令和７年度「地方やデジタル分野における専修学校理系転換等推進事業」

人工知能（ＡＩ）技術応用教材資料

本人工知能（ＡＩ）技術応用教材資料は、文部科学省の教育政策推進事業委託費による委託事業として、一般社団法人全国専門学校情報教育協会が実施した令和７年度「地方やデジタル分野における専修学校理系転換等推進事業」の成果物です。

情報成長分野の教育プログラム整備と教員育成による学科転換・新設推進事業

目　次

人工知能（ＡＩ）技術応用_導入編

人工知能（AI）技術応用 実習ガイド

人工知能（AI）技術応用

この教材は、『人工知能（AI）技術基礎』で学んだ理論を、実際に手を動かして確認するための**実践教材**です。基礎教材では全7章で構成された内容で、人工知能の理論や仕組みを体系的に学習しました。『**人工知能（AI）技術応用**』では、それらの理論を３つの実習で、機械学習・画像認識・自然言語処理について、**「理解」を「実践」に変える**ことを目指します。

『**人工知能(AI)技術基礎**』（理論・知識の学習）
- 第1章 人工知能の基礎
- 第2章 機械学習の基礎
- 第3章 ディープラーニングの基礎
- 第4章 機械学習と統計学
- 第5章 相関関係と因果関係
- 第6章 データマイニングの基礎
- 第7章 AIとビッグデータ

『人工知能（AI）技術応用』
（実践・体験による理解の深化）
- 実習１ 機械学習で手書き数字を認識
- 実習２ 画像認識AIで犬と猫を分類
- 実習３ 自然言語処理で感情分析AIを作る

学習のゴール

人工知能（AI）技術応用

- ✓ AIの「基礎知識」を「使えるスキル」に変える
- ✓ AI開発の基本的なプロセス（データ準備→学習→評価→応用）を体験する
- ✓ 3つの異なる分野（機械学習、画像、自然言語）のAI開発に挑戦する
- ▶ これらによって幅広い実践力を身につける

実習構成

人工知能（AI）技術応用

3つのステップで学ぶ AI実践

実習 1 機械学習	実習 2 画像認識	実習 3 自然言語処理
手書き数字認識 (MNIST) 決定木, KNN, ロジスティック回帰	犬と猫の分類 （CNN, 転移学習）	感情分析 （RNN/LSTM, BERT）

実習で使用する主なツール・環境

人工知能(AI)技術応用

Python:
AI開発の標準言語
Google Colaboratory (Colab): (推奨)
インストール不要のPython実行環境。無料でGPUも使えます
scikit-learn: (実習１で使用)
伝統的な機械学習のためのライブラリ
TensorFlow / Keras: (実習２,３で使用)
ディープラーニングのための主要ライブラリ
Hugging Face Transformers: (実習３で使用)
最新のNLPモデル（BERTなど）を簡単に使うためのライブラリ

ポイント
すべて無料で利用できるツール・環境を選定しています

実習の進め方

人工知能(AI)技術応用

実習	： Part１～４
導入ガイド	： 実習の概要、環境準備、データの入手方法等
オンライン講義（動画）	： 各Partの解説とデモ
ワークブック	： 実際のコードが記載、考察を記入する教材

実習ステップ

人工知能（AI）技術応用

準備	**導入ガイド**を読み、環境を準備する
視聴	**オンライン講義**（動画）を視聴する
実践	**ワークブック**のタスクに沿って、自分でコード実行する
記録	**ワークブック**に実行結果や気づきを記録する
考察	**考察問題**に挑戦し、自分の言葉でまとめる

人工知能（AI）技術応用

「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」

実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る

NEXT

人工知能（AI）技術応用

「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」

- **実習 1　機械学習で手書き数字を認識**
- 実習 2　画像認識AIで犬と猫を分類
- 実習 3　自然言語処理で感情分析AIを作る

人工知能（ＡＩ）技術応用_実習１

Part1


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習１　機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

実習１ 機械学習で手書き数字を認識

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → 実習３ 自然言語処理で感情分析AIを作る

☞ **Part１：環境準備とデータ確認**

Part２：3つのアルゴリズム実装

Part３：結果の比較と評価

Part４：実データでの検証

Part １：環境構築とデータ確認 の達成目標

実習１ 機械学習で手書き数字を認識

- Google Colabの基本操作ができる
- MNISTデータセットを読み込み、構造を確認できる
- 訓練データとテストデータの分割の目的を理解できる
- 手書き数字の画像を可視化し、データの特徴を理解できる
- 各数字のデータ分布をグラフで確認できる

Google Colaboratory

実習１ 機械学習で手書き数字を認識 Part１

Google Colab とは？

- ブラウザで Python が実行できる
- インストール不要
- 無料で使える
- Googleアカウントがあれば OK

Google Colabノートブック準備

実習１ 機械学習で手書き数字を認識 Part１

Google Colabにアクセスして、新しいノートブックを作成

1. colab.research.google.com にアクセス
2. 「ノートブックを新規作成」をクリック
3. ノートブックに名前をつける

（例：AI実習1_Part1_自分の名前）

ポイント
Googleアカウントでログインする必要があります

Part１：環境準備とデータ確認　人工知能（AI）技術応用 実習１

Google Colab 画面構成

実習１ 機械学習で手書き数字を認識 Part１

Google Colab の画面構成を理解する

- セル：コードを書く場所
- 実行ボタン：▶ マーク （Shift＋Enter でもOK）
- ファイル名の変更方法

Google Colab 動作確認

実習１ 機械学習で手書き数字を認識 Part１

Pythonを動かしてみよう

```
print("Hello, Machine Learning!")
print("2 + 3 =", 2 + 3)
print("10 * 5 =", 10 * 5)
```

▶ 期待される出力：

```
Hello, Machine Learning!
2 + 3 = 5
10 * 5 = 50
```

ライブラリの読み込み

実習１ 機械学習で手書き数字を認識 Part１

Python の便利な道具ライブラリをインポート

AI開発で頻繁に利用される標準的なライブラリを読み込みます

```
# 必要なライブラリをインポート
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import
train_test_split
```


MNIST

実習１ 機械学習で手書き数字を認識 Part１

MNISTデータセットとは？

- **MNIST ：Modified National Institute of Standards and Technology**
  - **手書き数字のデータベース**
  - 70,000枚の手書き数字画像
  - 28×28ピクセルの白黒画像
  - 0から9までの数字
  - 機械学習の「練習問題」として世界中で使用

MNISTデータ準備

実習１ 機械学習で手書き数字を認識 Part１

データをロードし、その構造を確認します

７万枚の手書き数字データ（MNIST）を読み込みます

- データをロードし、その構造（shape）を確認します

```
# MNISTデータのロード（初回のみ時間がかかる場合があります）
X, y = fetch_openml('mnist_784', version=1, return_X_y=True,
as_frame=False, parser='auto')

# データの構造（形状: shape）を確認
print("特徴量データ（X）の形状:", X.shape)
print("正解ラベルデータ（y）の形状:", y.shape)
```


データ準備

実習１ 機械学習で手書き数字を認識 Part１

訓練データとテストデータ

学習用の「訓練データ」6万枚と、確認用の「テストデータ」1万枚に分けます

種類	枚数	用途
訓練データ	60,000枚	AIに教える
テストデータ	10,000枚	実力を確認

```
# 訓練用6万枚、テスト用1万枚に分割
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000,
train_size=60000, random_state=42)

print("訓練データの数:", len(X_train))
print("テストデータの数:", len(X_test))
```

データ可視化

実習１ 機械学習で手書き数字を認識 Part１

データを見てみよう

実際のデータを画像として16枚表示し、中身を確認します

4×4のグリッドで16枚の数字を表示

- ▶ 各画像の上にラベル（正解）を表示
- ▶ 観察ポイント：
  - ✓ 同じ数字でも書き方が違う
  - ✓ 読みにくい数字もある
  - ✓ 7と1、3と8など似ている数字

```
plt.figure(figsize=(10, 10))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    # 784個の数値を28x28ピクセルの形に戻して表示
    plt.imshow(X_train[i].reshape(28,
28), cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```


データ可視化

実習１ 機械学習で手書き数字を認識 Part１

データを見てみよう

各数字が訓練データに何枚ずつ含まれているか棒グラフで確認します

- ▶ 各数字（0-9）の枚数を棒グラフで表示
- ➢ どの数字もほぼ同じ枚数
- ➢ バランスの良いデータセット

