2023年度 卒業論文

汎化性能を考慮した食品パッケージの画像分類

20X4110　鴇田 優太

指導教員

2024年1月31日

法政大学 理工学部 経営システム工学科

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 背景

近年，画像解析技術は，製造業の様々な分野での活用が見込まれている．例えば，品質管理の面では製品の外観検査や欠陥の検出に応用することで，自動的に不良品を排除している．その他にも，画像解析を用いたデータ収集と分析を活用することで，製造ラインを見える化し，効率的な改善策を導き出すことにも応用している．

画像解析技術の１つに画像分類が挙げられる．画像分類技術を応用することで，今まで人間が視覚を通して物を判別するような作業を自動的に機械で判別することが可能になる．食品を生産・管理する工場においても，適切な分類は，工程や物流の最適化，在庫管理などにおいて重要な要素であり，食料品の生産・管理の効率向上に繋がる．しかし，簡単な分類モデルでは，人間の視覚を通した判別に劣ってしまう問題が発生する．

このような背景から，食料品の工場向けのデータ分析に注目し，モデルの精度を示す評価指標を基準に，商品のパッケージが映った画像を食料か飲料かに自動分類するアルゴリズムを開発することが望まれている．精度の良い，適切な分類が可能なモデルを開発することで，人間の視覚を介することなく，分類作業の効率向上や更なる活用を期待できる．

また，より実用的なモデルを構築することを考えると，モデルの汎化性能に重点を置くことも重要と考えられている．汎化性能とは，学習データだけでなく，未知のデータに対しても，正しく予測できる性能のことを言う．そのためには，検証用のデータと予測用データの精度のフィードバックを通して，過学習が起こっ

ていないか，評価指標を通じて確認することで良い汎化性能を達成するというアプローチがとられる．

1.2 先行研究

先にキーワードを簡単に定義する．

- CNN : 畳み込みニューラルネットワークのことで，畳み込み層やプーリング層で構成される深いモデル．
- Vision Transformer： Google によって初めて提案されたで自然言語処理で使われてきた Transformer を使うモデル．
- 転移学習 : あるタスクで学習したモデルである学習済みモデルを異なるタスクに転用する手法．
- StratifiedKFold(K=5) : クラスの割合を保ちながらデータを分割する方法．K は分割数で，各分割セットをフォールド (Fold) という．

Liu et al. (2022) は，ConvNet の設計空間を再検討したモデルとして Convnext を提案し，Transformer 系手法と遜色なく、87.8 パーセントの ImageNet トップ 1 精度を達成している．また，吉次研二 and 中本幸一 (2021) は，画像分類において，CNN より Vision Transformer モデルの方が有効な場合があることを転移学習を用いて示している．そして，石田直也 (2021) は，不均衡データの取り扱いに着目した深層学習による運転者の眠気推定における提案モデルで StratifiedKFold(K=5) を分割に用いている．

1.3 目的

本研究では，汎化性能を考慮したモデルを提案することを目的とする．その目的を達成するアプローチとして，StratifiedKFold(K=5) を用いて，データを 5 つのフォールドに分割し，各フォールド毎に学習済みモデルのファインチューニングを通して，学習・検証を行い，最適なハイパーパラメータ，パラメータを調整，また保存する．こうして保存された５つのモデルを TTA を通して平均することで，１つの予測値を求める方法を行う．

これからのチャプターを通して，扱うデータ，データの前処理，訓練方法，そして予測の部分に分けて，具体的なテクニックとともに提案するアプローチの枠組みに踏み込んでいく．

Chapter 2

食品パッケージデータ

本研究では，株式会社 SIGNATE が運営するデータサイエンスプラットフォーム「SIGNATE」にて，株式会社テクノプロ　テクノプロデザイン社主催「テクノプロ・デザイン社　食品パッケージ画像解析チャレンジ」から提供されたデータを扱う．2023/8/11 から 2023/9/29 までの期間で開催で参加人数は 640 人であった．訓練データは 2176 枚の画像データと対応するラベルデータ，予測データは 2180 枚の画像データのみである．

2.1　データについて

コンペティションではいくつかのデータ形式がある．

2.1.1　訓練データ

訓練データは答えが与えられてるデータのことで，分析用に配布されるデータである．基本的には，この訓練データをデータ分析を通して，学習用データと検証用データに分割して，モデルのパラメータを最適に近づくように調整していく．

2.1.2　テストデータ

テストデータは，反対に答えが与えられていない予測するためのデータのことで，Public データと Private データに分けられる．

Public データは，コンペティションのページ上で，予測をまとめた csv ファイルを submit することで精度を確認できるデータのことをいう．コンペティション

が開催している期間中，この Public データでの予測制度の良さによって，暫定順位が決定される．

一方で，Private データに関しては，コンペティションが終了するまで，その精度を確認することができないデータである．また，この Private データの精度の良さによって，最終順位が決定されていく．そのため，コンペティション参加者は，訓練データと Public データの精度のフィードバックを利用して，Private のデータの制度の良さを上げることを目標に実験を試行錯誤していくことになる．具体的には，学習データと検証データの精度を比較して，学習データの特徴を学習しすぎて過学習が起きていないよう注意しつつ，Public データの精度を上げていくことで疑似的に Private データの精度を上げていくようにする．

2.2 評価指標

評価指標には AUC(Area Under the Curve) を使用する．図 2.1 が AUC を表している．AUC は，ROC 曲線の下の領域を示す指標で，二値分類モデルの性能を評価するために使われる代表的な指標の 1 つである．また，ROC 曲線は真陽性率 (True Positive Rate) と偽陽性率 (False Positive Rate) の関係を表すものである．

AUC は，モデルが異なる閾値でどれだけ優れているかを示しており，とり得る値は 0 から 1 までで 1 に近づくほど良いモデルであることを示す．この批評が本コンペティションで採択された理由は，代表的な 2 値分類モデルの評価指標というよりは，クラスに不均衡がある場合に有効な指標である点が大きいと考えている．極端にクラスに不均衡さがある場合，モデルが常に多数派のクラスを予測するだけで高い精度を得られてしまう問題が起こる可能性がある．しかし，AUC では異なる閾値での真陽性率と偽陽性率を示すので，この問題を防ぐことができる利点がある．

2.3 訓練データファイルの中身

コンペティションで提供された訓練データは，画像データと対応するラベル情報を対応付けた csv ファイル形式で与えられる．加えて，各画像が格納された zip 形式のファイルも与えられている．図 2.2 は，訓練データの csv ファイルを表示したものである．

