

# 2025 年度 ビジュアル情報処理 レポート課題

24G1133 安尾 晃貴

## はじめに

本研究では、5種類の抵抗に対して画像認識技術を用いた分類を行う。使用する画像は、各抵抗に印字されたカラーコードが明確に写っているものであり、これを入力として抵抗の種類を判別することを目的とする。

画像の識別手法としては、ニューラルネットワークとk-近傍法の2つを用いる。それぞれの手法について、以下に原理を説明する。

### ニューラルネットワークによる識別

ニューラルネットワークは、人間の脳の神経回路を模倣したモデルであり、複数の層（入力層、中間層、出力層）から構成される。各層のノード（ニューロン）は、前の層からの入力に対して重み付けと活性化関数による変換を行い、次の層に信号を伝える。学習の過程では、入力画像と正解ラベルを用いて誤差を計算し、その誤差を逆伝播法により各重みに反映させていくことで、分類精度を高めていく。特に画像認識では、畳み込みニューラルネットワークと呼ばれる構造が広く用いられ、画像の局所的な特徴（エッジ、色の分布、形状など）を効率的に抽出できるという利点がある。

### k-近傍法による識別

k-近傍法は、教師あり学習におけるシンプルな分類アルゴリズムの一つであり、新しいデータに対して、既知のデータの中で「距離が近い」 $k$ 個のデータを参照し、多数決により分類を行う。ここでの「距離」は通常、ユークリッド距離で定義され、特微量空間上での近さを表す。本研究では、画像から抽出された特徴ベクトルを用いて、訓練データとの距離を計算し、もっとも近い $k$ 個のラベルに基づいて、分類先を決定する。k-近傍法はモデルの学習が不要であり、シンプルである一方、分類対象の数が多くなると計算量が増加するという特性を持つ。

## 実験方法

### 画像に関する情報

本研究では、画像認識技術を用いて、5種類の抵抗（3.3R, 23KR, 33KR, 100KR, 1MR）の分類を行う。使用する画像は、各抵抗に印字されたカラーコードが明確に写っているものである。

撮影に用いたカメラはiPhone 16eであり、解像度は $3024 \times 3024$ ピクセルである。撮影環境としては、背景を白い紙で統一し、2024年7月21日の午後2時（晴れ）に照明を用いてできる限り均一に照明を当てた状

態で撮影を行った。

各抵抗に対して、真上、左側、右側、下方向、上方向の5方向からそれぞれ5枚ずつ、合計25枚の画像を撮影した。使用したデータの1部を図1、図2、図3、図4、図5に示す。このとき、各方向の画像は撮影位置を中心からわずかにずらして撮影し、視点の変化によるバリエーションを確保した。この操作を5種類の抵抗すべてに対して行い、 $25\text{枚} \times 5\text{クラス} = \text{計 } 125\text{枚}$ の画像を用意した。すべてのクラスで均等な枚数の画像を撮影している。

また、撮影画像はHEIC形式で保存されていたため、プログラムによってJPEG形式への変換を行った。さらに本研究では、データ拡張の有無による分類精度の比較も目的としており、後述する実験において、拡張前と拡張後のデータセットによる識別性能の違いを検証する予定である。



図1 上方向



図2 真上



図3 左側

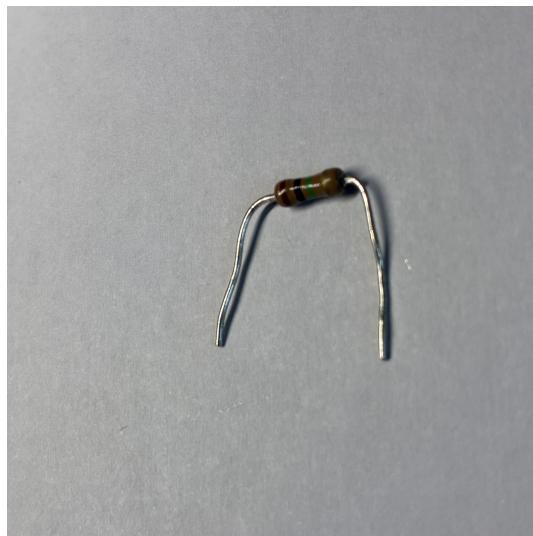


図4 右側

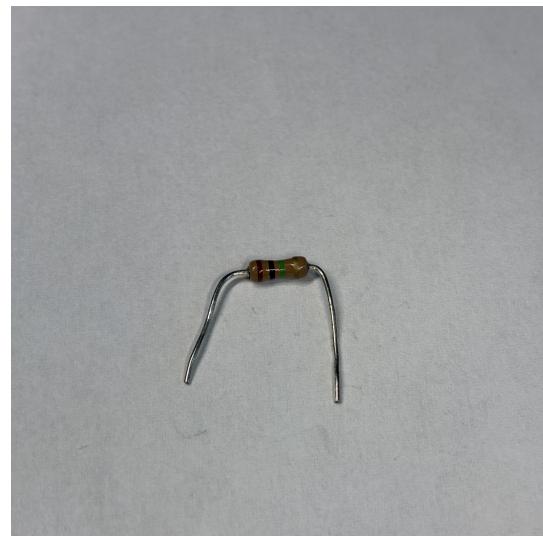


図5 下方向

## 識別方法に関する各種パラメータ

本研究では、画像から抵抗の色帯を識別するために、画像の色情報および形状情報に関する3種類の特微量を抽出し、それらを連結したベクトルを用いて分類を行った。具体的には、色のヒストグラム、色の平均値、

およびエッジ情報をそれぞれ抽出した。以下に各特微量の詳細を述べる。

### 色のヒストグラム

画像の色情報を RGB 各チャネルごとにヒストグラムとして表現することで、色の構成比や分布を特微量として抽出した。手順は以下のとおりである。

1. 画像を  $64 \times 64$  ピクセルにリサイズし、RGB 形式に変換する。
2. 各チャネル (R, G, B) に対して、輝度値 (0~255) を 16 個の bin (区間) に分割し、ヒストグラムを作成する。
3. 各チャネルのヒストグラム (長さ 16) を連結し、合計 48 次元の特微量ベクトルとする。
4. 値を総和で割ることで、ヒストグラムを正規化する (各画像で合計 1 になるように)。

