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）技術応用教材ワークブック

本人工知能（ＡＩ）技術応用教材ワークブックは、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

データ人工知能（AI）技術応用
実習１　機械学習で手書き数字を認識しよう
実習ガイド

📚実習の概要

この実習では、機械学習の基本的な流れを体験します。手書き数字の画像を使って、コンピュータに「これは何の数字か」を判断させるプログラムを作ります。

💡 **この実習で学ぶこと**

- 機械学習の基本的なワークフロー（データ準備 → 学習 → 予測 → 評価）
- Google Colab を使った Python プログラミングの基礎
- 3 つの機械学習アルゴリズムの使い方と比較
- モデルの性能を評価する方法

🎯学習目標

- **理解目標：**機械学習の基本的な処理の流れを説明できる
- **技能目標：**提供されたコードを使って機械学習モデルを動かせる
- **応用目標：**複数のアルゴリズムを比較し、最適なものを選択できる
- **評価目標：**モデルの正解率を計算し、結果を解釈できる

🗓️実習の流れ（全 4 回）

📘 Part 1

環境準備とデータ確認

- Google Colab の使い方
- MNIST データの読み込み
- データの可視化

📗 Part 2

3 つのアルゴリズム実装

- 決定木による分類
- k 近傍法の実装
- ロジスティック回帰

📕 Part 3

結果の比較と評価

- 正解率の計算

- 混同行列の作成
- 性能比較グラフ

📕 Part 4

実データでの検証

- 自分で撮影した数字
- 画像のアップロード
- 予測結果の確認

💻必要な環境・準備

✅ 必須環境

項目	要件	備考
インターネット環境	安定した接続	Google Colab を使用するため必須
Google アカウント	無料アカウント	事前に作成しておいてください
Web ブラウザ	Chrome 推奨	Firefox、Edge も可
カメラ付きデバイス	スマホまたは PC	Part 4 で手書き数字を撮影

📦 使用するツール・ライブラリ（すべて無料）

- **Google Colab：** Python をブラウザで実行できる環境（インストール不要）
- **scikit-learn：** 機械学習ライブラリ（Colab に標準搭載）
- **NumPy：** 数値計算ライブラリ（Colab に標準搭載）
- **Matplotlib：** グラフ描画ライブラリ（Colab に標準搭載）
- **MNIST データセット：** 手書き数字画像（自動ダウンロード）

この実習では、パソコンに何もインストールする必要がありません。Google アカウントがあれば、すぐに始められます！

🚀Google Colab の準備（初回のみ）

Step 1: Google Colab にアクセス

1. **1** ブラウザで https://colab.research.google.com/ にアクセス
2. **2**Google アカウントでログイン
3. **3**「ノートブックを新規作成」をクリック

Step 2: 基本操作の確認

📝 Colab の基本用語

- **セル：**コードを書く場所（1 つのブロック）
- **実行：**セルの左にある▶ボタンを押す、または Shift + Enter
- **テキストセル：**説明を書く場所（実行不要）
- **コードセル：**プログラムを書いて実行する場所

Step 3: 動作確認

新しいセルに以下を入力して実行してみましょう：

```
print("Hello, AI World!")
print(2 + 3)
```

「Hello, AI World!」と「5」が表示されれば成功です！

🔢 MNIST データセットとは？

MNIST は、機械学習の世界で最も有名な「練習用データ」です。

📊 データの内容

- **画像数：**訓練用 60,000 枚 + テスト用 10,000 枚
- **画像サイズ：**28×28 ピクセル（白黒画像）
- **内容：**0 から 9 までの手書き数字
- **特徴：**様々な人が書いた、様々な形の数字

🤔 なぜ MNIST を使うの？

- ✅ **初心者に最適：**データが単純で理解しやすい
- ✅ **すぐに使える：**ライブラリに含まれているため、ダウンロード不要
- ✅ **結果が分かりやすい：**「この数字は何？」という直感的な問題
- ✅ **標準的：**世界中で使われており、他の人と比較できる

📖 学習の進め方

🎯 各 Part の進め方

1. **ワークブックを開く：**各 Part のワークブックに記入しながら進めます
2. **動画を視聴（オンライン講座の場合）：**ナレーション付きで解説を聞きます
3. **実際にコードを実行：**Google Colab でコードを動かします
4. **結果を記録：**ワークブックに結果や気づきを書き込みます
5. **チェックポイント確認：**各セクションの理解度を確認します

💡 うまく進めるコツ

- **焦らない：**一つずつ確実に進めましょう

- **エラーを恐れない：**エラーは学習のチャンス。指導マニュアルに解決法があります
- **手を動かす：**見るだけでなく、必ず自分で実行しましょう
- **記録する：**気づいたことをワークブックに書き込みましょう
- **質問する：**分からないことは遠慮なく質問しましょう

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
Part 1 （環境準備とデータ確認）ワークブック

📝 実習記録

氏名	
実習日	年　　月　　日

🎯 Part 1 の学習目標

この Part では、次の内容を目標にします。

1. Google Colab にアクセスし、基本操作ができる。
2. MNIST データセットを読み込み、その構造を確認できる。
3. 訓練データとテストデータの分割の目的を理解できる。
4. 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。
5. 各数字のデータ分布をグラフで確認できる。

📍 タスク 1：Google Colab ノートブックの準備

🎯 タスク内容

Google Colab にアクセスして、新しいノートブックを作成します。

手順

1. ブラウザで https://colab.research.google.com/ にアクセスします。
2. 「ノートブックを新規作成」をクリックします。
3. ノートブック名を「AI 実習 1_Part1_氏名」に変更します。

✅ チェックポイント

- ☐ 新しいノートブックが開けた
- ☐ ノートブック名を変更できた
- ☐ セルが表示されている

💡 ヒント

ログインできない場合は、Google アカウントを持っているか確認しましょう。アカウントがない場合は、無料で作成できます。

📋 記録欄：うまくいったこと・困ったこと

（ここに自由に記入してください）

📍 タスク 2：Python 動作確認

🎯 タスク内容

Python が正しく動作するか、簡単なコードで確認します。

手順

1. 最初のセルに以下のコードを入力：

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

2. セルの左にある▶ボタンをクリック（または Shift + Enter）
3. 実行結果を確認

✅ チェックポイント

- ☐ 「Hello, Machine Learning!」と表示された
- ☐ 「2 + 3 = 5」と表示された
- ☐ 「10 * 5 = 50」と表示された

🎉 成功！

上記が表示されれば、Python が正しく動作しています。次に進みましょう！

📋 記録欄：実行結果（スクリーンショットを貼るか、結果を記入）

2️⃣ データのロードと確認

AI の学習には「データ」が必要です。今回は、手書き数字の有名なデータセット **MNIST** を使用します。

📍 タスク３：必要なライブラリのインポート

🎯 タスク内容

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。

手順： 以下のコードセルを実行し、必要なライブラリをインポートしてください。

```
# 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
```

📍 タスク４：MNIST データのロードと構造確認

🎯 タスク内容

7 万枚の手書き数字データ（MNIST）を読み込みます。

手順： データをロードし、その構造（shape）を確認するコードを実行してください。

```
# 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)
```

💻 実行結果を記録：

特徴量データ (X) の形状:

正解ラベルデータ (y) の形状:

問 1： この結果から、データが何件あり、1 つの数字がいくつの数値（特徴量）で表現されているか答えてください。

データ件数:

1 つの数字を構成する特徴量（ピクセル数）:

📍 タスク５：訓練データとテストデータの分割

🎯 タスク内容

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。

```
# 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))
```

考察：

なぜ学習に使ったデータとは別のデータでテストする必要があるのでしょうか？

タスク 6: 手書き数字画像の表示

タスク内容

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。

```
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()
```

実行結果の観察：

表示された画像を見て、人間でも間違えそうな数字や、特徴的な書き方の数字はありましたか？

タスク 5: データの分布確認

タスク内容

各数字（0〜9）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。

手順： 最初の数字（インデックス 0）を 28x28 の画像として表示するコードを実行してください。

```
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()
```

🖥️ **実行結果を記録：**

グラフを見て、特定の数字だけ極端に少なかったり多かったりしますか？

☑ Part 1 完了チェック

- ☐ Google Colab でコードを実行できた。
- ☐ MNIST データをロードし、形状（70000, 784）を確認した。
- ☐ データを訓練用とテスト用に分割できた。
- ☐ 16 枚の手書き数字画像を表示させた。
- ☐ ラベルの分布を棒グラフで確認した。

✅ Part 1 のまとめ

Part 1 では、以下のことを学びました：

- ✓ Google Colab の基本操作
- ✓ MNIST データセットの読み込み
- ✓ 訓練データとテストデータの分割
- ✓ 手書き数字画像の可視化
- ✓ データの基本的な統計情報の確認

☞ 次の Part に向けて

実習 1 Part 2 では、このデータを使って実際に機械学習モデルを作ります。
3 つの異なるアルゴリズムを試し、それぞれの特徴を比較します。

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習 1 Part 1）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part 2 （3 つのアルゴリズム実装）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 2 の学習目標