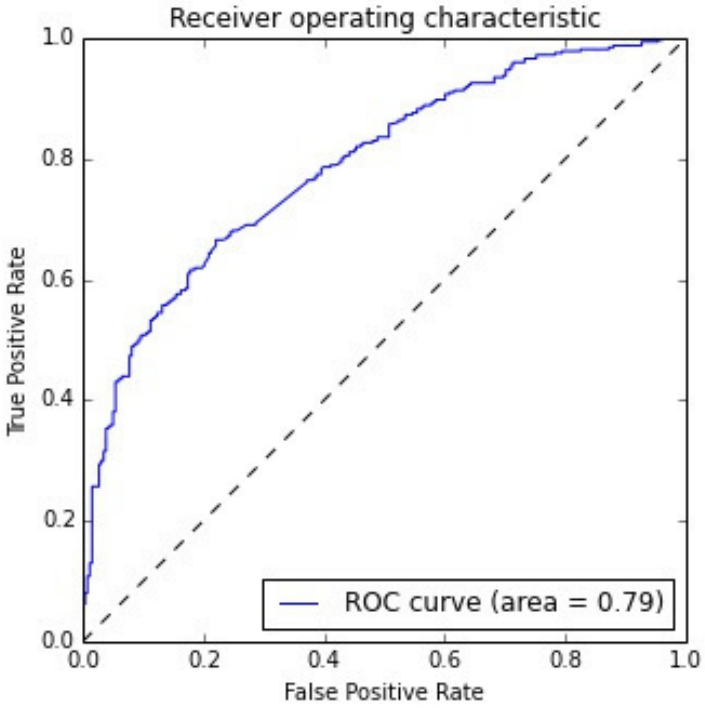

図 2.1: AUC(Area Under Curve).

2176 枚の各画像名と対応するラベルを格納したファイルとなっている．例えば，0 行目における 0000.png の正解ラベルは 0 である．なお，ラベル 0 は飲料を表し，ラベル 1 が食料を表す．また，ラベル 1 が 1182 枚，ラベル 0 が 994 枚で不均衡なクラス比率である．

	image_name	label
0	0000.png	0
1	0001.png	1
2	0002.png	1
3	0003.png	1
4	0004.png	0
...	...	...
2171	2171.png	1
2172	2172.png	1
2173	2173.png	1
2174	2174.png	0
2175	2175.png	0

2176 rows × 2 columns

図 2.2: 訓練データ.

Chapter 3

モデル

モデル構成として，手法を考える項目として，以下の事項がある．

- データ前処理
- ミニバッチ処理
- 訓練方法
- 転移学習
- 重みの更新
- 損失関数

以下に，提案する手法について説明する．

3.1　データ前処理

データ前処理段階を説明したい．まず，前処理とは，元データに対して何らかの変形であったり，形を整形することをいう．数量データであれば，組み合わせたり，累乗を取ることで新たな特徴量生成を行う．カテゴリ変数であれば，ダミー変数化して数値データに置き換える．他にも，欠損値である NaN(Not a Number) や外れ値の処理などが挙げられる．しかし，前章 2.3 で確認したが入力データが画像のみである．そのため，このような学習前の前処理ではなく，学習に移る上で，データに対して施した処理として，画像拡張と StratifiedKFold(K=5) について，深く解説したい．

3.1.1 データ拡張 (Data Augumentation)

データ拡張は，元の画像に対して回転・拡大のような変形を施すことで，データの水増しを機械的に行える．そのため，入力データの総数が少ないときは特に有効である．通常，深層学習では一度にすべての入力データを学習させるのではなく，データローダーを作成し，そこから入力データのセット，つまりミニバッチを取り出してモデルに学習させる「ミニバッチ学習」を行う．入力データ全てを学習するまでを「1 エポック」というが，通常エポック数を増やしてより多くのデータに学習させていく．しかし，エポック数を増やすことで，同じ画像データを入力することになるので，過学習の問題が発生してしまう．この問題を防ぐために，画像拡張を行い汎化性能を高めていく．

画像拡張ライブラリには，Albumentations ライブラリを採用する．他の画像拡張ライブラリと比較して各画像拡張手法の処理が速いためである．また，強いブラーをかけるような画像拡張を施し，予測対象が判別しにくい画像にならないように注意する．図 3.1 はラベルが 0 の画像をランダムに 4 つ表示させたものである．実際に確認してみると，画像の向きが定まっていないことがわかる．そのため，回転系の画像拡張が適切であると考えられる．そのため回転系の画像拡張である HolizontalFlip,VerticalFlip, そして RandomRotate90 全てを採用した．それぞれ水平方向，垂直方向，90 度回転させる拡張技術で，これらすべてを用いることで，拡張された画像のバリエーションを増やすことで汎化性を高めることを期待する．他には，標準化のために Normalize，モデル構造が求める画像サイズに変換する Resize，モデル構造を Pytorch ライブラリで扱うために tensor 型に変換する ToTensorV2 を用いる．回転系を利用することで，エポック数を増やしつつ過学習せずに精度向上につなげることができる．また，回転系の画像拡張は二分の一の確率で施している．

図 3.2 と図 3.3 では，0000.png の画像に対し，各画像拡張手法を施した画像を比較している．予測対象が判別できる画像となっている．

3.1.2 StratifiedKFold(K=5)

学習をする上ではデータの分割をする必要がある．訓練データから学習データと検証データに分けることで，最適なハイパーパラメータのチューニングを行うた

図 3.1: [ラベル 0 の画像の一部.

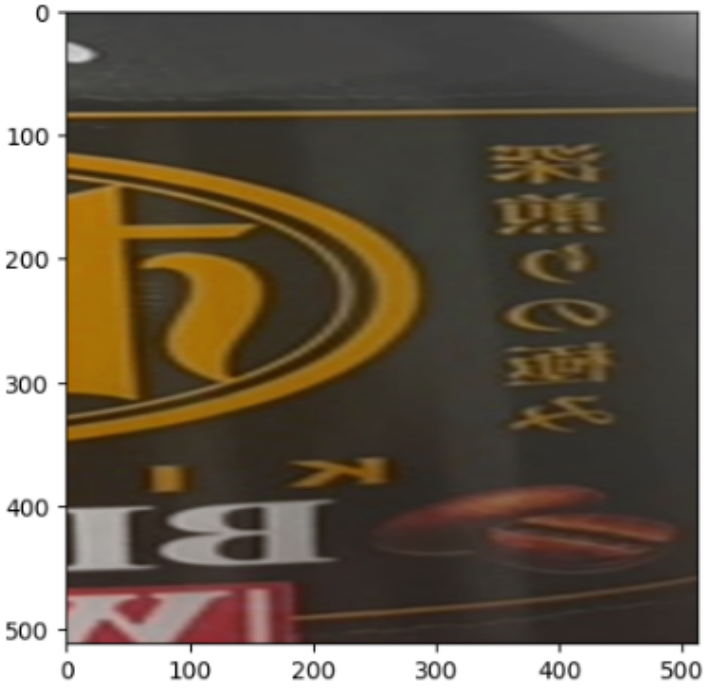

図 3.2: 画像拡張前の 0000.png.

