機械学習によるモータコアの傷検出に関する研究

長江　洋典※ Hirofumi Nagae
佐々木　渉真※ Shoma Sasaki
吉田　和弘※ Kazuhiro Yoshida
片平　洋一※ Yoichi Katahira

A Study on Defect Detection of Motor Cores using Machine Learning

1. はじめに

機械学習は、1959 年に Arthur Lee Samuel により「コンピュータに明示的なプログラミングなしで学習能力を与えるための学問分野」と定義された(1)。

それから 60 年以上経過した現在では、機械学習が画像認識、音声認識など数多くの分野で活用される時代に入っている。その糸口になったのが、Geoffrey E. Hinton らが 2006 年に発表した深層学習（Deep Learning）に関する論文である(2)。深層学習はニューラルネットワークを多層化（深層化）したもので、それを用いれば手書き数字を 98%以上の高い精度で認識できることが示された。その後、深層学習は様々な分野に応用されるようになった。

もちろん、コンピュータの飛躍的な性能向上がその背景にあることは言うまでもない。特に、NVIDIA 社から 2006 年に発表された GPU（Graphics Processing Unit）を使用した並列演算処理するための基盤である CUDA（Compute Unified Device Architecture）は、大規模な多次元配列の演算を必要とする機械学習の高速化に大きく貢献している。

今では、TensorFlow、PyTorch などの機械学習用ライブラリおよびそれらを活用する開発環境などが、インターネット上に公開されており、個人レベルでも十分機械学習を利用できる環境が整っている。

深層学習は、画像認識や音声認識などで優れた性能を示す。製造メーカでは、その特長を生かし生産ラインで行われる製品または部品の外観検査を自動化するために深層学習を取り入れつつある。当社も、自動化が困難であったモータコア（ステータコアおよびロータコア）の欠陥検出を目的として深層学習を適用するための研究を開始した。

モータコアの欠陥には、線傷や打痕の他に、鋼板のずれや浮き、欠け、錆び、および歪取り焼鈍で発生する積層面の変色がある。本稿では、まず研究の第一段階として、それらの欠陥の内、モータコア表面に発生する欠陥の検出について深層学習の適用を試みたのでその結果を報告する。

2. 機械学習

2.1 機械学習とテンソル

機械学習ではテンソルという用語が用いられる。まず、そのテンソルについて簡単に述べる。

大きさと方向を持つ量がベクトルで、線形性を有する。その量を複数掛け合わせたもので多重線形性を有するものがテンソルである。具体的には、2 つのベクトルの成分と基底ごとの積を取ったものがテンソルとなる。その積はテンソル積と呼ばれる。

また、テンソルが持つ方向の数により階数が付けられ、2 階のテンソルについては行列で表すことができる。スカラー、ベクトルもテンソルに含まれ、スカラーは 0 階のテンソル、ベクトルは 1 階のテンソルとみなされる。

座標変換しても物理法則や大きさが変わらないとう不変性を保つ前提もテンソルの概念に含まれる。この不変性の概念は数学、物理学のみならず工学でも極めて重要な意味を持つ。

微分幾何学の数学的対象として扱われてきたテンソルは、微分幾何学を取り入れて構築された一般相対性理論の定式化に用いられたことで広く知られるようになった。1930 年代には、Gabriel Kron により回転電気機械を統一的に解析する手段として適用された(3)。それ以降、テンソルは物理学だけでなく、工学でも広く用いられるようになった(4)。

機械学習ではその演算法が主体となる。そこで、不変性の概念とは関係なく、単にテンソルの演算則が適用される数値計算上の配列をテンソルと呼んでいる。ただし、機械学習が答えを導き出す構造を理論的に解析するためには、不変性の概念が要請される(5)。

2.2 機械学習の手法

機械学習は、表 1 に示すように回帰、分類、クラスタリングなどの統計学的手法を基礎に置く。そして、それらを解く演算法には、線形回帰、ロジスティック回帰、サポートベクトルマシン、k平均法など数多くある。ニューラルネットワークはその中の一つである。

なお、ニューラルネットワークはほとんどの場合、深層学習の基になった深層ニューラルネットワーク（Deep Neural Network：DNN）を指す。以降、DNN を単にニューラルネットワークと呼ぶ。

また、表 1 でニューラルネットワークは教師あり学習のみに記載してあるが、教師なし学習や強化学習にも応用され、それらの学習法も大きく発展した。

なお、本稿の欠陥検出では、答えが付いた訓練データのセットでコンピュータに学習させる教師あり学習法を、また、その手法として分類を適用した。

※モータカンパニー　研究・開発センター

表1　機械学習の主な演算法

学習法	手　法	演　算　法
教師あり学習	回帰	• 線形回帰 • 決定木（回帰木） • ベイズ線形回帰 • ニューラルネットワーク
	分類	• k近傍法 • ロジスティック回帰 • 決定木（分類木） • サポートベクトルマシン（SVM） • ニューラルネットワーク
教師なし学習	クラスタリング	• k平均法 • DBSCAN • 階層型蔵クラスタ分析（HCA）
	異常検知と変化検知	• 1クラスSVM • アイソレーションフォレスト
	次元削減と可視化	• 主成分分析（PCA） • カーネルPCA • 局所線形埋め込み法（LLE） • t-分布型確率的近傍埋め込み法（t-SNE）
	相関ルール抽出	• アプリオリ • Eclat
強化学習		• 動的計画法 • モンテカルロ法 • 時間差分（TD）学習 　• Q学習 　• SARSA

3. ニューラルネットワーク

3.1 ニューラルネットワークの基本的構造

ニューラルネットワークについて説明する前に、その基本となる線形回帰について簡単に述べる。

線形回帰方程式は、次式で表される。

$$y = b + W_1x_1 + W_2x_2 + \cdots + W_nx_n + \varepsilon \quad \cdots\cdots (1)$$

ここで、y：目的変数、x_1～x_n：説明変数、b：定数項、W_1～W_n：回帰係数、ε：擾乱項（雑音項）である。

$b = W_0\,x_0$、$x_0 = 1$ として定数項を回帰係数に含め、さらに、$\varepsilon = 0$ としてy を予測値とすれば、(1)式は、

$$y = \sum_{v=0}^{n} W_v x_v \quad \cdots\cdots (2)$$

として表される。ただし、$v = 0,1,2,\cdots,n$ である。

テンソルでは、(2)式の表記法を指標記法またはテンソル記法と呼ぶ。本稿では、W_v 、x_v のようにギリシャ文字の添字はテンソルの可変指標を表すものとする。また、指標の数とテンソルの階数は一致する。

(2)式は内積であり、右辺には同じ指標（v）が二度現れる。そのような場合には総和記号Σを省略するアインシュタインの総和規約が適用できる。その規約を用いれば、(2)式は次のように簡単に表すことができる。

$$y = W_v x_v \quad \cdots\cdots (3)$$

(3)式の W_v を求める手法としては、最小二乗法が一般的に用いられる。

線形回帰は、説明変数を入力データ、出力である目的変数を正解値データとして与える教師あり学習である。

(3)式を有向グラフで模式化すると図 1 (a)のようになる。図1において、円はノード、円と円を繋ぐ矢印の付いた線はエッジと呼ばれる。

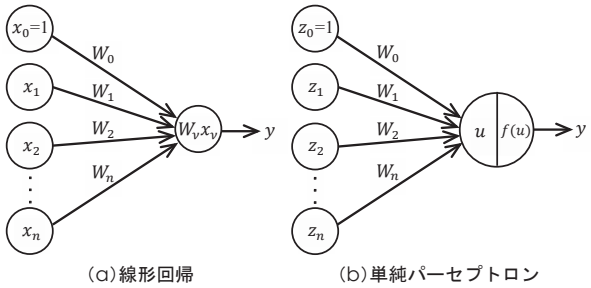


図1　線形回帰および単純パーセプトロンの模式図

線形回帰の場合は、出力が一つの場合にのみの対応となるが、ニューラルネットワークを用いれば複数の出力に対応することが可能である。

そのニューラルネットワークの基礎となるのが図 1(b)に示す単純パーセプトロンである。このモデルはロジスティック回帰と等価である。図 1(b)を式で表すと次のようになる。

$$\left.\begin{aligned} u &= W_v z_v \\ y &= f(u) \end{aligned}\right\} \quad \cdots\cdots (4)$$