この Part では、次の項目を目標にします。

- ✓ **目標 1：**決定木アルゴリズムを実装し、動作を確認できる
- ✓ **目標 2：**k 近傍法アルゴリズムを実装し、動作を確認できる
- ✓ **目標 3：**ロジスティック回帰アルゴリズムを実装し、動作を確認できる
- ✓ **目標 4：**各アルゴリズムの学習プロセスと予測の流れを理解できる
- ✓ **目標 5：**3 つのアルゴリズムの特徴の違いを体験できる

💡 Part 2 について

今回は、Part 1 で準備したデータを使って、実際に機械学習モデルを作ります。3 つの異なるアルゴリズムを試すことで、それぞれの特徴を体験しましょう。コードはすべて用意してあるので、安心してください！

🤖 今回学ぶ 3 つのアルゴリズム

1️⃣ 決定木（Decision Tree）

特徴：質問を繰り返して答えを導くアルゴリズム
例：「この線は曲がっている？」→「はい」→「丸い部分がある？」→「はい」→「それは 8 か 9 です」
メリット：結果がわかりやすい、速い

2️⃣ k 近傍法（k-Nearest Neighbors, KNN）

特徴：「似ている仲間」を探して答えを決めるアルゴリズム
例：「この形に一番似ている数字は何だろう？」と近くの 5 個を見て多数決
メリット：シンプルでわかりやすい

3️⃣ ロジスティック回帰（Logistic Regression）

特徴：確率を計算して分類するアルゴリズム
例：「この形が 3 である確率は 80%」のように数値で判断
メリット：確率がわかる、計算が速い

📍 タスク 1：決定木による分類

🎯 タスク内容

決定木アルゴリズムを使って、手書き数字を認識するモデルを作成します。

手順

1. Part 1 のノートブックを開く（または新しいセルを追加）
2. 以下のコードを入力して実行：

```
# 決定木モデルのインポート
from sklearn.tree import DecisionTreeClassifier import time

# モデルの作成
print("📊 決定木モデルを作成します...")
dt_model = DecisionTreeClassifier( max_depth=10, # 木の深さ（質問の回数）
random_state=42 # 結果を再現可能にする )

# 学習の開始（時間を測定）
print("⏱️ 学習を開始します...")
start_time = time.time()
dt_model.fit(X_train, y_train)
end_time = time.time()
training_time = end_time - start_time
print(f"☑️ 学習が完了しました！")
print(f" 学習時間: {training_time:.2f}秒")
```

3. 実行して結果を確認
4. 学習時間を記録

✅ チェックポイント

☐ エラーが出なかった
☐ 「学習が完了しました！」と表示された
☐ 学習時間が表示された

💡 コードの説明

- max_depth=10：質問を最大 10 回まで（深く考えすぎないように）

- random_state=42：毎回同じ結果になるようにする
- fit()：「学習する」という命令
- time.time()：時間を測る

予測してみよう

学習したモデルで、テストデータの予測をしてみましょう：

```
# 予測を実行
print("🤖 予測を実行します...")
dt_predictions = dt_model.predict(X_test)
# 最初の10件の予測結果を表示
print("¥n📋 最初の10件の予測結果:")
print(" 予測:", dt_predictions[:10])
print(" 正解:", y_test[:10].values)
```

✅ チェックポイント

- ☐ 予測結果が表示された
- ☐ 正解と比較できた
- ☐ いくつか当たっている

📋 記録欄：決定木の結果

項目	記録
学習時間	秒
最初の 10 件で何個正解？	個 / 10 個
気づいたこと	

📍 タスク 2：k 近傍法による分類

🎯 タスク内容

k 近傍法アルゴリズムを使って、同じデータで学習・予測してみます。

手順

1. **1** 新しいセルに以下のコードを入力：

```
# k 近傍法モデルのインポート
from sklearn.neighbors import KNeighborsClassifier

# モデルの作成
print("📊 k 近傍法モデルを作成します...")
knn_model = KNeighborsClassifier( n_neighbors=5 # 近くの 5 つを見て多数決 )

# 学習の開始（時間を測定）
print("⏱️ 学習を開始します...")
start_time = time.time()
knn_model.fit(X_train, y_train)
end_time = time.time()
training_time = end_time - start_time
print(f"☑️ 学習が完了しました！")
print(f" 学習時間: {training_time:.2f}秒")
```

✅ チェックポイント

- ☐ エラーが出なかった
- ☐ 学習が完了した
- ☐ 決定木と学習時間を比較できた

💡 n_neighbors とは？

「近くの何個を見るか」を決める数字です。5 にすると、一番似ている 5 つの数字を探して、多数決で決めます。
例：5 つのうち、3 個が「7」、2 個が「1」なら、答えは「7」になります。

予測してみよう

```
# 予測を実行
print("🔮 予測を実行します...")
knn_predictions = knn_model.predict(X_test)

# 最初の 10 件の予測結果を表示
```

```
print("¥n📋 最初の 10 件の予測結果:")
print(" 予測:", knn_predictions[:10])
print(" 正解:", y_test[:10].values)
```

📋 記録欄：k 近傍法の結果

項目	記録
学習時間	秒
最初の 10 件で何個正解？	個 / 10 個
決定木との違い	

📍 タスク 3：ロジスティック回帰による分類

🎯 タスク内容

3 つ目のアルゴリズム、ロジスティック回帰を試してみます。

手順

```
# ロジスティック回帰モデルのインポート
from sklearn.linear_model import LogisticRegression

# モデルの作成
print("📊 ロジスティック回帰モデルを作成します...")
lr_model = LogisticRegression( max_iter=100, # 計算の繰り返し回数
random_state=42 ) # 学習の開始（時間を測定）
print("⏱️ 学習を開始します...")
start_time = time.time()
lr_model.fit(X_train, y_train)
end_time = time.time()
training_time = end_time - start_time
print(f"☑️ 学習が完了しました！")
print(f" 学習時間: {training_time:.2f}秒")
```

⚠️ 警告メッセージが出るかもしれません

「ConvergenceWarning」という警告が出ることがありますが、動作には問題ありません。もっと長く計算すればより正確になる、という意味です。

予測してみよう

```
# 予測を実行
print("🔮 予測を実行します...")
lr_predictions = lr_model.predict(X_test) # 最初の 10 件の予測結果を表示
print("¥n📋 最初の 10 件の予測結果:")
print(" 予測:", lr_predictions[:10])
print(" 正解:", y_test[:10].values)

# 確率も見てみよう
print("¥n📊 最初の 1 件の予測確率:")
probabilities = lr_model.predict_proba(X_test[:1])[0]
for digit in range(10): print(f" {digit}: {probabilities[digit]*100:.1f}%")
```

💡 確率がわかる！

ロジスティック回帰は、「この数字である確率」を教えてくれます。例えば「3 である確率は 85%」のように。これは他のアルゴリズムにはない特徴です。

📋 記録欄：ロジスティック回帰の結果

項目	記録
学習時間	秒
最初の 10 件で何個正解？	個 / 10 個
最初の 1 件で一番高い確率	%
気づいたこと	

📍 タスク 4： 3 つのアルゴリズムを比較

🎯 タスク内容

3 つのアルゴリズムの学習時間と予測結果を比較してみましょう。

比較表を作ろう

ワークブックの記録欄を使って、以下の表を完成させてください：

アルゴリズム	学習時間	10 件の正解数	印象・気づき
決定木			
k 近傍法			
ロジスティック回帰			

🤔 考察問題

❓ 問題 1

3 つのアルゴリズムで、学習時間に違いはありましたか？その理由を考えてみましょう。

💭 **あなたの答え：**

❓ 問題 2

最初の 10 件の予測で、一番正解数が多かったアルゴリズムはどれでしたか？なぜそうなったと思いますか？

💡 ヒント

アルゴリズムごとに「得意な判断方法」が違います。

💭 **あなたの答え：**

❓ 問題 3

ロジスティック回帰だけ「確率」が表示されました。この情報はどんな時に役立つと思いますか？

💭 **あなたの答え：**

✅ Part 2 のまとめ

Part 2 では、以下のことを学びました：

- ✓ 決定木アルゴリズムの実装と予測
- ✓ k 近傍法アルゴリズムの実装と予測
- ✓ ロジスティック回帰アルゴリズムの実装と予測
- ✓ 各アルゴリズムの特徴と違い
- ✓ 機械学習の基本的な流れ（学習→予測）

次の Part に向けて

Part 3 では、これら 3 つのモデルの性能を詳しく評価します。正解率を計算したり、どの数字の組み合わせで間違えやすいかを分析したりします。

📝 確認事項

- ☐ 3 つのアルゴリズムをすべて実装した
- ☐ 各アルゴリズムの結果を記録した
- ☐ 比較表を完成させた
- ☐ 考察問題に答えた
- ☐ ノートブックを保存した

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習 1 Part 2）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part 3（結果の比較と評価）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 3 の学習目標

この Part では、次の項目を目標にします。

- ✓ 目標 1：各モデルの正解率を正確に計算できる
- ✓ 目標 2：混同行列を作成し、間違えやすいパターンを分析できる
- ✓ 目標 3：3 つのアルゴリズムを多角的に比較できる
- ✓ 目標 4：最適なアルゴリズムを選定し、理由を説明できる

📍 タスク 1：正解率の計算

🎯 タスク内容

3 つのモデルの正解率（Accuracy）を計算し、比較します。

```
# 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("📊 各モデルの正解率:")
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}%")
# 一番良いモデルは？
best_model = max([ ("決定木", dt_accuracy), ("k 近傍法", knn_accuracy), ("ロジスティック回帰", lr_accuracy) ], key=lambda x: x[1])
print(f"¥n🏆 最も正解率が高いのは: {best_model[0]} ({best_model[1]*100:.2f}%)")
```