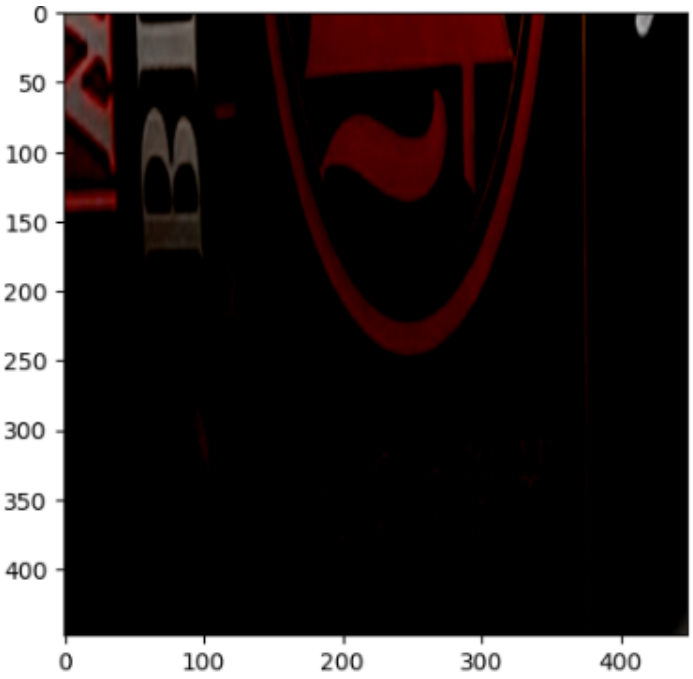

図 3.3: 画像拡張後の 0000.png.

めである．今回，扱ったデータ分割手法は StratifiedKFold(K=5) である．なお，K は分割数である．この分割手法では，全ての訓練データを学習に充てることができるだけでなく，ラベル比率を各分割内で保つことができる利点がある．そして，通常はデータ分割を通して最適なパラメータを見つけた後に，訓練データ全体で学習をし直して 1 つのモデルを作成する．本研究では，ハイパーパラメータを探した上で，各分割内で学習を進めていくことにする．なぜなら，今回は画像拡張を通して別のデータセットを疑似的に作成できるためである．こうすることで，より汎化性能に富んだ 5 つのモデルを学習させることが期待できる．

3.2 ミニバッチ処理

3.2.1 Imbalanced Dataset Sampler

深層学習では，入力画像に対して，画像変形を施してデータセットを作成した後，データローダーを通して，各エポック毎にミニバッチ学習をすることを前章で触れた．ImbalancedDatasetSampler を用いることで，ラベルの比率を考慮してミニバッチのサンプリングを行う．図 3.4 が ImbalancedDatasetSampler の仕組み

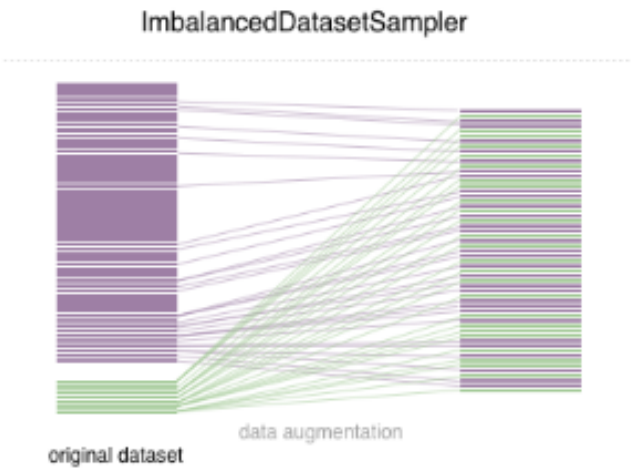

図 3.4: ImbalancedDatasetSampler の仕組み.

を表したものである．これは，多数派ラベルのサンプルを減らす over-sampling と少数派ラベルのサンプルを複製する under-sampling 手法を組み合わせた技術であり，片方のラベルに偏った学習を防ぐことを期待する．

また，ここで調整するハイパーパラメータとして，エポック数とバッチサイズの大きさがある．エポック数とは，トレーニングデータ全体を一通り学習する回数のことである．そのため，エポック数が大きくなるにつれて，モデルは学習して精度が上がるが，大きくしすぎると学習を繰り返すことになるので過学習が起きてしまう．そして，バッチサイズとは，ミニバッチ学習に使用されるサンプルの数のことである．バッチサイズが小さいと 1 つ 1 つのサンプルを細かく学習してしまい，反対に大きいと局所的な特徴しか学習できなくなる問題が発生する．そのため，適切なバッチサイズの大きさに調整することも不可欠である．

3.3 訓練方法

3.3.1 AMP(Automatic Mixed Precision)

ニューラルネットワークのサイズを大きくすると，精度は向上する一方で，モデルの学習に必要なメモリと計算量が増大してしまう．Micikevicius et al. (2018) は，モデルの精度を落とすことなく，またハイパーパラメータを変更することなく，半精度の浮動小数点数を使用してニューラルネットワークを訓練するほう手法

「AMP」を提案している．AMP は，メモリ要件をほぼ半減させつつ，GPU での演算を高速化することに成功している．本研究は，AMP を用いてメモリを節約することで，バッチサイズを大きくすることにメモリを割り当てる．

3.4 転移学習 (ファインチューニング)

モデル構築には，転移学習を利用する．研究で扱うデータでは，訓練データが 2176 枚であり，この分野では比較的に少ないことから，転移学習を採用する．転移学習では，学習済みの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必要がある．そして，学習をする際は学習済みモデルの全ての重みに対して更新は行うのではなく，新たに追加した末端の層の重みだけを更新していく．つまり，学習済みモデルで抽象的な特徴を抽出することに寄与し，新たに追加した末端の層で，学習データに特有な特徴を学習させる狙いがある．学習済みモデルには EVA-02 のラージモデルを利用する．