(4)式の第 1 式は回帰と同じ線形方程式で、z_v：入力、y：出力である。単純パーセプトロンでは線形方程式で演算した値 u をそのまま出力として返すのではなく、活性化関数と呼ばれる $f(u)$ で変換した値を出力として返す。単純パーセプトロンの活性化関数にはステップ関数が用いられる。

単純パーセプトロンを、図 2 のように複数接続したものがニューラルネットワークである。ニューラルネットワークでは、ノードをユニットあるいは生物の神経細胞を模擬したものとして、人工ニューロンと呼ぶ。なお、本稿ではユニットと呼ぶことにする。

ニューラルネットワークは、入力層と出力層との間に隠れ層とも呼ばれる中間層が設けられる。また、図 2 に示すように、その中間層を 2 層以上設けることで深層化される。

図 2 に示されるユニットの処理を一般化した式で表すと

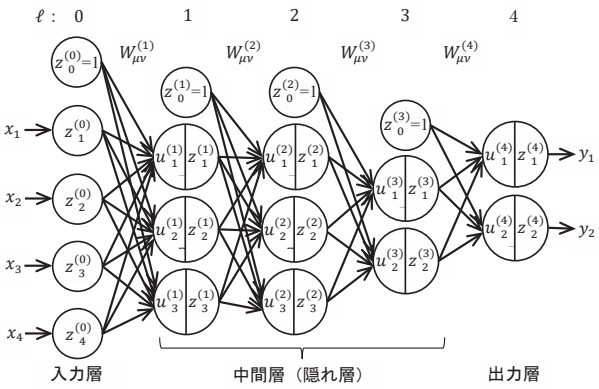


図2　ニューラルネットワークの例

次式のようになる。

$$\left.\begin{aligned} u_\mu^{(\ell)} &= W_{\mu\nu}^{(\ell)} z_\nu^{(\ell-1)} \\ z_\mu^{(\ell)} &= f(u_\mu^{(\ell)}) \end{aligned}\right\} \quad \cdots\cdots (5)$$

ただし、括弧付きの指標 $\ell = 1,2,\cdots N$ は、各層のテンソルを区別するものでテンソルの指標ではない。

ここで、$z_\nu^{(0)} \equiv x_\nu$、$y_\mu \equiv z_\mu^{(N)}$ であり、$\mu = 1,2,\cdots m$、$\nu = 0,1,2,\cdots n$ である。なお、m、n の値は層ごとに異なる。また、$z_0^{(\ell-1)} = 1$ である。

定数項 $b_\mu^{(\ell)}$ は、(5)式からは見えないが、(3)式と同様に(5)式でも $b_\mu^{(\ell)} = W_{\mu 0}^{(\ell)} z_0^{(\ell-1)}$ として係数 $W_{\mu\nu}^{(\ell)}$ の中に包含される。

なお、機械学習では、係数 $W_{\mu\nu}^{(\ell)}$ を重み、定数項 $b_\mu^{(\ell)}$ をバイアスと呼ぶ。そして、それら を求めることを学習と呼ぶ。

また、図 2 のように、入力データに対して出力である結果（予測値）が導き出される構造において、コンピュータで演算できる形式にしたものを学習モデルと呼ぶ。

(5)式の第 1 式は、実質的にはアフィン変換であり、第 2 式は非線形変換である。したがって、ニューラルネットワークはアフィン変換と非線形変換を交互に繰返すことで実現されると言える。それにより、ニューラルネットワークは複雑な形状でも認識することができる。

非線形変換には、シグモイド関数、双曲線正接（tanh）関数など複数種類の活性化関数が用いられる[1]。以下に、本稿で扱う分類問題に係る 2 つの関数について示す。

(1) 正規化線形（Rectified Linear Unit：ReLU）関数

$$f(u_\mu) = \max(0, u_\mu) \quad \cdots\cdots (6)$$

ReLU は、$u_\mu < 0$ の場合には出力値が $f(u_\mu) = 0$ で一定となり、$u_\mu \geq 0$ では u_μ をそのまま返す関数である。

(2) ソフトマックス関数

$$f(u_\mu) = \frac{e^{u_\mu}}{\sum_{\gamma=1}^{m} e^{u_\gamma}} \quad \cdots\cdots (7)$$

ソフトマックス関数は、主に分類問題で用いられる活性化関数であり、出力層の活性化関数として使用される。

この関数の機能は、出力される値の総和を 1 に正規化することである。分類の場合、ニューラルネットワークから出力される変数の総和が 1 となり、それぞれの出力値は $0 \leq f(u_\mu) \leq 1$ の範囲で値を取る。

3.2 重みの最適化

ニューラルネットワークではその演算過程で非線形変換が入るため、線形回帰のように最小二乗法を用いて直接重みを求めるようなことができない。そこで、(5)式で求められる出力値（予測値）と目的値（正解値）との誤差が最小になるように重みを調節しながら最適化していく。その誤差を求める関数を損失関数と呼ぶ。その代表的なものに、平均二乗誤差、二値交差エントロピー、交差エントロピーなどがある[1]。

本稿で扱う分類問題で一般的に用いられる交差エントロピーは、多クラス分類および二値分類を解く際によく用いられる損失関数であり、次式で表される。

$$L = -\frac{1}{M}\sum_{i=1}^{M}\sum_{\mu=1}^{m} y_\mu^{[i]} \log y_\mu^{[i]} \quad \cdots\cdots (8)$$

ここで、L：誤差、M：データ数、$y_\mu^{[i]}$ ：i 番目の入力データに対して、正解値として与えた確率を持つ出力ベクトル、$y_\mu^{[i]}$：ニューラルネットワークにより予測した確率を持つ出力ベクトル、m ：出力ベクトルの成分数である。ただし、L は損失と呼ばれる場合が多い。以降、本稿でも L を損失と呼ぶ。

なお、出力層の活性化関数にはソフトマックス関数を選択することになる。

L が最小になるように重みを修正するが、それには次式に示す確率的勾配降下法（Stochastic Gradient Descent：SGD）が基本的に用いられる。

$$\left(W_{\mu\nu}^{(\ell)}\right)_{t+1} = \left(W_{\mu\nu}^{(\ell)}\right)_t - \eta\left[\frac{\partial L}{\partial\left(W_{\mu\nu}^{(\ell)}\right)_t}\right]^{\mathrm{t}} \quad \cdots\cdots (9)$$

ここで、t：反復回数、η：学習率である。また、t は転置を表す。損失 L の勾配は、

$$\frac{\partial L}{\partial W_{\mu\nu}^{(\ell)}} = \frac{\partial L}{\partial u_\mu^{(\ell)}}\frac{\partial u_\mu^{(\ell)}}{\partial W_{\mu\nu}^{(\ell)}} \quad \cdots\cdots (10)$$

から求められる。したがって、(10)式右辺の $\partial L/\partial u_\mu^{(\ell)}$ は次層の各ユニットの入力 $u_\mu^{(\ell+1)}$ を経由した微分の連鎖により、

$$\frac{\partial L}{\partial u_\mu^{(\ell)}} = \frac{\partial L}{\partial u_{\mu'}^{(\ell+1)}}\frac{\partial u_{\mu'}^{(\ell+1)}}{\partial u_\mu^{(\ell)}} \quad \cdots\cdots (11)$$

として分解できる。ただし、$\ell+1$ 層の各ユニット入力の指標は、ℓ 層のそれと区別するため μ' とした。

(5)式より、$\partial u_{\mu'}^{(\ell+1)}/\partial u_\mu^{(\ell)} = W_{\mu'\mu}^{(\ell+1)}\left\{\partial f(u_\mu^{(\ell)})/\partial u_\mu^{(\ell)}\right\}$ となる。また、$\delta_\mu^{(\ell)} \equiv \partial L/\partial u_\mu^{(\ell)}$ とすれば、(11)式は、次式のように示される。

$$\delta_\mu^{(\ell)} = \delta_{\mu'}^{(\ell+1)} W_{\mu'\mu}^{(\ell+1)}\left\{\frac{\partial f(u_\mu^{(\ell)})}{\partial u_\mu^{(\ell)}}\right\} \quad \cdots\cdots (12)$$

したがって、(10)式は、その右辺第 2 項が (5)式の第 1 式より、$\partial u_\mu^{(\ell)}/\partial W_{\mu\nu}^{(\ell)} = z_\nu^{(\ell-1)}$ として求められるので、

$$\frac{\partial L}{\partial W_{\mu\nu}^{(\ell)}} = \delta_\mu^{(\ell)} z_\nu^{(\ell-1)} \quad \cdots\cdots (13)$$