✅ チェックポイント

3 つの正解率が表示され、最も高いモデルが表示された

モデル	正解率
決定木	%
k 近傍法	%
ロジスティック回帰	%

📍 タスク 2：結果の可視化

🎯 タスク内容

正解率を棒グラフで表示して、視覚的に比較します。

```
# グラフで比較
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.figure(figsize=(10, 6))
bars = plt.bar(models, accuracies, color=['#2ecc71', '#3498db', '#e74c3c'])
plt.ylabel('正解率', fontsize=12)
plt.title('3 つのアルゴリズムの性能比較', fontsize=14) plt.ylim(0.7, 1.0) # 70%〜100%
の範囲で表示
# 各棒の上に数値を表示
for i, bar in enumerate(bars): height = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2., height,
f'{accuracies[i]*100:.2f}%', ha='center', va='bottom', fontsize=11)
plt.grid(axis='y', alpha=0.3) plt.show()
```

📍 タスク 3：混同行列の作成

🎯 タスク内容

混同行列を作成して、どの数字のペアで間違えやすいかを分析します。

💡 混同行列とは？

「どの数字をどの数字と間違えたか」を表にしたものです。例えば「7 を 1 と間違えた」などが分かります。

```
# 混同行列の作成（決定木の例）
from sklearn.metrics import confusion_matrix import seaborn as sns
# 決定木の予測結果で混同行列を作成
cm = confusion_matrix(y_test, dt_predictions)
# ヒートマップで表示
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', fontsize=12)
plt.ylabel('実際の数字', fontsize=12)
plt.title('混同行列（決定木）', fontsize=14)
plt.show() # 間違えやすいペアを探す
print("¥n🔍 よく間違えるパターン（上位 5 つ）:")
mistakes = [] for i in range(10): for j in range(10): if i != j and cm[i][j]
> 0: mistakes.append((i, j, cm[i][j]))
mistakes.sort(key=lambda x: x[2], reverse=True)
for i, (true_label, pred_label, count) in enumerate(mistakes[:5]): print(f"
{i+1}. {true_label}を{pred_label}と間違えた: {count}回")
```

📋 よく間違えるパターン:

1. ___ を ___ と間違えた
2. ___ を ___ と間違えた
3. ___ を ___ と間違えた

🧐 考察問題

❓ 問題 1

3 つのモデルで、正解率に大きな差はありましたか？その理由を考えてみましょう。

❓ 問題 2

混同行列を見て、どの数字のペアが間違えやすかったですか？なぜそうなると思いますか？

❓ 問題 3

あなたが最も良いと思うアルゴリズムはどれですか？その理由を 3 つ挙げてください。

✅ Part３のまとめ

Part 3 では、モデルの評価方法を学びました。
次の Part 4 では、自分で撮影した数字で実際にテストします。

☑ 実習振り返りシート （人工知能（AI）技術応用 - 実習 1 Part 3）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part 4 （実データでの検証）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 4 の学習目標

この Part では、次の項目を目標にします。

- ✓ **目標 1：**自分で手書き数字を用意し、画像として取り込める
- ✓ **目標 2：**画像を適切に前処理（リサイズ、グレースケール変換）できる
- ✓ **目標 3：**学習したモデルで実データを予測できる
- ✓ **目標 4：**実習全体を振り返り、学びをまとめられる

📍 タスク 1：手書き数字の準備

🎯 タスク内容

自分で数字を書いて、AI に認識させてみましょう。

手順

1. 白い紙に 0 から 9 の数字を書く（各 1-2 個ずつ）
2. スマホのカメラで撮影（できるだけ明るい場所で）
3. Google Colab にアップロード

💡 きれいに撮影するコツ

- 白い紙に黒いペンではっきり書く
- 数字を大きく書く
- 影ができないように撮影する
- 真上から撮影する

📍 タスク 2：画像の前処理

🎯 タスク内容

撮影した画像を AI が認識できる形式に変換します。

```
# 画像処理ライブラリのインポート
from PIL import Image import numpy as np
# 画像をアップロード（Colab の左サイドバーから）
# アップロードした画像のパスを指定
image_path = "your_image.jpg" # ここを変更
# 画像を読み込み
img = Image.open(image_path)
# グレースケールに変換
img_gray = img.convert('L')
# 28x28 ピクセルにリサイズ
img_resized = img_gray.resize((28, 28))
# 反転（黒背景に白文字にする）
img_array = np.array(img_resized)
img_inverted = 255 - img_array
# 正規化（0-255 を 0-1 に）
img_normalized = img_inverted / 255.0
# 確認のため表示
plt.imshow(img_normalized, cmap='gray')
plt.title('前処理後の画像')
plt.show()
print(f"画像のサイズ: {img_normalized.shape}")
print(f"データの範囲: {img_normalized.min():.2f} 〜 {img_normalized.max():.2f}")
```

📍 タスク 3：AI に予測させてみよう

🎯 タスク内容

前処理した画像で、3 つのモデルに予測させます。

```
# 予測用にデータを整形
img_for_prediction = img_normalized.flatten().reshape(1, -1)
# 3 つのモデルで予測
dt_pred = dt_model.predict(img_for_prediction)[0]
knn_pred = knn_model.predict(img_for_prediction)[0]
lr_pred = lr_model.predict(img_for_prediction)[0]
print("🔮 予測結果:")
print(f" 決定木: {dt_pred}")
print(f" k 近傍法: {knn_pred}")
print(f" ロジスティック回帰: {lr_pred}") # 確率も見てみる（ロジスティック回帰）
probabilities = lr_model.predict_proba(img_for_prediction)[0]
print(f"¥n 確率が最も高い数字: {lr_pred}
({probabilities[int(lr_pred)]*100:.1f}%)")
```

💡 結果の見方

3 つのモデルが同じ答えなら、信頼度が高いです。バラバラなら、書き方が特殊だったのかもしれません。

📋 予測結果の記録:

書いた数字	決定木	k 近傍法	ロジ回帰	正解

📍 タスク 4：最終レポートの作成

🎯 タスク内容

実習全体を振り返り、学んだことをまとめます。

レポートに含める内容：

1. **実習の概要：**何をしたか（簡潔に）
2. **3 つのアルゴリズムの比較：**正解率、特徴
3. **実データでの結果：**うまくいったか、失敗したか
4. **学んだこと：**機械学習について理解したこと
5. **感想：**難しかった点、面白かった点

📄 最終レポート

（ここに記入するか、別途 Word/PDF で作成）

🎓 実習 1 全体のまとめ

達成したこと：

- ✅ Google Colab でプログラミング環境を構築
- ✅ 70,000 枚の手書き数字データを準備
- ✅ 3 つの機械学習アルゴリズムを実装
- ✅ モデルの性能を評価・比較
- ✅ 自分のデータで AI をテスト

これで、あなたも機械学習エンジニアの第一歩を踏み出しました！

次のステップ

もっと学びたい方は：

- 発展課題にチャレンジ
- Fashion-MNIST など他のデータセットを試す
- ディープラーニング（CNN）を学ぶ
- Kaggle のコンペティションに参加

✅ 提出前チェックリスト

✓	項目
	Part 1-4 のワークブックがすべて記入されている
	Colab ノートブックに実行結果が残っている
	3 つのアルゴリズムの正解率を記録した
	自分の手書き数字でテストした
	最終レポートを作成した
	考察問題にすべて答えた
	ノートブックの URL またはファイルを共有準備した

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習 1 Part 4）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

データ人工知能（AI）技術応用
実習 2 画像認識 AI で犬🐶と猫🐱を分類しよう
実習ガイド

📋 実習 2 の概要

🎯 何を作るの？

犬と猫の写真を見て、自動的に「これは犬」「これは猫」と分類する AI を作ります。 実習 1 では手書き数字でしたが、今回は**カラー写真**を扱います。 また、実習 1 の機械学習アルゴリズムではなく、**ディープラーニング（CNN）**という最新技術を使います。

実習 1 との違い

項目	実習 1（手書き数字）	実習 2（犬猫分類）
データ	28×28 の白黒画像	150×150 のカラー画像
分類数	10 種類（0-9 の数字）	2 種類（犬・猫）
アルゴリズム	決定木、k 近傍法、ロジスティック回帰	CNN（畳み込みニューラルネットワーク）

🎯 実習 2 の学習目標

🧠 ディープラーニングの理解

CNN の基本構造と仕組みを理解する

🖼️ 画像処理の技術

カラー画像の扱い方とデータ拡張を学ぶ

📊 学習プロセスの可視化

学習曲線から過学習を検出する

🔧 実用的な AI 開発

モデルの保存・読み込みと転移学習を体験する

📚 実習 2 の構成（全 4 Part）

📗 Part 1：データ準備と CNN 入門

- Dogs vs Cats データセットのダウンロード
- 画像データの読み込みと前処理
- CNN の基本構造の理解
- データ拡張（Data Augmentation）の実装