EVA-02 のラージモデルは，画像分類モデルの 1 つで，Merged-38M と呼ばれる合計 3800 万枚の画像データセットで事前学習した Vision Transformer 系のモデルである．Vision Transformer とは，画像認識や画像処理のためのニューラルネットワークアーキテクチャの一つで，Transformer アーキテクチャを画像処理に適用したものである．画像を小さなパッチに分割して、これらのパッチを Transformer モデルに供給することで，画像全体を一つの系列として扱うように応用している．

Fang et al. (2023) は，ImageNet-1K データセットにおけるファインチューニングによる実験で top-1 accuracy で 90.0 の精度を達成している．これは，公開されている学習済みモデルの中で一番の精度を達成している．そのため，EVA-02 を採用することが精度の高いモデルの構築につながることを期待している．

また，EVA-02 は ImageNet-1K でファインチューニングされている．ImageNet-1K は，約 1419 万枚の 1000 クラスの物体画像データのセットであるため，EVA-02 の最終層は 1000 個のニューロンである．本データでは，2 クラスに分類するので，学習可能な末端の層として，(1000×2) の線形層を追加する．

3.5 重みの更新

3.5.1 Ranger21

Optimizer とは，学習段階において，モデルの各重みの最適な更新方法を決定するアルゴリズムのことをいう．Wright and Demeure (2021) は，既存の Optimizer である AdamW と 8 つのコンポーネント組み合わせた新しい Optimizer，Ranger21 を提案している．結果として，AdamW に比べて検証精度と学習速度を大幅に改善することに成功している．そのため，本研究では，Ranger21 を用いることで，精度向上を期待する．

ここで調整するハイパーパラメータとして，学習率がある．学習率は各更新ステップでのパラメータの更新量を制御する役割を持つ．学習率が小さすぎる場合，最適なパラメータに収束するまでの学習に時間がかかるほか，局所的な最小値にとどまってしまう問題がある．一方で学習率が大きすぎる場合，更新量が大きすぎてしまうことで発散し，最適なパラメータに収束できない問題が発生してしまう．この問題を防ぐために，Ranger21 では，１つのコンポーネントである Warmdown を実装している．学習が後半になるにつれて最適なパラメータに近づくが，この段階で大きな学習率を設定すると，解が発散する可能性がある．これを防ぐために，学習率を徐々に小さくすることで，最適なパラメータに安定した更新量で進むことができるようにしたものが Warmdown である．

3.6 損失関数

3.6.1 CCE(Complement Cross Entropy)

損失関数は，モデルの性能を評価し，最適なモデルの重みを見つけるために最小化され，モデルの予測値と正解の差を計算する重要な関数である．Kima, Leea, and Jeona (2021) は，不均衡なクラス分布によるモデルの性能低下を解決するために，不正確なクラスの出力スコアをほぼ無視することで，不均衡な画像分類の事前予測精度が向上することに成功し，Complement Cross Entropy として提案している．この損失は，少数クラスのサンプルから重要な情報を学習することを容易にし，不均衡な分

表 3.1: 提案モデル

項目	手法
データ前処理	6 つのデータ拡張/StratifiedKFold(K=5)
ミニバッチ処理	ImbalancedDatasetSampler
訓練方法	AMP
転移学習	EVA-02
重みの更新	Ranger21
損失関数	CCE

布においてより正確で頑健な分類結果が得られることを実証している．本研究では，Complement Cross Entropy を用いることでモデルの汎化性能の向上に期待する．

3.7 提案モデル

最終的な提案モデルを表 3.1 にまとめる．

3.8 予測

予測のアプローチでは，StratifiedKFold を用いて学習した 5 つのモデルそれぞれで予測させた後，5 つの結果を平均化することで最終的な予測結果を取るアプローチを行う．ここで，5 つのモデルそれぞれでで予測する際に，データ拡張技術である TTAch を用いることで，より汎化性のある結果が得られることを期待している．

3.8.1 TTAch

TTAch は，Pytorch による予測時に行うデータ拡張技術である．学習時にデータ拡張を行う目的と同じで，TTAch を行う目的は予測画像に対してランダムな変更を施し，汎化性を高めることである．この手法では，予測画像を学習したモデルに一度だけ表示する代わりに，データ拡張した予測画像を数回表示する．そして，対応する各画像の予測値を平均化することで，それを最終的な予測結果とする手法である．図 3.5 は TTAch の構造を表している．Input という 1 つの予測画像に対して，いくつかのデータ拡張を施す．それぞれの画像で予測値を算出して平均化した値を Outout として最終的な予測結果としている．

図 3.5: TTAch の構造.

実際に扱ったデータ拡張は，HolizontalFlip,VerticalFlip, そして Rotate90 である．それぞれ，水平方向，垂直方向，90 度回転させる拡張手法で学習時に施したデータ拡張と同じである．これらを用いることで，画像の汎化性とモデル精度の向上を期待する．

Chapter 4

学習結果と予測

4.1 ハイパーパラメータ

モデル学習していく中で更新される値を重み，またはパラメータと呼ぶが，それとは別にハイパーパラメータと呼ばれる値がある．ハイパーパラメータとは，モデルの挙動を制御するために調整する重要な値である．本研究では，学習・検証実験を繰り返すことで，最適なハイパーパラメータを手動で調整する方法をとる．具体的には，影響が大きいハイパーパラメータから 1 つずつ調整していく方法を取った．エポック数，学習率，バッチサイズの大きさの順で，それぞれ過学習が起きたら，値を小さくして学習・検証を繰り返す．

バッチサイズの大きさが 17 で学習率が 6×10^{-5} の条件下でエポック数が 2 の場合，検証スコアの平均は 0.9558，Public データにおけるスコアは 0.8916 に対し，エポック数が 3 の場合，それぞれ 0.9704，0.8866 であった．学習回数を増やしたにもかかわらず，Public データにおけるスコアが下がってしまっているため，過学習が起きている可能性が高いことがわかった．そのため，エポック数は 2 に決定した．このように，検証スコア，または Public データにおけるスコアを通して過学習が起きていないか確認していきハイパーパラメータを調整した．以下，各ハイパーパラメータの値を表 4.1 にまとめた．