として表される。

(12)式は、活性化関数の微分が求められ、$\ell+1$ 層の δ を与えられれば ℓ 層の δ が求められることを意味する。つまり、最初に出力層各ユニットの δ が求められれば、入力層に向かって、順次各層各ユニットの δ を求めることができる[6]。

そこで、(5)式によって予測値が求められて行く過程が順伝播と呼ばれるのに対し、(9)式によって重みが更新されて行く過程は誤差逆伝播と呼ばれる。

3.3 ミニバッチ学習

ニューラルネットワークの学習、つまり、最適な重みを求めるためには大規模なテンソルの演算が要求される。そこで、並列演算処理ができるようミニバッチと呼ばれる一定数のデータの集合単位で、SGD を適用し重みを更新する。次式にミニバッチ学習に適用した SGD を示す。

$$\left(W_{\mu\nu}^{(\ell)}\right)_{t+1} = \left(W_{\mu\nu}^{(\ell)}\right)_{t} - \eta \frac{1}{N_t} \sum_{d\in\mathcal{D}_t} \left[\frac{\partial L_d}{\partial \left(W_{\mu\nu}^{(\ell)}\right)_t}\right]^{\mathrm{t}} \quad \cdots (14)$$

ここで、ミニバッチの 1 つを $\mathcal{D}_t$ $(t = 1,2,\cdots,n)$ とした。L_d はミニバッチごとの損失で、N_t は $\mathcal{D}_t$ が含むデータ数である。

ミニバッチを $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n$ と順番に取り出して使い、一巡したら、再度、同様にミニバッチを順番に取り出して使う。ミニバッチが一巡することをエポックと呼ぶ(6)。このようにして重みの最適化が行われる。

ここで、重みはパラメータと呼ばれることがある。また、エポック数、学習率、ミニバッチサイズなど事前に設定しておくパラメータはハイパーパラメータと呼ばれる。

なお、SGD では確率的な性質を持つため、エポックごとに重みの更新量がばらつく。これを安定化して、収束性を改善するモーメンタムという方法があり、SGD では一般的に用いられている(1)。

また、SGD による演算法は最適化アルゴリズムと呼ばれる。最適化アルゴリズムには、その他に、SGD を改良した NAG、RMSProp、Adam などがある(1)。特に、Adam は現在最も広く用いられているアルゴリズムの一つである。

3.4 交差検証

学習モデルに汎用性を持たせるためによく用いられる交差検証と呼ばれる手法がある。

機械学習に用いるデータの集合をデータセットと呼ぶ。交差検証は、そのデータセットを訓練セットと検証セットに分けて、訓練と検証を交互に行う方法である。

学習は訓練セットを用いて行うので、学習で得られる重みは訓練用セットを予測するのに適したものとなる．しかし、それでは訓練用セットに入っていないデータに対して信頼性（汎用性）が得られない。そこで、その信頼性を確認するために訓練セットとは別の検証セットを用いて検証を行う。

ただし、学習の過程では検証セットに対しても精度が上がるように学習モデルの構造やハイパーパラメータを見直すことになる。つまり、検証セットを用いて検証しても十分とは言えない。そのため、学習後にさらに訓練セット、検証セットとは別の試験セットを準備して最終試験を行い、信頼性を評価する必要がある。

4. 畳み込みニューラルネットワーク

ニューラルネットワークには、図 2 のような層間のユニットが互いに密に結合した全結合型だけでなく、画像処理などでよく用いられる畳み込み型、音声認識など時系列データの解析でよく用いられる再帰型など、いくつかの種類がある。

ここでは、本稿の目的を実現するための畳み込みニューラルネットワーク（Convolutional Neural Network：CNN）について述べる。

4.1 CNNの基本的構造

図 3 に基本的な構造をした CNN の例を示す。CNN は脳の視覚野の研究から生まれたもので、その核となるのが入力画像にフィルタを掛け合せることで特徴を抽出する畳み込み層とダウンサンプリングを行うプーリング層である。また、全結合層は、前述したニューラルネットワークの中間層と同じものである。

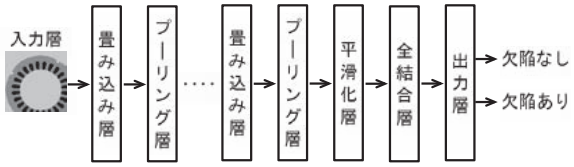


図3　CNNの例

(1) 畳み込み層

図 4 に畳み込み層の概念図を示す。画像は画素（pixel：単位は略記で px とする）と呼ばれる小領域で構成され、幅:w×高さ:h[px]の情報を持ったチャネルと呼ばれる複数の層が組み合わされたものから表現される。例えば、RGB（赤、緑、青）の 3 原色からなるカラー画像であれば、チャネル数 c は 3 となり、それらの情報は 1px ごとに 8bit 符号なし整数（0～255）の数値として 3 次元の配列に格納される。

その画像上にフィルタと呼ばれる小領域を畳み込んで、一つの特徴量を抽出する。それを画像の座標位置をずらしながら繰返す。それにより得られた出力は、特徴量マップと呼ばれる。複数のフィルタを用いることで様々な特徴を

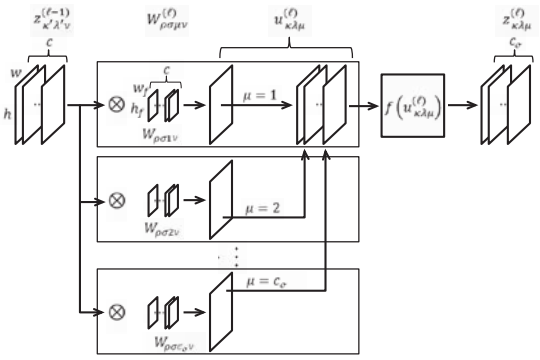


図4　畳み込み層の概念図(6)

抽出した特徴量マップの層が得られる。その層に活性化関数を適用することで、強い特徴量だけを抽出する。フィルタは畳み込みカーネル、または単にカーネルと呼ばれる。なお、カーネルのサイズと数は任意に決める必要がある。

図4の畳み込み層では、c 個の $w \times h$ の入力 $z_{\kappa'\lambda'\nu}^{(\ell-1)}$ に対し、c_o 個の $w_f \times h_f$ の重み（フィルタ係数）$W_{\rho\sigma\mu\nu}^{(\ell)}$ を有するカーネル を適用し、c_o 個の $w \times h$ の出力 $u_{\kappa\lambda\mu}^{(\ell)}$ を得る。

それにより得られた出力は、前述したニューラルネットワークと同様に活性化関数が適用され、次の層に伝播される。畳み込み層でのユニットの処理を式で表すと次のようになる。

$$\left.\begin{aligned} u_{\kappa\lambda\mu}^{(\ell)} &= W_{\rho\sigma\mu\nu}^{(\ell)} z_{\kappa'\lambda'\nu}^{(\ell-1)} + b_{\kappa\lambda\mu}^{(\ell)} \\ z_{\kappa\lambda\mu}^{(\ell)} &= f(u_{\kappa\lambda\,\mu}^{(\ell)}) \end{aligned}\right\} \quad \cdots\cdots (15)$$

$$\text{with} \quad \kappa' = \kappa + \rho,\ \lambda' = \lambda + \sigma$$

ただし、$\kappa = 1,2,\cdots w$、$\lambda = 1,2,\cdots h$、$\mu = 1,2,\cdots c_o$、$\nu = 1,2,\cdots c$、$\rho = 1,2,\cdots w_f$、$\sigma = 1,2,\cdots h_f$ である。バイアス項 $b_{k\lambda\mu}^{(\ell)}$ は、特徴量マップ全体の明るさを操作するためのノブであると考えることができる[1]。

畳み込みにより画素数が減少するが、元の画素数を維持するため、畳み込みの前に画像の周囲を 0 で埋めるパディング処理が一般的に行われる。そうすることで、画像の端の特徴も得られるようになる。また、活性化関数には ReLU が一般的に用いられる。

(2) プーリング層

プーリングとは、入力画像の特徴量の位置が変わらないように空間解像度を下げる処理のことを言う。

プーリング層では重みを用いた演算はなく、max（最大値）や mean（平均値）といった集計関数を用いて入力を集計するのみである[1]。

最後のプーリング層と全結合層の間の平滑化層では、3次元配列を 1 次元配列に並び替える平滑化、または、特徴量マップ全体の平均を計算する大域平均プーリング（Global Average Pooling : GAP）が行われる。平滑化（GAP）層を P とした場合、GAP は次式で表される。

$$u_{\mu}^{(P)} = \frac{1}{wh}\sum_{\kappa=1}^{w}\sum_{\lambda=1}^{h} z_{\kappa\lambda\mu}^{(P-1)} \quad \cdots\cdots (16)$$