成果物：前処理済みデータセット、データ拡張の実装

📘 Part 2：CNN モデルの構築と学習

- CNN モデルの設計（畳み込み層、プーリング層）
- モデルのコンパイルと学習
- 学習曲線の可視化と分析
- 過学習の検出と対策

成果物：学習済み CNN モデル、学習曲線グラフ

📙 Part 3：モデルの評価と改善

- テストデータでの性能評価
- 混同行列と分類レポート
- 間違えた画像の分析
- モデルの改善手法（ドロップアウト、正則化）

成果物：評価レポート、改善提案

📕 Part 4：転移学習と実データテスト

- VGG16 を使った転移学習
- ファインチューニング
- 自分の画像でテスト
- 最終レポートの作成

成果物：転移学習モデル、実データテスト結果、最終レポート

💻 必要な環境・準備

ソフトウェア環境

項目	内容
プラットフォーム	Google Colab（推奨）またはローカルの Jupyter Notebook
Python	3.7 以上
必要なライブラリ	TensorFlow 2.x, Keras, NumPy, Matplotlib, Seaborn
GPU	Google Colab の無料 GPU（T4）を使用（強く推奨）

データセット

- **Dogs vs Cats**（Kaggle データセット）
- 犬の画像：12,500 枚
- 猫の画像：12,500 枚
- 合計：25,000 枚のカラー画像
- サイズ：約 543MB（圧縮時）

⚠️ **注意：**

データセットのダウンロードには時間がかかります（5-10 分）。 授業前に事前ダウンロードしておくことを推奨します。

📖 前提知識

必須

- 実習 1（機械学習基礎）を完了していること
- Python の基本構文（変数、リスト、for 文など）
- Google Colab の基本操作
- 機械学習の基本概念（訓練データ、テストデータ、正解率など）

あると望ましい

- 行列の基本的な理解
- 画像処理の基礎知識
- ニューラルネットワークの概念

🏆 実習 2 完了後の到達目標

できるようになること

1. **CNN の構造を理解し、説明できる**
    - 畳み込み層、プーリング層、全結合層の役割
    - CNN が画像認識に適している理由
2. **画像分類モデルを 1 から作れる**
    - データの前処理
    - モデルの設計と学習
    - 性能評価と改善
3. **過学習を検出し、対策できる**
    - 学習曲線の読み方
    - ドロップアウト、データ拡張などの手法
4. **転移学習を活用できる**
    - 事前学習済みモデルの利用
    - 少ないデータでの高精度化
5. **実用的な AI システムの開発プロセスを理解している**
    - データ収集→モデル構築→評価→改善のサイクル
    - 実社会での応用例

🌎 実社会での応用例

🏥 医療分野

レントゲン画像から病変を検出、皮膚がんの診断支援

🚗 自動運転

歩行者・車両・信号の認識、車線検出

🏭 製造業

製品の不良品検出、品質管理の自動化

🛒 小売業

商品認識、在庫管理、顧客行動分析

🔒 セキュリティ

顔認証、不審者検知、監視カメラ分析

🌾 農業

作物の病害検出、収穫適期の判断

🚀 始める前に

実習 2 は、実習 1 よりも少し難しくなりますが、心配しないでください。 ディープラーニングという最新技術を使って、本格的な画像認識 AI を作ります。コードは難しく見えるかもしれませんが、一つ一つ丁寧に説明します。 分からないことがあったら、遠慮なく質問してください。自分で作った AI が、 犬と猫を見分けられるようになる瞬間を、楽しみにしていてください！

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）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 1 の学習目標

この Part では、次の項目を目標にします。

- ✓ **目標 1：**Dogs vs Cats データセットを準備できる
- ✓ **目標 2：**画像データの前処理方法を理解できる
- ✓ **目標 3：**CNN の基本構造を説明できる
- ✓ **目標 4：**AI 学習のための「前処理（正規化・データ拡張）」を行う

📍 タスク 1：Google Colab で GPU 設定

🎯 タスク内容

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

確認コード

```
# 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'))
```

✅ チェックポイント

- ☐ 「GPU available:」の後に何か表示される

📍 タスク 2：データセットのダウンロード

🎯 タスク内容

Dogs vs Cats データセットをダウンロードします（約 10 分）。

```
# 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")
```

💡 データセットの構造

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）

✅ チェックポイント

☐ エラーなくダウンロード完了

📍 タスク 3：画像の表示

🎯 タスク内容

ダウンロードした画像を表示してみます。

```
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()
```

✅ チェックポイント

- ☐ 猫 4 枚、犬 4 枚の画像が表示された

📍 タスク 4：画像の前処理

🎯 タスク内容

画像を 150×150 にリサイズし、0-1 の範囲に正規化します。

```
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")
```

💡 パラメータの説明

- `rescale=1./255`: 画素値を `0-1` に正規化
- `target_size=(150, 150)`: `150×150` にリサイズ
- `batch_size=20`: 一度に `20` 枚ずつ処理
- `class_mode='binary'`: `2` 値分類（犬 `or` 猫）

🧐 考察問題

❓ 問題 1

なぜ画像を 150×150 にリサイズする必要があるのですか？

❓ 問題 2

訓練データと検証データの違いは何ですか？

✅ Part 1 のまとめ

今日は、CNN で使う画像データの準備ができました。
次の Part 2 では、CNN モデルを作成して学習させます。

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習２ Part１）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

人工知能（AI）技術応用
実習２　画像認識 AI で犬と猫を分類
Part２（CNN モデルの構築と学習）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part２の学習目標

この Part では、次の項目を目標にします。

- ✓ 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

📍 タスク１：CNN モデルの構築

🎯 タスク内容

畳み込みニューラルネットワーク（CNN）を構築します。

```
from tensorflow.keras import layers, models
# CNN モデルの構築
model = models.Sequential([ # 第１畳み込み層 layers.Conv2D(32, (3, 3),
activation='relu', input_shape=(150, 150, 3)), layers.MaxPooling2D((2, 2)),
# 第２畳み込み層 layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)), # 第３畳み込み層 layers.Conv2D(128, (3, 3),
activation='relu'), layers.MaxPooling2D((2, 2)), # 第４畳み込み層
layers.Conv2D(128, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)),
# 全結合層 layers.Flatten(), layers.Dense(512, activation='relu'),
layers.Dense(1, activation='sigmoid') ])
# モデル構造の表示
model.summary()
```

💡 各層の役割

- **`Conv2D`：** 画像の特徴を抽出（エッジ、形など）
- **`MaxPooling2D`：** 画像を小さくして計算量削減
- **`Flatten`：** 2 次元を 1 次元に変換
- **`Dense`：** 分類を行う全結合層

✅ チェックポイント

☐ モデル構造が表示された
☐ 総パラメータ数が表示された

📋 記録：総パラメータ数 ＝ _____ 個

📍 タスク２：モデルのコンパイル

🎯 タスク内容

学習の設定（最適化手法、損失関数）を行います。

```
# モデルのコンパイル
model.compile( loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'] )
print("☑ コンパイル完了")
```

💡 設定の意味

- **`binary_crossentropy`：** 2 値分類用の損失関数
- **`adam`：** 効率的な最適化手法
- **`accuracy`：** 正解率を測定

📍 タスク３：モデルの学習

🎯 タスク内容

モデルを学習させます（約 10 分かかります）。

```
# モデルの学習
history = model.fit( train_generator, steps_per_epoch=100, epochs=15,
validation_data=validation_generator, validation_steps=50 )
print("☑ 学習完了")
```

💡 パラメータの説明

- **`steps_per_epoch`：** 1 エポックあたりのステップ数
- **`epochs`：** 全データを何回学習するか（15 回）
- **`validation_data`：** 検証用データ

✅ チェックポイント

☐ エポックごとの結果が表示された
☐ 最終的な正解率が表示された

📋 記録：

学習時間: _____ 分
最終の訓練正解率: _____ %
最終の検証正解率: _____ %

📍 タスク４：学習曲線の可視化

🎯 タスク内容

学習の過程をグラフで確認します。

```
import matplotlib.pyplot as plt
# 正解率のグラフ acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy over Epochs')
# 損失のグラフ
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.tight_layout()
plt.show()
```

💡 グラフの見方

- **理想的：** 訓練と検証が同じように上昇/下降
- **過学習：** 訓練は良いが、検証は悪化
- **未学習：** どちらも改善していない

🧐 考察問題

❓ 問題 1

学習曲線を見て、過学習は起きていますか？その理由を説明してください。

❓ 問題 2

訓練正解率と検証正解率に差がある場合、どうすれば改善できると思いますか？

✅ Part 2 のまとめ

CNN モデルを構築し、学習させることができました。

次の Part 3 では、モデルの性能を詳しく評価し、改善方法を学びます

☑ 実習振り返りシート （人工知能（AI）技術応用 - 実習２ Part２）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

人工知能（AI）技術応用
実習２　画像認識 AI で犬と猫を分類
Part３（モデルの評価と改善）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part３の学習目標

この Part では、次の項目を目標にします。

- ✓ テストデータで正確に性能評価できる
- ✓ AI の「間違い」を分析できる
- ✓ 混同行列や分類レポートを使い、多角的に評価できる
- ✓ 過学習への対策を理解する
- ✓ ドロップアウトと呼ばれる改善手法を実装できる

📍 タスク１：テストデータでの評価

🎯 タスク内容

検証データで最終的な性能を評価します。

```
# モデルの評価
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}%")
```

