int8)	高 (4 倍)	高 (2-3 × latency 低減)	低 (<2-4 %)
プルーニング +COO	中-高 (2-5 倍)	中 (全走査)	低 (<3 %)
プルーニング +CSR	高 (9-13 倍)	高 (FLOP 3-5 倍低減)	低 (0 %)
CSR + 量子化	最高 (36-52 倍)	最高 (50 %時間短縮)	低 (<5 %)

数値は論文のベンチマークを基にしたものであり、Table 4.1-4.3(4 倍メモリ削減，精度低下 <2-4 %，COCO で 50 %時間短縮)[1] ，Table 1/4/5 (9-13 ×圧縮，非ゼロ率 7.5-11 %，FLOP 3-5 ×低減，精度損失 0 %)[2] ．これらを小規模ニューラルネットワークにスケールダウンした推定値である．次節では，CSR+ 量子化の実装を詳細に記述し，MSE 検証で確認する．

4　実験方法

本節では，重み圧縮手法として CSR（Compressed Sparse Row）形式を第一優先とし，次に CSR 圧縮モデルに対する量子化を適用してさらなる圧縮を検討する．実験は，隠れ層次元 40 から開始し，「非圧縮」「CSR 圧縮」「CSR+ 量子化」の MSE を比較して品質影響を評価する．全体の流れは，PyTorch による訓練・圧縮生成，Arduino Uno R4 WiFi へのデプロイ，センサーデータによる推論・MSE 計算である．これにより，L1 正則化による疎化を活かし，メモリ制約下での画像復元品質向上と読み取り時間短縮を実現する．目標は，MSE 相対誤差を可能な限り抑えつつ，次元を 100-200 へ拡張可能とすることである．

4.1　全体実験フロー

実験の全体フローは以下の通りである．

1. **データ準備:** 授業配布の 3×3 カラーチャート RGB 値を訓練データ（9 サンプル，ノイズ付加）とし，基準画像を評価用とし，カラーセンサーで実測データを収集する．
2. **モデル訓練:** PyTorch で ColorNet（入力 3→ 隠れ $d_1 \to d_2 \to$ 出力 3，$d_1 = d_2 = 40$ 初期）を訓練する（epochs=50,000，Adam lr=0.01，$\lambda_{L1} = 10^{-6}$）．L1 正則化で非ゼロ率 50% 以下を目指す．
3. **圧縮生成:** プルーニング（閾値 0.01）後，CSR 変換（SciPy）し，`model_parameters.h` をエクスポートする．
4. **Arduino デプロイ:** RGB 入力 → 推論 →PPM 出力する．
5. **評価基準:** MSE/PSNR（PC ソフト）とレイテンシ（millis()）を測定し，相対誤差を可能な限り抑え，次元拡張を増加させる（40→60→100→200）．

また，結果は表にまとめ，圧縮率・誤差・時間のトレードオフを分析する．

4.2 CSR 形式の実験方法及び評価方法

CSR 形式を優先的に採用する理由は，L1 正則化による重みの疎化を最大限に活かした高い圧縮率（3-10 倍）と，行指向の matvec 演算による推論高速化にある．これにより，Arduino のメモリ制約をクリアしつつ，画像復元のリアルタイム性を確保できる．以下に，具体的な手順を示す．

1. **モデル訓練（Python）:**PyTorch を用いて ColorNet を訓練し，L1 正則化により重みを疎化（非ゼロ率 <50 %）する．活性化関数として ReLU 関数を採用し，全体の表現力を維持する．
2. **プルーニングと CSR 生成:** 閾値 0.01 で重みを 0 化する（`torch.abs(w) < 0.01`）．SciPy の `sparse.csr_matrix` で CSR 形式に変換する（data, indices, indptr を NumPy 配列として出力）．これを C 配列形式で `model_parameters.h` にエクスポートする．
3. **Arduino 実装:** CSR matvec 関数を実装し，カラーセンサーからの RGB 入力を CSR forward パス（ReLU 統合）で処理し，出力 RGB を PPM 形式で生成する．
4. **評価:** 生成した PPM 画像と基準画像の MSE 算出し，非圧縮モデルとの相対誤差を算出する．40 次元での確認後，次元を増加させて再訓練・再評価を行い，拡張性を検証する．

この一連のプロセスにより，CSR 形式の有効性を定量的に確認し，さらなる量子化との組み合わせへの基盤を築く．

4.3 CSR+ 量子化による追加圧縮の実験方法及び評価方法

CSR 形式を適用した後，8-bit 量子化（uint8 重み/活性化，int32 バイアス）を組み合わせることで，メモリをさらに約 4 倍削減する．論文のスキーム（$r = S(q - Z)$）を基に，CSR の疎性を維持しつつ 200 次元対応を目指す [1] ．量子化感知訓練（QAT）により精度を保ち，全体の効率を向上させる．以下に，手順を示す．

1. **量子化感知訓練（Python）:** 訓練グラフに fake quantization を挿入する（$\hat{r} = ((r; a, b)/S) \cdot S + Z$, $S = (b - a)/255$，範囲 $[a, b]$ を EMA で学習）．初期 50k ステップで量子化を無効化し，ReLU6 活性化で範囲を安定化させる．
2. **量子化実行（Python）:** CSR データ（float32）を uint8 へ変換する（$q = ((r - Z)/S)$, scale=$max(|r|)/127$）．バイアスは $S_w S_a$ スケールで int32 とし，multiplier $M = S_w S_a / S_o$ を fixed-point でオフライン計算する．indptr/indices を int16 で圧縮し，`model_parameters.h` を更新する．
3. **Arduino 実装:** matvec 内で uint8 乗算 +int32 蓄積 + スケーリングを実装する．ReLU を int8 clamp(0,127) に対応させ，EloquentTinyML ライブラリを活用して int8 演算を効率化させる．
4. **評価:** CSR 単独 vs CSR+ 量子化の MSE を比較する．また，200 次元への拡張を検証する．

この一連の実験により，CSR 基盤上で量子化を積層し，MSE 低減と時間短縮を両立させる．プルーニングとの相乗効果で，非ゼロ率の低減と整数演算の効率を定量的に確認する．

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.