Uniwersytet Bielsko-Bialski

Zadanie GL

Sprawozdanie z ćwiczeń
Nauka o Danych II

Data wykonania:
28.06.2025

Autor:
Bartosz Bieniek 058085

1. Cel ćwiczenia

Warstwa gęsta: Klasyfikacja IRIS z funkcją aktywacji Softplus.

Warstwa konwolucyjna: Płytka sieć konwolucyjna z tylko jedną warstwą Conv2D.

Warstwa rekurencyjna: Użycie prostej warstwy RNN do analizy sentymentu IMDB.

Warstwa Transformer: Zastosowanie mechanizmu self-attention bez pozycyjnego kodowania.

Zadania umieścić na Github.

2. Przebieg ćwiczenia

Krok 1: Import bibliotek i przygotowanie środowiska

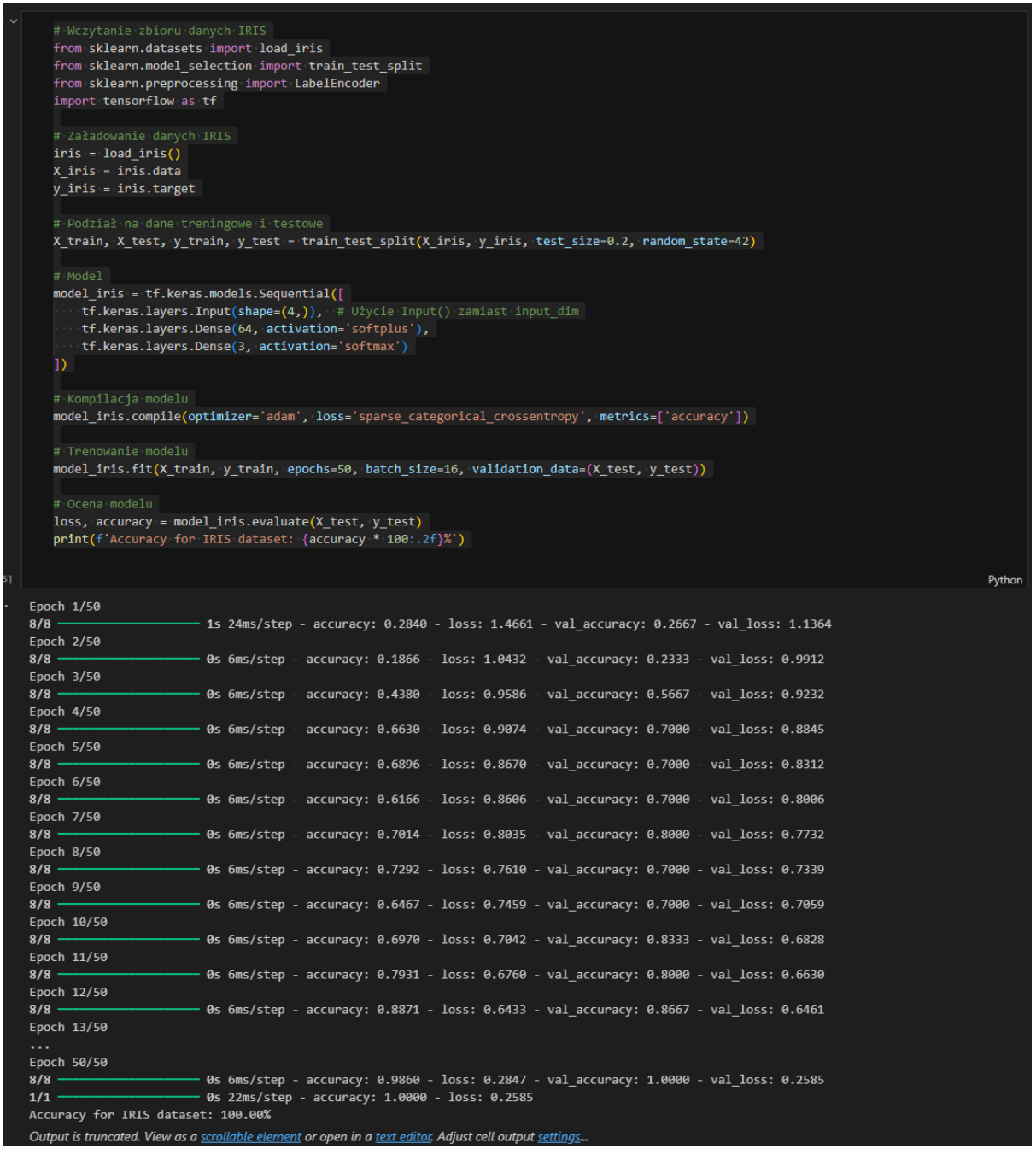
W tym etapie zaimportowano kluczowe biblioteki potrzebne do pracy z modelami głębokiego uczenia w TensorFlow oraz Keras. Uwzględniono moduły do budowy modeli, warstw takich jak Dense, Conv2D, SimpleRNN, a także narzędzia do przetwarzania danych, np. pad_sequences czy to_categorical. Dodatkowo załadowano biblioteki pomocnicze, takie jak NumPy i sklearn, które będą używane w kolejnych krokach. Etap ten stanowi podstawę dla dalszej implementacji modeli.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, SimpleRNN, LayerNormalization, MultiHeadAttention, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import numpy as np
```

Rys. 1. Import bibliotek i przygotowanie środowiska

Krok 2: Warstwa gęsta – klasyfikacja zbioru IRIS

W tym kroku przygotowano klasyfikator dla zbioru IRIS, wykorzystując prostą sieć neuronową z jedną warstwą ukrytą. Model zawiera warstwę Dense z funkcją aktywacji softplus oraz warstwę wyjściową softmax, dostosowaną do klasyfikacji wieloklasowej. Dane zostały podzielone na zbiór treningowy i testowy, a model trenowano przez 50 epok. Wynikowa dokładność pokazuje skuteczność modelu na prostym zbiorze cech morfologicznych kwiatów.