📋 記録：

テスト正解率: _____ %

📍 タスク２：予測結果の分析

🎯 タスク内容

間違えた画像を表示して分析します。

```
import numpy as np
# 予測の実行
validation_generator.reset() predictions =
model.predict(validation_generator, steps=50)
predicted_classes = (predictions > 0.5).astype(int).flatten()
# 正解ラベルの取得
true_classes = validation_generator.classes
# 間違えた画像のインデックス
wrong_indices = np.where(predicted_classes != true_classes)[0]
print(f"間違えた数: {len(wrong_indices)}枚 / {len(true_classes)}枚")
print(f"エラー率: {len(wrong_indices)/len(true_classes)*100:.2f}%")
# 間違えた画像を表示
if len(wrong_indices) > 0: import matplotlib.pyplot as plt fig, axes =
plt.subplots(2, 4, figsize=(12, 6))
for i, idx in enumerate(wrong_indices[:8]): img_path =
validation_generator.filepaths[idx]
from tensorflow.keras.preprocessing
import image
img = image.load_img(img_path)
ax = axes[i//4, i%4]
ax.imshow(img)
true_label = 'Dog'
if true_classes[idx] == 1 else 'Cat'
pred_label = 'Dog'
if predicted_classes[idx] == 1 else 'Cat'
ax.set_title(f'True: {true_label}¥nPred: {pred_label}')
ax.axis('off')
plt.tight_
layout()
plt.show()
```

📍 タスク３：改善版モデルの構築

🎯 タスク内容

ドロップアウトを追加して過学習を防ぎます。

```
# 改善版モデル（ドロップアウト追加）
model_improved = models.Sequential([ layers.Conv2D(32, (3, 3),
activation='relu', input_shape=(150, 150, 3)), layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)),
layers.Conv2D(128, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)),
layers.Conv2D(128, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)),
layers.Flatten(), layers.Dropout(0.5),
# ドロップアウト追加
layers.Dense(512, activation='relu'), layers.Dropout(0.5),
# ドロップアウト追加
layers.Dense(1, activation='sigmoid') ])
model_improved.compile( loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'] )
print("☑ 改善版モデル構築完了")
```

💡 ドロップアウトとは？

学習中にランダムにニューロンを無効化することで、モデルが特定のパターンに依存しすぎるのを防ぎます。 これにより過学習を抑制できます。

🤔 考察問題

❓ 問題 1

間違えた画像を見て、どのような特徴があると思いますか？

❓ 問題 2

ドロップアウト以外に、過学習を防ぐ方法はありますか？

✅ Part３のまとめ

モデルの性能評価と改善方法を学びました。
次の Part 4 では、転移学習を使ってさらに高精度なモデルを作ります。

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習２ Part３）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

人工知能（AI）技術応用
実習２　画像認識 AI で犬と猫を分類
Part４（転移学習と実データテスト）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part４の学習目標

この Part では、次の項目を目標にします。

- ✓ **目標 1：**転移学習(Transfer Learning)の概念を理解できる
- ✓ **目標 2：**事前学習済みモデル（VGG16）を使った転移学習を実装できる
- ✓ **目標 3：**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
- ✓ **目標 4：**自分の画像でテストできる

📍 タスク 1：VGG16 を使った転移学習

🎯 タスク内容

ImageNet で事前学習された VGG16 を使います。

```
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")
```

💡 転移学習とは？

大規模データセット（ImageNet：140 万枚）で学習済みのモデルを活用する手法です。 少ないデータでも高精度なモデルを作れます。

📍 タスク２：転移学習モデルの学習

🎯 タスク内容

転移学習モデルを学習させます。

```
# 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}%")
```

📋 比較記録：

モデル	正解率
通常の CNN	_____ %
転移学習	_____ %

📍 タスク３：自分の画像でテスト

🎯 タスク内容

自分で用意した犬・猫の画像で予測してみます。

```
from tensorflow.keras.preprocessing import image import numpy as np
# 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)
```

💡 画像の用意方法

1. 犬や猫の写真を撮影、またはネットから取得
2. Colab の左サイドバーからアップロード
3. 上記のコードでファイル名を指定

📍 タスク 4：最終レポート

🎯 タスク内容

実習 2 全体を振り返り、学びをまとめます。

レポートに含める内容：

1. 実習の概要（何をしたか）
2. CNN と転移学習の違い
3. 最も難しかった部分
4. 最も興味深かった発見
5. 実社会での応用アイデア

📄 最終レポート

（ここに記入、または別途作成）

🎓 実習 2 完了！

🎉 達成したこと：

- ✅ 2,000 枚の犬猫画像データを準備
- ✅ CNN モデルを 1 から構築
- ✅ 学習曲線から過学習を検出
- ✅ 転移学習で高精度化
- ✅ 自分の画像で AI をテスト

これで、ディープラーニングの基礎を習得しました！

次のステップ

- 発展課題にチャレンジ
- 他の画像分類タスクに挑戦（花、動物など）
- 物体検出（YOLO）を学ぶ

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習２ Part４）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

データ人工知能（AI）技術応用
実習３　自然言語処理で感情分析 AI を作ろう
実習ガイド

📋 実習 3 の概要

🎯 何を作るの？

映画のレビューや商品の口コミを読んで、それが**ポジティブ（良い評価）**なのか **ネガティブ（悪い評価）**なのかを自動判定する AI を作ります。 実習 1・2 では画像や数字でしたが、今回は**テキストデータ**を扱います。

これまでの実習との違い

項目	実習 1（数字）	実習 2（画像）	実習 3（テキスト）
データ	28×28 の数値	150×150 の画像	可変長のテキスト
分類	10 種類（0-9）	2 種類（犬・猫）	2 種類（ポジ・ネガ）
手法	機械学習	CNN	RNN/LSTM/Transformer
前処理	正規化	リサイズ	トークン化、埋め込み

🎯 実習 3 の学習目標

テキストデータの理解

自然言語処理の基礎とテキストの前処理方法

単語の数値化

単語埋め込み（Word Embedding）の概念と実装

系列データの処理

RNN、LSTM による時系列データの扱い

最新技術の活用

事前学習モデル（BERT）の転移学習

📚 実習 3 の構成（全 4 Part）

📗 Part 1：テキストデータの準備と前処理

- IMDb レビューデータセットの読み込み
- テキストのトークン化（単語分割）
- パディング（長さの統一）
- 単語埋め込みの理解

成果物：前処理済みテキストデータ、単語辞書

📘 Part 2：RNN/LSTM モデルの構築

- Embedding レイヤーの実装
- LSTM モデルの構築
- モデルの学習と評価
- 注意機構（Attention）の追加

成果物：感情分析 LSTM モデル、学習曲線

📙 Part 3：モデルの評価と可視化

- テストデータでの性能評価
- 混同行列と分類レポート
- 重要な単語の可視化
- 誤分類の分析

成果物：評価レポート、単語重要度の可視化

📕 Part 4：事前学習モデル（BERT）の活用

- BERT の概要と仕組み
- Hugging Face Transformers の使用
- ファインチューニング
- 実データでのテスト

成果物：BERT 感情分析モデル、最終レポート

💻 必要な環境・準備

ソフトウェア環境

項目	内容
プラットフォーム	Google Colab（推奨）
Python	3.7 以上
必要なライブラリ	TensorFlow, Keras, Transformers, NLTK, NumPy, Matplotlib
GPU	推奨（BERT のファインチューニングに必要）

データセット

- **IMDb Movie Reviews**（映画レビューデータ）
- ポジティブレビュー：25,000 件
- ネガティブレビュー：25,000 件
- 合計：50,000 件の英語テキスト
- サイズ：約 84MB

📖 前提知識

必須

- 実習 1・2 を完了していること
- Python の基本構文
- ディープラーニングの基礎（実習 2 で学習済み）
- 英語の基本的な読解力（データが英語）

あると望ましい

- 自然言語処理の基礎知識
- RNN や LSTM の概念
- Transformer の基本的な理解

🏆 到達目標

1. **テキストデータの前処理ができる**
    - トークン化、パディング、埋め込みの実装
    - データクリーニングの重要性

2. **RNN/LSTM モデルを理解し実装できる**
    - 系列データの処理方法
    - 長期依存性の問題と解決策
3. **感情分析システムを構築できる**
    - モデルの設計から評価まで
    - 実用的なシステムの開発
4. **事前学習モデルを活用できる**
    - BERT の仕組みの理解
    - ファインチューニングの実践
5. **自然言語処理の応用分野を理解している**
    - 実社会での活用例
    - 今後の発展可能性

🌎実社会での応用例

🛍️ EC サイト

商品レビューの自動分析、顧客満足度の測定

📱 SNS 分析

ブランドイメージ分析、炎上の早期検知

📞 カスタマーサポート

問い合わせの自動分類、緊急度判定

📰 メディア分析

ニュース記事の感情分析、世論調査

💼 金融

市場センチメント分析、投資判断の補助

🏥 医療

患者フィードバックの分析、メンタルヘルスの監視

🚀始める前に

実習 3 では、これまでの画像や数字とは違う**テキストデータ**を扱います。「コンピュータはどうやって言葉を理解するのか？」という疑問に答えながら、 実用的な感情分析 AI を作ります。