```
import collections
# 各ラベルの出現回数をカウント
counts = collections.Counter(y_train)
counts = dict(sorted(counts.items()))
# 棒グラフで表示
plt.bar(counts.keys(), counts.values())
plt.xlabel('Digit')
plt.ylabel('Count')
plt.title('Distribution of digits in training set')
plt.show()plt.show()
```

実習１ Part１のまとめ

実習１ 機械学習で手書き数字を認識 Part１

今回やったこと：

- ✅ Google Colabでプログラムを実行
- ✅ MNISTデータ（70,000枚）を読み込み
- ✅ データを訓練用とテスト用に分割
- ✅ 手書き数字を表示して観察
- ✅ データの統計情報を確認

NEXT

実習１ Part２：3つのアルゴリズム実装

実習１ 機械学習で手書き数字を認識

実習１ Part２では以下を行います

- 🤖 ３つのアルゴリズムを実装
- 📊 決定木
- 📊 k近傍法
- 📊 ロジスティック回帰
- ⚡ それぞれの特徴を比較


人工知能（AI）技術応用
実習１ 機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習１ 機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

完成イメージ

実習１ 機械学習で手書き数字を認識 Part１

何を作る？

手書き数字認識システム

1. 手書きで数字を書く
2. コンピュータが自動で判断
3. 「これは〇です」と表示

人工知能（ＡＩ）技術応用_実習１

Part2


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習１　機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

実習１ 機械学習で手書き数字を認識

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → 実習３ 自然言語処理で感情分析AIを作る

Part１：環境準備とデータ確認

☞ **Part２：3つのアルゴリズム実装**

Part３：結果の比較と評価

Part４：実データでの検証

Part２：3つのアルゴリズム実装 の達成目標

実習１ 機械学習で手書き数字を認識

- ✓ 3つの代表的な機械学習アルゴリズムの特徴を理解する
- ✓ 決定木 (Decision Tree) を実装できる
- ✓ k近傍法 (KNN) を実装できる
- ✓ ロジスティック回帰 を実装できる
- ✓ 各モデルの「学習」と「予測」のプロセスを体験する
- ✓ 各モデルの学習時間や予測結果の違いを観察する

3つのアルゴリズム概要

実習１ 機械学習で手書き数字を認識 Part2

今回試す3つのアルゴリズム

🌳 決定木
考え方
「質問」を繰り返す
例
「20の質問ゲーム」
特徴
理由が分かりやすい

👫 k近傍法 (KNN)
考え方
「多数決」で決める
例
「近くの友達に聞く」
特徴
シンプルで直感的

📈 ロジスティック回帰
考え方
「確率」を計算する
例
「天気予報」
特徴
確率が分かる

タスク1：決定木 (Decision Tree)

実習１ 機械学習で手書き数字を認識 Part2

「質問」を繰り返して分類するモデル

```
# 1. 決定木モデルのインポート
from sklearn.tree import DecisionTreeClassifier
import time
# 2. モデルの作成（質問は最大10回まで）
dt_model = DecisionTreeClassifier(max_depth=10, random_state=42)
# 3. 学習（訓練データで）
start_time = time.time()
dt_model.fit(X_train, y_train)
print(f"学習時間: {time.time() - start_time:.2f}秒")
# 4. 予測（テストデータで）
dt_predictions = dt_model.predict(X_test)
print("予測:", dt_predictions[:10])
print("正解:", y_test[:10].values)
```

タスク2：k近傍法 (KNN)

実習１ 機械学習で手書き数字を認識 Part2

「近くの仲間」の多数決で分類するモデル

```
# 1. KNNモデルのインポート
from sklearn.neighbors import KNeighborsClassifier
# 2. モデルの作成（近くの5人を参考にする）
knn_model = KNeighborsClassifier(n_neighbors=5)
# 3. 学習（データを覚えるだけなので速い）
start_time = time.time()
knn_model.fit(X_train, y_train)
print(f"学習時間: {time.time() - start_time:.2f}秒")
# 4. 予測
knn_predictions = knn_model.predict(X_test)
print("予測:", knn_predictions[:10])
print("正解:", y_test[:10].values)
```


タスク3：ロジスティック回帰

実習１ 機械学習で手書き数字を認識 Part2

「確率」を計算して分類するモデル

```
# 1. ロジスティック回帰モデルのインポート
from sklearn.linear_model import LogisticRegression
# 2. モデルの作成（計算を100回繰り返す）
lr_model = LogisticRegression(max_iter=100,
random_state=42)
# 3. 学習（最適な確率計算ルールを見つけるため時間がかかる）
start_time = time.time()
lr_model.fit(X_train, y_train)
print(f"学習時間: {time.time() - start_time:.2f}秒")
# 4. 予測
lr_predictions = lr_model.predict(X_test)
print("予測:", lr_predictions[:10])
print("正解:", y_test[:10].values)
```

タスク4：比較と考察

実習１ 機械学習で手書き数字を認識 Part2

3つのモデルを比較しよう

☞ ワークブックの比較表に、結果を記録しましょう

アルゴリズム	学習時間（例）	予測（10件中）
決定木	約 2-3 秒	(各自の結果)
k近傍法 (KNN)	約 1-2 秒	(各自の結果)
ロジスティック回帰	約 30-60 秒	(各自の結果)

🤔 考えてみよう
・なぜ学習時間がこんなに違う？ ・最初の10件の予測結果で、一番良かったのは？ ・ どのモデルが一番良さそう？

実習１ Part２のまとめ

実習１ 機械学習で手書き数字を認識 Part2

- ✅ ３つの機械学習アルゴリズム（決定木、KNN、ロジ回帰）を実装した
- ✅ `.fit()` で学習させ、`.predict()` で 予測する流れを学んだ
- ✅ アルゴリズムによって、学習時間や特徴が異なることを体験した

Part 2 ：3つのアルゴリズム実装　人工知能（AI）技術応用 実習１

NEXT

実習１ Part３：結果の比較と評価

実習１ 機械学習で手書き数字を認識

実習１ Part３では以下を行います

- ✓ 10,000件のテストデータで、本当の「正解率」を比較
- ✓ AIが「どんな間違い」をしているか分析する**「混同行列」**
- ✓ **一番優秀なモデルを決定**する

NEXT

人工知能（AI）技術応用

実習１　機械学習で手書き数字を認識

Part１：環境準備とデータ確認

Part２：3つのアルゴリズム実装

☞ **Part３：結果の比較と評価**

Part４：実データでの検証

人工知能（ＡＩ）技術応用_実習１

Part3


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習１　機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

実習１ 機械学習で手書き数字を認識

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → 実習３ 自然言語処理で感情分析AIを作る

Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
☞ **Part３：結果の比較と評価**
Part４：実データでの検証

Part３：結果の比較と評価 の達成目標

実習１ 機械学習で手書き数字を認識

- ✓ テストデータ（10,000件）**全体**でモデルの性能を評価する
- ✓ **正解率** (Accuracy) を計算し、客観的に比較する
- ✓ 比較結果をグラフで分かりやすく**「可視化」**する
- ✓ **混同行列** (Confusion Matrix) の見方を理解する
- ✓ **AIが「どんな間違い」をしているか**パターンを分析する
- ✓ データにもとづき、**最適なモデル**を選定する理由を考察する

タスク1：正解率の計算

実習１ 機械学習で手書き数字を認識 Part３

「10,000件のテストデータのうち、何件正解できたか？」を計算

```
# 正解率の計算
from sklearn.metrics import accuracy_score
# 各モデルの正解率を計算
dt_accuracy = dt_model.score(X_test, y_test)
knn_accuracy = knn_model.score(X_test, y_test)
lr_accuracy = lr_model.score(X_test, y_test)
print(f"決定木: {dt_accuracy*100:.2f}%")
print(f"k近傍法: {knn_accuracy*100:.2f}%")
print(f"ロジスティック回帰: {lr_accuracy*100:.2f}%")
```

ポイント
.score() メソッドを使えば一瞬でできます
Part２の「最初の10件」の結果と比べてどうですか？
10,000件のデータで評価する方が、モデルの「本当の実力」が分かります

タスク2：結果の可視化

実習１ 機械学習で手書き数字を認識 Part３

比較グラフの作成／数字の比較は棒グラフにすると一目瞭然

```
import matplotlib.pyplot as plt

models = ['決定木', 'k近傍法', 'ロジスティック回帰']
accuracies = [dt_accuracy, knn_accuracy, lr_accuracy]

plt.bar(models, accuracies, color=['#2ecc71', '#3498db', '#e74c3c'])
plt.title('3つのアルゴリズムの性能比較')
plt.ylabel('正解率')
plt.ylim(0.7, 1.0) # 70%〜100%の範囲で表示
plt.show()
```

タスク3：混同行列 (Confusion Matrix)

実習１ 機械学習で手書き数字を認識 Part３

AIがどんな間違い方をしたか を分析する表

混同行列の作成と可視化

実習１ 機械学習で手書き数字を認識 Part３

一番性能が良かった「k近傍法」で見る

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# k近傍法モデルの予測結果で混同行列を作成
cm = confusion_matrix(y_test, knn_predictions)
# ヒートマップで可視化
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.xlabel('予測した数字')
plt.ylabel('実際の数字')
plt.title('混同行列 (k近傍法)')
plt.show()
```

考察ポイント
対角線以外のマスで、数字が大きい（色が濃い）場所を探そう

Part３ ：結果の比較と評価　人工知能（AI）技術応用 実習１

実習１ Part３のまとめ

実習１ 機械学習で手書き数字を認識 Part３

- ✅ `.score()` でモデルの「正解率」を計算し、客観的に比較した
- ✅ 棒グラフで性能差を「可視化」した
- ✅ `confusion_matrix` で「間違いのパターン」を分析した
- ✅ 今回のデータでは、k近傍法が最も性能が良いことが分かった