GAP で平滑化する場合は全結合層を通さずそのまま出力層に接続されることもある。

CNN の重みも 3.2 節で述べた誤差逆伝播法により最適化が行われる。ただし、プーリング層などが含まれるためその計算はやや複雑となる。

出力層（$\ell = N$）では、(5)式で示した演算が行われる。それにより得られる出力ベクトル $y_\mu \equiv z_\mu^{(N)}$ の成分は、確率を表すものと解釈する。そこで、その成分はスコアと呼ばれることがある。そして、そのスコアで最も高い確率を示したものが予測値となる。

なお、CNN では重みの数が数百万にも達する場合がある。それだけ柔軟性が高いと言うことであるが、その代償として、過学習と呼ばれる現象を避けることができない。過学習とは図 5 に示すように、訓練セットに対しては損失が減少し続け学習が進んで行くが、未知の検証セットに対しては損失が減少から増加に転じ適合できなくなる状態を言う。

そこで、その過学習を抑制する方法として、ミニバッチ内のデータ分布をもとにチャネルごとに特徴を正規化するバッチ正規化や、ランダムにユニットの一部を欠落させるドロップアウトなどの処理層が追加される。

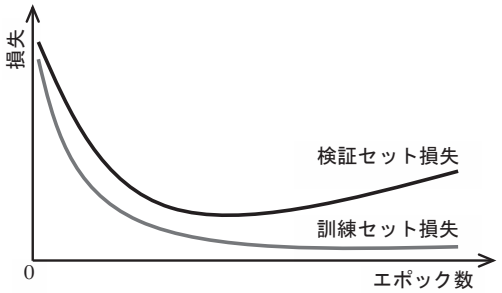


図5　過学習の例

4.2 転移学習とファインチューニング

ニューラルネットワークを用いて学習を行う場合、そのモデルは任意に構築するか、あるいは一般に公開されている学習モデルを利用する。ただし、学習には、モデルにもよるが数万～数十万のデータからなるデータセットの準備が必要となり、かなりの時間と労力を要する。

そこで、小規模なデータセットでも精度の高い学習済みモデルを得られれば労力の削減が可能となる。そのための有効な手段として、転移学習と呼ばれる手法がある。転移学習とは、ある領域で学習したモデル（事前学習済みモデル）を別の領域の学習に適用させる技術である。画像認識の分野では、大規模なデータセットで事前に学習した重みを用いることで、小規模なデータセットでも十分な性能を得ることが可能であると知られている[7][8][9]。

転移学習では事前学習済みモデルの末端に学習可能な重みを持つ層を新たに追加する。学習する際は事前学習済みモデルのすべての重みに対して更新は行わず、新たに追加した末端の層の重みだけを更新する。

それに対し、事前学習済みモデルの重みも含めて更新するファインチューニングと呼ばれる手法がある。一般的には、転移学習で新たに追加した層の重みをある程度最適化した後にファインチューニングを行ってさらに精度を高める方法が採られる[1]。

CNN の関係では、VGG16、ResNet50、MobileNet、DenseNet など様々なモデルが一般に公開されている。それらのモデルでは、ImageNet[10]の 1000 種類のカテゴリを約 100 万枚の画像を使って学習した重みも一緒に公開されている。その様々なモデルを転移学習の事前学習モデルとして利用することができる。

なお、ImageNet とは2万種以上のカテゴリに分類された1400万枚を超える画像を含んだ研究用標準データセットである。

4.3 特徴量の可視化

ニューラルネットワークではその計算の中身を説明することが難しく、よくブラックボックスに例えられる。しかし、いかにしてその結果が導かれたかを判断する根拠が必要となる。

CNN ではその判断根拠を与える一つの方法として、クラス活性化マッピング (Class Activation Mapping：CAM)[11]と呼ばれる手法がある。CAM は、入力画像のどこの領域の特徴量が判定に強く結びついているかを可視化する。

CNN の構造が、

〔最終畳み込み層〕→〔GAP 層〕

→〔出力層（ソフトマックス関数)〕

であれば、出力層の重みを畳み込み、特徴マップに投影し直すことで画像領域の重要度を特定することができる。

具体的には、出力層を N 、最終畳み込み層出力画像の幅および高さの画素数をそれぞれ w、h とすれば、(16)式よりその出力は、次式として表される。 ただし、出力層のバイアス項は分類性能にほとんど影響を与えないので無視する。

$$u_{\mu}^{(N)} = W_{\mu\nu}^{(N)} \frac{1}{wh} \sum_{\kappa=1}^{w} \sum_{\lambda=1}^{h} z_{\kappa\lambda\nu}^{(N-2)}$$

$$= \frac{1}{wh} \sum_{\kappa=1}^{w} \sum_{\lambda=1}^{h} W_{\mu\nu}^{(N)} z_{\kappa\lambda\nu}^{(N-2)} \quad \cdots\cdots (17)$$

上式の κ と λ で総和をとる前の

$$M_{\kappa\lambda\mu} = W_{\mu\nu}^{(N)} z_{\kappa\lambda\nu}^{(N-2)} \quad \cdots\cdots (18)$$

がクラス活性化マップ $M_{\kappa\lambda\mu}$ である。

(18)式を計算し、ヒートマップ化した画像を入力画像に重ね合わせて表示すれば、特徴として注目している部位がわかる。

CAM は、出力層の前段に GAP 層があることを前提としたマッピング法であるが、CNN では必ずしもそのような構造にするわけではない。その場合には、誤差逆伝播法の勾配を重みとして用いる Grad-CAM (Gradient-weighted CAM)[12]がよく用いられる。

ただし、CAM および Grad-CAM は、最後の畳み込み層からの出力の空間分解能が低いため、粗い物体領域しか特定できないという欠点を有している。その欠点を改善する手法として、SmoothGrad[13]、LayerCAM[14]などが開発されている。

本稿では、CAM の他に SmoothGrad による結果についても示した。そこで、SmoothGrad について簡単に述べておく。

入力画像の各画素の微小変化で出力に大きく影響するものを重要な画素として強調する感度マップと呼ばれる手法がある。

入力画像データを $\mathbf{x}$ 、出力層で演算されるスコアを $\mathbf{y} \equiv S(\mathbf{x})$ として表すものとすると、感度マップ $\mathbf{M}_s$ は次式で表される。なお、アルファベットの太字はベクトルと同様にテンソルを表す。

$$\mathbf{M}_s = \frac{\partial S(\mathbf{x})}{\partial \mathbf{x}} \quad \cdots\cdots (19)$$

感度マップでは、認識対象でない部分（背景ノイズ）も強調されてしまう欠点を有する。それを改善したものが SmoothGrad である。

SmoothGrad は、入力画像に人間の目では判別できない微小なノイズを加えたものを幾つか用意し、それらで感度マップを求めてその平均 をとる。それにより背景ノイズを低減する手法である。そのマップ $\mathbf{M}_{sg}$ を式で表せば次のようになる。

$$\left.\begin{aligned} \mathbf{x}_i' &= \mathbf{x} + \mathbf{d}_i \\ \mathbf{M}_{sg} &= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial S(\mathbf{x}_i')}{\partial \mathbf{x}_i'} \end{aligned}\right\} \quad \cdots\cdots (20)$$

ただし、n はサンプル数、$\mathbf{d}_i$ は平均が 0 で分散 σ^2 に従う正規分布を有するガウシアンノイズ $\mathcal{N}(0, \sigma^2)$ からのランダムなサンプルである。

5. 機械学習の環境構築

5.1 GPU演算処理環境

ニューラルネットワークでは大規模なテンソルの演算が必要となり、データセットの画像データが増えれば増えるほど、演算に時間がかかる。しかも、所望の学習結果を得るためには、学習モデルおよびハイパーパラメータを調整しながら何度も学習を繰返す必要がある。そこで、GPU 演算処理環境を構築することにした。

GPU は3次元グラフィックスを扱うゲームや3次元 CAD などの描画を高速処理するための装置であり、並列処理技術が利用されている。この GPU が持つ並列処理能力を画像処理以外の汎用的な計算にも行えるようにした技術は、GPGPU (General-Purpose computing on Graphics Processing Units)[15]もしくは GPU コンピューティングと呼ばれる。