```python
# Wczytanie zbioru danych IRIS
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf

# Załadowanie danych IRIS
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

# Podział na dane treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

# Model
model_iris = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(4,)),  # Użycie Input() zamiast input_dim
    tf.keras.layers.Dense(64, activation='softplus'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Kompilacja modelu
model_iris.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Trenowanie modelu
model_iris.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))

# Ocena modelu
loss, accuracy = model_iris.evaluate(X_test, y_test)
print(f'Accuracy for IRIS dataset: {accuracy * 100:.2f}%')
```

Python

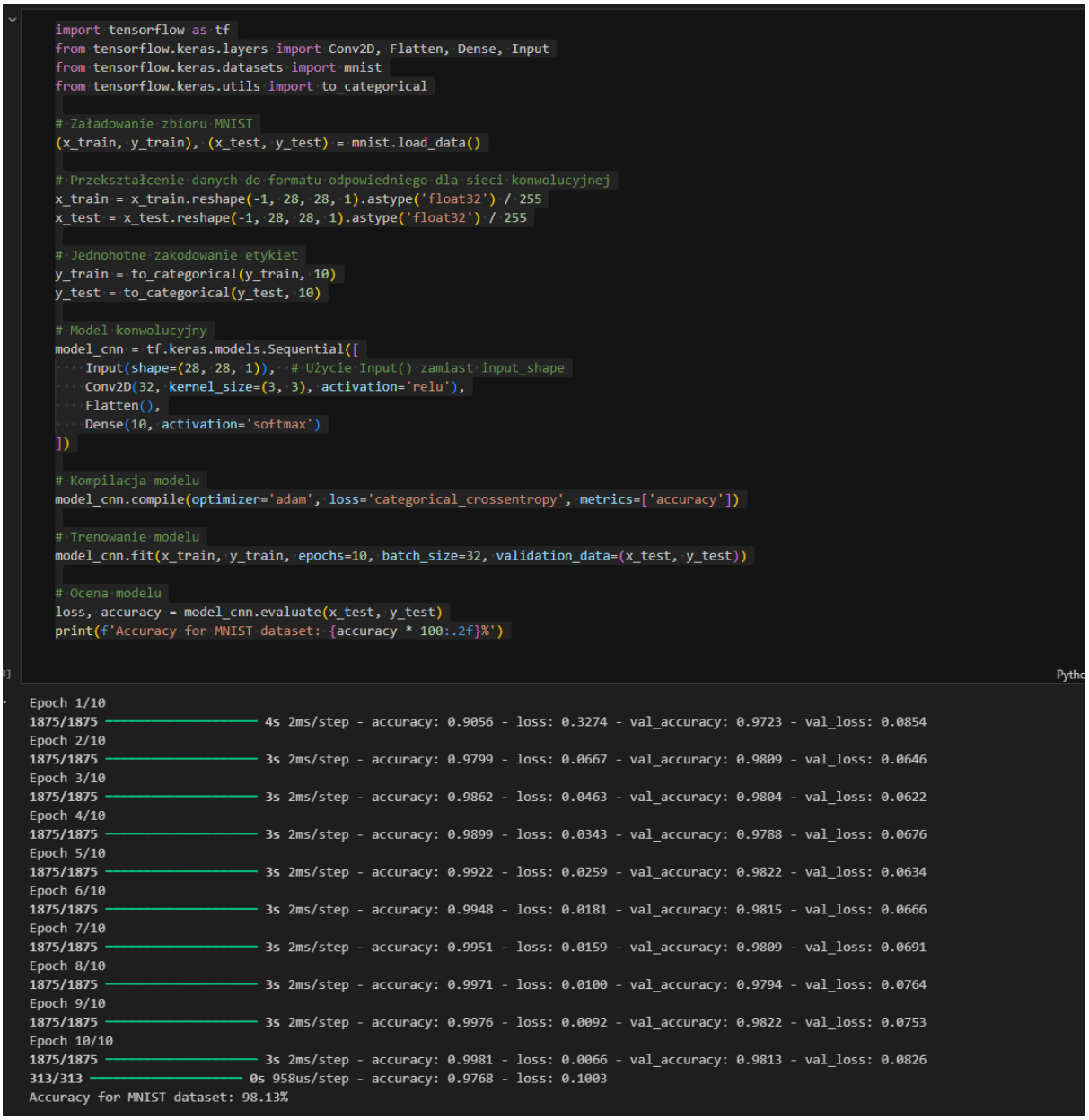
```
Epoch 1/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.2840 - loss: 1.4661 - val_accuracy: 0.2667 - val_loss: 1.1364
Epoch 2/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.1866 - loss: 1.0432 - val_accuracy: 0.2333 - val_loss: 0.9912
Epoch 3/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4380 - loss: 0.9586 - val_accuracy: 0.5667 - val_loss: 0.9232
Epoch 4/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6630 - loss: 0.9074 - val_accuracy: 0.7000 - val_loss: 0.8845
Epoch 5/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6896 - loss: 0.8670 - val_accuracy: 0.7000 - val_loss: 0.8312
Epoch 6/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6166 - loss: 0.8606 - val_accuracy: 0.7000 - val_loss: 0.8006