テキスト処理は最初は難しく感じるかもしれませんが、一つ一つのステップを 丁寧に説明します。完成した AI に自分の文章を入力して、反応を見るのは とても面白い体験です！

自然言語処理は、ChatGPT などの最新 AI 技術の基礎にもなっています。 この実習を通じて、AI がどのように言葉を扱うのか、その仕組みを学びましょう。

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
Part 1 （環境準備とテキスト前処理の基礎）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 1 の学習目標

この Part では、次の項目を目標にします。

- ✓ **目標 1：**テキストデータの前処理方法を理解する
- ✓ **目標 2：**自然言語処理モデルを構築できる
- ✓ **目標 3：**感情分析の仕組みを説明できる
- ✓ **目標 4：**実用的な AI システムを開発できる

📍 タスク 1：データセットの読み込み

🎯 タスク内容

IMDb 映画レビューデータセットを読み込みます。

```
# IMDb データセットの読み込み
from tensorflow.keras.datasets
import imdb
# データのダウンロード
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
print(f"訓練データ: {len(X_train)}件")
print(f"テストデータ: {len(X_test)}件")
print(f"最初のレビュー（数値化済み）: {X_train[0][:10]}...")
print(f"ラベル（0=ネガティブ, 1=ポジティブ）: {y_train[0]}")
```

💡 num_words=10000 とは？

最も頻出する 10,000 語だけを使用します。これにより、データサイズを削減できます。

✅ チェックポイント

☐　データが正しく読み込まれた

☐　25,000 件の訓練データがある

📍 タスク２：テキストの前処理

🎯 タスク内容

可変長のテキストを固定長に統一します（パディング）。

```
from tensorflow.keras.preprocessing import sequence
# 最大長を 250 単語に設定
max_length = 250
# パディング（短い文は 0 で埋める、長い文は切り捨てる）
X_train = sequence.pad_sequences(X_train, maxlen=max_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_length)
print(f"パディング後の形状: {X_train.shape}")
print(f"最初のレビュー: {X_train[0]}")
print(f"パディングにより、すべてのレビューが{max_length}単語になりました")
```

📋 記録：

データの形状: (_____ , _____)

📍 タスク３：モデルの構築

🎯 タスク内容

感情分析用の LSTM モデルを構築します。

```
from tensorflow.keras import models, layers
# モデルの構築
model = models.Sequential([ layers.Embedding(10000, 128,
input_length=max_length), layers.LSTM(128, dropout=0.2,
recurrent_dropout=0.2), layers.Dense(1, activation='sigmoid') ])
model.compile( optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'] )
model.summary()
```

💡 各層の役割

- **Embedding:** 単語を 128 次元のベクトルに変換
- **LSTM:** 文章の流れを理解（128 ユニット）
- **Dense:** ポジティブ/ネガティブを判定

📍 タスク４：モデルの学習

🎯 タスク内容

モデルを学習させます（約 10-15 分）。

```
# モデルの学習
history = model.fit( X_train, y_train, epochs=5, batch_size=128,
validation_split=0.2 )
print("☑ 学習完了")
```

📋 記録：

最終の訓練正解率: _____ %
最終の検証正解率: _____ %

🧐 考察問題

❓ 問題 1

なぜテキストを数値化する必要があるのですか？

❓ 問題 2

LSTM はどのような特徴を持っていますか？

✅ Part 1 のまとめ

Part 1 では、自然言語処理の基礎を学びました。
次の Part では、さらに高度な技術を学びます。

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習 3 Part 1）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part２（RNN/LSTM モデルの構築）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part２の学習目標

この Part では、次の項目を目標にします。

- **目標 1：**単語埋め込み（Embedding）レイヤーの役割を理解し、実装できる
- **目標 2：**SimpleRNN モデルを構築し、系列データ処理の基本を理解する
- **目標 3：**LSTM モデルを構築し、SimpleRNN との違い（長期記憶）を理解する
- **目標 4：**モデルをコンパイルし、学習させる（.fit()）ことができる
- **目標 5：**学習曲線（Accuracy, Loss）をプロットし、結果を観察できる

📖 Part 2 について

Part 1 では、テキストデータを AI が読める「数値の列」（パディング済み）に変換しました。
Part 2 では、いよいよ AI モデル、特にテキストのような「順序」が重要なデータ（＝系列データ）を扱うのが得意な **RNN (Recurrent Neural Network)** とその発展形である **LSTM (Long Short-Term Memory)** を構築し、学習させます。

🔄 タスク 1：準備（Part 1 のデータ引き継ぎ）

📌 Task 1-1: ライブラリのインポートとデータの復元

Part 1 と同様のライブラリと、Part 1 で作成した前処理済みデータ（X_train, y_train, X_test, y_test）を準備します。

```
# 必要なライブラリのインポート
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras import models, layers

# --- Part 1 のコード（データの読み込みと前処理） ---
# （ノートブックを再起動した場合などは、ここで Part 1 のコードを再実行してデータを用意する）

# 例：
# (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
# max_length = 250
# X_train = sequence.pad_sequences(X_train, maxlen=max_length)
# X_test = sequence.pad_sequences(X_test, maxlen=max_length)
# print("データ準備完了")
# print(f"X_train shape: {X_train.shape}")
# print(f"y_train shape: {y_train.shape}")

# --- ここでは、X_train, y_train, X_test, y_test が準備できているものとして進めます ---
# （もし変数が存在しないエラーが出たら、上のコメントアウトを外して実行してください）
```

🧠 タスク 2：LSTM モデルの構築とコンパイル

テキストの「順序」を記憶しながら学習できる LSTM モデルを構築します。これは Part 1 の最後と重複しますが、各層の役割を再確認します。

📌 Task 2-1: モデルの定義

Keras の Sequential API を使って、層を積み重ねてモデルを定義します。

```
# 使用する単語数（語彙数）
max_features = 10000
# パディングした文の長さ
max_length = 250

model = models.Sequential()

# 1. Embedding レイヤー（単語埋め込み層）
# 10000 個の単語を、それぞれ 128 次元のベクトルに変換する
model.add(layers.Embedding(max_features, 128, input_length=max_length))

# 2. LSTM レイヤー（系列処理層）
# 128 個のユニット（ニューロン）で文章の文脈を学習する
# dropout=0.2, recurrent_dropout=0.2 は過学習を防ぐためのおまじない
model.add(layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))

# 3. Dense レイヤー（出力層）
# 最終的に「ポジティブ(1)」か「ネガティブ(0)」かを判定する
# 2 値分類なので、activation='sigmoid' を使う
model.add(layers.Dense(1, activation='sigmoid'))

print("モデルの定義完了。")
```

💡 Embedding レイヤーとは？

「単語をベクトルに変換する」層です。 コンピュータは「good」や「bad」という単語を直接理解できません。Embedding 層は、これらの単語を「意味が近い単語は、ベクトル空間上でも近い位置に来る」ような数値のベクトル（今回は 128 次元）に変換する役割を持ち、この変換ルールも学習中に最適化されます。

📌 Task 2-2: モデルのコンパイル

モデルに「学習の方法」を教えます（損失関数、最適化手法、評価指標）。

```
model.compile(
    optimizer='adam',            # 最適化手法（Adam が一般的）
```

```
    loss='binary_crossentropy',    # 損失関数（2値分類の標準）
    metrics=['accuracy']           # 評価指標（今回は「正解率」）
)

print("モデルのコンパイル完了。")

# モデルの構造をサマリー（概要）として表示
model.summary()
```

モデル構造の記録

model.summary() の結果を見て、各層のパラメータ数を確認し、Total params（総パラメータ数）を記録してください。

🏃 タスク３：モデルの学習

定義したモデルに、訓練データ（X_train, y_train）を与えて学習（fit）させます。

📌 Task 3-1: 学習の実行

.fit() を実行します。**GPU が有効になっていることを確認してください。**

⚠️注意： 学習には時間がかかります（Colab の T4 GPU で 約 5〜15 分）。

```
%%time
# ↑ セル全体の実行時間を計測

print("モデルの学習を開始します... （約 5〜15 分かかります）")

# 学習の実行
history = model.fit(
    X_train, y_train,
    epochs=5,           # 訓練データを 5 周学習する
    batch_size=128,      # 128 件ずつまとめて処理する
    validation_split=0.2 # 訓練データのうち 20%を検証用として使用する
)

print("☑ 学習完了！")
```

⚠️ GPU の確認

Colab のメニュー「ランタイム」→「ランタイムのタイプを変更」で「T4 GPU」が選択されていることを確認してください。
CPU だと数時間かかる可能性があります。

📝 学習結果の記録

Epoch 5/5 の最後に出力された 4 つの値を記録してください。

📊 タスク 4：学習曲線の可視化

学習の過程（history）をグラフ化して、モデルがうまく学習できたか（過学習していないか）を確認します。

📌 Task 4-1: 正解率（Accuracy）の推移

```
# history オブジェクトから学習結果を取得
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)

# グラフ描画
plt.figure(figsize=(10, 5))
plt.plot(epochs, acc, 'bo-', label='Training accuracy (訓練)')
plt.plot(epochs, val_acc, 'ro-', label='Validation accuracy (検証)')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs (学習回数)')
plt.ylabel('Accuracy (正解率)')
plt.legend()
plt.grid(True)
plt.show()
```