この特微量は、照明条件の変化に対して頑健であり、色の比率を捉えるのに有効である。

### 色の平均値

画像の RGB 各チャネルの平均輝度を計算することで、画像全体の支配的な色を 3 次元ベクトルで表現した。

1. 画像を  $64 \times 64$  ピクセルにリサイズし、RGB 形式に変換する。
2. NumPy 配列に変換後、各チャネル (R, G, B) に対して平均輝度値を計算する。
3. それ 255 で割って正規化し、値の範囲を  $[0, 1]$  にスケーリングする。

色の平均値は非常に低次元 (3 次元) ながら、背景色や全体的な色調の違いを簡易に捉えることができる。

### エッジ情報

抵抗の形状や色帯の輪郭などの構造的特徴を捉えるため、Sobel フィルタを用いたエッジ検出を行った。

1. 画像を  $32 \times 32$  ピクセルにリサイズし、グレースケール (1 チャネル) に変換する。
2. NumPy 配列に変換し、OpenCV の Sobel フィルタを用いて X 方向および Y 方向の勾配を計算する。
3. 各画素におけるエッジの強度を、勾配の大きさ  $\sqrt{G_x^2 + G_y^2}$  として計算する。
4. 勾配の最大値で割って正規化し、得られたエッジマップ ( $32 \times 32$ ) を 1 次元配列 (1024 次元) に変換する。

このエッジ情報により、抵抗の色帯の位置や本体形状など、色情報だけでは区別しづらいパターンも捉えることが可能になる。

### 分類器における $k$ の値の選定

$k$  近傍法 ( $k$ -Nearest Neighbors) は、入力された特微量ベクトルに対して、学習データ中の  $k$  個の最近傍点を参照し、多数決によりクラスを決定する手法である。ここで、 $k$  の値の設定は分類性能に大きく影響を及ぼすため、適切な  $k$  の値を選定する必要がある。

本研究では、最も分類精度が高くなった  $k$  の値を基準とし、その前後 2 つずつの値を含めた範囲で評価を行った。たとえば、 $k = 5$  のときに最も高い正答率が得られた場合には、 $k = 3, 4, 5, 6, 7$  の 5 つの値について分類性能を比較した。

分類精度が同程度である場合 (例: 正答率が 0.5, 0.6, 0.5 のような場合) には、より高い正答率を持つ  $k$

(この例では 0.6) を代表値として採用した。このようにして、モデルの過学習や分類の粗さを避けつつ、最も安定して高い精度を示す  $k$  を選定した。

選定された  $k$  の値は、その後の評価実験や混同行列の分析において一貫して使用されている。

## 実験結果

### ニューラルネットワークモデルの学習と評価

#### データ拡張の有無による識別精度の比較

元画像のみで学習・評価した結果とデータの拡張を行った後の学習・評価結果を比較を以下に示す。また、以下に元画像を元に行ったデータ拡張を示す。

- 元画像（変更なし）
- 左右反転（水平フリップ）
- 上下反転（垂直フリップ）
- 180 度回転
- 少しだけ回転（例：± 10 度など）
- 明るさを上げる（画像を明るくする）
- 明るさを下げる（画像を暗くする）
- 彩度を上げる（色を濃くする）
- 彩度を下げる（色を薄くする）
- コントラストを上げる（明暗差を強調）
- コントラストを下げる（全体的にのっぺり）
- ぼかし（ガウシアンブラーなど）
- ランダムな領域を切り出す（クロップ処理）

図 6 および図 7 は、それぞれ元画像のみで学習・評価を行った場合と、データ拡張を適用した場合における、学習データおよびテストデータに対する精度と損失関数の推移を示している。

まず、元画像のみで学習を行った場合では、学習データに対する損失関数はエポックの進行とともに順調に低下している一方で、テストデータに対する損失はほとんど変化しておらず、また精度についても学習データとテストデータの間に顕著な差が見られた。これは、モデルが学習データに対して過学習しており、未知のデータに対する汎化性能が十分に得られていないことを示唆している。

一方、データ拡張を適用した場合では、学習データとテストデータの両方に対して、精度が約 0.8 程度まで向上し、かつ収束していることが確認できる。また、損失関数についても両データセットで安定的に減少しており、拡張によって学習が安定化し、モデルの汎化性能が向上していることが示された。

また、それぞれの混同行列の結果を図 8 および図 9 に示す。図 8（拡張なし）では、正解ラベルに対して他クラスへの誤分類が多く見られ、特に 100KR や 1MR といったクラスでは誤認識が目立つ。一方で、図 9（拡張あり）では、ほとんどのクラスにおいて対角成分（正解の分類数）が明瞭に多く、高い識別精度が得られていることがわかる。これにより、データ拡張によってモデルの識別能力が全体的に向上していることが視覚的にも確認できる。

