画像認識技術による火星大気シミュレーションデータの解析

師 智薫

神戸大学 理学部 惑星学科
流体地球物理学教育研究分野

2025/03/07

要旨

本論文では, 機械学習による解析の試みとして畳み込みニューラルネットワーク (Convolutional Neural Network, CNN) による画像認識技術を用い, 火星大気のシミュレーションデータの解析を行う. 火星 SCALE-GM による鉛直風のシミュレーションデータから, 鉛直対流を認識するための畳み込みニューラルネットワークを作成し, モデルの学習とモデルによる識別をを行った.

目 次

第1章　はじめに

1.1　火星の鉛直対流

ある高さにある空気塊を 1000hPa まで断熱的に移動させた時の絶対温度を温位という．この温位の勾配が大気の鉛直方向の時に負の値であるときに、下層の温位が高い空気が上昇し上層の温位が低い空気が下降する鉛直対流が発生する．本研究では, この熱的な要因によって発生する鉛直対流を識別の対象として扱う．

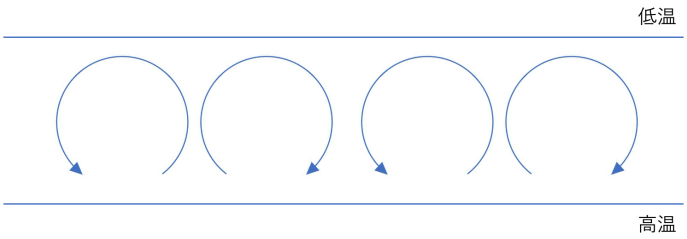

図 1.1:　鉛直対流の模式図.

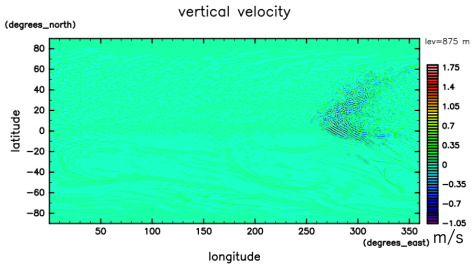

図 1.2:　温位勾配が負の時に発生する鉛直対流による鉛直風．鉛直風の単位は m/s．鉛直対流は数 km から発生するスケールの小さい気象現象であるため, これは glevel8 (水平格子間隔 26km) で表せる程度のスケールの鉛直風である．

1.2 鉛直対流の推測

上で述べたように, 鉛直対流は大気の温位勾配のデータがあれば存在を確かめることが可能である. しかし, 温位勾配のデータが存在しない場合, 鉛直風や密度, 圧力のデータから鉛直対流が発生しているかどうかを推測することができる. 本論文では, シミュレーションデータから鉛直風を描画した画像を畳み込みニューラルネットワークによる画像認識で解析することで, 温位勾配のデータが存在しない場合でも鉛直対流を推測する手法を述べる.

1.3 人工知能

人工知能 (Artificial Intelligence) の定義は研究者ごとに異なっている. これは『知性』や『知能』自体の定義がそもそも定まっていないためであり, したがって人工知能の統一した定義を行うことも困難であると考えられているからである. [*1] 本論文では, 人工知能を「人間と同じような知的処理を行うことのできる技術や機械」と定義し, 学習を行ったコンピュータが人間と同じように画像の中にあるものを識別ができるようになることを目指す. また, 人工知能は発展段階に着目した分類がされている. 以下で紹介する機械学習と深層学習は人工知能の発展段階に応じた分類である.

1.4 機械学習

機械学習 (Machine Learning, ML) とは, 人工知能の分類の 1 つである. コンピュータが人間に近い高度な認識能力を得るためには, パラメータと呼ばれる認識のための基準が必要となる. 人間と同じようにコンピュータがデータを元に学習することとでパラメータを自動的に決定, すなわち学習するための理論体系が機械学習である.

[*1] 平成 28 年度版 情報通信白書 (総務省) より.

1.5 深層学習 (ディープラーニング)

深層学習 (ディープラーニング) とは人工知能の分類の 1 つである. 従来の機械学習では, 学習を行う際, コンピュータがデータを識別するのに必要な特徴量と呼ばれる数値を人間が設定する必要があった. これは, モデルにとって最適な特徴量であるとは限らないため, 学習精度を向上させる上で課題となっていた. しかし, 深層学習では特徴量をコンピュータ自身が決定する. したがって, よりコンピュータがデータの特徴を捉えやすい特徴量を設定し, 高い認識精度を実現することができる. 深層学習では, モデルが自ら特徴量を設定することでより効率的な学習を行うことができる. 画像認識の分野では突出しており, 人間が行うより正確に, かつ膨大な枚数の画像を扱うことができる.

1.5.1 ニューラルネットワーク

ニューラルネットワークとは人間の脳の神経回路を模した形状のモデルであり, 深層学習の基本的な仕組みとなっている. 人間の神経回路は, ニューロンと呼ばれる神経細胞の集まりであり, ニューロン間の信号の伝達によって情報を処理する. このニューロンを模した単一のモデルをパーセプトロンと呼ぶ. 図 1.3 で丸で示したところをノードといい, パーセプトロンのこのノードからノードへの伝達の際に, ノード間の結び付きの強さ (重み) をかけ, その総和と 1 に重みをかけたバイアス項 (定数項) の和に活性化関数をかけることで次のノードへと入力される. パーセプトロンをつなげていくことで, 入力層と出力層の間にいくつも枝分かれをした隠れ層を持つネットワークが完成する. このネットワークをニューラルネットワークと呼ぶ. ニューラルネットワークの隠れ層を増やしていくことでより複雑な出力が可能なディープニューラルネットワークとなる. 本論文では, 作成したニューラルネットワークをモデルとする.

1.5.2 損失関数

学習モデルによる出力と正解となる出力が, どの程度離れているかの指標となる関数を損失関数 (損失) という. 本論文では, 交差エントロピー誤差 (クロスエントロピー誤差) を損失関数として使用する. 交差エントロピー誤差 Q は, 学習モデルによる出力が $\hat{z}_c$, 正解の出力が z_cの時,

[*2]物体・画像認識と時系列データ処理入門 (チーム・カルポ) より.

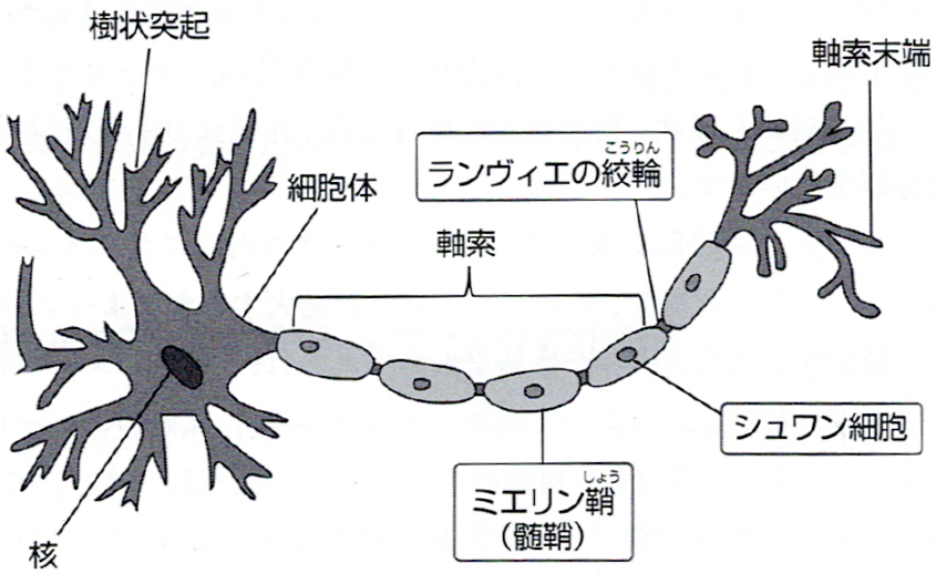

図 1.3: 人間の神経細胞 (ニューロン) の図. ニューロンが繋がったネットワークを神経回路という. ニューロンは別のニューロンへと信号を伝える. [*2]