Epoch 7/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7014 - loss: 0.8035 - val_accuracy: 0.8000 - val_loss: 0.7732
Epoch 8/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7292 - loss: 0.7610 - val_accuracy: 0.7000 - val_loss: 0.7339
Epoch 9/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6467 - loss: 0.7459 - val_accuracy: 0.7000 - val_loss: 0.7059
Epoch 10/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6970 - loss: 0.7042 - val_accuracy: 0.8333 - val_loss: 0.6828
Epoch 11/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7931 - loss: 0.6760 - val_accuracy: 0.8000 - val_loss: 0.6630
Epoch 12/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8871 - loss: 0.6433 - val_accuracy: 0.8667 - val_loss: 0.6461
Epoch 13/50
...
Epoch 50/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9860 - loss: 0.2847 - val_accuracy: 1.0000 - val_loss: 0.2585
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 1.0000 - loss: 0.2585
Accuracy for IRIS dataset: 100.00%
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Rys. 2. Warstwa gęsta – klasyfikacja zbioru IRIS

Krok 3: Warstwa konwolucyjna – rozpoznawanie cyfr MNIST

Ten etap obejmuje budowę prostej sieci konwolucyjnej z jedną warstwą Conv2D, zastosowanej do klasyfikacji obrazów z ręcznie pisanymi cyframi. Dane zostały odpowiednio przekształcone i znormalizowane, a etykiety zakodowano metodą one-hot. Model osiągnął bardzo wysoką dokładność już po kilku epokach, co świadczy o skuteczności podejścia konwolucyjnego w rozpoznawaniu obrazów.