表 2 に今回使用した機械学習用コンピュータの基本的環境を示す。同表に示すように、GPU には NVIDIA 社の GeForce GTX1070 を使用した。

NVIDIA 社では、GPGPU 基盤である CUDA の他に、GPGPU 環境でニューラルネットワークの演算を行うためのライブラリである cuDNN（CUDA Deep Neural Network）も提供している。GPU で機械学習を行うための環境を整えるためにはそれら 2 つのライブラリが必要となる。なお、cuDNN を使用するためにはデータ圧縮用ライブラリ Zlib が必要となる。

表2　機械学習用コンピュータの基本的環境

項　目	品　　名	メーカ
CPU	Core i7-6700(3.4GHz)	Intel
GPU	GeForce GTX1070	NVIDIA
OS	Windows 10	Microsoft
ライブラリ	CUDA(Ver. 11.5)	NVIDIA
	cuDNN(Ver. 8.3.2)	NVIDIA

GeForce GTX1070 の RAM サイズは 8Gbyte、並列演算に係わる CUDA プロセッサーコア数は 1920 基で、GPU がサポートする機能を示す Compute Capability は 6.1 である。同価格帯の最新 GPU と比較すればスペックは低いが、それでも最新の CPU で機械学習を行うよりも十分速い。

なお、GPGPU では、CPU の RAM は共用できないので使用できる RAM は GPU の 8Gbyte のみとなる。この RAM サイズは CNN で扱うデータセットのサイズ考えれば決して大きくはない。ただし、小規模なデータセットで前述の転移学習を用いる、あるいは、データセットのサイズが RAM サイズを超える場合は、ミニバッチごとに画像データを取り込む方法を取るなどすれば十分対応可能である。

5.2 開発環境

開発環境の構築には、プログラミング言語に Python、機械学習用のライブラリに TensorFlow、およびプログラムの実行環境に JupyterLab を用いることにした。

TensorFlow は、Google 社が 2015 年に公開した機械学習用のオープンソースライブラリで、Python を標準プログラミング言語として使用する。TensorFlow には、Python で記述されたオープンソースのニューラルネットワークライブラリである Keras が付属している。

Keras の API（Application Programming Interface）を用いることで少ないコード数で比較的簡単にニューラルネットワークのプログラムを組むことができる。また、Keras には MobileNet など複数の事前学習済みモデルも付属しているので転移学習も容易に行うことができる。

JupyterLab は、Web ブラウザ上で動作する対話型プログラム実行環境で、入力したプログラムを即実行して、実行結果をすぐに確認できることを最大の特徴とする。

開発環境構築に用いたライブラリなど主なソフトウェアの一覧を、使用したバージョンとともに表 3 に示す。

表3　開発環境構築に用いた主なソフトウェアの一覧

ソフトウェア	バージョン	説　　明
Python	3.10.2	インタープリタ型汎用プログラミング言語
NumPy	1.22.2	Python 用数値計算用拡張モジュール
Pillow	9.0.1	Python 用画像処理ライブラリ
Matplotlib	3.5.1	Python 用グラフ描画ライブラリ
TensorFlow	2.8.0	機械学習用ライブラリ
Keras	2.8.0	ニューラルネットワークライブラリ
JupyterLab	3.3.0	対話型プログラム実行環境

6. データセットと学習モデル

6.1 データセット

図 6 にデータセットとして用いたモータコアの欠陥画像の一例を示す。ステータコア、ロータコアの欠陥品サンプルは生産ラインより入手した。

なお、本稿に示した画像には、欠陥を認識しやすくするため鮮鋭化フィルタがかけてある。また、欠陥およびその周辺以外は都合上ぼかしてある。

撮影には、青色系の照明を用いた。また、図 6 に示すようにモータコアの 1/2～1/4 が収まる任意範囲で、位置、方向、照度を任意に変えて白黒撮影した。

データセットの画像として、欠陥ありでは、表面に欠陥がある複数のステータコアおよびロータコアよりそれぞれ6種類の欠陥に対して計300枚を撮影した。また、欠陥なしでは、4個ずつのステータコアおよびロータコアの両面から計800枚を撮影した。そして、撮影した画像については幅×高さの画素数を640×640[px]にして学習用コンピュータに保存した。

学習では、モータコアの形状に関係なく欠陥部以外は背景ノイズであると考える。したがって、データセットではステータコアとロータコアを区別せず、欠陥あり画像の集合と欠陥なし画像の集合とに分けた。また、それらの画像の内、76%を訓練用とし、残りの 24%を検証用として用いることにした。

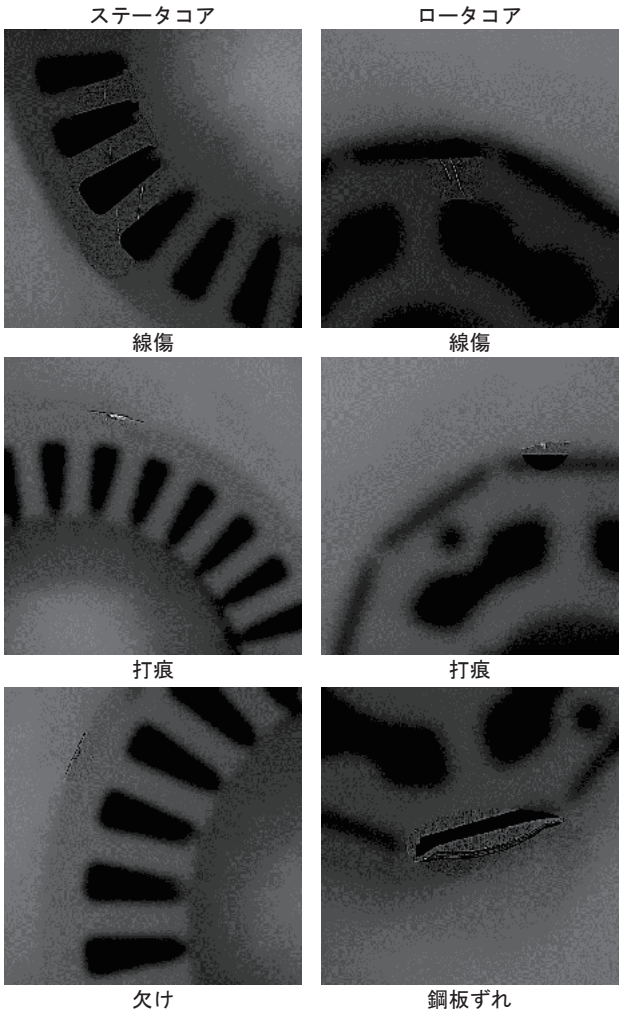


図6　モータコアの欠陥画像一例

6.2 学習モデル

学習には転移学習およびファインチューニングを適用することにし、その事前学習モデルには MobileNet[16]を用いた。MobileNet は、2017 年に Google 社から発表された CNN モデルで、モバイル端末や組み込み用途でも使用できるよう計算量やメモリ使用量が抑えられるよう設計されている。

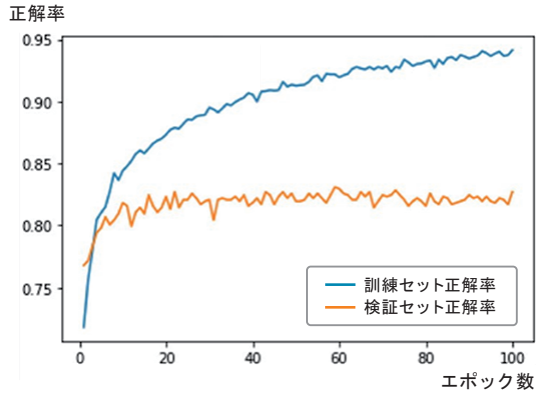
その構造は、深さ方向分離可能畳み込み（Depthwise Separable Convolution）処理と通常の畳み込み処理からなる層を 13 ブロック積み上げた形をとる。

深さ方向分離可能畳み込みとは、一般的な 2 次元畳み込み層のカーネル群による 3 次元畳み込み処理を，その処理と等価な、2 次元の深さ方向畳み込みと 1 次元のチャネル方向畳み込みの二つに分解し、重みを削減するとともに計算を簡素化するものである。