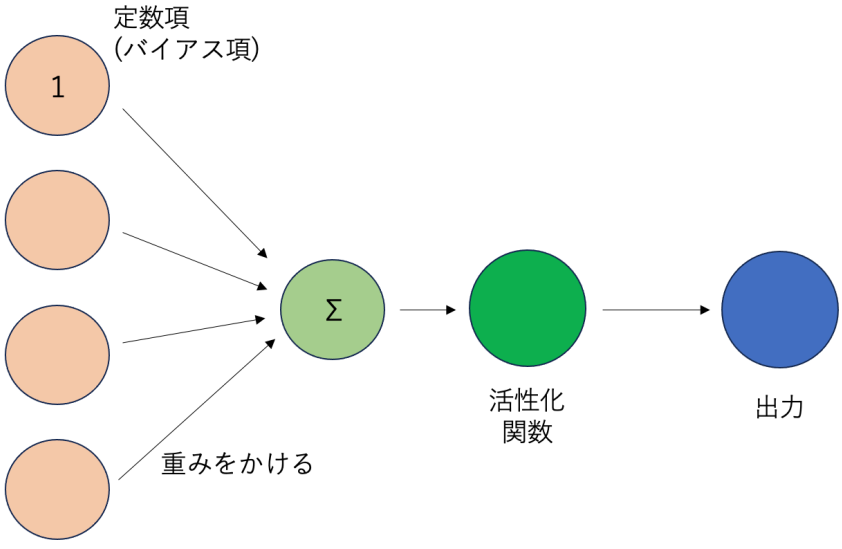

図 1.4: パーセプトロンの模式図.

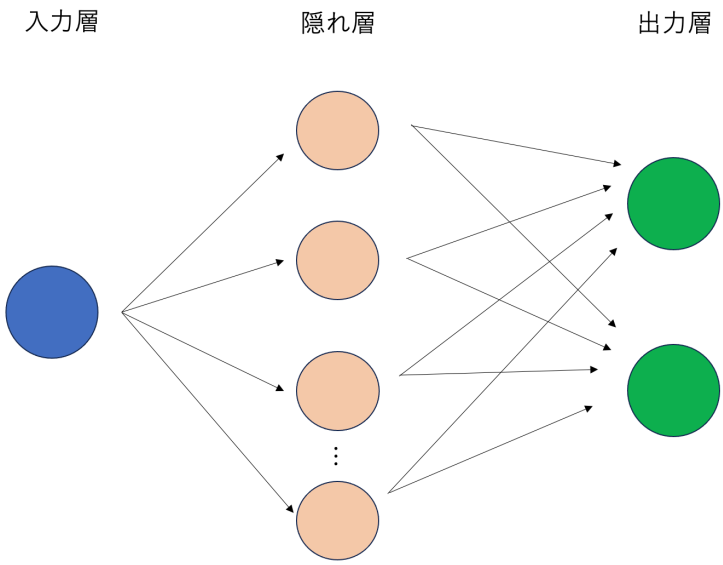

図 1.5: ニューラルネットワークの模式図. パーセプトロンを繋げていくことで図のような形状のモデルとなる.

$$Q = -\sum_{c=1}^{C}\sum_{k=1}^{K} z_c(k)\log \hat{z}_c(k) \tag{1.1}$$

で表される. 損失関数モデルが正解に近づくためには, この損失関数の値が小さくなるように重みを更新し最適解を求める必要がある. 学習モデルによる出力と正解となる出力の差は学習時とは逆方向, すなわち出力層から入力層に向かって伝播され, 伝播された差によって各ノードは重みを更新する. 本論文で作成する学習モデルは確率的勾配降下法 (Stochastic Gradient Descent, SGD) と呼ばれるアルゴリズムを用いて重みを最適解に近づけている.

確率的勾配降下法

図 1.5 のように損失関数を縦軸, 重みとバイアスを横軸とした時, 損失関数が 0 に近づくほど勾配は小さくなる. このことを利用し, 勾配を求め重みを更新していくことで, 最適解に近づけていく方法を勾配降下法という. 勾配降下法のうち, 確率的勾配降下法でははランダムに取り出した 1 つのデータのみを用いて勾配を求め, パラメータを更新していく作業をデータの数だけ行う. 現在時刻が k の時, 損失関数を Q とすると, 重み $w_m(m = 1, 2, ..., M)$ の最適解は以下の手順で見つけることができる.

初期値：

$$w_m(0) = (\text{適切な任意値}) \tag{1.2}$$

繰り返し処理：for$k = 0, 1, ..., K$:

$$\Delta(k) = \frac{\partial Q}{\partial w_m}(w_1(k), w_2(k), ..., w_M(k)) \tag{1.3}$$

$$w_m(k+1) = w_m(k) - \eta\Delta(k) \tag{1.4}$$

式中の η は学習率であり, η の値を変更することで一度にどの程度更新するかを調整できる. また, $\eta > 0$ である.

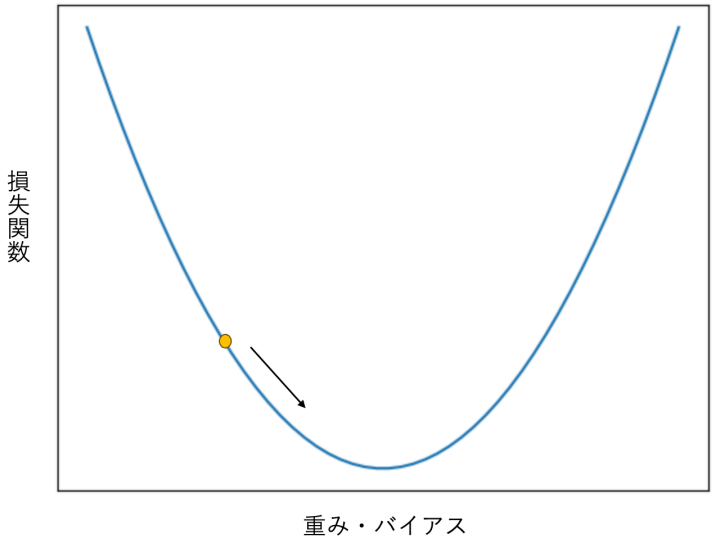

図 1.6:　勾配降下法の模式図. 損失関数を縦軸, 重みとバイアスを横軸とした時, 損失関数が 0 に近づくほど勾配は小さくなる.

1.6 研究の背景と目的

大型計算機の性能の向上により, 解像度の高いシミュレーションデータが得られるようになった. 人の手による大量のデータの解析には量的限界が存在し, 解析のために膨大な時間をかけることもあり得るだろう. このような背景から生じた人の手による解析の量的限界を克服するため, 機械学習の技術を用い人間と同等の認識能力を有した学習モデルによるより効率的な解析を実現することを目指す. 本論文では, 機械学習による解析の試みとして火星の鉛直風のシミュレーションデータを用いた鉛直対流の画像認識を行っていく.

1.7 本論文の構成

本論文の構成は以下の通りになっている. 第 2 章では, 解析に用いたデータを述べる. 第 3 章では解析に用いた手法, 構築した CNN について説明する. 第 4 章では, 学習した CNN による画像の推測の結果を, 第 5 章では考察を述べる.