表 4.1: 提案するハイパーパラメータの値

エポック数：	2
バッチサイズの大きさ：	17
学習率：	6×10^{-5} から 3×10^{-5} に Warmdown

表 4.2: エポックによる AUC の変化

Fold	Train Auc		Valid Auc	
	エポック 1	エポック 2	エポック 1	エポック 2
0	0.7800	0.9380	0.8990	0.9620
1	0.8200	0..9620	0.9142	0.9667
2	0.8175	0.9510	0.8946	0.9716
3	0.8155	0.9642	0.8978	0.9702
4	0.8127	0.9587	0.8837	0.9379

なお，学習率は，初期段階では 6×10^{-5}，学習 72 パーセント地点で 3×10^{-5} に Warmdown させている．

4.2 検証結果

検証結果について，各 Fold 毎に学習データ，検証データに対する AUC をそれぞれ Train AUC,Valid AUC としてエポック 1 からエポック 2 への変化を表 4.2 に記載する．

どの Fold においても，Train AUC,Valid AUC ともにスコアが伸びていることから，過学習を防いだモデルを構築できているといえる．また，図 4.1 では訓練データにおける混合行列を示している．モデルがラベル 0 を正しく 0 と予測する真陽性は 900，反対にできなかった場合の偽陽性は 94 つである．モデルがラベル 1 を正しく 1 と予測する真陰性は 1047, 反対にできなかった場合である偽陰性は 135 つである．どちらかのラベルに予測が偏っているのではなく，ともに真陽性と真陰性が多く，偽陽性と偽陰性が少ないことから，手元のデータ，つまり学習データと検証データにおいて過学習せずに汎化性能に富んだモデル性能を達成することに成功している．

次に，テストデータに対する結果を見ていく．コンペティション開催中に確認できる Public データにおける AUC スコアは 0.888，終了後の Private データにおける AUC スコアは 0.9052 で参加者 640 人の中で 2 位の精度を達成することに成功している．コンペティション期間中に確認できない Private データにおける AUC スコアが Public データに劣らず伸びていることから，高い汎化性能を持つモデルを構築できていることがわかる．

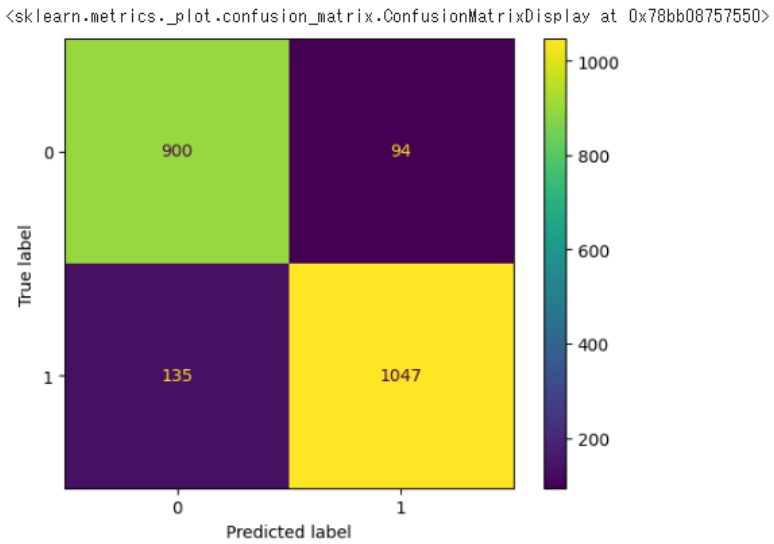

図 4.1: 訓練データにおける混合行列.

表 4.3: ImbalancedDatasetSampler あり

検証スコア (平均)	0.9596
Public データにおけるスコア	0.8880
Private データにおけるスコア	0.9050

4.3 ImbalancedDatasetSampler 無しとの比較

モデル提案の中でもミニバッチ処理に ImbalancedDatasetSampler を用いることが，高い汎化性能を発揮する．表 4.3，表 4.4 にそれぞれ ImbalancedDatasetSampler を用いた場合，用いない場合の AUC スコアをまとめた．コンペティション終了後に確認できる Private データにおけるスコアでは，ImbalancedDatasetSampler を用いることで，0.8999 から 0.9050 へとスコアを 0.0051 伸ばすことができた．実際，参加したコンペティションでは，ImbalancedDatasetSampler を用いることで，他の参加者に差をつけることができ，最終 10 位の精度から最終 2 位の精度まで更新できたことからも，その効果を知ることができる．

表 4.4: ImbalancedDatasetSampler なし

検証スコア (平均)	0.9558
Public データにおけるスコア	0.8916
Private データにおけるスコア	0.8999

表 4.5: Convnext で用いたハイパーパラメータの値

エポック数：	3
バッチサイズの大きさ：	8
学習率：	7×10^{-5} から 3×10^{-5} に Warmdown

表 4.6: Convnext におけるスコア

検証スコア (平均)	0.9583
Public データにおけるスコア	0.8762
Private データにおけるスコア	0.8861

4.4 CNN モデルとの比較

CNN モデルの転移学習には，Convnext を用いた．調整したハイパーパラメータと各スコアは表 4.5 と表 4.6 の通りである．

本研究でも，先行研究同様に CNN 系のモデルより Vision Transformer 系のモデルの転移学習の方が有効であった．

Chapter 5

まとめ

本研究では，簡単な画像分類モデルでは，人間の視覚を通した判別に劣ってしまい活用が難しい問題に触れた．この背景から，食料品の工場向けのデータ分析に注目し，精度だけではなく，実用性を考え，汎化性能にも重点を置いたモデルを提案することを目的とした．そのために，データの前処理，ミニバッチ処理，訓練方法，転移学習，重みの更新，そして損失関数の項目でモデルを考え，手法を提案した．

本研究で提案しているアプローチを取ることで，AUC スコアが 0.9 という高い精度を保ちつつ，未知な Private データに対しても十分な汎化性能を持つモデルを構築することに成功し，目的が達成された．

謝辞

本研究を進めるにあたり，指導教員であられる作村建紀先生には，大変丁寧で熱心なご指導をいただきました．心より感謝申し上げます．

References

Fang, Y., Sun, Q., Wang, X., Huang, T., Wang, X., & Cao, Y. (2023). Eva-02: A visual representation for neon genesis. *arXiv*, 1-20.

Kima, Y., Leea, Y., & Jeona, M. (2021). Imbalanced image classification with complement cross entropy. *arXiv*, 1-8.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xiem, S. (2022). A convnet for the 2020s. *arXiv*, 1-15.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., ... Wu, H. (2018). Mixed precision training. *arXiv*, 1-12.