さらに表 1 より、データ拡張の有無によって分類性能に明確な違いが見られた。全体の精度（Accuracy）

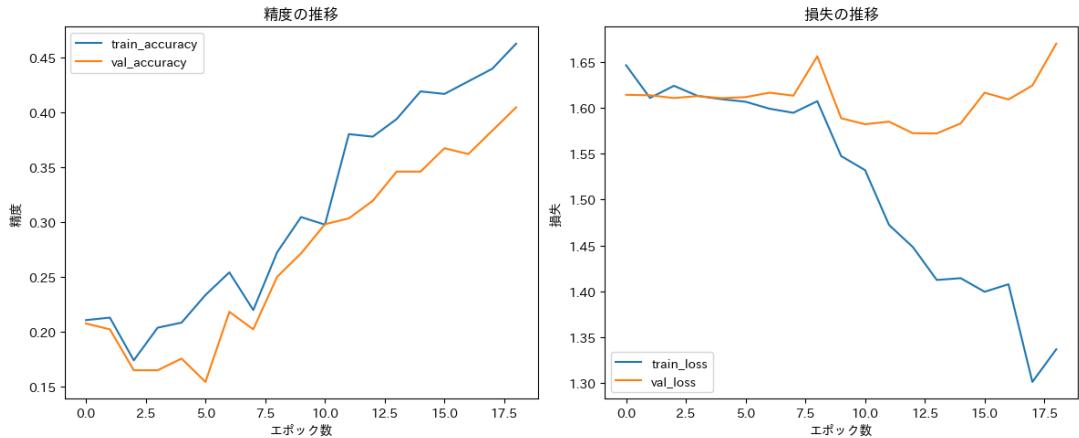


図 6 元画像のみで学習・評価した場合の精度と損失関数の推移

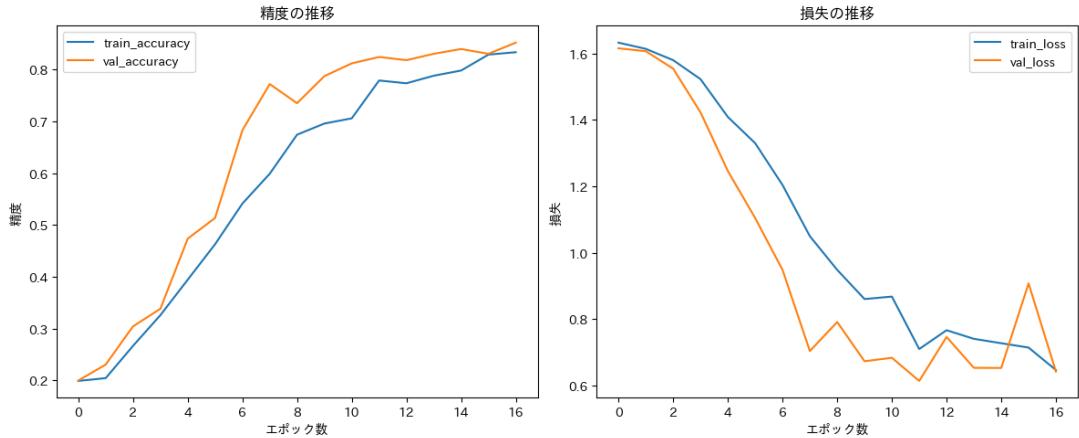


図 7 データ拡張を適用した場合の精度と損失関数の推移

は、拡張なしの場合は 0.35、拡張ありの場合は 0.82 であった。また、各クラスにおいても精度の差が顕著であり、以下のような傾向が確認された。

- 100KR : F1-score が 0.29（拡張なし）から 0.88（拡張あり）に向上。
- 1MR : F1-score が 0.29 から 0.80 に向上。
- 23KR : F1-score が 0.39 から 0.77 に向上。
- 33KR : F1-score が 0.49 から 0.97 に向上。
- 3\_3KR : F1-score が 0.28 から 0.69 に向上。

また、macro 平均および weighted 平均の各指標も以下のように向上した。

- Macro 平均の F1-score : 0.35（拡張なし）→ 0.82（拡張あり）
- Weighted 平均の F1-score : 0.34（拡張なし）→ 0.83（拡張あり）

以上の結果から、データ拡張はモデルの過学習を抑制し、テストデータに対する分類精度の向上に有効であることが明らかとなった。

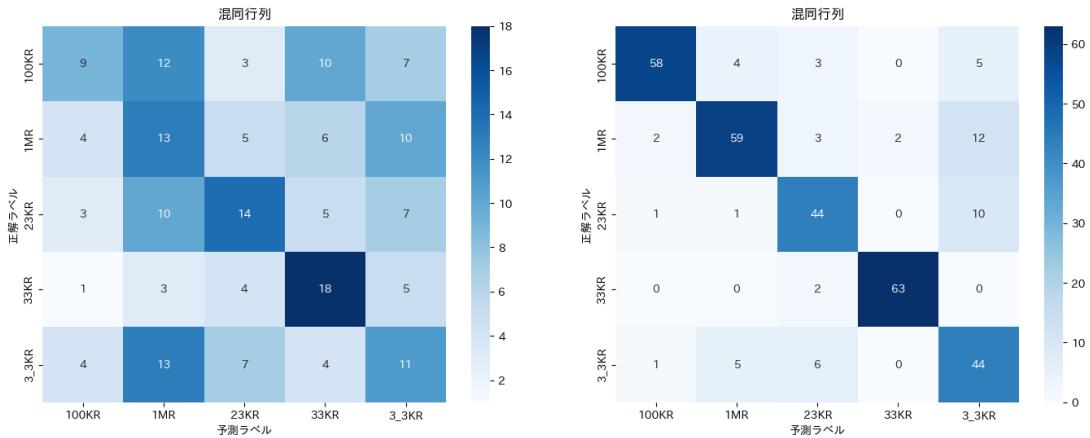


図 8 元画像の混合行列

図 9 データ拡張後の混合行列

表 1 データ拡張の有無による分類結果の比較

2*クラス	拡張なし				拡張あり			
	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
100KR	0.43	0.22	0.29	41	0.94	0.83	0.88	70
1MR	0.25	0.34	0.29	38	0.86	0.76	0.80	78
23KR	0.42	0.36	0.39	39	0.76	0.79	0.77	56
33KR	0.42	0.58	0.49	31	0.97	0.97	0.97	65
3_3KR	0.28	0.28	0.28	39	0.62	0.79	0.69	56
<b>Accuracy</b>	0.35 (全体 188 枚)				0.82 (全体 325 枚)			
Macro avg	0.36	0.36	0.35	188	0.83	0.83	0.82	325
Weighted avg	0.36	0.35	0.34	188	0.84	0.82	0.83	325

### 背景の違いによる識別精度の変化

データ拡張をした学習済みのモデルを用いて、背景の色や背景上のノイズが識別精度に与える影響を調査した。

まず、背景色を黒に変更し、その他の条件は変更せずに真上から撮影した画像を用いて評価を行った。図10は、背景を黒色に変更した場合の予測結果を示している。5種類の抵抗をそれぞれ1枚ずつモデルに入力したところ、3枚が正しく分類され、正答率は60%であった。この結果は、学習時に使用した背景（白）と異なる条件下では、識別精度が低下する可能性があることを示している。