第2章　使用データ

2.1　使用データ

本論文で使用したデータは火星 SCALE-GM で火星大気のシミュレーションを行ったデータである. 火星 SCALE-GM は地球の大気大循環モデル SCALE-GM を元に作られた火星の大気大循環モデルである. 鉛直方向の運動を無視する静力学系の方程式を用いる従来の全球モデルとは異なり, 計算機の性能向上による高解像度の計算に対応した非静力学系の方程式を用いている. そのため, 火星 SCALE-GM では水平規模が数キロメートルの現象も表現できる高解像度全球大気計算が可能となっている. シミュレーション時の水平解像度は glevel8, すなわち水平格子間隔が 26km, 地形無し条件の元で北半球春分から約 60 日計算したデータとなっている.

2.2　データの加工

シミュレーションデータは NetCDF データとなっているため, 画像認識を行うために画像データへと加工をする必要がある. また, モデルの学習では学習に用いるデータ (学習データ) と学習した後に予測ができるかどうかを確かめるためのデータ (テストデータ) を用意する. 学習に用いるデータは事前に正解 (ラベル) をつけておくことで, モデルは画像と正解を照らし合わせながら学習を行うことができる. 学習を終えたモデルにテストデータで予測をさせることで, モデルの性能を評価することができる. 本論文では Python を用いて加工を行った. 最初に, NetCDF4 モジュールを用いて読み込んだシミュレーションデータを 描画のために Numpy モジュールで扱える npy ファイルの配列に変換した. 次に, 全球を識別のための最小単位として 7.5°×7.5° の領域に分割し, それぞれの領域で鉛直対流があるか, あるいはないかを判断するための二値分類モデルを作成した. したがって, 鉛直対流がある場所を「1」, 鉛直対流がない場所を「0」と分類し, 描画用のライブラリで

ある Matplotlib で出力された画像 1 枚ずつににラベル付けを行った. これにより, 人間の目による分類をモデルに学習させることができる.

2.2.1 データのラベル付け

ラベル付けは, 各ラベルごとにフォルダにまとめた後に, glob モジュールと tensorflow モジュールで各画像ごとの 0 もしくは 1 のラベルを新たな配列として作成する. これにより, 画像データとそれに対応したラベルの 2 つの配列を用意することができる. 学習データは, 高度 1km で切り出した計算最終時刻までの 10 火星日間, 全球を 7.5°×7.5° の領域に分割し, 1 枚ずつラベル付けを行った計 92160 枚の 30×30 ピクセルの PNG データを用意した. テストデータは計算最終時刻の瞬間場を 高度 1125 m で切り出したものを, 学習データと同様に 7.5°×7.5° の領域に分割した後ラベル付けをした 1152 枚のデータを用いた. 作成した画像は学習モデルに入力するためにグレースケールの画像に変換している.

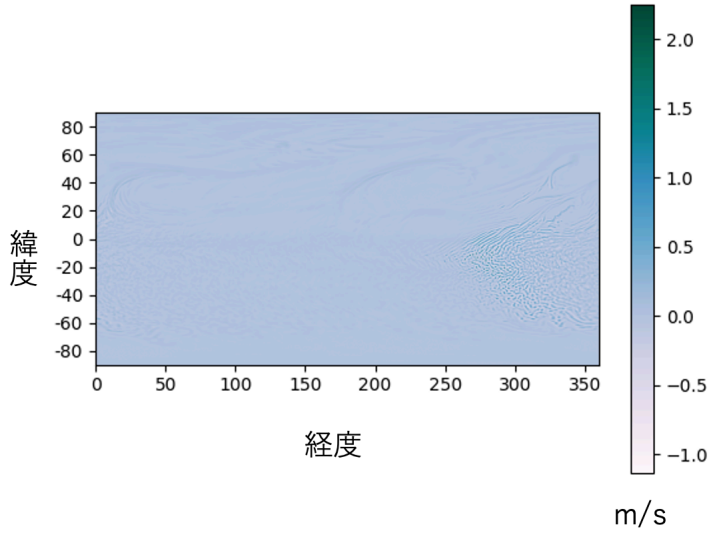

図 2.1: 全球の鉛直風を描画した画像. 縦軸は緯度, 横軸は経度を示す.

2.2.2 オーバーサンプリング

全球を描画した画像を見ると, 鉛直風がある範囲は全球に比べて面積が大幅に狭く, ラベル付けを行った時に「1」のラベルがついたデータが「0」のデータに比べ少

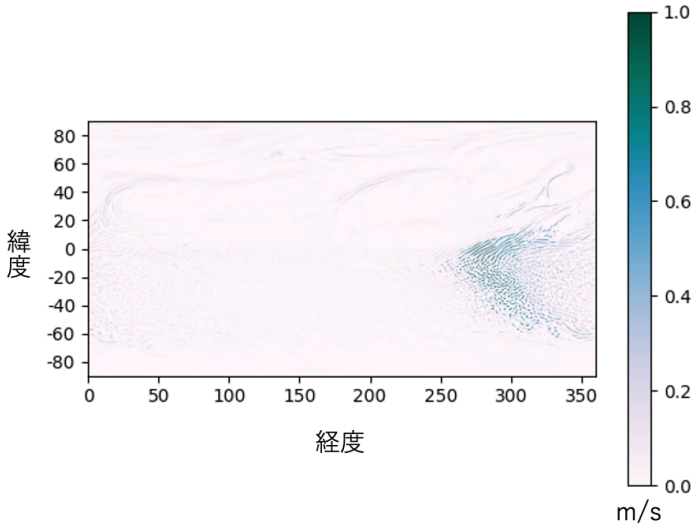

図 2.2: 全球の鉛直風を描画した画像. 視認性を上げるために描画範囲を 0 から 1 に変更している. 鉛直対流と見られる強い鉛直風が画像右端と左端で発生している.

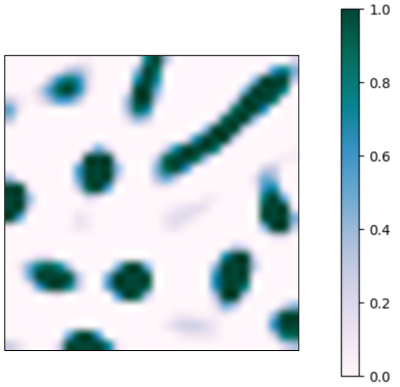

図 2.3: 30×30 ピクセルで分割した画像の一例. 鉛直風が確認できるため, 「1」のラベルをつける.

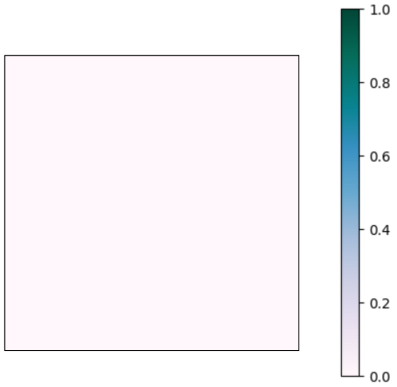

図 2.4: 30×30 ピクセルで分割した画像の一例. 強い鉛直風は確認できない. 「0」のラベルをつける.

なくなってしまうことがわかる. このようなデータの不均衡は学習の精度の低下を招く. したがって, 学習の精度を上げるために少ないラベルのデータを増やす「オーバーサンプリング」によるデータの均衡化を行った. Python の tensorflow.keras モジュールを使用し, 学習データの中から鉛直対流がある「1」のラベルがついた画像データに回転や左右反転, 上下反転の処理を加え, 92160 枚から 148160 枚に全体の画像データを増やすことで, 不均衡による学習精度の低下が起こらないようにした.