NEXT

実習１ Part４：実データでの検証

実習１ 機械学習で手書き数字を認識

実習１ Part４では以下を行います

- ✓ **「自分で書いた数字」でAIをテスト**します
- ✓ 画像のアップロードと**「前処理」の方法を学びます**
- ✓ 学習したAIは、**自筆の文字を正しく認識**できるか？
- ✓ **実習全体のまとめレポート**を作成


人工知能（AI）技術応用
実習１ 機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習１ 機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

人工知能（ＡＩ）技術応用_実習１

Part4


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習１　機械学習で手書き数字を認識
Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
Part４：実データでの検証
Mirai no tobira

実習１ 機械学習で手書き数字を認識

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → 実習３ 自然言語処理で感情分析AIを作る

Part１：環境準備とデータ確認
Part２：3つのアルゴリズム実装
Part３：結果の比較と評価
☞ **Part４：実データでの検証**

Part４：実データでの検証 の達成目標

実習１ 機械学習で手書き数字を認識

- ✓ 自分で用意した画像（手書き数字）を Google Colabにアップロードできる
- ✓ 画像をAIが認識できる形に「前処理」する方法を学ぶ
- ✓ 学習済みのモデル（Part２,３で作成）を使って、自分の画像を予測させる
- ✓ AIが「なぜ間違えるのか」を考察する
- ✓ 実習１全体の学びを「最終レポート」としてまとめる

タスク1：手書き数字の準備

実習１ 機械学習で手書き数字を認識 Part４

AIにテストさせる「問題」を自分で作る

手順

1. **書く：** 白い紙に、黒いペンで、0～9の数字をはっきりと書く
2. **撮る：** スマホなどで撮影する（影や傾きに注意）
3. **送る：** PCに送信する（メール、クラウドストレージなど）

タスク2：画像の前処理

実習１ 機械学習で手書き数字を認識 Part４

同じ形に揃える

撮影した画像は、AIが学習したMNISTデータと「形」が違うので、
同じ形に揃える必要がある

実行する処理

- ▶ **アップロード**　：Colabに画像をアップロード
- ▶ **読み込み**　：Pythonで画像を読み込む (PILライブラリ)
- ▶ **グレースケール化**：カラー画像を白黒に変換
- ▶ **リサイズ**　：画像を 28×28 ピクセルに縮小
- ▶ **反転**　：「白背景に黒文字」を「黒背景に白文字」（MNISTに合わせる）
- ▶ **正規化**　：0～255の色の濃さを 0～1 の範囲に変換
- ▶ **平坦化**　：28×28の画像を、784個の数字の列に変換

前処理

実習１ 機械学習で手書き数字を認識 Part４

各処理を一括で行う

```
from PIL import Image
import numpy as np
# 1. アップロード（Colabの左サイドバーから手動で）
image_path = "your_image.jpg" # 自分のファイル名に変更
# 2-5. 読込、グレースケール、リサイズ、反転
img = Image.open(image_path).convert('L') # 白黒読込
img_resized = img.resize((28, 28))
img_array = np.array(img_resized)
img_inverted = 255 - img_array # 白黒反転
# 6. 正規化
img_normalized = img_inverted / 255.0
# 7. 平坦化（予測用）
img_for_prediction = img_normalized.flatten().reshape(1, -1)
plt.imshow(img_normalized, cmap='gray') # 確認用表示
```


タスク3：予測実行

実習１ 機械学習で手書き数字を認識 Part４

AIに予測させてみよう

Part３で一番性能が良かったモデル（k近傍法）などで予測させます

```
# 一番良かったモデル（例：knn_model）で予測
my_prediction = knn_model.predict(img_for_prediction)
print(f"AIの予測結果: {my_prediction[0]}")

# （参考）ロジスティック回帰で確率も見る
lr_prob = lr_model.predict_proba(img_for_prediction)
print(f"AIの自信度 （ {lr_model.predict(img_for_prediction)[0]}である確率 ):
{np.max(lr_prob)*100:.1f}%")
```

タスク4：最終レポートの作成

実習１ 機械学習で手書き数字を認識 Part４

実習１の総まとめとして、分析レポートを作成

レポートに含める内容：

1. **分析の目的：** 何をしようとしたか？（手書き数字認識）
2. **使用データと手法：** MNISTデータ、3つのアルゴリズム（決定木, KNN, ロジ回帰）
3. **分析結果（性能比較）：**
   - 各モデルの正解率（棒グラフ）
   - 混同行列からわかった「間違いやすい数字」
   - どのモデルが一番良かったか？
4. **実データ検証：** 自分の手書き数字での結果（成功したか、失敗したか）
5. **考察と学び：**
   - なぜ性能に差が出た？ なぜ自分の字は間違えた？
   - この実習で学んだこと、難しかったこと、面白かったこと

実習１ まとめ

実習１ 機械学習で手書き数字を認識

今実習で達成したこと

- ✅ データ分析環境 (Colab) を使いこなした
- ✅ 70,000件のデータを準備・分析した
- ✅ ３種類のAIモデルを実装・学習・評価した
- ✅ 自分のデータでAIをテストした
- ✅ 機械学習の基本プロセスをすべて体験した

実習１で身につけたスキル

実習１ 機械学習で手書き数字を認識

- ✓ **Google Colab** と **Python**、**Scikit-learn** を使った分析環境の構築・実行スキル
- ✓ AI開発における**「訓練データ」と「テストデータ」**の重要性の理解
- ✓ **決定木、KNN、ロジスティック回帰**という３つの機械学習モデルの実装スキル
- ✓ **.fit()（学習）**と**.predict()（予測）**という基本操作の習得
- ✓ **「正解率」**や**「混同行列」**を使った、AIの性能評価スキル
- ✓ AIの限界と、**学習データ（MNIST）**と、**実データ**（自分の文字）の違いの考察力

NEXT

実習２ 画像認識AIで犬と猫を分類

実習１ 機械学習で手書き数字を認識

実習２では以下を行います

- ☞ 手書き数字より難しい「犬と猫」のカラー写真に挑戦
- ☞ より強力なAI技術**「ディープラーニング(CNN)」**を使います


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る


NEXT
人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る

人工知能（ＡＩ）技術応用_実習２

Part1


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習２　画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

実習２ 画像認識AIで犬と猫を分類

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → **実習２ 画像認識AIで犬と猫を分類** → 実習３ 自然言語処理で感情分析AIを作る

- **Part１：データ準備とCNN入門**
- Part２：CNNモデルの構築と学習
- Part３：モデルの評価と改善
- Part４：転移学習と実データテスト

Part １：データ準備とCNN入門 の達成目標

画像認識AIのためのデータ準備

今回の目標

- ✓ 実習データセットを準備する
- ✓ 画像データの前処理方法を理解する
- ✓ 画像認識の心臓部「CNN」の必要性と基本を知る
- ✓ AI学習のための「前処理（正規化・データ拡張）」を行う

画像データとは？

実習２ 画像認識AIで犬と猫を分類 Part１

RGBと3層構造

AIにとって、画像は巨大な「数値の配列（行列）」

▶実習１（数字）：白黒（1層）

▶**実習２（写真）：カラー（３層：赤・緑・青）**

▶各ピクセルは 0～255 の諧調（数値）で表される

ディープラーニング（CNN）

実習２ 画像認識AIで犬と猫を分類 Part１

なぜ CNN なのか？

従来の機械学習（実習1）では、複雑な写真の分類は困難

➢ **理由：**写真はずれる、傾く、明るさが変わる
ピクセル単位の比較では対応できない

☞ **解決策：CNN（畳み込みニューラルネットワーク）**

✓ **CNNの特徴：**画像から「エッジ」「形」「模様」といった
特徴を自動で抽出可能

環境準備とデータセット

実習２ 画像認識AIで犬と猫を分類 Part１

環境設定とデータの分割

▶**環境設定：**計算量が多いため、Colab の**「GPU」**を必ず有効に

▶**データセット：**犬と猫の画像（合計 約25,000枚）

▶**データの分割：**

- **訓練データ** (Train)　　：AIが勉強するため用
- **検証データ** (Validation)：勉強の成果のチェック用

Google Colab で GPU設定

実習２ 画像認識AIで犬と猫を分類 Part１

GPU を有効化して学習を高速化します

手順

1. Google Colab で新しいノートブックを作成
2. メニューから「ランタイム」→「ランタイムのタイプを変更」
3. 「ハードウェアアクセラレータ」で「T4 GPU」を選択
4. 「保存」をクリック

```
# GPUが使えるか確認
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("GPU available:",
tf.config.list_physical_devices('GPU'))
```

データセットのダウンロード

実習２ 画像認識AIで犬と猫を分類 Part１

Dogs vs Cats データセットをダウンロード

```
# データセットのダウンロード
!wget https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
!unzip -q cats_and_dogs_filtered.zip
print("✅ ダウンロード完了")
```

▶ データセット：cats_and_dogs_filtered

- train （訓練用）
  - cats （猫の画像 1,000枚）
  - dogs （犬の画像 1,000枚）
- validation （検証用）
  - cats （猫の画像 500枚）
  - dogs （犬の画像 500枚）