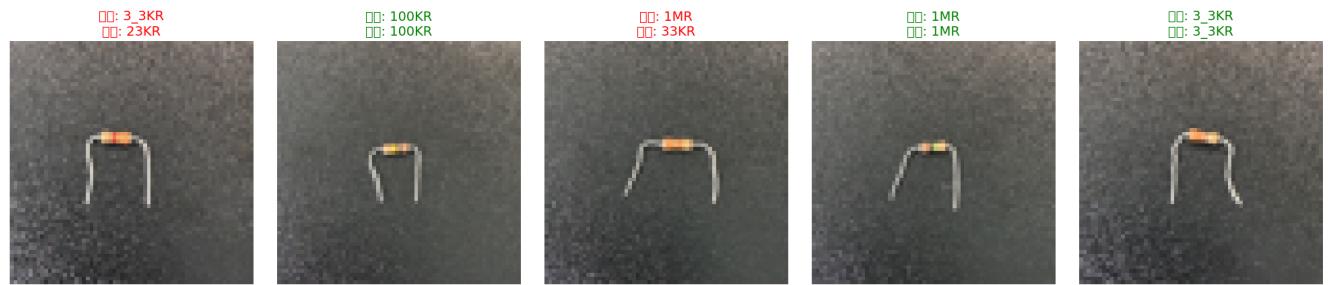


図10 背景を黒色に変更した場合の識別精度

次に、背景色は学習データと同様の白とし、その上に文字のノイズを加えた画像を用いてモデルの識別性能を評価した。図11にその結果を示す。5種類の抵抗を各1枚ずつ入力したところ、正しく分類されたのは1枚のみであり、正答率は20%であった。この結果から、背景に含まれるノイズ（特に文字情報）がモデルの判断を妨げる要因となりうることが示唆される。



図11 背景に文字を加えたことに識別精度の変化

### 形状の違いによる識別精度の変化

学習済みモデルを用いて、抵抗の形状が識別精度に与える影響を調査した。具体的には、抵抗のリード（足）を伸ばした状態の画像を用いて、学習済みモデルによる分類精度を評価した。

図12に、リードを伸ばした抵抗画像を用いた場合の予測結果を示す。5種類の抵抗を1枚ずつ入力したところ、正しく分類されたのは1枚のみであり、正答率は20%であった。これは、抵抗のリードの有無という形状の変化がモデルの識別性能に悪影響を及ぼしていることを示唆している。



図 12 抵抗の足を伸ばした場合の識別精度

## k-近傍法の学習と評価

元画像からニューラルネットワーク同様にデータを拡張し、その画像を元に k-近傍法を用いて識別を行った。

### 特徴量の違いによる識別精度の変化 (k の値はそれぞれ最適な値)

k-近傍法を用いて抵抗の識別を行い、特徴量の違いによる分類性能への影響を調査した。各特徴量に対して、最も精度が高くなる k の値を選定して分類を実施した。使用した特徴量は以下の 3 種類である：色のヒストグラム、色の平均値、およびエッジ情報。表 2 に、各特徴量を用いた場合の分類性能を示す。

色のヒストグラムを用いた場合は、Precision, Recall, F1-score の全体的なバランスが良く、Accuracy は 0.78 であった。とくに「33KR」クラスにおいては、高い再現率 (Recall 0.95) と F1 スコア (0.93) が得られた。

一方、色の平均値を用いた場合は、全体の Accuracy が 0.71 とやや低下し、「3<sub>3</sub>KR」クラスでは Recall は比較的高い (0.77) もの、Precision が低下し、全体の F1 スコアは他の特徴量と比べて劣っていた。

エッジ情報を特徴量とした場合、Accuracy は 0.78 とヒストグラムと同等の精度が得られた。とくに「100KR」や「1MR」、「23KR」などのクラスで安定したスコアが得られ、全体的にバランスの良い分類結果となった。

以上の結果から、色のヒストグラムとエッジ情報の双方が有効な特徴量であり、単純な色の平均よりもより高い識別精度が得られることがわかった。

表 2 特徴量ごとの分類性能比較（各クラスの精度）

5*クラス	色のヒストグラム			色の平均			エッジ		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
100KR	0.69	0.89	0.78	0.69	0.71	0.70	0.81	0.80	0.81
1MR	0.81	0.72	0.76	0.71	0.71	0.71	0.79	0.77	0.78
23KR	0.71	0.62	0.67	0.72	0.59	0.65	0.73	0.77	0.75
33KR	0.90	0.95	0.93	0.84	0.78	0.81	0.79	0.80	0.79
3 <sub>3</sub> KR	0.83	0.71	0.77	0.62	0.77	0.69	0.75	0.73	0.74
Accuracy	0.78			0.71			0.78		

### 特徴量を複数にした場合

k-近傍法において、色のヒストグラム、色の平均、エッジ情報の特徴量を組み合わせた場合の分類性能を調査した。

■k 近傍法における k の値の変化と分類性能の比較 本実験では、k 近傍法のパラメータ k を 3, 4, 5 と変化させ、それぞれの分類性能を比較した。評価指標として、Precision, Recall, F1-score, および Accuracy を用いた。表 3 に示すように、k=3 の場合、33KR クラスで最も高い F1 スコア 0.85 を記録し、他のクラスも比較的安定した性能を示した。k=4 では、全体的に精度が向上し、特に 100KR クラスで F1 スコアが 0.83 に達したこと、総合 Accuracy は 0.79 と最も高かった。k=5 では、33KR クラスの F1 スコアはわずかに低下したが、他クラスでは安定した成績を維持し、Accuracy は 0.78 であった。これらの結果から、k=4 が最もバランス良く分類性能を発揮し、今回のデータセットにおける最適な k の選択肢であると考えられる。一方、k=3 や k=5 でも大きく性能が劣ることではなく、k の値を調整することで性能の微調整が可能であることが示された。

■特徴量を複数にした場合の分類性能 色のヒストグラム、色の平均、エッジ情報の 3 種類の特徴量を組み合わせた特徴ベクトルを用いて、k-近傍法による分類を行った。表 3 に、k の値を 3, 4, 5 と変化させた際の分類性能を示す。

$k = 4$  のときに最高の Accuracy 0.79 が得られ、单一の特徴量を用いた場合（最大 Accuracy 0.78）よりもわずかに向上している。各クラスの F1 スコアもおおむね高く、特に「100KR」や「1MR」では  $k = 4$  で 0.83, 0.77 と良好な性能を示した。

一方、单一特徴量の場合は色のヒストグラムやエッジ情報が高い性能（Accuracy 0.78）を示したもの、色の平均ではやや精度が劣っていた。

これらの結果から、複数の特徴量を組み合わせることで、より安定した識別性能が得られることが示された。特に、k の値を適切に調整することで、分類精度の微調整が可能であった。

また、単一予測の色の平均の予測精度が良くないため、色の平均のみを外した場合も確認したが、評価は 0.79 と変化はなかった。このことから、色の平均はこのモデルには必須ではないことがわかった。