第3章　解析手法

3.1　畳み込みニューラルネットワーク (CNN)

畳み込みニューラルネットワーク (Convolutional Neural Network, CNN) とは, 多次元配列に特化したニューラルネットワークである. 通常のニューラルネットワークでは 2 次元の画像データなども 1 次元の配列に変換して学習を行うが, CNN では 1 つのニューロンが 2 次元空間の情報, すなわち画像のピクセル同士の位置関係の情報を保持したまま学習を行うことができるため, 画像認識の分野において活用される.

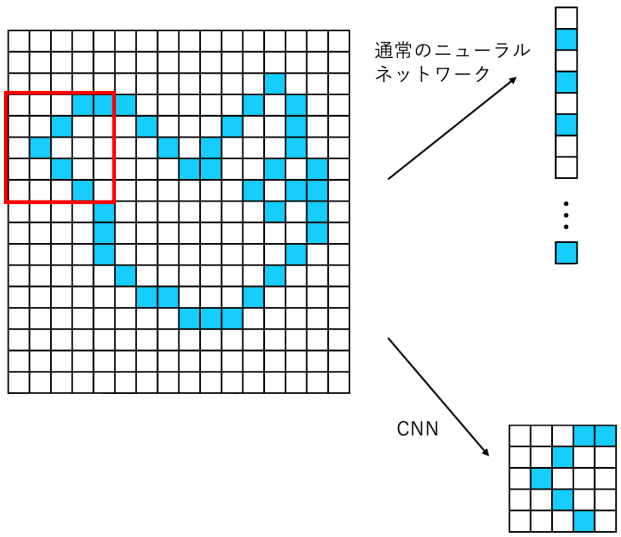

図 3.1:　CNN の学習の模式図. 通常のニューラルネットワークでは赤線で囲った部分を 1 次元の配列に変換するが, CNN では囲われた部分の位置関係を保ったまま学習を行う.

3.1.1 CNN の構築

CNN は入力された画像に対し特定の演算を行うための処理方法を持ち, これをフィルターという. また, CNN も通常のニューラルネットワークと同じように複数の層を持ち, 本論文で使用する CNN は入力層, 畳み込み層, Flatten 層, 出力層で構成されている. 畳み込み層ではフィルターを用いて, 画像の中の特徴的な部分を強調する処理を行う. Flatten 層では畳み込み層から出力した多次元配列を 1 次元配列に変換し, 出力層では, 分類するクラスの数によってノード数が変わる. 本研究では, Python の tensorflow.keras モジュールを使用し, 鉛直対流が存在する場所と存在しない場所の 2 クラスに分類するための CNN モデルを作成した.

学習の流れ

本論文で扱う CNN はミニバッチ勾配降下法と呼ばれる手法で重みの更新を行う. これは, 学習データから無作為に抽出した少量のデータを用いて学習と確率的勾配降下法による重みの更新を行い, 全てのデータを使い切るまで少量のデータによる学習を繰り返す手法である. (ミニバッチ 1 つあたりのデータ数) ×(ミニバッチの数) = (学習データ全体の数) となり, 学習 1 回 (1 エポック) につきミニバッチの数だけ重みの更新が行われる.

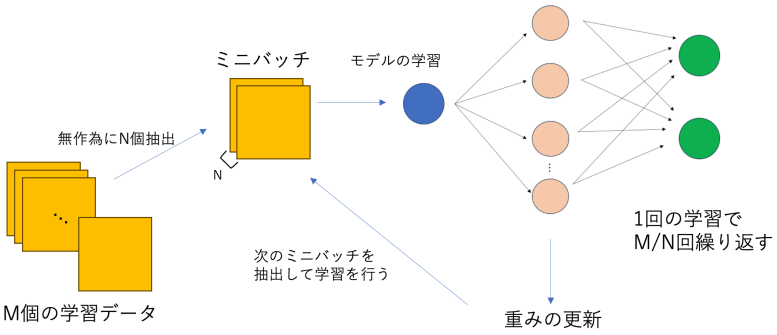

図 3.2: ミニバッチ勾配降下法の模式図. 学習データ全体を M, ミニバッチのサイズを N とすると, 1 エポックにつき M/N 回の重みの更新が行われる.

入力層

入力層では, 画像データを (データサイズ, 行データ, 列データ, チャネル) の 4 次元配列に変換して入力を行う. データサイズは画像データの枚数, 行データと列

データは 1 枚の画像の縦横のサイズ, そしてチャネルは 1 ピクセルにおける 1 値のグレースケールの数値を示す. 本論文では配列が (148160, 30, 30, 1) となる.

畳み込み層

画像内の特定の形状に反応するフィルタを入力されたデータに重ね, 検出する処理を行うための層となる. 本論文では, tensorflow.keras モジュールの Conv2D() メソッドで畳み込み層を設定した. フィルタのサイズは 3×3 ピクセル, 30×30 の画像全体にスライドしながら重ねていくことで, 画像内の特徴的な部分を検出し強調する. ニューラルネットワークにおける重みにあたるのがこのフィルタであり, フィルタがどのような特徴を捉えるかを学習していくことで画像認識の精度を学習によって高めていく. 使用するフィルタは 32 枚, フィルタの数が畳み込み層のノード数となり, フィルタの数とフィルタのサイズをかけた数値が畳み込み層の重みの数となる. フィルタ (ノード) と通した時の活性化関数は, 以下の式で表される ReLU(Rectified Linear Unit) 関数 (正規化線形関数) を用いる. これは入力が 0 を超えていれば入力された値をそのまま出力し, 入力が 0 以下であれば 0 を出力する関数である.

$$ReLU(x) = \begin{matrix} x(x > 0) \\ 0(x \leq 0) \end{matrix} \tag{3.1}$$

Flatten 層

畳み込み層からの出力である (30, 30, 32) の 3 次元配列を 1 次元配列に変換するための層が Flatten 層である. これは最終的な出力が 1 次元で行われるためである. Flatten 層を通すことで (30, 30, 32) は 28800 の 1 次元配列となる.

出力層

本論文では二値分類を行うため, 出力層のニューロンの数は 2, 重みの数は $28800 \times 2 = 57600$ となる. 出力層でも活性化関数が適用され, 以下の式で表されるソフトマックス関数を使用する. 各クラスを 0 から 1.0 の間の実数で出力し, 出力値の総

和は 1 となる関数であり, ノード数が n 個の時, k 番目の出力 $y_k z$ は

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \tag{3.2}$$

となる.

3.1.2 検証データ

148160 枚の学習データのうち 20% にあたる 29632 枚の画像は学習に使わず, 学習と並行してモデルが未知のデータに対応できるかどうかを確かめるための検証データとして使う. この検証データ用の画像は 1 回の学習ごとにランダムに選ばれ, その回の学習には使われない. 残りの 118528 枚を一度の学習で学習データとして使用し, この学習データから少量のデータを 1 つのミニバッチとして無作為に抽出しながら学習を行う. 今回は 128 枚のデータを 1 つのミニバッチとして扱い, 合計 926 個のミニバッチを 1 回の学習で用いる.

層の種類	出力されるデータの形状	定数項を含むパラメータ数
畳み込み層	(30, 30, 32) の 3 次元配列	320
Flatten 層	28800 (1 次元配列)	0
出力層	2 (1 次元配列)	57602

3.1.3 損失と正解率

前章で述べたように, 正解の出力と学習モデルの出力である損失を減らすように重みを更新することでモデルは正解の出力に近づく. 正解率は (予想と正解が一致するデータの数)/(全体のデータの数) で求めることができる. 正解率も損失と同様に学習の精度を評価する基準となる.

3.1.4 早期終了

学習回数が 100 に達する前に, モデルの精度が限界まで向上しそれ以上改善しない状態になることがある. そういった場合に, 学習を 100 よりも手前で終了する

[*3]物体・画像認識と時系列データ処理入門 (チーム・カルポ) を参考に作成.