Wright, L., & Demeure, N. (2021). Ranger21: a synergistic deep learning optimizer. *arXiv*, 1-10.

吉次研二, & 中本幸一. (2021). 画像分類における cnn と vision transformer の精度比較. *2021* **年情報処理学会関西支部支部大会**, 1-4.

石田直也. (2021). 不均衡データの取り扱いに着目した深層学習による運転者の眠気推定. **大学院研究年報理工学研究科編**, 1-4.

Appendices

Appendix A

ソースコード

Listing A.1: test

```
!python -m pip install git+https://github.com/lessw2020/Ranger21.
    git

#モジュールインストール
!pip install torchsampler
!pip install timm
!pip install ttach

#必要なライブラリのインポート
from ranger21 import Ranger21
import ttach as tta
from torchsampler import ImbalancedDatasetSampler
import timm
import os, shutil
import re, gc, sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from glob import glob

import warnings, random
import cv2

warnings.filterwarnings("ignore")

from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import StratifiedShuffleSplit

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torch.optim import lr_scheduler

```

```
import torchvision
from torchvision import transforms
import torchvision.models as models
from torch.cuda.amp import GradScaler

import yaml
from tqdm import tqdm
import time
import copy
from collections import defaultdict

from sklearn.model_selection import train_test_split
#ドライブのマウント
from google.colab import drive
drive.mount('/content/drive')

#環境・バージョンの確認
!nvidia-smi

#train.zip, test.が格納されている場所から上にファイルをコピー zipcontent
#上のディレクトリ colab(上 content)にファイルを移動し解凍を行うと、画像の読み込みが速くなるため zip
if "train.zip" not in os.listdir("/content/"):
    shutil.copy("/content/drive/MyDrive/comp/data/train.zip",
                "/content/train.zip")
if "train" not in os.listdir("/content/"):
    !unzip /content/train.zip

if "test.zip" not in os.listdir("/content/"):
    shutil.copy("/content/drive/MyDrive/comp/data/test.zip",
                "/content/test.zip")
if "test" not in os.listdir("/content/"):
    !unzip /content/test.zip

#ディレクトリの移動
cd "/content/drive/MyDrive/comp"

#自作関数#####
def get_logger(filename):
    from logging import getLogger, INFO, StreamHandler,
        FileHandler, Formatter
    logger = getLogger(__name__)
    logger.setLevel(INFO)
    handler2 = FileHandler(filename=f"{filename}.log")
    handler2.setFormatter(Formatter("%(message)s"))
    logger.addHandler(handler2)
    return logger

#再現性を出すため
def worker_init_fn(worker_id):
    torch.manual_seed(worker_id)
    random.seed(worker_id)
```

```
86      np.random.seed(worker_id)
87      torch.cuda.manual_seed(worker_id)
88      os.environ['PYTHONHASHSEED'] = str(worker_id)
89
90  def set_seed(seed=42):
91      '''Sets the seed of the entire notebook so results are the
            same every time we run.
92      This is for REPRODUCIBILITY.'''
93      random.seed(seed)
94      np.random.seed(seed)
95      torch.manual_seed(seed)
96      torch.cuda.manual_seed(seed)
97      torch.cuda.manual_seed_all(seed)
98      # When running on the CuDNN backend, two further options
           must be set
99      torch.backends.cudnn.deterministic = True
100     torch.backends.cudnn.benchmark = False
101     # Set a fixed value for the hash seed
102     os.environ['PYTHONHASHSEED'] = str(seed)
103
104 set_seed(seed=42)
105
106 #データの読み込み・確認2.
107 data = pd.read_csv("data/train␣(4).csv")
108 data.head()
109
110 #0000.の画像を確認png
111 image = cv2.imread(f"/content/train/0000.png")
112 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
113 plt.imshow(image)
114 plt.show()
115
116 #画像拡張の設定
117 import albumentations as albu
118 from albumentations.pytorch import transforms as AT
119
120 image_transform = albu.Compose([
121     albu.Resize(size[0],size[1]),
122  # albu.Crop(x_max=448,y_max=448),
123  # albu.CenterCrop(440,440),
124  # albu.Resize(size[0],size[1]),
125  # albu.Cutout(),
126      albu.HorizontalFlip(p=0.5),
127     albu.RandomRotate90(p=0.5),
128     albu.VerticalFlip(p=0.5),
129    # albu.RandomBrightnessContrast(),
130    # albu.RandomGamma(gamma_limit=(85, 115)),
131      #albu.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.10,
           rotate_limit=90, p=0.5),
132     albu.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
133 ])
134
135 #画像拡張後の0000.の確認png
```

```
136 transformed = image_transform(image=image)
137 transformed_image = transformed["image"]
138 plt.imshow(transformed_image)
139 plt.show()
140
141 #ラベルがの画像をランダムにつ表示して画像の特徴を確認04
142 data_0 = data[data["label"] == 0].sample(n=4).reset_index(drop=
        True)
143 fig, axes = plt.subplots(2, 2, figsize=(6,6), tight_layout=True)
144 for i , d in data_0.iterrows():
145   image_name = d["image_name"]
146   image = cv2.imread(f"/content/train/{image_name}")
147   image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
148   axes[i//2, i%2].imshow(image)
149   axes[i//2, i%2].set_axis_off()
150   axes[i//2, i%2].set_title(image_name)
151 plt.show()
152
153
154 #ハイパーパラメータの設定
155
156 #エポック数
157 EPOCHS = 2
158 #学習率
159 LR =6e-5
160 #バッジサイズ
161 TRAIN_BATCH_SIZE, VALID_BATCH_SIZE =17,17
162 #画像サイズ
163 size = (448,448)
164 #乱数777
165 seeds = 777
166
167
168 #を行う関数stratifiedkfold
169 def create_folds(data, num_splits, seed):
170     data["kfold"] = -1
171
172     mskf = StratifiedKFold(n_splits=num_splits, shuffle=True,
            random_state=seed)
173     labels = ["label"]
174     data_labels = data[labels].values
175
176     for f, (t_, v_) in enumerate(mskf.split(data, data_labels)):
177         data.loc[v_, "kfold"] = f
178
179     return data
180 train = create_folds(data, num_splits=5, seed=seeds)
181 print("Folds␣created␣successfully")
182
183 train.head()
184
185
186 #データセットを作成するクラス
```

```
187 class Custom_Dataset(Dataset):
188     def __init__(self, df, transform, data_type):
189         self.df = df
190         self.data_type = data_type
191         self.images = []
192 
193         if self.data_type == "train":
194             self.image_paths = df['image_name']
195             self.labels = df['label']
196         if self.data_type == "test":
197             self.image_paths = df[0]
198 
199         self.transform= transform
200 
201         # Load images into RAM beforehand
202         for image_path in self.image_paths:
203             if self.data_type == "train":
204                 image = cv2.imread(f"/content/train/{image_path}"
                        )
205             if self.data_type == "test":