表 3 k 近傍法における k の値の違いによる分類性能の比較

5* クラス	$k = 3$			$k = 4$			$k = 5$		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
100KR	0.76	0.81	0.79	0.83	0.83	0.83	0.81	0.80	0.81
1MR	0.71	0.79	0.75	0.79	0.76	0.77	0.78	0.76	0.77
23KR	0.75	0.70	0.72	0.71	0.75	0.73	0.71	0.71	0.71
33KR	0.91	0.80	0.85	0.86	0.85	0.85	0.84	0.83	0.84
3.3KR	0.78	0.75	0.76	0.74	0.75	0.74	0.72	0.77	0.74
Accuracy	0.78			0.79			0.78		

### 背景の違いによる識別精度の変化

データ拡張をし、特徴量を色のヒストグラムとエッジのみを用いた学習済みのモデルを用いて、背景の色や背景上のノイズが識別精度に与える影響を調査した。

まず、背景色を黒に変更し、その他の条件は変更せずに真上から撮影した画像を用いて評価を行った。図15は、背景を黒色に変更した場合の予測結果を示している。5種類の抵抗をそれぞれ1枚ずつモデルに入力したところ、一つも正しく分離されず、正答率は0%であった。この結果は、学習時に使用した背景（白）と異なる条件下では、識別精度が低下する可能性があることを示している。



図13 背景を黒色に変更した場合の識別精度

次に、背景色は学習データと同様の白とし、その上に文字のノイズを加えた画像を用いてモデルの識別性能を評価した。図14にその結果を示す。5種類の抵抗を各1枚ずつ入力したところ、正しく分類されたのは1枚のみであり、正答率は20%であった。この結果から、背景に含まれるノイズがモデルの判断を妨げる要因となりうることが示唆される。

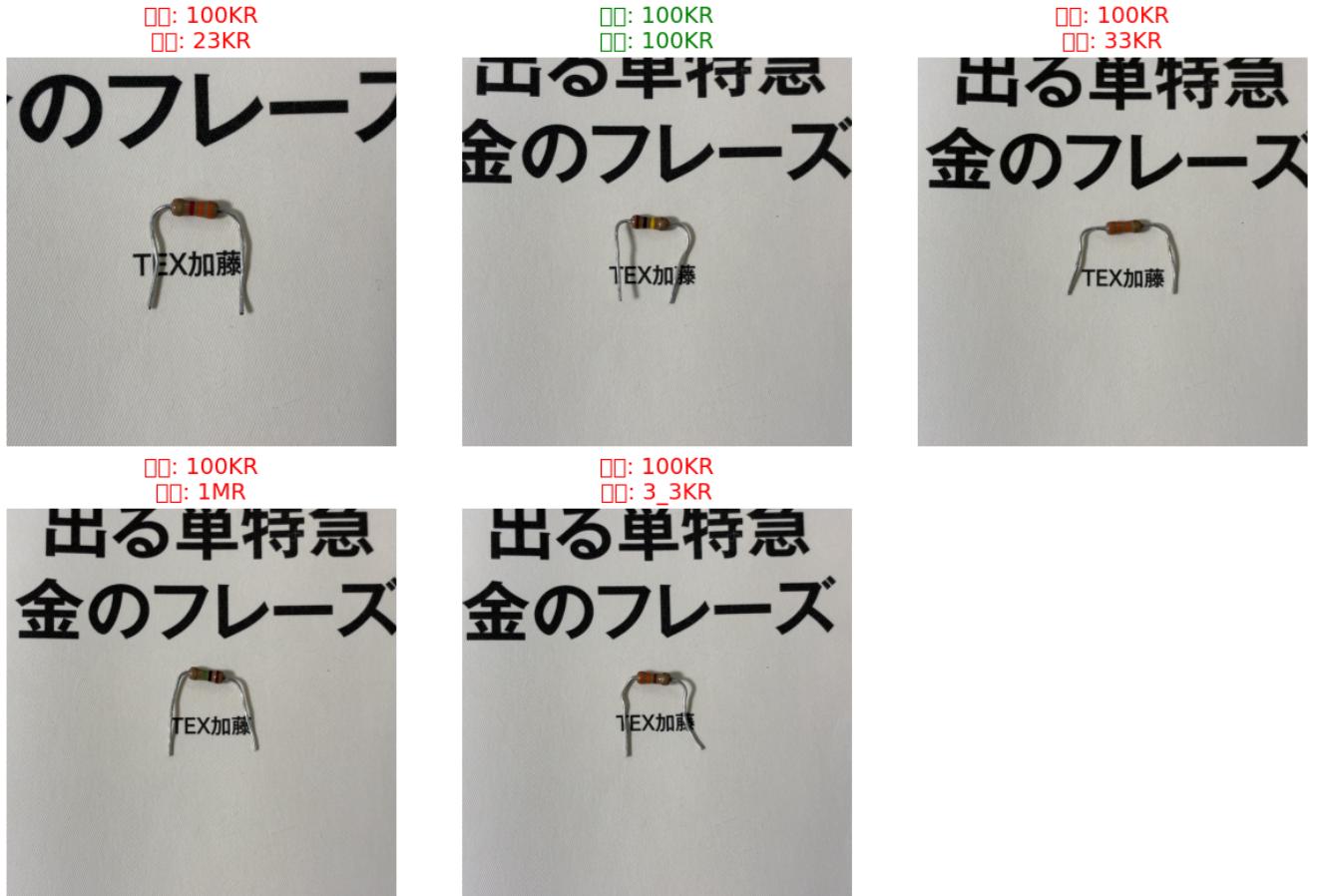


図 14 背景に文字を加えたことに識別精度の変化

#### 形状の違いによる識別精度の変化

学習済みモデルを用いて、抵抗の形状が識別精度に与える影響を調査した。具体的には、抵抗のリード（足）を伸ばした状態の画像を用いて、学習済みモデルによる分類精度を評価した。

図 15 に、リードを伸ばした抵抗画像を用いた場合の予測結果を示す。5種類の抵抗を1枚ずつ入力したところ、正しく分類されたのは2枚のみであり、正答率は40%であった。これは、抵抗のリードの有無という形状の変化がモデルの識別性能に悪影響を及ぼしていることを示唆している。