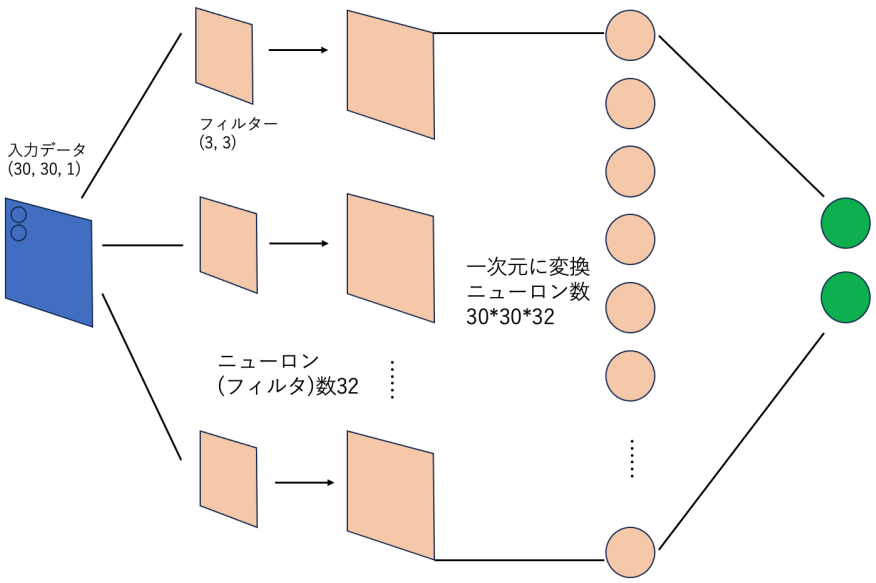

図 3.3: 作成した CNN モデルの模式図. 左から順に入力層, 畳み込み層, Flatten 層, 出力層となっている. [*3]

ように設定する. 本論文のモデルでは損失の値を早期終了のための基準とし, 損失の値が 5 回以上改善されなかった場合に学習を終了する.

第4章　結果

4.1　学習の推移

学習は 18 回目で早期終了した. 検証データによる最終的な正解率は 81.46%, テストデータによる最終的な正解率は, 95.31% である. 学習の各回ごとにどのように正解率と損失が変化したかを図 4.1 に示す.

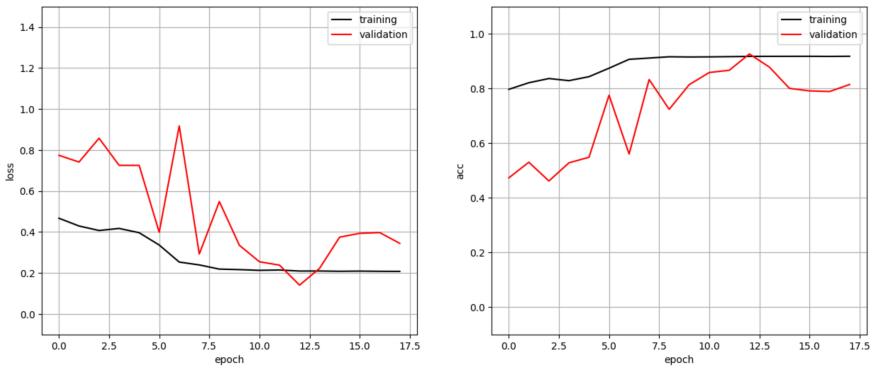

図 4.1:　学習の推移の時系列データを示したグラフ. 左が損失, 右が正解率である. 赤線は検証データ, 黒線は訓練データの推移を示す.

4.2　学習したモデルによる識別

学習を行ったモデルとテストデータを用いて実際に識別を行った. 学習時と同じようにテストデータ用の画像を分割し, 各領域に鉛直対流があるかどうかを, 鉛直対流がない場合は「0」ある場合は「1」で答える. 以下の表ではモデルによる予測のラベルと正解のラベルの比較を行う. 上段のカッコ内は正解が 0 の画像に対し, 0 と予測した割合, 1 と予測した割合を示す. 下段も同様に, 正解が 1 の画像に対し, どちらのラベルで予測したかの割合を示す. 図 5.2 から図 5.4 では予測したラ

ベルとテストデータの比較を行っている. 学習モデルによる識別が行われていることが確認できるが, 本論文で識別したい鉛直対流による鉛直風ではない画面中央付近の鉛直風も鉛直対流がある部分として識別している.

予測と正解の比較	予測が 0	予測が 1
正解が 0	916(98.39%)	15(1.61%)
正解が 1	39(17.65%)	182(82.35%)

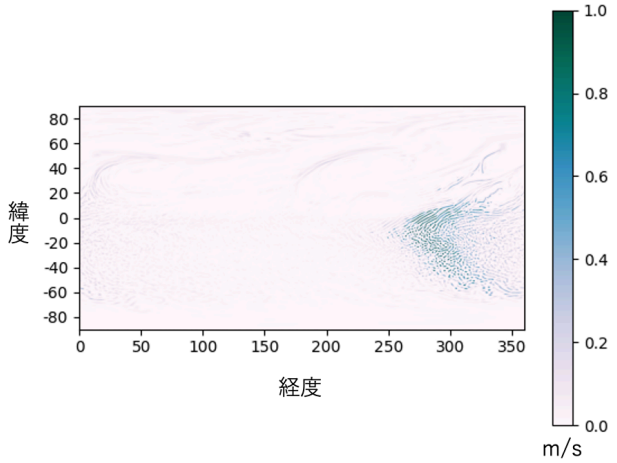

図 4.2: テストデータを描画した画像. 横軸は経度, 縦軸は緯度を示す. 画像右端と左端に鉛直対流と見られる鉛直風が確認できる.

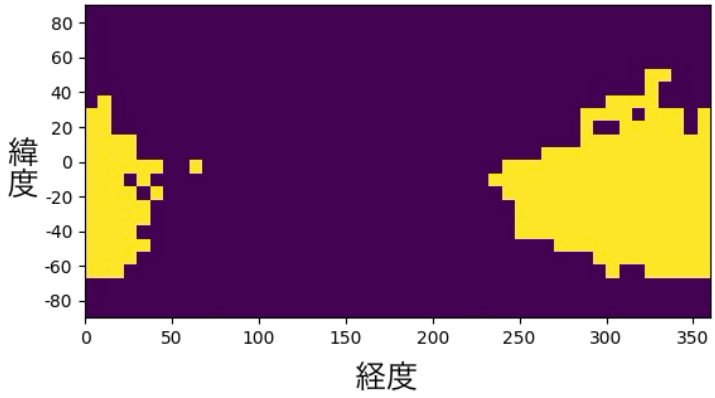

図 4.3: テストデータの正解のラベルを描画した画像. 横軸と縦軸はそれぞれ経度と緯度に沿ったラベルの枚数を示す. 鉛直対流による鉛直風がない「0」のラベルを紫, 鉛直風がある「1」のラベルを黄色で描画している.

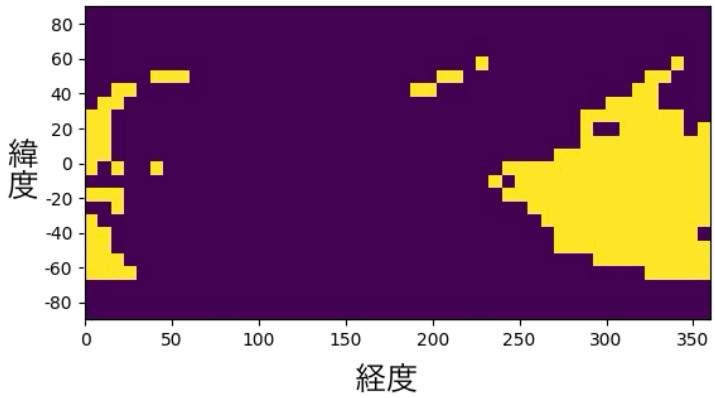

図 4.4: 7.5°×7.5° に分割したテストデータの各領域に鉛直対流があるかどうかをモデルに識別させ, その結果を全球に並べ直し描画した画像. 「0」を紫, 「1」を黄色で描画している. 画面右端と左端を「1」と予測していることがわかる.