画像の確認

実習２ 画像認識AIで犬と猫を分類 Part１

ダウンロードした画像を表示します

```
import matplotlib.pyplot as plt import os from tensorflow.keras.preprocessing
import image
# 画像パスの設定
train_cats_dir = 'cats_and_dogs_filtered/train/cats'
train_dogs_dir = 'cats_and_dogs_filtered/train/dogs'
# 猫と犬の画像を4枚ずつ表示
fig, axes = plt.subplots(2, 4, figsize=(12, 6))
# 猫の画像
for i in range(4): img_path = os.path.join(train_cats_dir, os.listdir(train_cats_dir)[i])
img = image.load_img(img_path) axes[0, i].imshow(img) axes[0, i].set_title('Cat') axes[0, i].axis('off')
# 犬の画像
for i in range(4):
img_path = os.path.join(train_dogs_dir, os.listdir(train_dogs_dir)[i])
img = image.load_img(img_path) axes[1, i].imshow(img) axes[1, i].set_title('Dog') axes[1, i].axis('off')
plt.tight_
layout()
plt.show()
```

前処理①：正規化 (Normalization)

実習２ 画像認識AIで犬と猫を分類 Part１

データの数値を整える

AIが学習しやすくするためにデータの数値を整える

- ▶ **元のデータ：** 0 ～ 255　（範囲が広い）
- ▶ **変換後　　：** **0.0 ～ 1.0**（範囲が狭い）
- ▶ **方法　　：** 全ての値を 255 で割る

メリット
計算が安定し、学習のスピードと精度が向上します

前処理②：データ拡張 (Data Augmentation)

実習２ 画像認識AIで犬と猫を分類 Part１

AIを「応用力のある賢い子」にするための工夫

- ▶ **課題：「過学習（Overfitting）」**
  決まった画像しか認識できなくなる
- ☞ **対策：画像を加工して、擬似的にデータを増やす**
  - 回転させる、ズームする、左右反転させるなど

実装：ImageDataGenerator

実習２ 画像認識AIで犬と猫を分類 Part１

前処理の自動化

Keras の便利な機能を使って、前処理を自動化する

▶画像をフォルダから読み込みながら、リアルタイムで加工してAIに渡す

▶メモリを節約しながら大量のデータを学習できる

実装：ImageDataGenerator

実習２ 画像認識AIで犬と猫を分類 Part１

前処理の自動化

```
from tensorflow.keras.preprocessing.image
import ImageDataGenerator
# 画像の前処理設定
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)
# データジェネレータの作成
train_generator =
train_datagen.flow_from_directory( 'cats_and_dogs_filtered/train', target_size=(150, 150),
batch_size=20, class_mode='binary' )
validation_generator =
validation_datagen.flow_from_directory( 'cats_and_dogs_filtered/validation',
target_size=(150, 150), batch_size=20, class_mode='binary' )
print(f"訓練データ: {train_generator.samples}枚")
print(f"検証データ: {validation_generator.samples}枚")
```

実習１ Part１のまとめ

実習２ 画像認識AIで犬と猫を分類 Part１

今回やったこと：

- ✅ 画像データ（カラー）の仕組みを確認した
- ✅ GPU環境を整えた
- ✅ 実習用の画像データを準備した
- ✅ 「正規化」と「データ拡張」で、学習の準備を整えた

NEXT

実習２ Part２：CNNモデルの構築と学習

実習２ 画像認識AIで犬と猫を分類

実習２ Part２では以下を行います

- ✓ **CNNモデル**をゼロから設計
- ✓ AIを実際に学習させ**「学習曲線」**からその成長を見守る
- ✓ 強力なAIの力の裏にある**「過学習」の課題**に直面する


人工知能（AI）技術応用
実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

人工知能（ＡＩ）技術応用_実習２

Part2


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習２　画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

実習２ 画像認識AIで犬と猫を分類

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → **実習２ 画像認識AIで犬と猫を分類** → 実習３ 自然言語処理で感情分析AIを作る

Part１：データ準備とCNN入門

☞ **Part２：CNNモデルの構築と学習**

Part３：モデルの評価と改善

Part４：転移学習と実データテスト

Part２：CNNモデルの構築と学習 の達成目標

実習２ 画像認識AIで犬と猫を分類

「AIの『脳』をゼロから設計する」

- ✓ **CNN（畳み込みニューラルネットワーク）の基本構造を理解する**
- ✓ **Kerasを使ってCNNモデルをゼロから構築（`models.Sequential`）できる**
- ✓ **モデルの学習設定（`.compile()`）の意味を理解する**
- ✓ **モデルを学習（`.fit()`）させ、そのプロセスを監視できる**
- ✓ **学習曲線（正解率・損失）を可視化し、読み取れる**
- ✓ **「過学習（Overfitting）」とは何かを説明できる**

CNN 各層の役割

実習２ 画像認識AIで犬と猫を分類 Part２

CNN各層のイメージの理解

Conv2D (畳み込み層)
- 役割：画像から「特徴」（エッジ、線、模様）を抽出する

MaxPooling2D (プーリング層)
- 役割：画像（特徴マップ）を縮小し、重要な特徴だけを残す

Flatten (平坦化層)
- 役割：2次元の画像を1次元の列データに変換する

Dense (全結合層)
- 役割：抽出された特徴を基に、最終的な分類（犬か猫か）を行う

タスク１：CNNモデルの構築

実習２ 画像認識AIで犬と猫を分類 Part２

AIの脳を設計する

Keras の Sequential を使い、層を積み重ねてAIの「脳」を設計する

```
from tensorflow.keras import layers, models
model = models.Sequential([
  # 150x150x3 の画像を入力
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  layers.MaxPooling2D((2, 2)),
  # 畳み込み層をさらに追加
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),
  # ...（さらに層を追加）...
  # 1次元に平坦化
  layers.Flatten(),
  # 最後の分類層
  layers.Dense(512, activation='relu'),
  layers.Dense(1, activation='sigmoid') # 犬(1)か猫(0)の2値分類
])
```

Part２：CNNモデルの構築と学習　人工知能（AI）技術応用 実習２

タスク２：モデルのコンパイル

実習２ 画像認識AIで犬と猫を分類 Part２

モデルに「学習の方法」を教える

```
model.compile(
  loss='binary_crossentropy', # 2値分類（犬/猫）用の損失関数
  optimizer='adam', # 効率的な最適化アルゴリズム
  metrics=['accuracy'] # 評価指標として「正解率」を使用
)
# モデルの構造を確認
model.summary()
```

ポイント
- `loss` ：AIの「間違い」を測るモノサシ
- `optimizer` ：「間違い」を最小限にするための学習方法
- `metrics` ：学習中に人間が確認したい指標（正解率）

タスク３：モデルの学習

実習２ 画像認識AIで犬と猫を分類 Part２

.fit() コマンドで学習を開始する

.fit() コマンドでPart１で準備したデータ (train_generator) を使い学習を開始

```
# モデルの学習（GPUで約5〜15分かかります）
history = model.fit(
  train_generator, # 訓練データ
  steps_per_epoch=100, # 1エポックあたり100ステップ （100*20=2000枚）
  epochs=15, # 訓練データを15周学習
  validation_data=validation_generator, # 検証データ
  validation_steps=50 # 1エポックあたり50ステップ （50*20=1000枚）
)
```

⚠ **GPUの確認**
必ずColabのランタイムが「T4 GPU」になっているか確認する

学習中の待機時間

実習２ 画像認識AIで犬と猫を分類 Part２

学習中の観察

- ▶Epoch (エポック)　：全訓練データを何周学習したか。
- ▶loss (損失)　：AIの間違い度合い（低いほど良い）
- ▶accuracy (正解率)　：訓練データでの正解率（高いほど良い）
- ▶val_loss (検証損失)　：未知のデータ（検証用）での間違い度合い
- ▶val_accuracy (検証正解率)：未知のデータ（検証用）での正解率 ←重要

タスク４：学習曲線の可視化

実習２ 画像認識AIで犬と猫を分類 Part２

学習の履歴を使って、グラフで検証

```
# 正解率(Accuracy)のグラフ
plt.plot(history.history['accuracy'], label='Training Accuracy (訓練)')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy (検証)')
plt.legend()
plt.title('Accuracy over Epochs')
plt.show()

# 損失(Loss)のグラフ
plt.plot(history.history['loss'], label='Training Loss (訓練)')
plt.plot(history.history['val_loss'], label='Validation Loss (検証)')
plt.legend()
plt.title('Loss over Epochs')
plt.show()
```

学習曲線の分析

グラフから何を読み取るか？

🤔 **考察ポイント（過学習）**

▶**症状**　：訓練データの成績（acc ↑, loss ↓）は良いのに、
検証データの成績（val_acc ↓, val_loss ↑）が悪化する

▶**原因**　：AIが訓練データを「暗記」しすぎて、
未知のデータに対応できなくなった

▶**グラフは？**：「過学習」に近い形になっているはず

実習２ Part２のまとめ

- ✅ **CNNモデルをゼロから Keras で構築**した
- ✅ **モデルを学習**させた
- ✅ **学習にはGPUが必要**で時間がかかることを体感した
- ✅ 学習曲線を可視化し、**「訓練」と「検証」の差**を見る方法を学んだ
- ✅ **「過学習（Overfitting）」**という、AI開発の重要な課題を発見した