📌 Task 4-2: 損失（Loss）の推移

```
# 損失の推移
loss = history.history['loss']
val_loss = history.history['val_loss']

# グラフ描画
plt.figure(figsize=(10, 5))
plt.plot(epochs, loss, 'bo-', label='Training loss (訓練)')
plt.plot(epochs, val_loss, 'ro-', label='Validation loss (検証)')
plt.title('Training and validation loss')
plt.xlabel('Epochs (学習回数)')
plt.ylabel('Loss (損失)')
plt.legend()
plt.grid(True)
plt.show()
```

🤔 Part 2　考察問題

作成した 2 つのグラフ（学習曲線）を見て、以下の問いについて考え、記録してください。

問 1： 正解率（Accuracy）のグラフを見てください。「訓練データ (Training)」と「検証データ (Validation)」の正解率は、エポック（学習回数）が進むにつれてどのように変化しましたか？

問 2： 損失（Loss）のグラフを見てください。「訓練データ」と「検証データ」の損失は、どのように変化しましたか？

問 3： 問 1 と問 2 の結果から、このモデルは「過学習（Overfitting）」を起こしていると考えられますか？ 理由とともに説明してください。（ヒント：過学習とは、訓練データに最適化されすぎて、未知のデータ（検証データ）に対する性能が悪化することです）

✅ Part 2　のまとめ

🎉 **お疲れ様でした！**

Part 2 では、RNN/LSTM モデルを構築し、学習させ、その結果を可視化・考察しました。

📚 **今日学んだこと**

- 単語埋め込み (Embedding) レイヤーの役割
- LSTM モデルの構築とコンパイルの方法
- モデルの学習（.fit()）と学習時間の長さ（GPU の必要性）
- 学習曲線の可視化と、そこから「過学習」を読み取る方法

➡️ **次のステップ：Part 3**

Part 3 では、学習させたモデルを使って、未知の「テストデータ」に対する最終的な性能を評価します。また、AI が「どの単語」に注目してポジティブ/ネガティブを判断したのかを可視化するテクニックにも挑戦します！

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習３ Part２）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part３（モデルの評価と可視化）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part３の学習目標

この Part では、次の項目を目標にします。

- **目標 1**:学習済みモデルを使い、テストデータで最終的な性能評価（正解率）ができる
- **目標 2**:「混同行列」を作成し、AI がどのような間違い方をしているか分析できる
- **目標 3**:「分類レポート」を読み解き、Precision, Recall, F1-score の意味を理解する
- **目標 4**:実際に AI が間違えたレビューを確認し、その原因を考察できる

📖 Part 3 について

Part 2 では、LSTM モデルを「訓練データ」と「検証データ」を使って学習させ、過学習の兆候を見ました。

Part 3 では、モデルが一度も見たことのない**「テストデータ（X_test, y_test）」**を使い、モデルの**最終的な実力**を評価します。正解率だけではなく、「どのような間違いをしたのか」を詳しく分析するのがゴールです。

🔄 タスク１：準備（モデルとデータの引き継ぎ）

📌 Task 1-1: ライブラリのインポートとデータの復元

Part 2 で学習させたモデル(model)と、Part 1 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

# --- ここに Part 1, 2 のコードを実行して model, X_test, y_test を作成する ---
# （ノートブックを再起動した場合などは、ここで Part 1, 2 のコードを再実行する）

# 例：
# (X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=10000)
# max_length = 250
# X_train = sequence.pad_sequences(X_train, maxlen=max_length)
# X_test = sequence.pad_sequences(X_test, maxlen=max_length)
# model = models.Sequential([...]) # Part 2 のモデル定義
# model.compile(...)
# history = model.fit(...) # Part 2 の学習
# print("データと学習済みモデルの準備完了")

# --- ここでは、model, X_test, y_test が準備できているものとして進めます ---
# （もし変数が存在しないエラーが出たら、上のコメントアウトを外して実行してください）
```

💯 タスク２：テストデータによる最終評価

学習済みモデルの「本当の実力」を、未知のテストデータで評価します。

📌 Task 2-1: テストデータでの正解率を計算

.evaluate() メソッドを使って、テストデータでの損失（Loss）と正解率（Accuracy）を計算します。

```
print("テストデータでの最終評価を開始します...")
start_time = time.time() # 参考：評価にも時間がかかる

# evaluate でテストデータ (X_test, y_test) に対する損失と正解率を計算
test_loss, test_accuracy = model.evaluate(X_test, y_test,
verbose=1)

end_time = time.time()
print(f"評価完了 ({end_time - start_time:.2f} 秒)")

print("¥n--- 最終評価結果 ---")
print(f"テスト損失 (Test Loss): {test_loss:.4f}")
print(f"テスト正解率 (Test Accuracy): {test_accuracy*100:.2f} %")
```

📝 最終評価結果の記録

Part 2 の最後で確認した「検証データ（validation data）の正解率」と、今回計算した「テストデータ（test data）の正解率」を記録し、比較してください。

🔍 タスク３：混同行列による間違い分析

正解率（Accuracy）だけでは、「どのような間違いをしたか」が分かりません。そこで「混同行列」を使います。

📌 Task 3-1: 予測の実行と混同行列の作成

まず、テストデータ全体に対して予測を行い、その予測結果（0 か 1）を取得します。

```
# 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)
```

📌 Task 3-2: 混同行列の可視化

数字のままでは分かりにくいので、Seaborn のヒートマップを使って可視化します。

```
# 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()
```

💡 混同行列の見方

- **左上（True Negative）**：本当はネガティブで、AI も「ネガティブ」と正解した数。
- **右下（True Positive）**：本当はポジティブで、AI も「ポジティブ」と正解した数。
- **右上（False Positive）**：本当はネガティブなのに、AI が「ポジティブ」と間違えた数。
- **左下（False Negative）**：本当はポジティブなのに、AI が「ネガティブ」と間違えた数。

👉 **対角線（左上と右下）が正解数**、それ以外が間違いの数です。

📝 混同行列の記録

グラフを見て、4 つのマスの数字を記録してください。

📈 タスク４：分類レポートによる詳細評価

正解率以外にも、モデルの性能を測る指標があります。

📌 Task 4-1: 分類レポートの表示

classification_report を使うと、主要な指標をまとめて表示できます。

```
# 分類レポートの表示
report = classification_report(y_test, predictions,
target_names=['Negative (0)', 'Positive (1)'])

print("=== 分類レポート ===")
print(report)
```

💡 分類レポートの見方

- **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。（冤罪の少なさ）
- **Recall (再現率)**: 本当にポジティブだったもののうち、AI が「ポジティブ」と正しく見つけられた割合。（見逃しの少なさ）
- **F1-score (F1 値)**: Precision と Recall のバランスを取ったスコア。（総合評価）
- **Accuracy (正解率)**: 全体のうち、正解した割合。

📝 レポート結果の記録

Negative(0)と Positive(1)の Precision, Recall, F1-score を記録してください。

🤔 Part３　考察問題

分析結果を見て、以下の問いについて考え、記録してください。

問 1：混同行列を見て、AI は「ネガティブなレビューをポジティブと間違える」（右上）のと、「ポジティブなレビューをネガティブと間違える」（左下）のとでは、どちらの間違いが多かったですか？

問 2：分類レポートの「Negative(0)」と「Positive(1)」の F1-score を比べてみましょう。AI はどちらの感情を判定する方が得意（または苦手）だと考えられますか？

問 3：もしこの AI を「不適切なコメントを自動で非表示にする」システムに使う場合、「Precision（適合率）」と「Recall（再現率）」のどちらをより重視すべきだと思いますか？　理由とともに答えてください。（ヒント：不適切＝ネガティブと仮定）

✅ Part３のまとめ

Part 3 では、モデルの性能を「正解率」だけでなく、「混同行列」や「Precision/Recall」といった多角的な指標で評価する方法を学びました。

📚 今日学んだこと

- `.evaluate()` でテストデータに対する最終性能を評価した
- `confusion_matrix` で AI の間違いパターンを可視化した
- `classification_report` で Precision, Recall, F1-score の意味を学んだ
- 評価指標は「目的」によって使い分ける必要があることを理解した

🔜 次のステップ：Part 4

いよいよ実習 3 の最終章です！

Part 4 では、今回作った LSTM モデルよりも遥かに強力な、現代の自然言語処理の主流である**「BERT（バート）」**という事前学習モデルを使った転移学習に挑戦します！

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習３ Part３）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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
Part４（事前学習モデル（BERT）の活用）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part４の学習目標

この Part では、次の項目を目標にします。

- **目標 1**:「事前学習モデル」と「転移学習（ファインチューニング）」の概念を理解する
- **目標 2**:Hugging Face transformers ライブラリの基本操作を習得する
- **目標 3**:BERT モデルを使って IMDb データセットの感情分析をファインチューニングできる
- **目標 4**:自作した LSTM モデルと BERT モデルの性能を比較できる
- **目標 5**:学習済み BERT モデルを使って、任意の文章の感情を予測できる

📍 📖 Part 4 について

Part 2, 3 では、LSTM というモデルを「ゼロから」学習させました。結果は約 86%の正解率でしたね。

Part 4 では、現代の自然言語処理（NLP）で主流となっている**「事前学習モデル」**、その中でも特に有名な**「BERT」**を使います。BERT は、インターネット上の膨大なテキストを予め学習しており、非常に賢いモデルです。

この「賢い」モデルをベースに、私たちの「映画レビュー」データで追加学習させる（＝**転移学習 / ファインチューニング**）ことで、どれだけ性能が向上するかを体験します。

🛠️ タスク 1：準備（Hugging Face ライブラリのインストール）

📌 Task 1-1: 必要なライブラリのインストール

BERT モデルを簡単に扱うためのライブラリ transformers と、データセットを扱う datasets をインストールします。