206                 image = cv2.imread(f"/content/test/{image_path}")
207 
208             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
209             self.images.append(image)
210 
211     def __len__(self):
212         return len(self.df)
213     def get_labels(self):
214       return self.labels
215     def __getitem__(self, index: int):
216         image = self.images[index]
217 
218         image = self.transform(image=image)["image"]
219 
220         if self.data_type == "train":
221             label = self.labels[index]
222             label = torch.tensor(label, dtype=torch.long)
223             return image, label
224 
225         if self.data_type == "test":
226             return image
227 
228 
229 #各エポックを学習させる関数
230 def train_one_epoch(model, optimizer, train_loader, device, epoch
        ):
231     model.train()
232     dataset_size = 0
233     running_loss = 0.0
234     running_score = []
235     running_score_y = []
236     scaler = torch.cuda.amp.GradScaler(enabled=True)
237 
```

```
238     train_loss = []
239     bar = tqdm(enumerate(train_loader), total=len(train_loader))
240     for step, (images, targets) in bar:
241       images = images.to(device)
242       targets = targets.to(device)
243       optimizer.zero_grad()
244       batch_size = targets.size(0)
245       with torch.autocast(device_type='cuda', dtype=torch.float16)
              :
246           outputs = model(images)
247           loss = criterion(outputs, targets)
248           scaler.scale(loss).backward()
249           scaler.step(optimizer)
250           scaler.update()
251           train_loss.append(loss.item())
252       #train_loss.append(loss.item())
253       running_loss += (loss.item() * batch_size)
254       dataset_size += batch_size
255
256       epoch_loss = running_loss / dataset_size
257
258       running_score.append(outputs.detach().cpu().numpy())
259       running_score_y.append(targets.detach().cpu().numpy())
260
261       score = get_score(running_score_y, running_score)
262
263       bar.set_postfix(Epoch=epoch, Train_Loss=epoch_loss,
264                       Train_Acc=score[0],
265                       Train_Auc=score[1],
266                       LR=optimizer.param_groups[0]['lr']
267                       )
268     gc.collect()
269     return epoch_loss, score
270
271
272 #検証データで評価するための関数
273 @torch.no_grad()
274 def valid_one_epoch(model, optimizer, valid_loader, epoch):
275     model.eval()
276
277     dataset_size = 0
278     running_loss = 0.0
279     preds = []
280     valid_targets = []
281     softmax = nn.Softmax()
282
283     bar = tqdm(enumerate(valid_loader), total=len(valid_loader))
284     for step, (images, targets) in enumerate(valid_loader):
285       images = images.to(device)
286       targets = targets.to(device)
287       batch_size = targets.size(0)
288       with torch.no_grad():
289         outputs = model(images)
```

```
290         predict = outputs.softmax(dim=1)
291         loss = criterion(outputs, targets)
292 
293       running_loss += (loss.item() * batch_size)
294       dataset_size += batch_size
295 
296       epoch_loss = running_loss / dataset_size
297 
298       preds.append(predict.detach().cpu().numpy())
299       valid_targets.append(targets.detach().cpu().numpy())
300 
301       if len(set(np.concatenate(valid_targets))) == 1:
302           continue
303       score = get_score(valid_targets, preds)
304 
305       bar.set_postfix(Epoch=epoch, Valid_Loss=epoch_loss,
306                       Valid_Acc=score[0],
307                       Valid_Auc=score[1],
308                       LR=optimizer.param_groups[0]['lr'])
309 
310     return epoch_loss, preds, valid_targets, score
311 
312 #各フォールドの学習ステップ関数
313 def one_fold(model, optimizer, device, num_epochs, fold):
314 
315     if torch.cuda.is_available():
316         print("[INFO] Using GPU: {}\n".format(torch.cuda.get_
                device_name()))
317 
318     start = time.time()
319     best_model_wts = copy.deepcopy(model.state_dict())
320     best_epoch_score = np.inf
321     best_prediction = None
322 
323     best_score = -np.inf
324     for epoch in range(1, 1+num_epochs):
325       train_epoch_loss, train_score = train_one_epoch(model,
              optimizer,
326                                                       train_loader=train_
                                                          loader,
327                                                       device=device, epoch=
                                                          epoch)
328 
329       train_acc, train_auc = train_score
330 
331       val_epoch_loss, predictions, valid_targets, valid_score =
              valid_one_epoch(
332                                                                    mod
333                                                                    opt
```

```
                                                                              val
                                                                              epo
      valid_acc, valid_auc = valid_score

      print(f'Epoch {epoch} - avg_train_loss: {train_epoch_loss:.4f}  avg_val_loss: {val_epoch_loss:.4f}')
      print(f'Epoch {epoch} - Train Acc: {train_acc:.4f}  Train Auc: {train_auc:.4f}  Valid Acc: {valid_acc:.4f}  Valid Auc: {valid_auc:.4f}')

      if valid_auc >= best_score:
        best_score = valid_auc

        print(f"Validation Score Improved ({best_epoch_score} ---> {valid_auc})")
        best_epoch_score = valid_auc
        best_model_wts = copy.deepcopy(model.state_dict())
        # PATH = f"Score-Fold-{fold}.bin"
        PATH = "./" + f"Score-Fold-{fold}.bin"
        torch.save(model.state_dict(), PATH)
        # Save a model file from the current directory
        print(f"Model Saved")

        best_prediction = np.concatenate(predictions, axis=0)[:,1]

    end = time.time()
    time_elapsed = end - start

    print('Training complete in {:.0f}h {:.0f}m {:.0f}s'.format(
        time_elapsed // 3600, (time_elapsed % 3600) // 60, (time_elapsed % 3600) % 60))
    print("Best Score: {:.4f}".format(best_epoch_score))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, best_prediction, valid_targets

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')#setting gpu

#モデル作成
class create_model(nn.Module):
  def __init__(self):
    super().__init__()