NEXT

実習２ Part３：モデルの評価と改善

実習２ 画像認識AIで犬と猫を分類

実習２ Part３では以下を行います

- 学習済みモデルの「本当の実力」を**テストデータで評価**
- AIはどんな画像で間違えるのか？**（誤分類分析）**
- 過学習を抑えるテクニック**「ドロップアウト」**を実装

人工知能（AI）技術応用

実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト


NEXT
人工知能（AI）技術応用
実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no Tobira

人工知能（ＡＩ）技術応用_実習２

Part3


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習２　画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

実習２ 画像認識AIで犬と猫を分類

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → **実習２ 画像認識AIで犬と猫を分類** → 実習３ 自然言語処理で感情分析AIを作る

Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
☞ **Part３：モデルの評価と改善**
Part４：転移学習と実データテスト

Part３：モデルの評価と改善 の達成目標

実習２ 画像認識AIで犬と猫を分類

「AIの『弱点』を見つけて賢くする」

- ✓ 学習済みモデルの「最終的な実力」をテストデータで評価できる
- ✓ AIが「どんな間違い」をしたかを具体的に分析できる
- ✓ 混同行列や分類レポートを使い、性能を多角的に評価する
- ✓ Part２で見つけた「過学習」への対策を理解する
- ✓ 過学習を抑えるテクニック「ドロップアウト」を実装できる

タスク１：最終評価

実習２ 画像認識AIで犬と猫を分類 Part３

テストデータでの最終評価

学習に使っていない「検証用データ (validation_generator)」で
モデルの本当の実力をテストする

```
# モデルの評価 (Part 2で学習させた model を使用)
test_loss, test_acc = model.evaluate(validation_generator)

print(f"テスト損失: {test_loss:.4f}")
print(f"テスト正解率: {test_acc*100:.2f}%")
```

🎯 ポイント
Part 2の学習曲線 (val_accuracy) の最終地点と、この test_acc の値が近ければ、
学習は正しく行われています
(例えば、Part 2 のval_accuracyが75%なら、test_accも75%前後になるはず)

タスク２：間違い分析

実習２ 画像認識AIで犬と猫を分類 Part３

AIがなぜどこで間違えたのか分析する

```
# 予測を実行し、間違えた画像のインデックスを取得
predictions = model.predict(validation_generator)
predicted_classes = (predictions > 0.5).astype(int).flatten()
true_classes = validation_generator.classes
wrong_indices = np.where(predicted_classes != true_classes)[0]

# 間違えた画像を表示
# （ワークブックのコード参照）
```

間違いのパターン

実習２ 画像認識AIで犬と猫を分類 Part３

AIが間違えやすい画像の理解

- 例１：犬だけど猫っぽいポーズの画像
- 例２：猫だけど犬っぽい毛並みの画像
- 例３：背景がごちゃごちゃしていて、犬/猫が小さい画像
- 例４：暗い、または不鮮明な画像

タスク３：改善策（過学習対策）

実習２ 画像認識AIで犬と猫を分類 Part３

モデルの改善（過学習対策）のテクニック

▶学習中にランダムに一部のニューロンを「休ませる」ことで、AIが訓練データを「丸暗記」するのを防ぐ

- これにより、未知のデータ（検証データ）に対する対応力（汎化性能）が向上する

改善版モデルの実装

実習２ 画像認識AIで犬と猫を分類 Part３

ドロップアウト追加

```
model_improved = models.Sequential([
  # ... (Conv/Pool層は同じ) ...
  layers.Conv2D(128, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),
  layers.Flatten(),
  layers.Dropout(0.5), #  追加！ (50%のニューロンを無効化)
  layers.Dense(512, activation='relu'),
  layers.Dropout(0.5), #  追加！
  layers.Dense(1, activation='sigmoid')
])
# 再度コンパイルして学習
model_improved.compile(...)
history_improved = model_improved.fit(...) # 再び学習
```


改善結果の比較

実習２ 画像認識AIで犬と猫を分類 Part３

改善版モデルの学習曲線

考察
ドロップアウトによって過学習が抑制され、モデルの汎化性能が向上しました
しかしゼロから学習するCNNでは、これ以上の大幅な性能向上は難しい場合があります

実習２ Part３のまとめ

実習２ 画像認識AIで犬と猫を分類 Part３

- ✅ `.evaluate()` でテストデータに対する最終性能を評価した
- ✅ 予測が間違った画像を確認し、AIの弱点を分析した
- ✅ 過学習が性能低下の原因であることを理解した
- ✅ 過学習対策として「ドロップアウト」を実装し、効果を確認した

NEXT

実習２ Part４：転移学習と実データテスト

実習２ 画像認識AIで犬と猫を分類

実習２ Part４では以下を行います

- ✓ ゼロから学習するのをやめ「賢いAI」の知識を借りる**「転移学習」**に挑戦
- ✓ **VGG16**という超強力なモデルを使う
- ✓ **正解率**はどこまで上がるのか？
- ✓ 自分で用意した犬・猫の写真でAIをテストする


人工知能（AI）技術応用
実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習２ 画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

人工知能（ＡＩ）技術応用_実習２

Part4


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習２　画像認識AIで犬と猫を分類
Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
Part４：転移学習と実データテスト
Mirai no tobira

実習２ 画像認識AIで犬と猫を分類

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → **実習２ 画像認識AIで犬と猫を分類** → 実習３ 自然言語処理で感情分析AIを作る

Part１：データ準備とCNN入門
Part２：CNNモデルの構築と学習
Part３：モデルの評価と改善
☞ **Part４：転移学習と実データテスト**

Part４：転移学習と実データテスト の達成目標

実習２ 画像認識AIで犬と猫を分類

- ✓ **「転移学習 (Transfer Learning)」の概念と利点を理解する**
- ✓ **「事前学習済みモデル（VGG16）」を読み込む方法を学ぶ**
- ✓ **事前学習済みモデルを使って、高精度な分類モデルを構築できる**
- ✓ **自作CNNモデルと転移学習モデルの性能を比較・考察できる**
- ✓ **自分で用意した画像で、学習済みAIの予測をテストできる**
- ✓ 実習２全体の学びを「最終レポート」としてまとめる

転移学習とは？

実習２ 画像認識AIで犬と猫を分類 Part４

非常に強力で一般的なテクニック転移学習とは？

ゼロから学習 (Part２,３)

例

赤ちゃんに「犬」と「猫」を**ゼロから教える**

特徴

- ✓ 大量のデータ（数万枚）が必要
- ✓ 学習に時間がかかる
- ✓ 性能に限界がある

転移学習 (Part４)

例

「動物博士」に「犬」と「猫」の**見分け方だけ追加で教える**

特徴

- ✓ 少ないデータでOK
- ✓ 学習時間が短い
- ✓ 高精度が出やすい

タスク１：VGG16を使った転移学習

実習２ 画像認識AIで犬と猫を分類 Part４

ImageNetで事前学習されたVGG16を使う

大規模データセット（ImageNet:140万枚）で学習済みモデルを活用する

```
from tensorflow.keras.applications import VGG16
# VGG16の読み込み（事前学習済み）
base_model = VGG16( weights='imagenet', include_top=False, input_shape=(150, 150, 3) )
# ベースモデルの重みを固定
base_model.trainable = False
# 新しい分類層を追加
model_transfer = models.Sequential([ base_model, layers.Flatten(), layers.Dense(256,
activation='relu'), layers.Dropout(0.5), layers.Dense(1, activation='sigmoid') ])
model_transfer.compile( loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'] )
print("転移学習モデル構築完了")
```

Part４：転移学習と実データテスト　人工知能（AI）技術応用 実習２

タスク２：転移学習モデルの学習

実習２ 画像認識AIで犬と猫を分類 Part４

転移学習モデルを学習させる

```
# 転移学習モデルの学習
history_transfer = model_transfer.fit( train_generator, steps_per_epoch=100,
epochs=10, validation_data=validation_generator, validation_steps=50 )
# 評価
test_loss, test_acc = model_transfer.evaluate(validation_generator)
print(f"転移学習モデルの正解率: {test_acc*100:.2f}%")
```

🎯 性能比較（例）

モデル	検証正解率 (val_accuracy)
自作CNN (Part３)	約 75～80%
転移学習 (Part４)	**約 90% 以上**

タスク３：自分の画像でテスト

実習２ 画像認識AIで犬と猫を分類 Part４

学習させた最強のモデルで、用意した犬や猫の画像を予測させる

📝 手順

1. 犬や猫の画像を（スマホで撮る、ネットで探すなど）用意する
2. Colabに画像をアップロードする
3. `predict_image(img_path)` 関数を使って予測