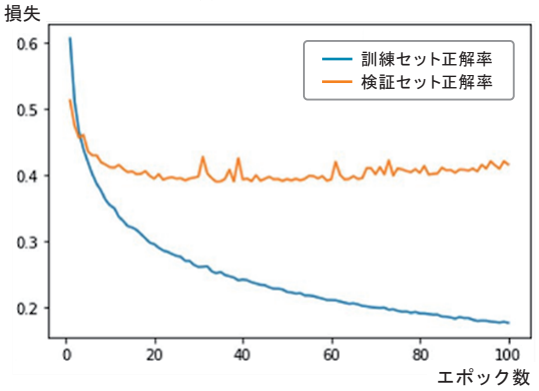
MobileNet の実装では、それに付属する出力層および全結合層は含めず、次のように GAP 層と活性化関数にソフトマックス関数を用いた出力層を追加した。

〔MobileNet 最終畳み込み層〕→〔GAP 層〕
→〔出力層（ソフトマックス関数）〕

なお、MobileNet を使用するにあたっては、事前に金属表面の傷のデータセットである DAGM2007[17]の 6 クラスすべてについて学習を行い、十分な精度が得られることを確認した。



(a) 正解率曲線



(b) 損失曲線

図7　入力画像画素数 224 × 224 [px] における学習曲線

7. 学習結果と視覚的検証

7.1 学習結果

(1) 入力画像画素数と正解率の関係

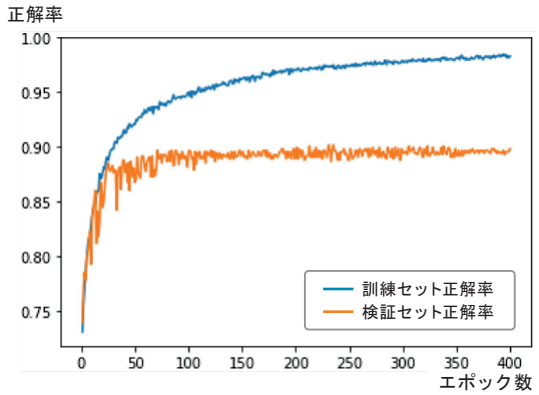
MobileNet 事前学習済みモデルの入力画像デフォルトの画素数は 224×224[px]である。ただし、要求する入力画像画素数について、Keras では、32×32[px]以上とされているのみである。そこで、デフォルトサイズ以上の入力画像を用いた場合の影響について、転移学習を行い調べた。

なお、データセットの入力画像は 180 度回転およびその画像を左右反転することで画像数を 3 倍に拡張した。

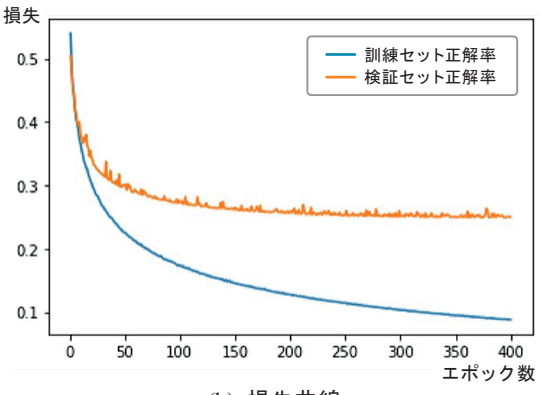
学習は、最適化アルゴリズムに Adam を用い、学習率を 2.0×10^{-4}〜3.0×10^{-4} の間で任意に設定した。また、エポック数は 100〜400 の間とし、画素数が大きいほどエポック数を増すようにした。

図7に入力画像画素数をデータセットとして保存した640×640[px]から224×224[px]にリサイズして学習した場合の学習曲線を、図8に640×640[px]で学習した場合の学習曲線を示す。

図7の224×224[px]の場合では、検証セットでの正解率（=正解数/データ数）および損失とも60エポック程度で飽和し、それ以降は過学習の傾向が出始める。一方、図8の640×640[px]の場合では、100エポック以降から正解率、損失とも飽和し始める。なお、入力画像の画素数が増えるほど、飽和に達するエポック数が大きくなり、過学習も抑制された。



(a) 正解率曲線



(b) 損失曲線

図8　入力画像画素数 640 × 640 [px] における学習曲線

図9に入力画像画素数を MobileNet デフォルトの224×224[px]から640×640[px]まで変えた場合の検証セットでの正解率と損失の関係を示す。同図は、各画素数での学習で、検証セットの損失が飽和したところ（過学習が発生した場合は損失が上昇し始める前のところ）での平均的な正解率と損失をプロットしたものである。

同図より、画素数を増やせば正解率が上昇し、損失が減少していくことがわかる。つまり、事前学習済みモデルの重みは画素数に関係なく対応できる。むしろ、画素数が増すことで解像度が上がり、その分正解率が高くなる。

ただし、正解率については、520×520[px]から飽和傾向になる。なお、その画素数での検証セットの正解率は 91.9%、損失は 0.270 であった。

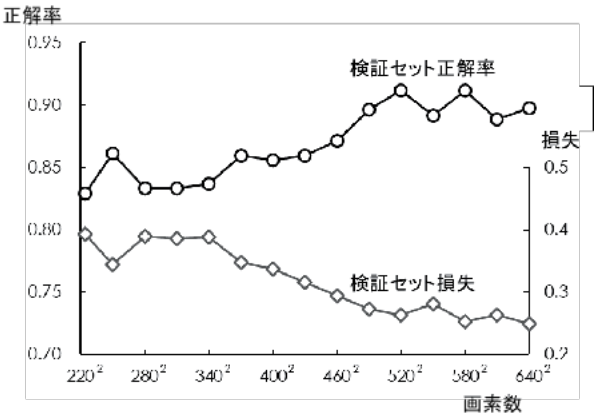


図9　入力画像画素数と正解率および損失の関係

(2) 再学習ブロック数と正解率の関係

前述したように MobileNet は 13 ブロックから構成される。そこで、出力側の第 13 ブロックから入力側の第 1 ブロックにかけて再学習するブロック数を 1 ブロックずつ増加させて、ファインチューニングを行った場合の正解率と損失の関係について調べた。その結果を図 10 に示す。なお、入力画像は 520×520[px]とした。また、学習率は転移学習と同様に 2.0×10^{-4}〜3.0×10^{-4} の間で任意に設定した。また、エポック数は 60 とした。すべての画素数に対して、エポック数が 30 程度以内で検証セットの正解率、損失とも飽和した。

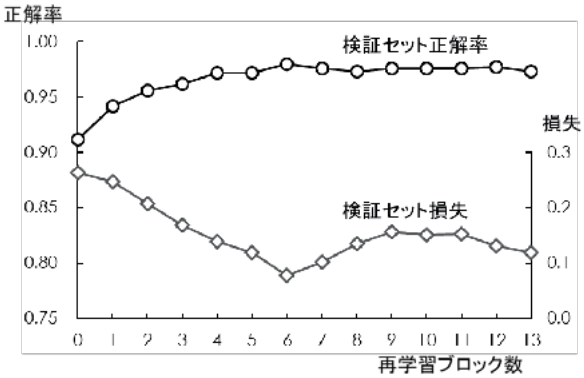


図10　再学習ブロック数と正解率および損失の関係

図 10 より、正解率は再学習ブロック数が 3 以上で飽和状態になり、損失はブロック数が 6 で最も小さくなる。このことより、最適な再学習ブロック数は 6 であることがわかる。このときの正解率は 97.9%で、損失は 0.079 であった。

7.2 特徴量の可視化による視覚的検証

上記の最適な学習済みモデルが、実際にモータコアの欠陥を認識できているか検証するため CAM および SmoothGrad による可視化を行った。なお、SmoothGrad の設定は、$\sigma = 0.1$、サンプリング数を 50 とした。また、可視化には、次の(1)〜(4)項に示す欠陥種別に対して、検証セットの画像を主に用いた。

以降、欠陥種類別ごとに可視化結果を見て行くことにする。

(1) 線傷

図 11 に線傷が付いたモータコアのヒートマップを示す。同図より、CAM では、欠陥部の粗い領域しか特定できていないが SmoothGrad では欠陥部のみが特定できていることがわかる。ただし、SmoothGrad では背景ノイズが若干強

