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２

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 3

人工知能（ＡＩ）技術応用_実習１ 9

Part1 9
Part2 20
Part3 28
Part4 35

人工知能（ＡＩ）技術応用_実習２ 43

Part1 43
Part2 53
Part3 62
Part4 70

人工知能（ＡＩ）技術応用_実習 3..... 79

Part1 79
Part2 87
Part3 96
Part4 105

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

実習２ 画像認識AIで犬と猫を分類 Part２

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

実習２ 画像認識AIで犬と猫を分類 Part２

- ✅ **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")
```

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

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 Part 1

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 Part 1

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 Part 1

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 Part 1

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

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**



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

Part２：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 Part 3

モデルとデータの引き継ぎ

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
```


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 Part 3

テストデータによる最終評価

Part 2 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
Part 2 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 2

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

●本書の内容を無断で転記、掲載することは禁じます。