```
# Hugging Face ライブラリをインストール
!pip install transformers datasets -q

print("☑ transformers, datasets のインストール完了")
```

⚠️ GPU の確認

BERT の学習は計算量が非常に大きいため、**必ず GPU ランタイムを有効**にしてください。（メニュー「ランタイム」→「ランタイムのタイプを変更」→「T4 GPU」）

📌 Task 1-2: ライブラリのインポート

今回使うライブラリをインポートします。

```
import numpy as np
import pandas as pd
import torch # PyTorch (Transformers が内部で使用)
from datasets import load_dataset
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, TrainingArguments, Trainer
from sklearn.metrics import accuracy_score,
precision_recall_f1_score

print("☑ ライブラリのインポート完了")
```

📁 タスク２：データセットの準備（Hugging Face 版）

BERT でファインチューニングするために、Part 1 とは少し違う方法でデータを準備します。

📌 Task 2-1: IMDb データセットの読み込み

```
datasets ライブラリを使って、IMDb データセットを読み込みます。今回はテキストデータ（文字列）として直接読み込みます。

# IMDb データセットを Hugging Face Hub から読み込む
# train['text'] にレビュー本文、train['label'] に 0（Neg）か 1（Pos）が入る
dataset = load_dataset("imdb")

# 時間短縮のため、訓練データ・テストデータをそれぞれ 5000 件に減らします
# （Colab 無料枠では全データ（25000 件）を使うと時間がかかりすぎるため）
train_dataset =
dataset['train'].shuffle(seed=42).select(range(5000))
test_dataset =
dataset['test'].shuffle(seed=42).select(range(5000))

print("データセットの読み込みと削減完了")
print(train_dataset)
print("¥n--- サンプルデータ ---")
print(f"レビュー: {train_dataset[0]['text'][:200]}...")
print(f"ラベル: {train_dataset[0]['label']}")
```

📌 Task 2-2: BERT 用トークナイザーの準備

BERT は独自の「辞書」と「ルール」（トークナイザー）を持っています。AutoTokenizer を使って、モデルに合ったトークナイザーを読み込みます。

```
# 使用するモデル名（今回は小規模で高速な distilbert）
model_name = "distilbert-base-uncased-finetuned-sst-2-english"

# モデル名に対応したトークナイザーを読み込み
tokenizer = AutoTokenizer.from_pretrained(model_name)

print(f"'{model_name}' 用のトークナイザーを読み込みました。")
```

📌 Task 2-3: トークン化（前処理）

テキストをトークナイザーで数値に変換します。Part 1 で行った「数値化」と「パディング」を一度に行います。

```
# トークン化を行う関数を定義
def tokenize_function(examples):
    # padding="max_length" で最大長（512）にパディング
    # truncation=True で最大長を超える場合は切り捨て
    return tokenizer(examples["text"], padding="max_length",
truncation=True)

# 訓練データとテストデータに関数を適用
tokenized_train_dataset = train_dataset.map(tokenize_function,
batched=True)
tokenized_test_dataset = test_dataset.map(tokenize_function,
batched=True)

print("¥n トークン化とパディング完了。")
print("例 （最初のデータ）:")
print(tokenized_train_dataset[0]['input_ids'][:20]) # 数値に変換されたリスト
```

💡 **.map(batched=True)**

Hugging Face datasets ライブラリの強力な機能で、データセット全体に高速に関数を適用できます。

🚀 タスク３：BERT モデルのファインチューニング

事前学習済みの BERT モデルを、IMDb データで追加学習（ファインチューニング）します。

📌 Task 3-1: 事前学習モデルの読み込み

AutoModelForSequenceClassification を使って、感情分析タスク用の事前学習済みモデルを読み込みます。

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

print(f"'{model_name}' モデルを読み込みました。GPU に転送します...")

# モデルを GPU に移動 （GPU が利用可能な場合）
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model.to(device)
print(f"使用デバイス: {device}")
```

📌 Task 3-2: 学習の設定

Trainer を使うために、学習の詳細設定（ハイパーパラメータ）を TrainingArguments で定義します。

```
# 学習の設定
training_args = TrainingArguments(
    output_dir="test_trainer",          # 結果の保存先
    evaluation_strategy="epoch",        # 1 エポックごとに評価
    num_train_epochs=3,                 # 学習するエポック数 （3 周）
    per_device_train_batch_size=16,     # 訓練バッチサイズ
    per_device_eval_batch_size=16,      # 評価バッチサイズ
    logging_steps=100,                  # 100 ステップごとにログ表示
)

# 評価指標を計算する関数を定義
def compute_metrics(eval_pred):
    logits, labels = eval_pred
```

```
    predictions = np.argmax(logits, axis=-1)
    return {"accuracy": accuracy_score(labels, predictions)}

# Trainer を初期化
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_test_dataset,
    compute_metrics=compute_metrics,
)

print("Trainer の準備完了。")
```

📌 Task 3-3: ファインチューニングの実行

.train() を実行します。
⚠️注意： GPU を使用していても、約 5〜15 分かかります。

```
%%time
print("ファインチューニングを開始します... (約 5〜15 分かかります)")

# 学習の実行
trainer.train()

print("☑ ファインチューニング完了！")
```

📌 Task 3-4: 最終評価

.evaluate() で、テストデータに対する最終性能を確認します。

```
# 最終評価
eval_results = trainer.evaluate()
print("¥n--- BERT モデル 最終評価結果 ---")
print(f"テスト損失 (Test Loss): {eval_results['eval_loss']:.4f}")
print(f"テスト正解率 (Test Accuracy):
{eval_results['eval_accuracy']*100:.2f} %")
```

📝 BERT モデルの性能記録

BERT モデルの最終的なテスト正解率を記録し、Part 3 で計算した LSTM モデルのテスト正解率（約 86%）と比較してください。

✍️ タスク４：実データでの予測

学習した BERT モデルを使って、自分で書いたレビューがポジティブかネガティブか判定させてみましょう。

📌 Task 4-1: 予測パイプラインの作成

transformers の pipeline を使うと、トークン化から予測までを簡単に行えます。

```
from transformers import pipeline

# 感情分析パイプラインを作成（学習済みモデルとトークナイザーを使用）
sentiment_pipeline = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer, device=0 if torch.cuda.is_available() else -
1)

print("☑ 予測パイプライン準備完了")
```

📌 Task 4-2: 予測の実行

my_review の中身を書き換えて、AI の反応を見てみましょう。

```
# --- 自由にレビューを書き換えて試してみよう ---
my_review_1 = "This movie was absolutely fantastic! The acting was
great and the story was amazing."
my_review_2 = "I really hated this film. It was boring and too
long."
my_review_3 = "The movie was not bad, actually it was pretty
good." # 少し複雑な文

# 予測の実行
results = sentiment_pipeline([my_review_1, my_review_2,
my_review_3])

# 結果の表示
for review, result in zip([my_review_1, my_review_2, my_review_3],
results):
    print(f"¥n レビュー: {review}")
    if result['label'] == 'LABEL_1' or result['label'] ==
'POSITIVE': # モデルによってラベル名が異なる
        print(f"  予測: ポジティブ 👍 (スコア: {result['score']:.4f})")
    else:
        print(f"  予測: ネガティブ 👎 (スコア: {result['score']:.4f})")
```

📝 予測実験

自分で考えたレビュー（ポジティブな文、ネガティブな文）をいくつか試し、AI が正しく判定できるか記録してください。

🤔 Part４　考察問題

問 1: なぜ BERT モデル（転移学習）は、私たちがゼロから学習させた LSTM モデルよりも高い性能（正解率）を出せたのでしょうか？「事前学習」という言葉を使って説明してください。

問 2: 実習 1（数字）、実習 2（画像）、実習 3（テキスト）を終えて、データ形式（数値、画像、テキスト）によって AI の作り方（前処理やモデル）にどのような違いがありましたか？　あなたが最も重要だと感じた違いを 1 つ挙げてください。

✅ Part4 ／ 実習3 のまとめ

🎉 これで実習 3、そして応用教材の全実習が完了です！

Part 4 では、最新の NLP モデルである BERT を使い、高い精度の感情分析 AI を構築しました。

📚 実習 3 全体で学んだこと

- テキストデータ特有の前処理（トークン化、パディング）
- 系列データを扱う RNN/LSTM モデルの構築と学習
- モデルの評価（混同行列、Precision/Recall）
- Hugging Face Transformers を使った BERT のファインチューニング

🌟 全応用教材の完了

実習 1（機械学習基礎）、実習 2（画像 CNN）、実習 3（テキスト NLP）を通して、データサイエンスと AI の主要な分野の実践的スキルを幅広く習得しました。この経験は皆さんの大きな力となるはずです。本当にお疲れ様でした！

✅ 実習振り返りシート （人工知能（AI）技術応用 - 実習３ Part４）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

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）技術応用教材ワークブック

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