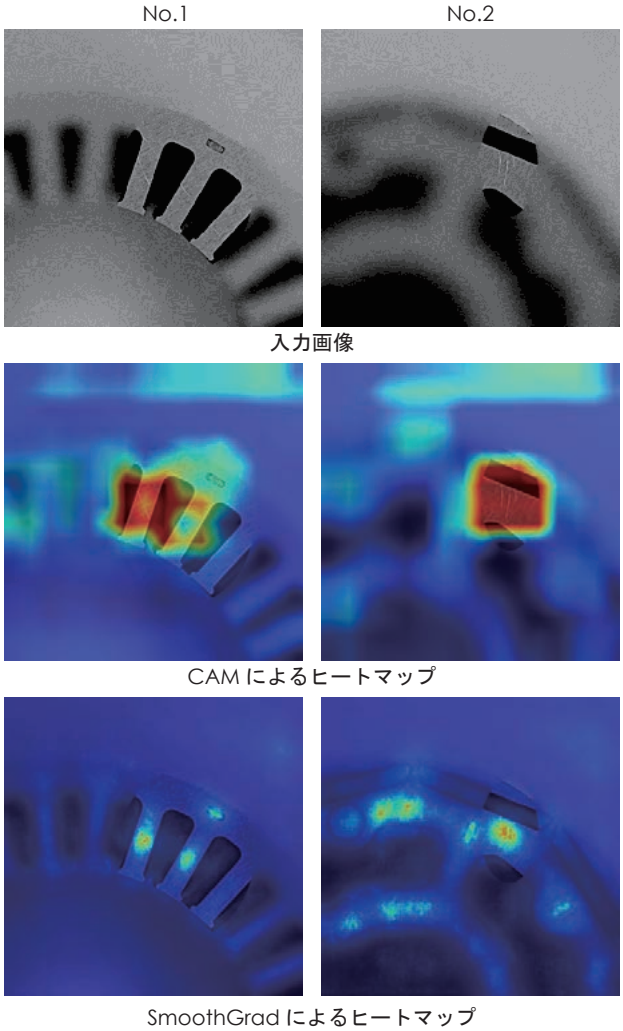


図11　線傷が付いたモータコアのヒートマップ1

調されているようである。サンプリング数を増してもその傾向は変わらなかった。そこで、以降もCAMとSmoothGradの両方を用いて視覚的検証を行う。なお、紙面の都合上、欠陥部およびその周辺のみを示すことにする。

図12に、図11の他の線傷が付いたモータコアのヒートマップを示す。図11、12 より、No.1〜6のどの線傷も特徴として抽出できていることがわかる。

線傷については、少量ではあるが試験用サンプルを確保することができたので、学習済みモデルを用いて予測を行うとともに、特徴量を可視化した。その結果を図13に示す。

予測では、同図に示したNo.7〜10のすべての画像について欠陥ありとの結果が出された。ヒートマップでも線傷が特徴として抽出されている。このことより、予測では線傷を検出して結果が出されたものと見て間違いないと考えられる。

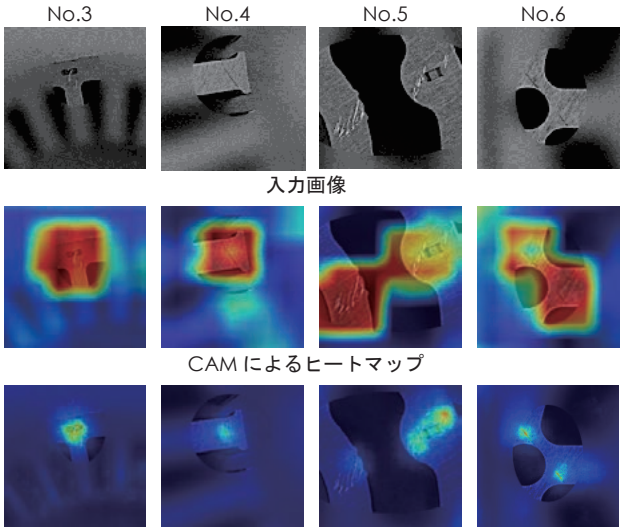


図12　線傷が付いたモータコアのヒートマップ2

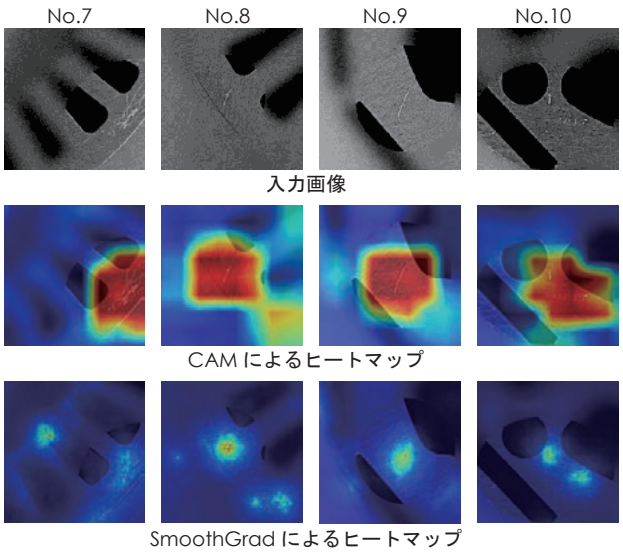


図13　線傷が付いたモータコアのヒートマップ3

(2) 打痕

図14に打痕が付いたモータコアのヒートマップを示す。同図より、No.11〜14 どの打痕も線傷と同様に特徴として抽出されていることがわかる。打痕については、比較的小さなものが多いが、それらについても検出できている。

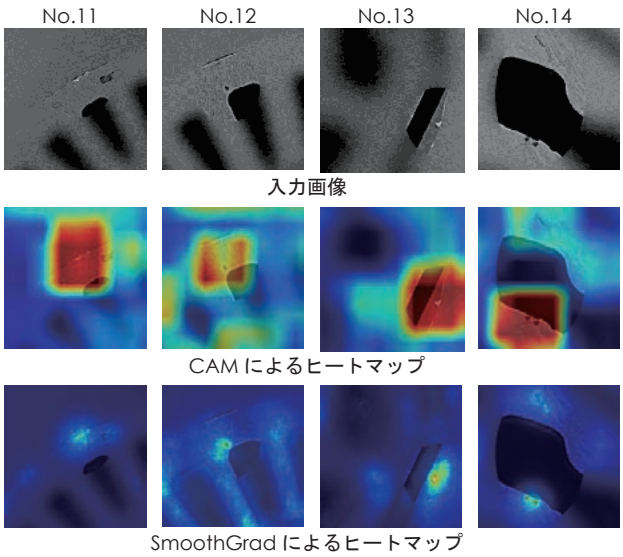


図14　打痕が付いたモータコアのヒートマップ

(3) 鋼板ずれ

鋼板ずれがあるモータコア画像では、次の(4)項に示す欠けがある画像とともに欠陥を特徴として抽出しているか判別できなかった。

そこで、鋼板ずれ、欠けの欠陥がある別の2個ずつのモータコアサンプルを撮影した画像を計50枚追加したデータセットを用意し、それによる学習も行った。その学習済みモデルによる特徴量の可視化を同じ入力画像で行い、その変化を見ることにした。

追加で行った学習は、画像を追加したデータセットを前述したのと同様に3倍にデータ拡張して行った。その学習済みモデルから得られたヒートマップも含めたものを図15に示す。

サンプル画像を追加していない学習済みモデルによる結果では、CAMによるヒートマップを見るとNo.15〜18のどの画像も鋼板ずれ部ではなく、その周辺を特徴として抽出しているようである。SmoothGradによるヒートマップを見るとNo.15〜17の画像については、鋼板ずれ部が薄っすらと明るくなっており、特徴として抽出しているようであるが、No.18 の画像では鋼板ずれ部を全く抽出していない。この学習済みモデルでは、図15の入力画像に対しては何れも欠陥ありと予測されたが、鋼板ずれを認識して欠陥ありと予測しているとは言い難い。

次に、サンプル画像を追加した学習済みモデルによる結果では、CAM、SmoothGrad によるヒートマップともに鋼板ずれ部を抽出していることがよくわかる。このことから鋼板ずれのあるサンプルが不足していたものと考えられる。