Part４：転移学習と実データテスト　人工知能（AI）技術応用 実習２

タスク３：自分の画像でテスト

実習２ 画像認識AIで犬と猫を分類 Part４

```
# 画像のアップロードと予測
def predict_image(img_path): img = image.load_img(img_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0 prediction = model_transfer.predict(img_array)[0][0]
if prediction > 0.5: result = f"犬（確率: {prediction*100:.1f}%）" else: result =
f"猫（確率: {(1-prediction)*100:.1f}%）"
# 画像表示
import matplotlib.pyplot as plt
plt.imshow(img)
plt.title(result)
plt.axis('off')
plt.show()
return result
# 使用例（画像をアップロード後）
# result = predict_image('your_image.jpg')
# print(result)
```


タスク４：最終レポート

実習２ 画像認識AIで犬と猫を分類 Part４

実習２全体の総まとめとして分析レポートを作成

レポートに含める内容

1. 実習の概要： 何をしたか？（犬猫分類AIの構築）

2. 自作CNN (Part２, ３)：

モデル構造、学習曲線、最終的な正解率

過学習の問題点とドロップアウトによる改善

3. 転移学習 (Part４)：

転移学習とは何か？ VGG16の使用

自作CNNと転移学習モデルの性能比較

4. 考察と学び：

なぜ転移学習は高精度なのか？実データでテストした結果

実習２全体で学んだこと、難しかったこと

実習２まとめ

実習２ 画像認識AIで犬と猫を分類

今回学んだこと：

- ✅ ディープラーニング (CNN) の基本構造の理解
- ✅ ゼロから画像認識モデルを構築と学習
- ✅ 「過学習」を発見し、ドロップアウトで対策
- ✅ 「転移学習」という強力なテクニックを実装、高精度を達成
- ✅ 自分の画像でAIをテストする実践的プロセス

NEXT

実習３ 自然言語処理で感情分析AIを作る

実習２ 画像認識AIで犬と猫を分類

実習３では以下を行います

- ✓ 数字、画像の次は、**テキスト（自然言語）**に挑戦
- ✓ 文章を読んで**感情を分析**するAIを作る
- ✓ 新しいディープラーニング技術 **RNN** や **LSTM** を学ぶ


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


NEXT
人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira

タスク２：転移学習モデルの学習

実習２ 画像認識AIで犬と猫を分類 Part４

新しく追加した分類層だけ学習させる

```
model_transfer.compile(...) # Part 2と同様
# 学習実行（GPUで約5〜10分）
history_transfer = model_transfer.fit(
  train_generator,
  epochs=10, # エポック数は少なめでもOK
  validation_data=validation_generator,
  ...)
```

🎯 性能比較（例）

モデル	検証正解率 (val_accuracy)
自作CNN (Part３)	約 75〜80%
転移学習 (Part４)	**約 90% 以上**

モデルの結合

実習２ 画像認識AIで犬と猫を分類 Part４

ベースモデルと分類層の結合

凍結したベースモデル（特徴抽出機）の後ろに
解きたい問題（犬猫分類）用の分類層を新たに追加する

```
model_transfer = models.Sequential([
  base_model, # 賢いベースモデル（凍結済み）
  layers.Flatten(),
  layers.Dense(256, activation='relu'), # 自分で追加する分類層
  layers.Dropout(0.5),
  layers.Dense(1, activation='sigmoid') # 最終的な犬(1)猫(0)判定
])

model_transfer.summary() # 構造確認
```

人工知能（ＡＩ）技術応用_実習３

Part1


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１　環境準備とテキスト前処理の基礎
Part２　RNN/LSTMモデルの構築
Part３　モデルの評価と可視化
Part４　事前学習モデル（BERT）の活用
Mirai no tobira

実習3 自然言語処理で感情分析AIを作る

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → **実習３ 自然言語処理で感情分析AIを作る**

- **Part１：環境準備とテキスト前処理の基礎**
- Part２：RNN/LSTMモデルの構築
- Part３：モデルの評価と可視化
- Part４：事前学習モデル（BERT）の活用

Part １：テキストデータの準備 の達成目標

実習3 自然言語処理で感情分析AIを作る

自然言語処理の準備とデータ確認

「映画レビューの感情分析AI」

ユーザーが書いたレビューから、

それが「ポジティブ」か「ネガティブ」かを判定するAIを作る

Part１：

- **テキストデータの準備と確認**
- **コンピュータが理解できる形に変換する プロセス**

データセットの設定と確認

実習3 自然言語処理で感情分析AIを作る Part 1

タスク１：IMDb映画レビューデータセットの設定

データ内容は、レビューの文章ではなく、数値のリスト

- ▶ データ源　：インターネット・ムービー・データベース (IMDb) のレビュー
- ▶ データ数　：合計 50,000件（学習用 25,000件、テスト用 25,000件）
- ▶ データ形式：テキスト と ラベル のペア
- ▶ テキスト　：実際のレビュー文章（例: "This movie was great!"）
- ▶ ラベル　　：感情（0: ネガティブ, 1: ポジティブ）

テキストの「前処理」

実習3 自然言語処理で感情分析AIを作る Part 1

コンピュータは「文章」を理解できない

文章をAIが扱える 数値ベクトル に変換します

テキストの前処理ステップ

1. トークン化：文章を単語や記号に区切る
2. IDへの変換：各単語にID (数値) を割り当てる
3. パディング：文章の長さを揃える

🤔 ポイント
この前処理が、文章をAIに「読ませる」ための最も重要なステップです

ステップ１：トークン化

実習3 自然言語処理で感情分析AIを作る Part１

タスク２：文章を単語に区切る (Tokenization)

- ▶ 目的：AIに文章の「意味の単位」を認識させる
- ▶ 実行内容：文: "This movie was great!“
  トークン: ["This", "movie", "was", "great", "!"]

タスク ２：トークナイザの準備と実行

- ▶ ワークブック タスク２のコードで、実際にテキストを
  トークン化（`Tokenizer`）を実行します

ステップ２：IDへの変換

実習3 自然言語処理で感情分析AIを作る Part１

単語を数値 (ID) に変換する

- ▶ 目的：トークンをAIが計算できる 数値ベクトル に変換する
- ▶ 実行内容：

  トークン: ["This", "movie", "was", "great", "!"]

  ID: [10, 5, 2, 80, 25]

💡 注意点：
出現頻度が高い単語から、1, 2, 3... とIDが割り当てられます
登場回数が少ない単語は無視（**Unknown**として処理）されます

ステップ３：長さを揃える（パディング）

実習3 自然言語処理で感情分析AIを作る Part１

タスク３：AIへの入力サイズを統一する（ Padding ）

長さがバラバラのレビューの長さを、すべて同じ長さに揃える

- ▶ **長いレビュー：途中をカット** (Truncating)
- ▶ **短いレビュー：不足分を `0` (ゼロ) で埋める** (Padding)

タスク３：パディングの実行

今回は最大長 256単語 に揃えます。このコードを実行して、データがAIの入力形式として整ったことを確認する

実習３ Part１のまとめ

実習3 自然言語処理で感情分析AIを作る Part１

今回やったこと：

- ✅ 自然言語処理の**データセット（IMDb）**を確認した
- ✅ コンピュータが理解できるようにするための **前処理 の重要性**を理解した
- ✅ 前処理の３ステップ **「トークン化」「ID変換」「パディング」**を実行した

☞ これで、文章をAIに「読ませる」準備ができました

Part 1：テキストデータの準備　人工知能（AI）技術応用 実習３

NEXT

実習３ Part２ ：RNN/LSTMモデル

実習3 自然言語処理で感情分析AIを作る

実習１Part２では以下を行います

- ✓ 文章の「順序」を理解するニューラルネットワーク**「LSTM」**を使う
- ✓ テキストのIDを「意味のある点」にする **埋め込み層 (Embedding)** を学ぶ
- ✓ AIを実際に学習させ、**過学習 (Overfitting)** の課題を発見する

人工知能（AI）技術応用

実習３自然言語処理で感情分析AIを作る

Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用


NEXT
人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用
Mirai no tobira

人工知能（ＡＩ）技術応用_実習３

Part2


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１　環境準備とテキスト前処理の基礎
Part２　RNN/LSTMモデルの構築
Part３　モデルの評価と可視化
Part４　事前学習モデル（BERT）の活用
Mirai no tobira

実習3 自然言語処理で感情分析AIを作る

人工知能（AI）技術応用

実習 1　機械学習で手書き数字を認識 → 実習 2　画像認識AIで犬と猫を分類 → **実習 3　自然言語処理で感情分析AIを作る**

Part 1 ：環境準備とテキスト前処理の基礎

☞ **Part 2 ：RNN/LSTMモデルの構築**

Part 3 ：モデルの評価と可視化

Part 4 ：事前学習モデル（BERT）の活用

Part 2 ：RNN/LSTMモデルの構築 の達成目標

実習3 自然言語処理で感情分析AIを作る

- ✓ 単語埋め込み (Embedding) の役割を理解する
- ✓ 系列データを扱う RNN/LSTM の基本概念を理解する
- ✓ Keras で LSTMモデルを構築・コンパイルできる
- ✓ モデルを学習 (`.fit()`) させ、学習プロセスを監視する
- ✓ 学習曲線（Accuracy, Loss）を可視化し、読み取る
- ✓ 「過学習（Overfitting）」の兆候を発見する