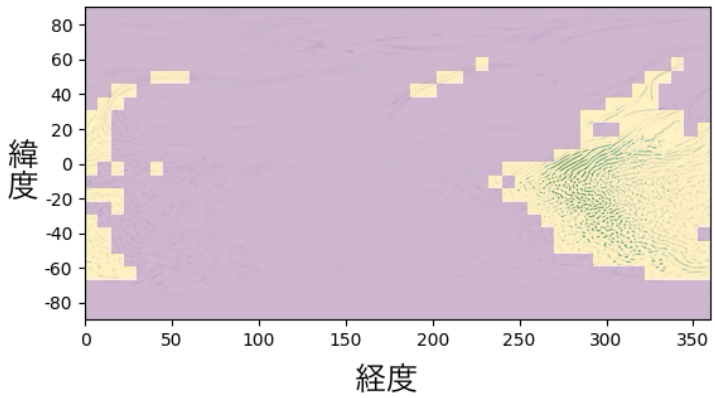

図 4.5: 図 5.2 と図 5.4 を重ねた画像. 鉛直対流による鉛直風が発生している位置とモデルが鉛直対流があると識別した位置が重なっていることがわかる.

第5章　考察

5.1　データの不均衡による精度の低下

本論文ではデータの加工の際, オーバーサンプリングにより少ないラベルのデータを増やすことで不均衡を解消し精度を上げて学習を行った. 本節では研究の過程の紹介として, オーバーサンプリングを行わない状態での学習を行いその精度を検証した. データの加工や学習に使ったモデルの条件は変わらず, 学習に使う画像データの枚数をオーバーサンプリングによる均衡化を行った 148160 枚から 不均衡な状態の 92160 枚に削減する. この時, 「0」のラベルがついた画像データは 77643 枚, 「1」のラベルがついた画像データは 14517 枚となる. 学習の後に同様にテストデータを使ったモデルによる予測を行い, 正解との比較を行ったものが表 5.1 である. また, 予測されたラベルを全球に変換したものを図 5.1 に示す. 図の通り, 鉛直対流による鉛直風がある領域のデータが少なかったことで識別の精度が下がり, モデルが鉛直対流が学習できていないことがわかる.

予測と正解の比較	予測が 0	予測が 1
正解が 0	931(100.0%)	0(0.0%)
正解が 1	211(100.0%)	0(0.0%)

5.2　データ数を削減した場合の学習

機械学習では学習に用いるデータ数を減らすことで計算にかかる時間を削減できる. したがって, データ数を減らした場合でも元の同様の精度を維持できれば, 今後の学習でデータ量を削減しより短時間で研究を進めることができると考えられる. 本節では, 元のデータの 1/2, 1/4 と画像データの枚数を減らした時に識別の精度がどのように変化するかを検証していく.

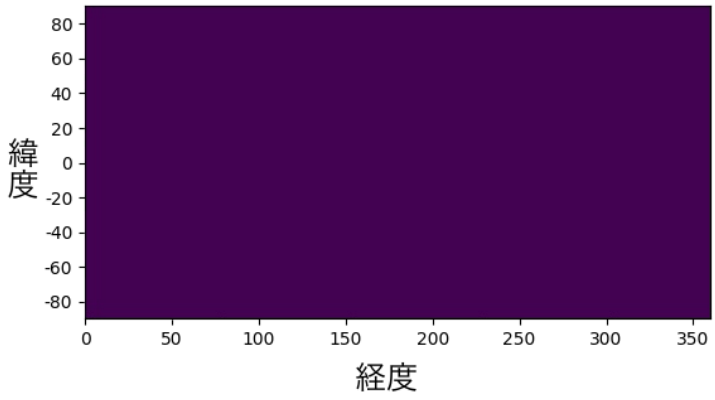

図 5.1: 不均衡なデータによる学習を行ったモデルが予測したラベル. 第 4 章と同じテストデータを使用し黄色が鉛直対流がある「1」のラベル, 紫色が鉛直対流がない「0」のラベルを示す. 縦軸の緯度方向に 24 枚, 横軸の経度方向に 48 枚, 計 1152 の領域全てが「0」のラベルとなっている. これは鉛直対流による鉛直風を識別できていないことを示す.

1/2 に削減した場合の学習

学習データの枚数を元の 148160 枚からその半分である 74080 枚に削減し, 学習を行った. 表は予測と正解のラベルの比較を行ったもの, 図 5.2 はモデルによる予測結果を描画した画像である. 元のデータ量 (第 4 章参照) の時と比べ, 左側の鉛直風を識別できなくなっているが, 右側の鉛直対流による強い鉛直風の識別はできていることがわかる.

予測と正解の比較	予測が 0	予測が 1
正解が 0	928(99.68%)	3(0.32%)
正解が 1	80(36.2%)	141(63.80%)

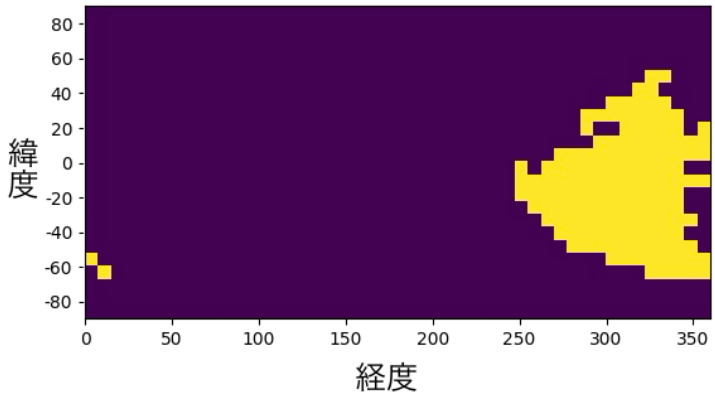

図 5.2: データ量を元のデータの半分に減らして学習を行ったモデルによるラベルの予測結果.

1/4 に削減した場合の学習

学習データの枚数を 74080 枚から更にその半分である 37040 枚に削減し, 学習を行った. 表は予測と正解のラベルの比較を行ったもの, 図 5.3 はモデルによる予測結果を描画した画像である. 1/2 に減らした時と大きな差はないことがわかる. このことから. 各ラベルごとのデータの量が均衡であれば, データの量を削減して学習を行っても強い鉛直風の識別は可能であると考えられる.

予測と正解の比較	予測が 0	予測が 1
正解が 0	929(99.79%)	2(0.21%)
正解が 1	82(37.10%)	139(62.90%)

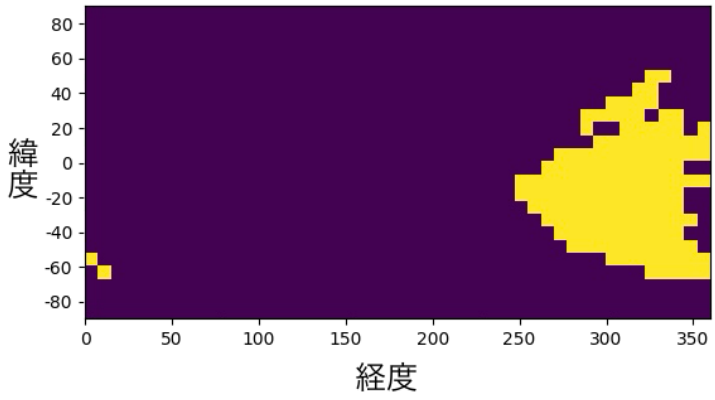

図 5.3: データ量を元のデータの 1/4 に減らして学習を行ったモデルによるラベルの予測結果.