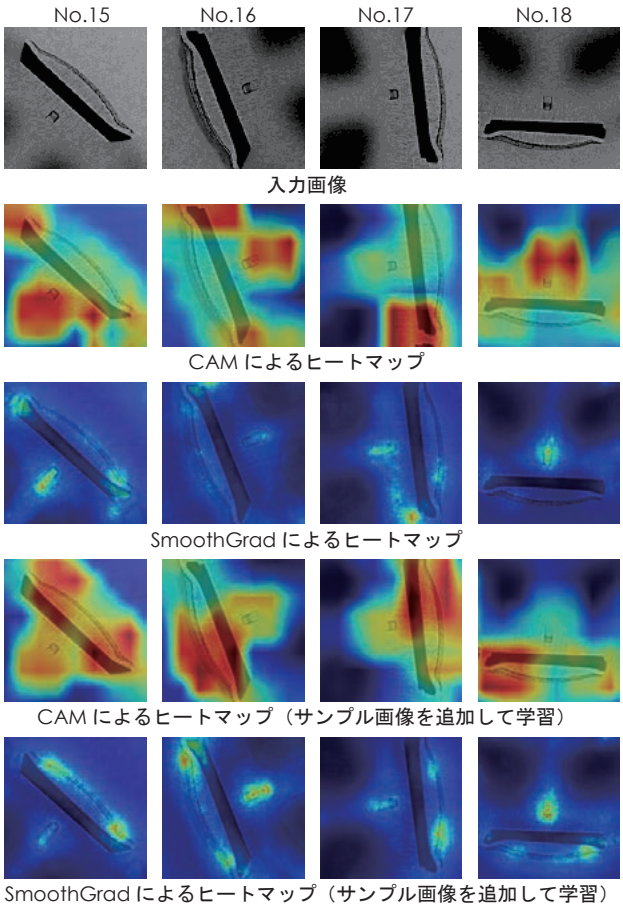


図15　鋼板ずれがあるモータコアのヒートマップ

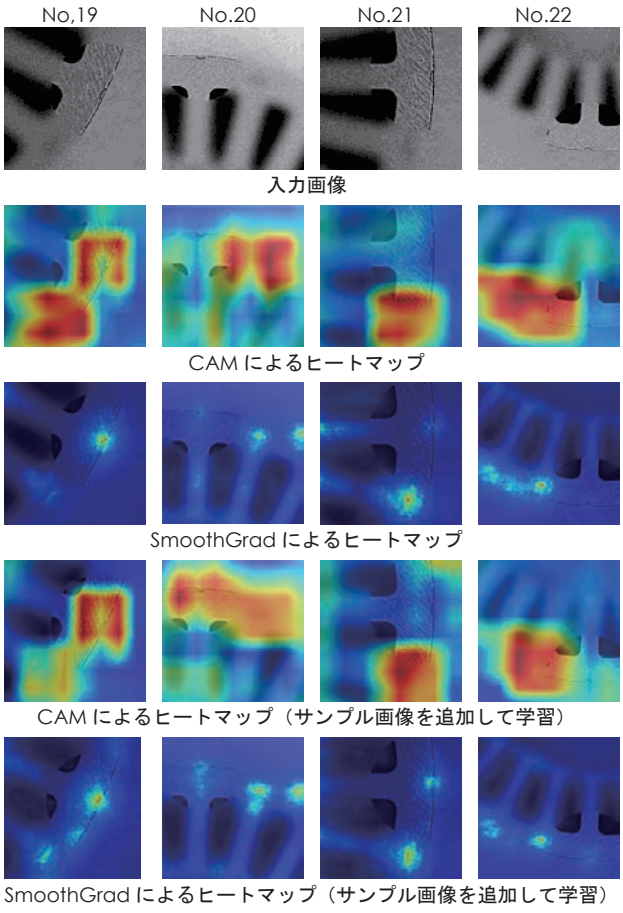


図16　欠けがあるモータコアのヒートマップ

(4) 欠け

図 16 に、欠けのあるモータコアのヒートマップを、サンプル画像を追加して学習した場合の結果も含めて示す。

サンプルの画像を追加していない学習済みモデルによる結果では、No.19 の画像では欠けを特徴として抽出しているが、他の No.20～22 の画像では CAM および SmoothGrad によるヒートマップから判断して欠けを特徴として抽出していないように見える。鋼板ずれと同様に、この学習済みモデルでは、図 16 の入力画像に対しては何れも欠陥ありと予測されたが、鋼板ずれの場合と同様、欠けを認識して欠陥ありと予測しているとは言い難い。

次に、サンプル画像を追加した学習済みモデルによる結果について見てみる。No.19 の画像では、CAM によるヒートマップでは欠けが明らかに強調されていることがわかる。No.20 の画像では、まだはっきりしないが欠けを抽出しているように見える。しかし、No.21、22 ではサンプル画像を追加する前の学習済みモデルの結果とほとんど変わらない。

以上のことより、欠けを判別するためには、まだ、欠けのあるサンプルが不足しているものと考えられる。

線傷、打痕については、ステータコア、ロータコアともに複数のサンプルがあり欠陥を十分抽出できた。しかし、鋼板ずれおよび欠けについては、最初の段階では鋼板ずれがロータコアのみ 1 種類 、欠けがステータコアのみ 1 種類のサンプルを用いた学習であったため、両者とも欠陥をはっきり抽出することができなかった。

当初、モータコア欠陥検出用データセットの欠陥ありのセットについては、欠陥の種類に関係なく一つの集合とすればよいと考えていた。しかし、上記の結果より欠陥の種別ごとに複数のサンプルが必要であることがわかった。

今後、試験セットを用いた検証も行わなければならないが、今回の学習済みモデルの分類精度はまだ低いと考えられる。したがって、欠陥種別ごとのサンプルをさらに収集し学習を行うことで分類精度を高めていく必要がある。

また、本稿で示した欠陥あり、なしの二値分類ではなく、欠陥種別ごとにデータセットを用意して多クラス分類を行った方が、欠陥検出の要因を特定できるより有用な学習済みモデル（分類器）を獲得できるものと考える。

8. あとがき

今回行ったモータコア欠陥検出のための機械学習により、入力画像の最適画素数、ファインチューニングの再学習ブロック数など多くの知見を得ることができた。

機械学習では、データセットを用意することに最も労力

を費やす。今後は、改善すべき点があったデータセットを見直すとともに、サンプルの撮影法など効率よく学習を行うための方法について検討していく予定である。

最後に、現在目的としているモータコア欠陥検出を実用化するためにはまだ幾つかのハードルがあるが、なるべく早く生産ラインに導入できるよう努力していく所存である。

参考文献

(1) A. Gé*r*on :『scikit-learn、Keras、TensorFlow による実践機械学習 第 2 版』(下田(監訳)、長尾(訳)) オライリージャパン (2022)

(2) G.E. Hinton, S.Osindero, and Y.W. Teh :「A fast learning algorithm for deep belief nets.」 Neural Computation, Vol.18, Issue 7 (2006) 1527-1554

(3) G. Kron :『The application of tensor to the analysis of rotating electrical machinery, Parts I-XVI Elementary engineering treatment』General Electric Review (1938)

(4) 上原 :「工学力学系におけるエネルギ変換の幾何学的理論」山形大学紀要 (工学) Vol.18　No.2 (1985) 91-100

(5) 甘利、長岡 :『情報幾何の方法』岩波書店 (1993)

(6) 岡谷 :『深層学習 改定第 2 版』講談社 (2022)

(7) A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing :「Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks」 In Proc. ECCV (2008)

(8) Y. Aytar and A. Zisserman :「Tabula rasa: Model transfer for object category detection」 In Proc. ICCV (2011)

(9) M.Oquab, L.Bottou, I.Laptev and J.Sivic :「Learning and transferring mid-level image representations using convolutional neural networks」 In Proc. CVPR (2014) 1717-1724

(10) O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei :「ImageNet large scale visual recognition challenge」 arXiv:1409.0575 (2015)

(11) B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba :「Learning deep features for discriminative localization」 In Proc. CVRP (2016) 2921-2929

(12) R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra :「Grad-CAM: Visual explanations from deep networks via gradient-based localization」 In Proc. ICCV (2017) 618-626

(13) D. Smilkov, N. Thorat, B. Kim, F. Vi*é*gas, and M. Wattenberg :「SmoothGrad: Removing noise by adding noise」 arXiv:1706.03825 (2017)

(14) PT. Jiang, CB. Zhang, Q.Hou, MM. Cheng, and Y. Wei :「LayerCAM: Exploring hierarchical class activation maps for localization」 IEEE Transactions on Image Processing 30 (2021) 5875-5888

(15) S. Che, J. Meng, J. W. Sheaffer, K. Skadron :「A performance study of general-purpose applications on graphics processors using CUDA」 Journal of Parallel and Distributed Computing 68.10, 1370 (2008)

(16) A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam :「MobileNets: Efficient convolutional neural networks for mobile vision applications」 arXiv:1704.04861 (2017)

(17) M. Wieler, T. Hahn :「Weakly supervised learning for industrial optical inspection」 In DAGM symposium in 2007