タスク１：準備の確認

実習3 自然言語処理で感情分析AIを作る Part2

データの引き継ぎ

Part１で作成したデータを引き継ぎます

- ▶ `X_train, y_train` (訓練データとラベル)
- ▶ `X_test, y_test` (テストデータとラベル)

```
# Part 1 の処理
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
X_train = sequence.pad_sequences(X_train, maxlen=250)
X_test = sequence.pad_sequences(X_test, maxlen=250)
```

🎯 ポイント
データは「250語の長さ」に揃えられた「数値の列」になっていて、AIはまだ「単語の意味」を理解していません

タスク２：モデル構築① Embedding

実習3 自然言語処理で感情分析AIを作る Part2

モデル構築① Embedding層

- ▶ **コンピュータに「単語の意味」を教えるための層**
- ▶ **「単語の番号」を「意味を持つベクトル、数値の配列」に変換する層**

```
model = models.Sequential()

# 1. Embeddingレイヤー
model.add(layers.Embedding(max_features, 128, input_length=max_length))
```

タスク２：モデル構築② LSTM

実習3 自然言語処理で感情分析AIを作る Part２

モデル構築② LSTM層

- 文章の「順序」と「文脈」を理解するための層（心臓部）
- 文章を単語のベクトル列として、最初から順番に読み込み、文脈を記憶する

```
# 2. LSTMレイヤー
model.add(layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))
```


タスク２：モデル構築③ Dense

実習3 自然言語処理で感情分析AIを作る Part２

モデル構築③ Dense層（出力）

- 文脈を理解した結果、最終的な「判定」を下す層
- LSTMが読み取った文脈全体をもとに「ポジティブ，1に近い値」か、「ネガティブ，ゼロに近い値」かを判定

```
# 3. Denseレイヤー（出力層）
model.add(layers.Dense(1, activation='sigmoid'))
# モデルのコンパイル
model.compile(
  optimizer='adam',
  loss='binary_crossentropy', # 2値分類（ポジ/ネガ）
  metrics=['accuracy'])
model.summary() # 構造の確認
```

タスク３：モデルの学習

実習3 自然言語処理で感情分析AIを作る Part2

.fit() コマンドによるモデルの学習

▶ model.fit() ：学習開始の命令

```
# モデルの学習（GPUで約5〜15分かかります）
history = model.fit(
  X_train, y_train,
  epochs=5, # 訓練データを5周学習
  batch_size=128, # 128件ずつ処理
  validation_split=0.2 # 訓練データの20%を検証用に
)
```

⚠ GPUの確認！
必ずColabのランタイムが「T4 GPU」になっているか確認してください

タスク４：学習曲線の可視化

実習3 自然言語処理で感情分析AIを作る Part2

学習曲線のグラフ化

▶ 学習の履歴 (history) をグラフにし学習がうまくいったか確認

```
# 正解率(Accuracy)のグラフ
plt.plot(history.history['accuracy'], label='訓練 (Training)')
plt.plot(history.history['val_accuracy'], label='検証 (Validation)')
plt.legend()
plt.title('Accuracy (正解率) の推移')
plt.show()
```

タスク４：学習曲線の可視化

実習3 自然言語処理で感情分析AIを作る Part2

正解率、損失、のグラフ

- Task4-1：正解率（accuracy）のグラフを描画
  - history.history['accuracy']　　：訓練データの正解率、
    history.history['val_accuracy']：検証データの正解率の推移
- Task 4-2：損失（Loss）のグラフを描画
  - history.history['loss']　　：損失率
    history.history['val_loss']　　：損失率の推移
    - 損失は「間違いの度合い」で、低いほど良い

学習曲線の分析（過学習）

実習3 自然言語処理で感情分析AIを作る Part2

損失（Loss）のグラフを見ての疑問

「過学習 (Overfitting)」

症状： 訓練データでは成績が良いのに、
未知のデータ（検証データ）では成績が悪化する現象

原因： AIが訓練データを「丸暗記」しすぎて、
応用が効かなくなっている状態

対策： Part３で「評価」、Part４で「改善（転移学習）」を行う

Part 2 ：RNN/LSTMモデルの構築　人工知能（AI）技術応用 実習３

実習３ Part２のまとめ

実習3 自然言語処理で感情分析AIを作る Part２

今回やったこと：

- ✅ テキストを「意味のあるベクトル」に変換する Embedding 層を学んだ
- ✅ 文章の「順序」や「文脈」を理解する LSTM 層を学んだ
- ✅ LSTMモデルを構築し、学習 (.fit()) させた
- ✅ 「過学習」というAI開発の重要な課題を発見した

NEXT

実習３ Part３：モデルの評価と可視化

実習3 自然言語処理で感情分析AIを作る

実習３Part３では以下を行います

- ▶ 学習したモデルの「本当の実力」を**テストデータで評価**
- ▶ AIは「どんな間違い」をしている？ → **混同行列**
- ▶ AIの「冤罪」と「見逃し」を測る → **Precision / Recall**


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用
Mirai no tobira

人工知能（ＡＩ）技術応用_実習３

Part3


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai no tobira


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１　環境準備とテキスト前処理の基礎
Part２　RNN/LSTMモデルの構築
Part３　モデルの評価と可視化
Part４　事前学習モデル（BERT）の活用
Mirai no tobira

実習3 自然言語処理で感情分析AIを作る

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → **実習３ 自然言語処理で感情分析AIを作る**

Part１：環境準備とテキスト前処理の基礎
Part２：RNN/LSTMモデルの構築
☞ **Part３：モデルの評価と可視化**
Part４：事前学習モデル（BERT）の活用

Part３：評価と可視化 の達成目標

実習3 自然言語処理で感情分析AIを作る

AIの『通知表』を読み解く

- ✓ 学習済みモデルで「未知のテストデータ」に対する最終性能を評価する
- ✓ **混同行列 (Confusion Matrix)** を作成し、AIの間違いパターンを分析する
- ✓ **分類レポート (Classification Report)** を読み解く
- ✓ **Precision (適合率)** と **Recall (再現率)** の意味とトレードオフを理解する
- ✓ AIがなぜ間違えたのかを考察する

タスク１：準備

実習3 自然言語処理で感情分析AIを作る Part３

モデルとデータの引き継ぎ

Part２で学習させたモデル（model）と、
Part１で準備したテストデータ（X_test, y_test）を準備します

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras import models, layers
from sklearn.metrics import confusion_matrix,
classification_report
```


タスク２：最終評価

実習3 自然言語処理で感情分析AIを作る Part３

テストデータによる最終評価

Part２で学習させた model を、
一度も見ていない「テストデータ」 (X_test, y_test) で評価します

```
# 準備：Part 1, 2のコードを実行し、model, X_test, y_test を用意
# （ノートブックが継続している場合は不要）

print("テストデータでの最終評価を開始します...")
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

print(f"テスト正解率 (Test Accuracy): {test_accuracy*100:.2f} %")
```

ポイント
この「テスト正解率」が、モデルの「本当の実力」です
Part２の「検証正解率 (val_accuracy)」 と近い値（例：86%前後）なら学習は妥当だったと言えます

タスク3：混同行列

実習3 自然言語処理で感情分析AIを作る Part3

混同行列（Confusion Matrix）

- 「正解率」だけでは、AIが「どんな間違い」をしているか分からない
  - ☞ 混同行列を使う

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# 予測の実行（0.5以上を1=ポジティブと判定）
probabilities = model.predict(X_test)
predictions = (probabilities > 0.5).astype(int)
# 混同行列の作成と可視化
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['ネガティブ（予測）', 'ポジティブ（予測）'], yticklabels=['ネガティブ（正解）', 'ポジティブ（正解）'])
```


混同行列の見方

実習3 自然言語処理で感情分析AIを作る Part3

混同行列の読み解き方

	予測: **ネガティブ**(0)	予測: **ポジティブ**(1)
正解: **ネガティブ**(0)	**左上 (TN)** 正しく「ネガティブ」と予測	**右上 (FP)** ネガティブを「ポジティブ」と誤認
正解: **ポジティブ**(1)	**左下 (FN)** ポジティブを「ネガティブ」と誤認	**右下 (TP)** 正しく「ポジティブ」と予測

🎯 **考察ポイント**
- **対角線 (TN, TP)**: 正解した数
- **右上 (FP)**: 間違ってポジティブと予測（冤罪）
- **左下 (FN)**: ポジティブを見逃した数（見逃し）
- 「右上」と「左下」、どちらの間違いが多いか？

Part3 ：評価と可視化　人工知能（AI）技術応用 実習3

予測の実行と混同行列の作成

実習3 自然言語処理で感情分析AIを作る Part3

テストデータ全体に対して予測を行い、その予測結果を取得

- テストデータ (X_test) に対する予測を実行
- .predict() は各レビューがポジティブである「確率」を返す (例: 0.9, 0.1)

```
probabilities = model.predict(X_test)
```

- 確率が 0.5 より大きい場合は 1 (ポジティブ)
  そうでなければ 0 (ネガティブ) に変換

```
predictions = (probabilities > 0.5).astype(int)
# 混同行列を計算
# y_test (正解ラベル) と predictions (予測ラベル) を比較
cm = confusion_matrix(y_test, predictions)
print("混同行列:")
print(cm)
```