#### k-近傍法とニューラルネットワークの比較

本研究では、ニューラルネットワークと k 近傍法の両手法を用いて、抵抗カラーコード画像の分類を行い、それぞれの識別精度を比較した。両モデルに対して、データ拡張を施した同一条件の画像データを使用し、学習条件を揃えた上で評価を行った。

表 4 に各モデルにおける Accuracy および macro 平均 F1-score の比較結果を示す。ニューラルネットワークでは、Accuracy が 0.82、macro F1-score が 0.82 となり、高い性能を記録した。一方、k 近傍法（混合特徴量モデル）では、Accuracy が 0.79、macro F1-score が 0.78 と、ニューラルネットワークにやや劣る結果



図 15 抵抗の足を伸ばした場合の識別精度

となつた。

また、混同行列の分析において、ニューラルネットワークでは全体的に対角成分が明瞭であり、多くのクラスで高い再現率が得られていた。一方で、 $k$  近傍法では一部のクラス（特に 23KR と 33KR）において混同が見られ、クラス間での判別がやや不安定であることが確認された。

さらに、形状を変化させた画像（抵抗のリードを伸ばしたもの）に対する識別性能についても評価を行った。その結果、 $k$  近傍法ではこの条件下でも一定の分類精度を維持していたのに対し、ニューラルネットワークでは識別精度が著しく低下した。

表4 ニューラルネットワークと  $k$  近傍法の分類結果の比較

2*クラス	ニューラルネットワーク			$k$ 近傍法		
	Precision	Recall	F1-score	Precision	Recall	F1-score
100KR	0.94	0.83	0.88	0.83	0.83	0.83
1MR	0.86	0.76	0.80	0.79	0.76	0.77
23KR	0.76	0.79	0.77	0.71	0.75	0.73
33KR	0.97	0.97	0.97	0.86	0.85	0.85
3_3KR	0.62	0.79	0.69	0.74	0.75	0.74
Accuracy		0.82			0.79	
Macro Avg	0.83	0.83	0.82	0.78	0.79	0.79
Weighted Avg	0.84	0.82	0.83	0.79	0.79	0.79

## 考察

### ニューラルネット

ニューラルネットワークを用いた識別において、データ拡張の有無による精度や損失関数、混同行列の結果を比較したところ、データを拡張することでモデルの識別精度が大幅に向上去ることが確認された。これは、データ拡張によって学習データの多様性が高まり、モデルがより一般的なパターンを学習できたためであり、過学習の抑制にもつながったと考えられる。

一方で、使用された拡張データはすべて元の画像を基に生成されているため、背景や物体の形状などに関しては元画像と同様の情報が多く残っている。このため、背景色を変更した画像や、背景にノイズ（文字など）を加えた画像、あるいは抵抗のリード（足）を伸ばした画像といった、元データと異なる条件の画像に対しては、識別精度が著しく低下する結果が得られた。

これらの結果から、単に元画像を加工してデータを拡張するだけでは、未知の条件下での識別性能を十分に高めることは難しいといえる。今後は、単純な拡張処理に加えて、元画像そのものの多様性を高めることが重要である。さらに、背景の違いや形状の変化に強いモデルを構築するためには、入力画像から抵抗部分のみを検出・抽出する前処理モデルを導入し、対象物に対してのみ識別を行う仕組みを構築することが有効である。このようなアプローチを探ることで、背景に依存しない頑健な分類モデルが実現でき、未知のデータに対する汎化性能の向上が期待される。

### k 近傍法

k 近傍法を用いた識別においては、使用する特徴量の種類および組み合わせが分類性能に大きく影響することが確認された。色のヒストグラム、色の平均値、エッジ情報の3種類の特徴量をそれぞれ個別に用いた場合、色のヒストグラムおよびエッジ情報は比較的高い識別性能を示したが、色の平均値単独では精度がやや低下する傾向が見られた。一方、これらの特徴量を組み合わせた混合モデルによって分類を行った場合、より高い識別精度が得られ、特定のクラスに対する分類の偏りも少なく、バランスの取れた性能が実現された。

これは、複数の特徴量を組み合わせることで、単一の特徴量では捉えきれない情報を相互に補完し、より安定した識別が可能となったためであると考えられる。また、k の値を適切に調整することによって、ノイズへの感度や識別の粗さを制御できることも示された。特に k=4 のときに最も高い精度が得られたことから、データの密度やクラス分布に応じてパラメータ調整を行うことの重要性が確認された。

一方で、k 近傍法は距離に基づく分類であるため、入力画像が訓練データと異なる条件（例：背景の変化やノイズの混入、形状の変化など）を含む場合には、類似度の判断が難しくなり、識別精度が大きく低下することが観察された。特に背景を黒にした場合や、背景に文字を加えた場合などは、特徴量の分布が大きく変化し、誤分類が多発する傾向があった。

のことから、k 近傍法においてもニューラルネットワークと同様に、単に拡張されたデータを用いるだけでは未知条件への頑健性を高めるには不十分であるといえる。今後は、前処理段階で抵抗の本体部分のみを抽出する仕組みを導入し、背景や不要領域の影響を除去した上で特徴抽出を行うことが有効である。また、特徴量の選択や次元削減を適切に行うこと、ノイズ成分を除去しつつ識別性能の向上が期待できる。このような対策により、k 近傍法においても未知の条件下で安定した分類が可能となり、より実用的なモデルの構築につながると考えられる。

## k 近傍法とニューラルネットの比較

ニューラルネットワークは畳み込み演算を通じて、画像の空間的・色彩的な特徴を自動的に学習できるため、通常の撮影条件においては k 近傍法よりも高い分類性能を示した。特に明瞭なカラーコードが写った画像に対しては、高い識別精度と再現率を維持できていたことから、パターン抽出においてニューラルネットの優位性が確認された。

一方で、抵抗のリードを伸ばすといった形状の変化が加わった画像に対しては、ニューラルネットワークが形状に過剰に適応していた可能性があり、精度の低下が見られた。これに対し、k 近傍法では特微量空間に基づいた分類を行うため、入力画像にある程度の構造的变化あっても、訓練データ内の類似例に基づいて柔軟な分類ができていたと考えられる。このことから、形状変化に対しては k 近傍法の方が頑健であり、実環境のように物体の配置や形が一定でない場面においては、k 近傍法の活用が有効である可能性が示唆される。