5.3 マルチクラスの分類

前節まで行っていた研究では, 単直な鉛直風の有無による二値分類だったため, 鉛直対流とは直接関係のない鉛直風の画像データも鉛直流がある領域のデータとして識別することが課題となっている. したがって, 鉛直風がある部分のデータをさらに細分化することで, より正確に鉛直対流の識別が実現できると考えられる. 本節では鉛直風のデータを 2 クラスから 3 クラスに分類し直し, 学習を行った際の精度を検証する. 鉛直風のない領域は二値分類と同じように「0」, 鉛直対流によるものではない鉛直風を「1」, そして鉛直対流による鉛直風を「2」として再度分類を行う. 使用データや加工方法は変えず, モデルの出力層のニューロン数を分類したい数に合わせて 2 から 3 に変更する. また, オーバーサンプリングによるデータの均衡化により, 学習データに使われる画像は合計で 218160 枚となる.

以下に学習を行ったモデルによる識別の結果を示す. テストデータは第 4 章と同じデータを用いている. 表はラベルごとの予測と正解を比較し, 各正解のラベルご

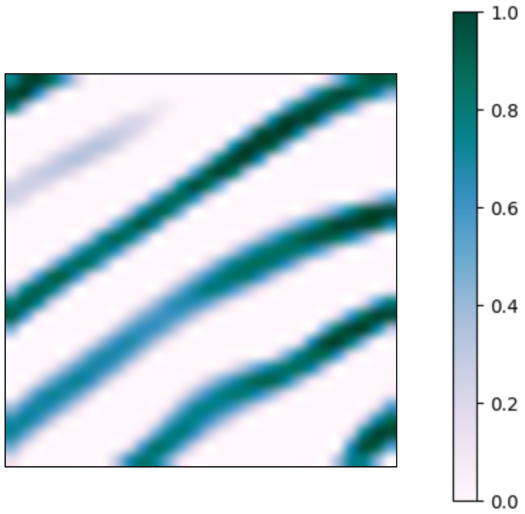

図 5.4: 「1」に分類し直した画像の一例. 筋状に描画された鉛直風が確認できる.

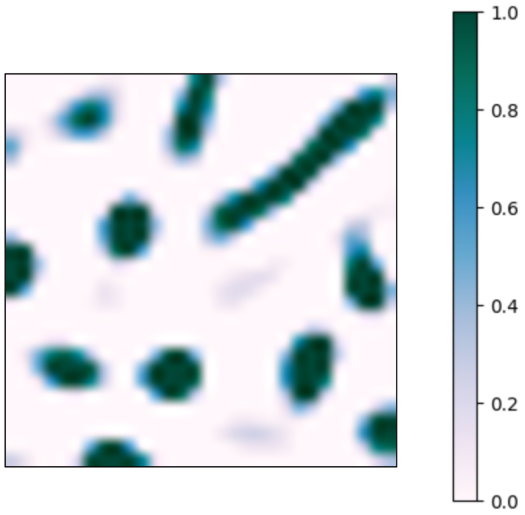

図 5.5: 「2」に分類し直した画像の一例. 斑点のような模様で描画された鉛直風が確認できる.

との予測結果の割合を出したものである. 図 4.6 はテストデータの正解のラベルを全球で描画した画像, 図 4.7 は学習モデルが行った予測のラベルを全球で描画した画像, 図 4.8 はテストデータを描画した画像と学習モデルによる予測のラベルを重ねた画像である. 2 クラスでの分類と比較して, 強い鉛直風の有無は識別できているが左端のやや弱い鉛直風の識別ができず, 鉛直風を含む 1 や 2 のラベルの正答率が低下している. 鉛直風がある領域内での, 鉛直対流による鉛直風かそうでないかの識別は不可能ではないことがわかる.

予測と正解の比較	予測が 0	予測が 1	予測が 2
正解が 0	930(99.89%)	1(0.11%)	0(0.00%)
正解が 1	86(60.99%)	53(37.59%)	2(1.42%)
正解が 2	10(12.50%)	52(65.00%)	18(22.50%)

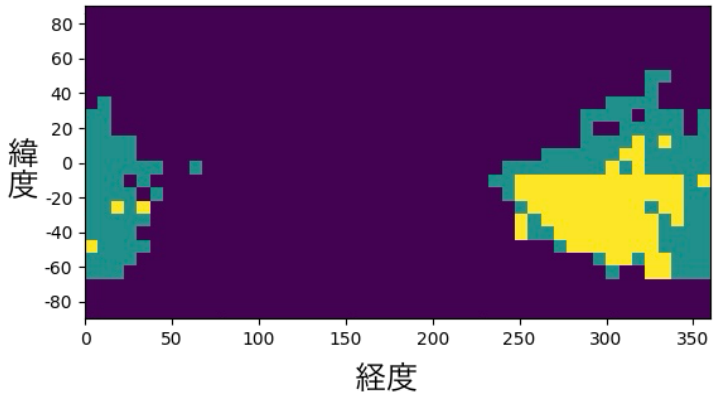

図 5.6: テストデータの正解のラベルを描画した画像. 横軸と縦軸はそれぞれ経度と緯度に沿ったラベルの枚数を示す. 鉛直対流による鉛直風がない「0」のラベルを紫, 鉛直対流ではない鉛直風である「1」のラベルを緑, 鉛直対流による鉛直風である「2」を黄色で描画している.

さらに, データの量を 1/2 である 109080 枚に削減し, 同様に学習とモデルによる予測を行った. 予測と正解の比較を表に, 予測したラベルを全球で描画した画像を図 4.9 で示している. 2 のラベルの正答率は元のデータ量より高くなっているが, 1 のラベルの正答率は低下している.

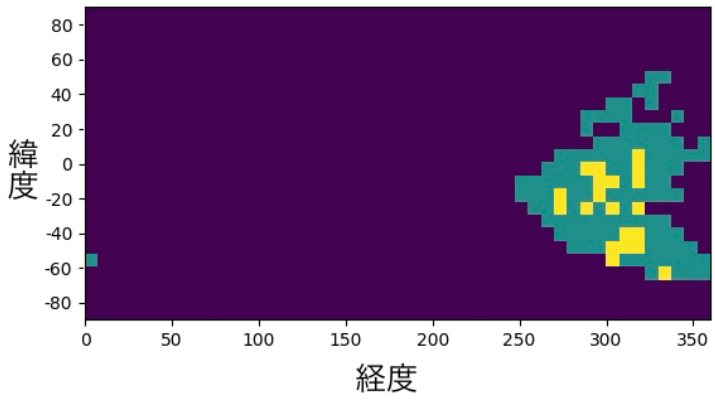

図 5.7: 学習モデルによる予測のラベルを描画した画像. 鉛直対流による鉛直風がない「0」のラベルを紫, 鉛直対流ではない鉛直風である「1」のラベルを緑, 鉛直対流による鉛直風である「2」を黄色で描画している.