混同行列の可視化

実習3 自然言語処理で感情分析AIを作る Part3

Seaborn のヒートマップを使って可視化

```
# ヒートマップで可視化
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative (Pred)', 'Positive (Pred)'],
            yticklabels=['Negative (True)', 'Positive (True)'])
plt.title('Confusion Matrix (混同行列)')
plt.xlabel('予測ラベル (AIの答え)')
plt.ylabel('正解ラベル (本当の答え)')
plt.show()
```

タスク４：分類レポート

実習3 自然言語処理で感情分析AIを作る Part３

より詳しい指標を計算

混同行列の数値をもとに、さらに詳しい性能指標を計算します

```
from sklearn.metrics import classification_report
report = classification_report(y_test, predictions,
target_names=['Negative (0)', 'Positive (1)'])
print(report)
```


Precision vs Recall

実習3 自然言語処理で感情分析AIを作る Part３

Precision (適合率) vs Recall (再現率)

「正解率」よりも重要な指標

- **Precision (適合率)**
  - AI がポジティブと予測したもののうち、本当にポジティブだった割合
  - **「AIの予測の正確さ」（冤罪の少なさ）**
  - Precision が高い → AI がポジティブと言ったら、信用できる
- **Recall (再現率)**
  - 本当にポジティブだったもののうち、AI が正しくポジティブと見つけられた割合
  - **「AIの見つける能力」（見逃しの少なさ）**
  - Recall が高い → AI はポジティブなものを、あまり見逃さない

Part３：評価と可視化　人工知能（AI）技術応用 実習３

実習３ Part３ まとめ

実習3 自然言語処理で感情分析AIを作る Part３

今回やったこと：

✅ .evaluate() でテストデータに対する最終性能を評価した

✅ confusion_matrix（混同行列）で間違いのパターンを分析した

✅ classification_report で Precision と Recall の意味を学んだ

✅ AIの評価は「目的」に応じて使い分ける必要があると理解した

NEXT

実習３ Part４ ：事前学習モデル（BERT）の活用

実習3 自然言語処理で感情分析AIを作る

実習３Part４では以下を行います

✓ ゼロから作るのをやめ、最強の**「事前学習モデル」**を使う

✓ 現代NLPの標準 **「BERT」** と **Hugging Face** ライブラリを使う

✓ **正解率は 90% を超える**か？

✓ **自分で書いた文章**で、感情分析AIをテストする


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用
Mirai no tobira


NEXT
人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１ 環境準備とテキスト前処理の基礎
Part２ RNN/LSTMモデルの構築
Part３ モデルの評価と可視化
Part４ 事前学習モデル（BERT）の活用
Mirai no tobira

人工知能（ＡＩ）技術応用_実習３

Part4


人工知能（AI）技術応用
「基礎から実践へ、3つの実習で学ぶ
機械学習・画像認識・自然言語処理」
実習１　機械学習で手書き数字を認識
実習２　画像認識AIで犬と猫を分類
実習３　自然言語処理で感情分析AIを作る
Mirai-no tobira


人工知能（AI）技術応用
実習３自然言語処理で感情分析AIを作る
Part１　環境準備とテキスト前処理の基礎
Part２　RNN/LSTMモデルの構築
Part３　モデルの評価と可視化
Part４　事前学習モデル（BERT）の活用
Mirai-no tobira

実習3 自然言語処理で感情分析AIを作る

人工知能（AI）技術応用

実習１ 機械学習で手書き数字を認識 → 実習２ 画像認識AIで犬と猫を分類 → **実習３ 自然言語処理で感情分析AIを作る**

Part１：環境準備とテキスト前処理の基礎
Part２：RNN/LSTMモデルの構築
Part３：モデルの評価と可視化
☞ **Part４：事前学習モデル（BERT）の活用**

Part４：事前学習モデル（BERT）の活用 の達成目標

実習3 自然言語処理で感情分析AIを作る

現代 NLPの巨人『BERT』を使ってみよう

- ✓ **「事前学習モデル」**と**「転移学習（ファインチューニング）」**の概念理解
- ✓ **Hugging Face transformers** ライブラリの基本を学ぶ
- ✓ BERTモデルを読み込み、**IMDbデータでファインチューニング**できる
- ✓ 自作 LSTMモデルと **BERTモデルの性能（正解率）を比較**する
- ✓ 学習済み BERTモデルで、自分の書いた文章の感情を予測する

転移学習

実習3 自然言語処理で感情分析AIを作る Part４

転移学習 (Transfer Learning) とは？

AI開発を「料理」に例えると...

ゼロから学習 (Part２)

例：**米、野菜、スパイスからカレーを自作**

特徴

✓ 大変（大量のデータと時間）
✓ そこそこの味 (正解率 約86%)

転移学習 (Part４)

例：**市販のカレールーを使い具材だけ追加**

特徴

✓ **簡単（少ないデータと時間）**
✓ **プロの味 (正解率 90%超え!?)**

ポイント
「カレールー」＝**事前学習済みモデル (BERT)**
BERTは、インターネット上の膨大なテキストを読んで「言語の基本」を既に学習しています
その賢いモデルに「感情分析」だけを追加で教えます（＝**ファインチューニング**）

Hugging Face Transformers

実習3 自然言語処理で感情分析AIを作る Part４

Hugging Face transformersライブラリインストール

☞ **BERTを、数行のコードで簡単に利用できるようにするライブラリ**
▶ **タスク１：インストール、タスク２：IMDbデータセット読み込み**

```
# 1. ライブラリのインストール
!pip install transformers datasets -q
# 2. データセットの読み込み (Hugging Face版)
dataset = load_dataset("imdb")
# (時間短縮のため5000件に削減)
train_dataset = dataset['train'].select(range(5000))
test_dataset = dataset['test'].select(range(5000))
# 3. BERT用の辞書（トークナイザー）を読み込む
tokenizer = AutoTokenizer.from_pretrained(model_name)
# 4. 全データをトークン化（数値化+パディング）
tokenized_train_dataset = train_dataset.map(tokenize_function, batched=True)
tokenized_test_dataset = test_dataset.map(tokenize_function, batched=True)
```

タスク３：BERTのファインチューニング

実習3 自然言語処理で感情分析AIを作る Part４

BERTのファインチューニング

▶ Hugging Faceの Trainer を使って学習させます

ステップ1：モデルの読み込み

```
# 感情分析用の事前学習済みモデルを読み込む
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

ステップ2：学習設定

```
# 学習の設定（エポック数など）を定義
training_args = TrainingArguments(...)
# 評価方法を定義
def compute_metrics(eval_pred): ...
# Trainerを準備
trainer = Trainer(model=model, args=training_args, ...)
```

ステップ3：学習実行 (約5〜15分)

```
trainer.train() # 学習開始！
```


性能比較

実習3 自然言語処理で感情分析AIを作る Part４

LSTM vs BERT

▶ ファインチューニングしたBERTモデルを評価します

```
eval_results = trainer.evaluate()
print(f"BERT テスト正解率: {eval_results['eval_accuracy']*100:.2f} %")
```

モデル	テスト正解率 (Accuracy)
LSTM (Part３)	約 86%
BERT (Part４)	**約 92% 以上**

🎯 **結論**
「事前学習」の力で、LSTMモデルを**大幅に上回る性能**を達成できた

Part４：事前学習モデル（BERT）の活用　人工知能（AI）技術応用 実習３

タスク4：実データ予測

実習3 自然言語処理で感情分析AIを作る Part4

自分の文章で試してみよう

☆ Hugging Face の pipeline を使って、学習済みBERTモデルに好きな文章を予測させてみましょう

```
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer, device=0)
my_review = "This movie was absolutely fantastic!"
print(sentiment_pipeline(my_review))
# 出力例: [{'label': 'POSITIVE', 'score': 0.999...}]
my_review = "I really hated this film. It was boring."
print(sentiment_pipeline(my_review))
# 出力例: [{'label': 'NEGATIVE', 'score': 0.999...}]
```


実習３まとめ

実習3 自然言語処理で感情分析AIを作る

- ✅ テキストを「数値」に変換する前処理（トークン化、ID化、パディング）を学んだ
- ✅ 文章の「順序」を学習できるRNN/LSTMモデルを構築した
- ✅ モデルの評価（混同行列、Precision/Recall）の方法を学んだ
- ✅ 「転移学習」と「BERT」の強力さを体験した
- ✅ 自分の文章で感情分析AIをテストした

人工知能（AI）技術応用 まとめ

3つの実習を通して、AIの主要分野を制覇しました

- ▶ **実習1：** 機械学習の基礎（数字認識）
- ▶ **実習2：** コンピュータビジョン（犬猫分類）
- ▶ **実習3：** 自然言語処理（感情分析）

令和7年度「地方やデジタル分野における専修学校理系転換等推進事業」
情報成長分野の教育プログラム整備と教員育成による学科転換・新設推進事業

人工知能（ＡＩ）技術応用教材資料

令和８年２月
一般社団法人全国専門学校情報教育協会
〒164-0003 東京都中野区東中野 1-57-8 辻沢ビル３F
電話：03-5332-5081 FAX.03-5332-5083