ただし、今回使用した形状変化画像や背景変化画像の枚数は極めて少なく、統計的な評価には不十分であった。したがって、これらの条件下での手法間の比較結果はあくまで傾向の一例であり、今後より多くの追加画像を収集した上で、定量的な評価を行う必要がある。その上で、各手法の得意・不得意な条件を明確にし、用途に応じた手法選択を検討することが今後の課題である。

## まとめ

本研究では、抵抗のカラーコード画像に対して、ニューラルネットワークおよび k 近傍法を用いた分類を行い、データ拡張の有無や背景・形状の違いが識別精度に与える影響を検証した。その結果、特にニューラルネットワークでは、データ拡張を行うことでモデルの過学習が抑制され、精度と汎化性能の大幅な向上が確認された。k 近傍法においても、複数の特微量を組み合わせることでバランスの取れた分類が可能となり、混合特微量を用いたモデルでは、分類の偏りが少ないと判明した。

一方で、使用された拡張データはすべて元画像に基づいて生成されており、背景や物体形状が学習時と異なる条件では識別精度が大きく低下する傾向が見られた。このことから、単なる拡張処理では対応しきれない入力の変動に対しては、背景の除去や抵抗部分の検出といった前処理の導入が有効であると考えられる。

今後の改善点として、まず元画像の追加によるデータ多様性の向上が挙げられる。今回の拡張処理は限られた撮影条件の画像から派生したものであるため、異なる環境・照明・背景下で新たに画像を収集し、より現実的なデータセットの構築を行うことで、未知の条件下でも頑健に動作するモデルの構築が期待される。

さらに、今回の実験では検証できなかったアンサンブル学習の導入も、今後の有効な手法として挙げられる。アンサンブル学習は、複数の異なるモデルを統合して判断を行う手法であり、単一のモデルでは捉えきれないパターンを補完することで、精度や安定性の向上が期待される。特に、CNN と k 近傍法のように異なる性質を持つ分類器を組み合わせることで、より高い汎化性能を得られる可能性がある。

以上のような改善策を踏まえた拡張的な研究を今後行うことで、抵抗カラーコードの画像識別における精度と実用性のさらなる向上が期待される。

## ニューラルネットのプログラム

以下にニューラルネットのプログラムと評価のコードを示す。ライブリーとデータの分割、モデルの作成、評価のみを掲載する。

---

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import colormaps
5 import japanize_matplotlib
6
7 import glob
8 from sklearn.svm import SVC
9 from sklearn.model_selection import train_test_split
10 from sklearn.metrics import accuracy_score
11 from sklearn.tree import DecisionTreeClassifier, plot_tree
12 from sklearn.ensemble import AdaBoostClassifier
13
14 import torch
15 import torch.nn as nn
16 import torch.optim as optim
17 from torch.utils.data import TensorDataset, DataLoader
18
19 import networkx as nx
20
21 %matplotlib inline
```

---

```
1 import os
2 import numpy as np
3 from PIL import Image
4 from sklearn.model_selection import train_test_split
5 from tensorflow.keras.utils import to_categorical
6
7 # パラメータ
8 IMG_SIZE = 64
9 DATA_DIR = 'img/resistor_images_augmented_2'
10
11 # ラベル名を取得（フォルダ名）
12 # ラベル名を取得（フォルダ名のみ、隠しファイルやファイルを除外）
13 labels = sorted([
14     name for name in os.listdir(DATA_DIR)
15     if os.path.isdir(os.path.join(DATA_DIR, name)) and not name.startswith(
16         '.')
16 ])
17
18 label_to_index = {label: i for i, label in enumerate(labels)}
19
20 # データ読み込み
```

```
21 X = []
22 y = []
23
24 for label in labels:
25     label_dir = os.path.join(DATA_DIR, label)
26     for subfolder, _, files in os.walk(label_dir):
27         for file in files:
28             if file.endswith('.jpg'):
29                 img_path = os.path.join(subfolder, file)
30             try:
31                 img = Image.open(img_path).convert('RGB')
32                 img = img.resize((IMG_SIZE, IMG_SIZE))
33                 X.append(np.array(img))
34                 y.append(label_to_index[label])
35             except:
36                 print(f"読み込み失敗:{img_path}")
37
38 X = np.array(X) / 255.0 # 正規化
39 y = to_categorical(y)    # One-hot encoding
40
41 # データ分割
42 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20,
43                                                 random_state=42)
44 print(f"データ形状:{X_train.shape},{y_train.shape}")
```

---

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
3     Dropout
4 from tensorflow.keras.optimizers import Adam
5 from tensorflow.keras.callbacks import EarlyStopping
6
6 # モデル構築
7 model = Sequential([
8     Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE,
9             3)),
10    MaxPooling2D((2, 2)),
11
11    Conv2D(64, (3, 3), activation='relu'),
12    MaxPooling2D((2, 2)),
13
14    Conv2D(128, (3, 3), activation='relu'),
15    MaxPooling2D((2, 2)),
16
17    Flatten(),
18    Dense(128, activation='relu'),
19    Dropout(0.5),
20    Dense(len(labels), activation='softmax') # クラス数に合わせる
21 ])
```

```

22
23 # コンパイル
24 model.compile(
25     optimizer=Adam(),
26     loss='categorical_crossentropy',
27     metrics=['accuracy']
28 )
29
30 # コールバック（早期終了）
31 early_stopping = EarlyStopping(monitor='val_loss', patience=5,
32                                 restore_best_weights=True)
33
34 history = model.fit(
35     X_train, y_train,
36     validation_data=(X_val, y_val),
37     epochs=30,
38     batch_size=32,
39     callbacks=[early_stopping]
40 )
41
42 # モデル保存
43 model.save('resistor_model.h5')
44 print("▣ モデル学習と保存が完了しました")

```

---

```

1     from sklearn.metrics import classification_report, confusion_matrix
2     import seaborn as sns
3
4     # ラベル名を取得（フォルダ名）※隠しファイル除外
5     #labels = sorted([d for d in os.listdir(DATA_DIR) if not d.startswith
6     #    ('.')])
6     #label_to_index = {label: i for i, label in enumerate(labels)}
7
8
9     # 予測と正解を取得
10    y_pred = model.predict(X_val)
11    y_pred_labels = np.argmax(y_pred, axis=1)
12    y_true_labels = np.argmax(y_val, axis=1)
13
14    # 混同行列
15    cm = confusion_matrix(y_true_labels, y_pred_labels)
16    plt.figure(figsize=(8, 6))
17    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
18                xticklabels=labels, yticklabels=labels)
19    plt.xlabel('予測ラベル')
20    plt.ylabel('正解ラベル')
21    plt.title('混同行列')
22    plt.show()
23