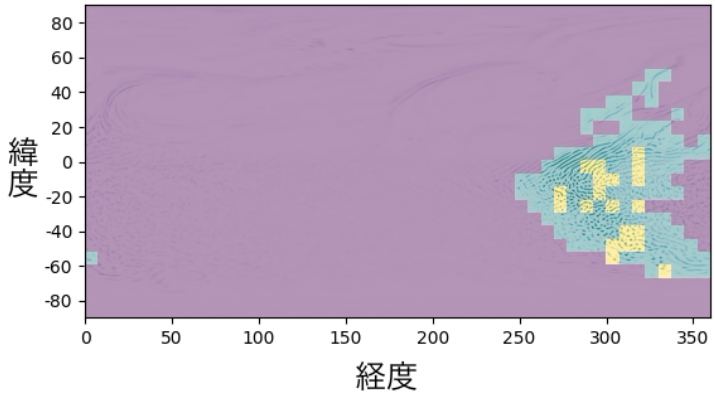

図 5.8: テストデータを描画した画像とモデルによる予測のラベルを重ねた画像. 右側の鉛直風の領域の一部で識別ができていることがわかる.

予測と正解の比較	予測が 0	予測が 1	予測が 2
正解が 0	930(99.89%)	0(0.00%)	1(0.11%)
正解が 1	79(56.03%)	38(26.95%)	24(17.02%)
正解が 2	8(10.00%)	6(7.50%)	66(82.50%)

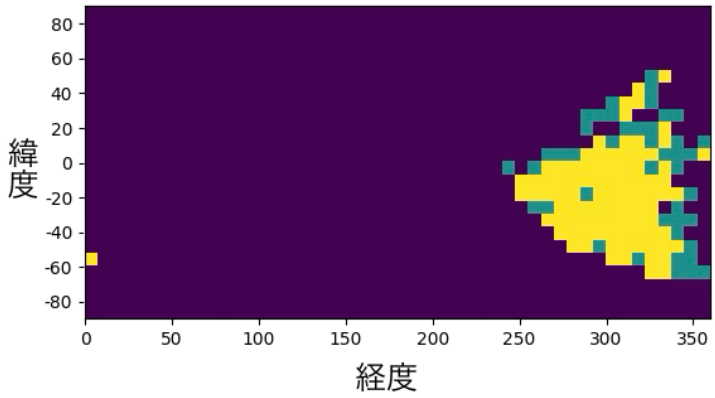

図 5.9: 画像データの枚数を 109080 枚に減らして学習したモデルによる予測のラベルを描画した画像. 鉛直対流による鉛直風がない「0」のラベルを紫, 鉛直対流ではない鉛直風である「1」のラベルを緑, 鉛直対流による鉛直風である「2」を黄色で描画している.

5.4 考察

本論文での研究の結果, 画像認識技術による膨大な量のデータの処理を実現できた. このような膨大な量のデータの解析は統計的な特徴を捉えることにつながると考えられる. また, 本論文では火星の鉛直対流に着目して研究を行ったが, 今後火星以外の天体や気象現象などにも応用ができる可能性がある.

付録 A ソースコード

ここでは本論文で使用したライブラリのバージョンと学習モデルのソースコードを紹介する. 使用した言語は, Python3.11.11, netCDF のデータを読み込むために使用した netCDF4 は 1.7.2 である.

構築したモデルのソースコード

```
# tensorflow.keras モジュールからインポートする
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten
from tensorflow.keras.optimizers import SGD

model = Sequential ()

# 畳み込み層の設定
model.add(
Conv2D(filters=32, # フィルタの枚数
kernel_size=(3, 3), # フィルタのサイズ
padding="same", # 出力する画像データを同サイズにする
input_shape=(30, 30, 1), # 入力する画像データの形状
activation="relu" # 活性化関数は ReLU 関数
))

# Flatten 層の設定
model.add(Flatten())
model.add(Dropout(0.5)) # ドロップアウト

# 出力層の設定
model.add(Dense(2, # 出力層のニューロン数を決定する
activation="softmax" # 活性化関数はソフトマックス関数
))

model.compile(
loss="sparse_categorical_crossentropy", # 損失関数の設定, 正解のラベルと予測の間の誤差を求めるスパース行列対応交差エントロピー誤差
optimizer=SGD(learning_rate=0.1), # 重みの最適化を確率的勾配降下法で行う. 学習率は 0.1
metrics=["accuracy"]) # 学習評価を正解率に設定する

model.summary()  モデルの概要を出力
```

tensorflow.keras モジュールのバージョンは 3.8.0 である. ソースコードを実行

することで, 構築された CNN のモデルの概要が出力される.

学習を実行するソースコード

```
%%time
# tensorflow.keras モジュールからインポートする
from tensorflow.keras.callbacks import EarlyStopping
training_epochs = 100 # 学習回数の決定
batch_size = 128 # ミニバッチのサイズ (1 パッチあたりのデータ
# 128 枚 のミニバッチを 926 回, 1epoch=926 回
# 148160 枚のうち検証データを除いて 118528 枚を使う

# EarlyStopping(早期終了) 監視対象回数内で改善されなければ学習を止める
early_stopping = EarlyStopping(
monitor="val_loss", # 監視対象は損失関数
patience=5, # 回数は 5 回 verbose=1 # 早期終了をログとして出力)

history = model.fit(
x_train, # 学習データ (x_train)
y_train, # 学習データの正解ラベル (y_train)
epochs=training_epochs, # 学習を繰り返す回数 (エポック) を指定する
batch_size=batch_size, # ミニバッチのサイズを指定
verbose=1, # 学習の推移を出力する
validation_split=0.2, # 検証データの割合, 20%を使用する
shuffle=True, # 検証データをランダムに抽出する
callbacks=[early_stopping] # 1 エポック が終了する度に EarlyStopping を呼び出す
)

# テストデータを用いて学習したモデルの評価を行う. x_test はテストデータ, y_test はテストデータの正解ラベル
score = model.evaluate(x_test, y_test, verbose=0)

# テストデータを用いた損失関数を出力
print("Test loss:", score[0])
# テストデータを用いた正解率を出力
print("Test accuracy:", score[1])
```

ソースコードを実行することで, 学習が行われ各学習ごとの損失関数と正解率の推移が出力される.

モデルによる予測を行うソースコード

```
# 必要なライブラリをインポートする
import numpy as np
from sklearn.metrics import confusion_matrix
import pandas as pd

predict = model.predict(x_test, batch_size=16) # テストデータを使用してモデルによる予測を行う
results = predict.argmax(axis=1) # 予測結果の中で最も確率の高いラベルの配列を作成する

# テストデータの正解ラベルとモデルによる予測のラベルの混合行列を出力する
print(confusion_matrix(y_test, results))

pd.options.display.precision = 4 # 表示桁数の設定

# 混合行列を各ラベルの予測に対する正解のラベルの割合として出力する
conf_mat = confusion_matrix(y_test, results, normalize='true')
display(pd.DataFrame(conf_mat))
```

Numpy のバージョンは 1.26.4, scikit-learn モジュールのバージョンは 1.6.1, pandas モジュールは 2.2.2 を使用した.

謝辞

本研究を行うにあたって, 樫村博基講師には研究内容の方針をご指導頂き, 研究や論文執筆に関する相談等で大変お世話になりました. 高橋芳幸准教授には基礎理論読書会やセミナーを通し, 地球流体力学の基礎的な知識や手法についてご指導頂きました. また, 所属する地球および惑星大気科学研究室の皆様には研究室活動を通して発表や論文に関する助言を頂きました. 本論文の執筆に関わってくださった皆様に心からの感謝を申し上げます.

参考文献

[1] 物体・画像認識と時系列データ処理入門 (チーム・カルポ)

[2] 図解即戦力　機械学習&ディープラーニングのしくみと技術がこれ 1 冊でしっかりわかる教科書 (株式会社アイデミー　山口 達輝／松田 洋之)

[3] 入門 ディープラーニング　— NumPy と Keras を使った AI プログラミング— (藤野 巖)

[4] 火星 SCALE-GM