```

```
372     self.timm = timm.create_model("eva02_large_patch14_448.mim_
            m38m_ft_in22k_in1k", pretrained=True)
373     self.fc = nn.Linear(1000, 2)
374   def forward(self, x):
375     x = self.timm(x)
376     x = self.fc(x)
377     return x
378 #損失関数
379 class CCE(nn.Module):
380     def __init__(self, device, balancing_factor=1):
381         super(CCE, self).__init__()
382         self.nll_loss = nn.NLLLoss()
383         self.device = device # {'cpu', 'cuda:0', 'cuda:1', ...}
384         self.balancing_factor = balancing_factor
385 #順伝播
386     def forward(self, yHat, y):
387         # Note: yHat.shape[1] <=> number of classes
388         batch_size = len(y)
389         # cross entropy
390         cross_entropy = self.nll_loss(F.log_softmax(yHat, dim=1),
                y)
391         # complement entropy
392         yHat = F.softmax(yHat, dim=1)
393         Yg = yHat.gather(dim=1, index=torch.unsqueeze(y, 1))
394         Px = yHat / (1 - Yg) + 1e-7
395         Px_log = torch.log(Px + 1e-10)
396         y_zerohot = torch.ones(batch_size, yHat.shape[1]).scatter_
                (
397             1, y.view(batch_size, 1).data.cpu(), 0)
398         output = Px * Px_log * y_zerohot.to(device=self.device)
399         complement_entropy = torch.sum(output) / (float(batch_size
                ) * float(yHat.shape[1]))
400 
401         return cross_entropy - self.balancing_factor * complement_
                entropy
402 
403 #評価指標
404 def get_score(y_trues, y_preds):
405 
406     predict_list, targets_list = np.concatenate(y_preds, axis=0),
            np.concatenate(y_trues)
407     predict_list_proba = predict_list.copy()[:, 1]
408     predict_list = predict_list.argmax(axis=1)
409 
410     accuracy = accuracy_score(predict_list, targets_list)
411     auc_score = roc_auc_score(targets_list, predict_list_proba)
412 
413     return (accuracy, auc_score)
414 
415 #データローダー作成
416 def prepare_loaders(train, image_transform, fold):
417     df_train = train[train.kfold != fold].reset_index(drop=True)
418     df_valid = train[train.kfold == fold].reset_index(drop=True)
```

```
419
420     train_dataset = Custom_Dataset(df_train, image_transform, data
            _type="train")
421     valid_dataset = Custom_Dataset(df_valid, valid_image_transform
            , data_type="train")
422     train_loader = DataLoader(train_dataset, batch_size=TRAIN_
            BATCH_SIZE,
423                               worker_init_fn=worker_init_fn(seeds),
                                      sampler=ImbalancedDatasetSampler(
                                      dataset=train_dataset),
424                               num_workers=4,
425                               shuffle=False, pin_memory=True, drop_
                                      last=True)
426     valid_loader = DataLoader(valid_dataset, batch_size=VALID_
            BATCH_SIZE,
427                               num_workers=4,
428                               shuffle=False, pin_memory=True)
429
430     return train_loader, valid_loader
431
432 #学習・検証の実行
433 train_copy = train.copy()
434 #LOGGER.info(ARGS)
435 for fold in range(0, 5):
436     print(f"======␣Fold:␣{fold}␣======")
437    # LOGGER.info(f"========== fold: {fold} training
           ==========")
438
439     # Create Dataloaders
440     train_loader, valid_loader = prepare_loaders(train=train,
            image_transform=image_transform, fold=fold)
441     model = create_model()
442     model = model.to(device)
443
444     #損失関数・最適化関数の定義
445     optimizer = Ranger21(model.parameters(), lr=LR, num_batches_
            per_epoch=len(train_loader), num_epochs=EPOCHS, warmdown_
            min_lr=3e-5)
446     criterion = CCE(device).to(device)
447     model, predictions, targets = one_fold(model, optimizer,
            device=device, num_epochs=EPOCHS, fold=fold)
448
449     train_copy.loc[train_copy[train_copy.kfold == fold].index, "
            oof"] = predictions
450
451     del model, train_loader, valid_loader
452     _ = gc.collect()
453     torch.cuda.empty_cache()
454     print()
455
456 scores = roc_auc_score(train_copy["label"].values, train_copy["
        oof"].values)
457 print("CV")
```

```
458 print("scores")
459
460 #sample_submit.を読み込む csv
461 submit = pd.read_csv("data/sample_submit.csv", header=None)
462 submit.head()
463
464 #テストデータ用のデータローダー
465 class Custom_Test_Dataset(torch.utils.data.Dataset):
466     def __init__(self, df, transform):
467         self.df = df
468         self.image_paths = df[0]
469         self.transform= transform
470
471     def __len__(self):
472         return len(self.df)
473
474     def __getitem__(self, index: int):
475         image_path = self.image_paths[index]
476         #画像読み込み
477         image = cv2.imread(f"/content/test/{image_path}")
478         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
479
480         image = self.transform(image=image)["image"]
481
482         return image
483
484 test_data = Custom_Test_Dataset(submit, image_transform)
485 test_loader = DataLoader(test_data, batch_size=32, shuffle=False,
      num_workers=4)
486
487 #setting tta
488 transforms = tta.Compose(
489     [
490         tta.HorizontalFlip(),
491         tta.VerticalFlip(),
492         tta.Rotate90(angles=[0, 90])
493     ]
494 )
495
496 #推論コード
497 def inference(model_paths, dataloader, device):
498     final_preds = []
499     for i, path in enumerate(model_paths):
500         model = create_model()
501         model = model.to(device)
502
503         #学習済みモデルの読み込み
504         model.load_state_dict(torch.load(path))
505         model.eval()
506         tta_model = tta.ClassificationTTAWrapper(model, transforms
              )
507         print(f"Getting␣predictions␣for␣model␣{i+1}")
508         preds = valid_fn(tta_model, dataloader, device)
```

```
509         final_preds.append(preds)
510
511     final_preds = np.array(final_preds)
512     final_preds = np.mean(final_preds, axis=0)
513     return final_preds
514
515 #モデルパラメータ保存先
516 MODEL_PATHS = [
517     f"./Score-Fold-{i}.bin" for i in range(5)
518 ]
519
520 #推論
521 predict_list = inference(MODEL_PATHS, test_loader, device)
522 submit[1] = predict_list
523 submit.head()
524
525 save to csvfile
526 submit.to_csv(f"./ttasubmission_CV{scores:.4f}.csv", index=False
    , header=None)
```