```

```
24      # 詳細な分類レポート
25      print(classification_report(y_true_labels, y_pred_labels, target_names=
labels))
```

---

## k-近傍法のプログラム

以下に k-近傍法のプログラムと評価のコードを示す。ライブリーとデータの分割、モデルの作成、評価のみを掲載する。コードの長さの都合上、特徴量を複数使用した時のみを掲載する。

```
1   import cv2
2   import numpy as np
3   import matplotlib.pyplot as plt
4   from matplotlib import colormaps
5   import japanize_matplotlib
6
7   import glob
8   from sklearn.svm import SVC
9   from sklearn.model_selection import train_test_split
10  from sklearn.metrics import accuracy_score
11  from sklearn.tree import DecisionTreeClassifier, plot_tree
12  from sklearn.ensemble import AdaBoostClassifier
13
14  import torch
15  import torch.nn as nn
16  import torch.optim as optim
17  from torch.utils.data import TensorDataset, DataLoader
18
19  import networkx as nx
20
21 %matplotlib inline
```

```
1   import os
2   import numpy as np
3   from PIL import Image
4   from sklearn.model_selection import train_test_split
5   from sklearn.neighbors import KNeighborsClassifier
6   from sklearn.metrics import classification_report
7
8   # パラメータ
9   IMG_SIZE = 32 # 特徴量数を抑えるため縮小
10  DATA_DIR = 'img/resistor_images_augmented_2'
11
12  # ラベル取得（フォルダ名から）
13  labels = sorted([
14      name for name in os.listdir(DATA_DIR)
15      if os.path.isdir(os.path.join(DATA_DIR, name)) and not name.
16         startswith('..')
17  ])
18  label_to_index = {label: i for i, label in enumerate(labels)}
19
20  # データ読み込み
21  X = []
```

```

21     y = []
22
23     for label in labels:
24         label_dir = os.path.join(DATA_DIR, label)
25         for root, _, files in os.walk(label_dir):
26             for file in files:
27                 if file.lower().endswith('.jpg'):
28                     path = os.path.join(root, file)
29                     try:
30                         img = Image.open(path).convert('L') # グレースケール
31                         img = img.resize((IMG_SIZE, IMG_SIZE))
32                         feature = np.array(img).flatten() / 255.0 # して正規化
33                         flatten
34                         X.append(feature)
35                         y.append(label_to_index[label])
36                     except Exception as e:
37                         print(f"■読み込み失敗:{path}:{e}")
38
39     X = np.array(X)
40     y = np.array(y)
41
42     # データ分割
43     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
44                                                       random_state=42)
45
46     # モデルKNN
47     knn = KNeighborsClassifier(n_neighbors=5)
48     knn.fit(X_train, y_train)
49
50     # 評価
51     y_pred = knn.predict(X_test)
52     print(classification_report(y_test, y_pred, target_names=labels))

```

---

```

1     # 色のヒストグラム
2     def extract_color_histogram(img, bins=16):
3         img = img.resize((64, 64)).convert('RGB')
4         r, g, b = img.split()
5         hist_r = np.histogram(np.array(r).flatten(), bins=bins, range=(0,
50             256))[0]
6         hist_g = np.histogram(np.array(g).flatten(), bins=bins, range=(0,
50             256))[0]
7         hist_b = np.histogram(np.array(b).flatten(), bins=bins, range=(0,
50             256))[0]
8         hist = np.concatenate([hist_r, hist_g, hist_b])
9         return hist / np.sum(hist) # 正規化
10
11
12     # 色の平均
13     def extract_color_mean(img):

```

```
14     img = img.resize((64, 64)).convert('RGB')
15     r, g, b = np.array(img).transpose(2, 0, 1) # (3, H, W)
16     mean_r = np.mean(r)
17     mean_g = np.mean(g)
18     mean_b = np.mean(b)
19     return np.array([mean_r, mean_g, mean_b]) / 255.0
20
21
22     import cv2
23
24     # エッジ特徴量の抽出
25     def extract_edge_feature(img, size=32):
26         img = img.resize((size, size)).convert('L') # グレースケール
27         img_np = np.array(img)
28         sobelx = cv2.Sobel(img_np, cv2.CV_64F, 1, 0, ksize=3)
29         sobely = cv2.Sobel(img_np, cv2.CV_64F, 0, 1, ksize=3)
30         edge = np.hypot(sobelx, sobely) # 勾配の大きさ
31         edge = edge / np.max(edge) # 正規化
32         return edge.flatten()
```

---

```
1     def extract_combined_features(img):
2         feat1 = extract_color_histogram(img)
3         feat2 = extract_color_mean(img)
4         feat3 = extract_edge_feature(img)
5         return np.concatenate([feat1, feat2, feat3])
```

---

```
1     import os
2 import numpy as np
3 from PIL import Image
4 from sklearn.model_selection import train_test_split
5 from sklearn.neighbors import KNeighborsClassifier
6 from sklearn.metrics import classification_report
7
8 DATA_DIR = 'img/resistor_images_augmented_2'
9 labels = sorted([
10     name for name in os.listdir(DATA_DIR)
11     if os.path.isdir(os.path.join(DATA_DIR, name)) and not name.startswith(
12         '.'))
13 label_to_index = {label: i for i, label in enumerate(labels)}
14
15 X = []
16 y = []
17
18 for label in labels:
19     label_dir = os.path.join(DATA_DIR, label)
20     for root, _, files in os.walk(label_dir):
21         for file in files:
```

```
22     if file.lower().endswith('.jpg'):
23         path = os.path.join(root, file)
24         try:
25             img = Image.open(path)
26             feature = extract_combined_features(img)
27             X.append(feature)
28             y.append(label_to_index[label])
29         except Exception as e:
30             print(f"読み込み失敗:{path}:{e}")
31
32 X = np.array(X)
33 y = np.array(y)
34
35 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
36 random_state=42)
37 knn = KNeighborsClassifier(n_neighbors=4)
38 knn.fit(X_train, y_train)
39 y_pred = knn.predict(X_test)
40
41 print(classification_report(y_test, y_pred, target_names=labels))
```

---