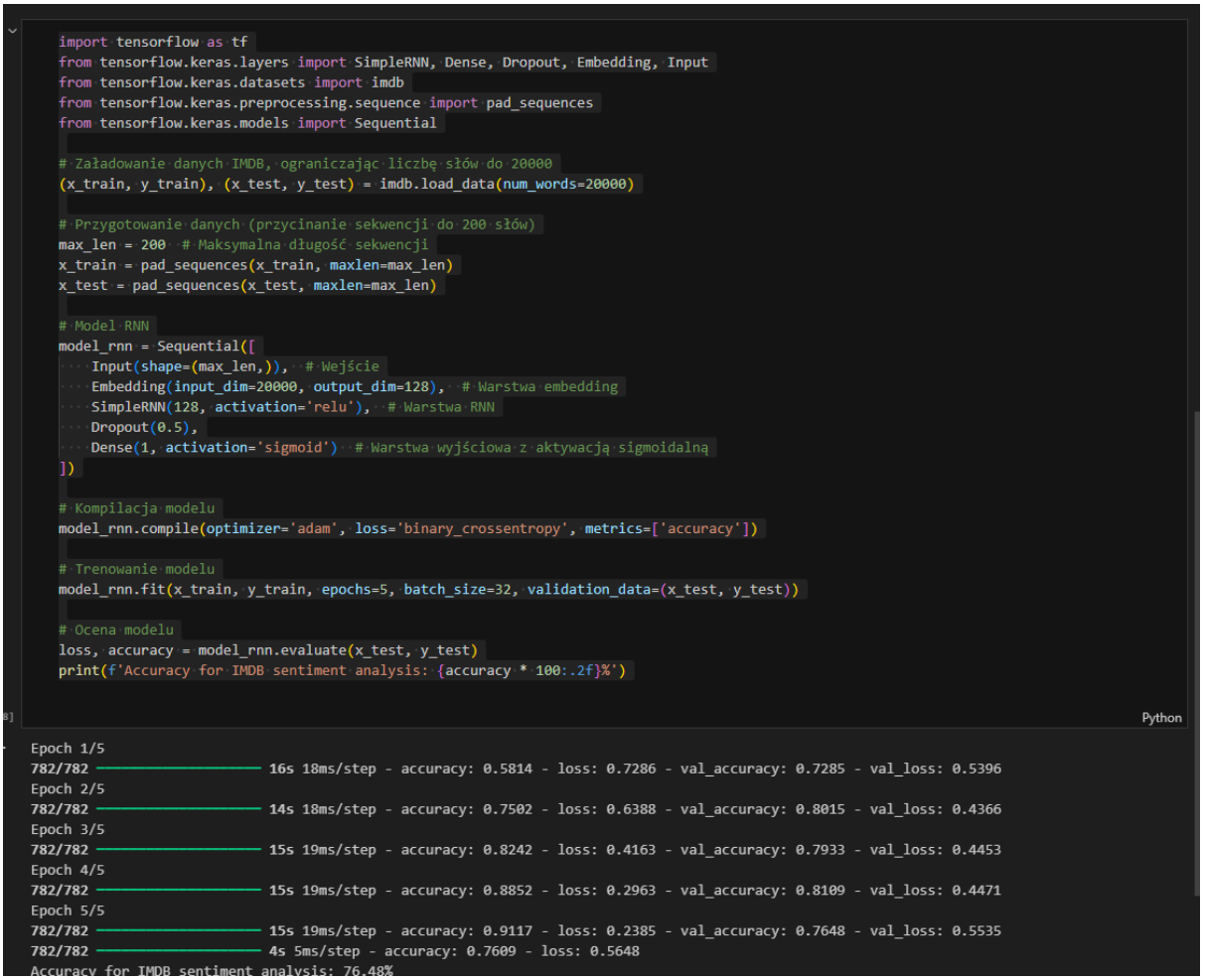
```python
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Input
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Załadowanie zbioru MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Przekształcenie danych do formatu odpowiedniego dla sieci konwolucyjnej
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255

# Jednohotne zakodowanie etykiet
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Model konwolucyjny
model_cnn = tf.keras.models.Sequential([
    Input(shape=(28, 28, 1)),  # Użycie Input() zamiast input_shape
    Conv2D(32, kernel_size=(3, 3), activation='relu'),
    Flatten(),
    Dense(10, activation='softmax')
])

# Kompilacja modelu
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Trenowanie modelu
model_cnn.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))

# Ocena modelu
loss, accuracy = model_cnn.evaluate(x_test, y_test)
print(f'Accuracy for MNIST dataset: {accuracy * 100:.2f}%')
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9056 - loss: 0.3274 - val_accuracy: 0.9723 - val_loss: 0.0854
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9799 - loss: 0.0667 - val_accuracy: 0.9809 - val_loss: 0.0646
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9862 - loss: 0.0463 - val_accuracy: 0.9804 - val_loss: 0.0622
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9899 - loss: 0.0343 - val_accuracy: 0.9788 - val_loss: 0.0676
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9922 - loss: 0.0259 - val_accuracy: 0.9822 - val_loss: 0.0634
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9948 - loss: 0.0181 - val_accuracy: 0.9815 - val_loss: 0.0666
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9809 - val_loss: 0.0691
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9971 - loss: 0.0100 - val_accuracy: 0.9794 - val_loss: 0.0764
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9976 - loss: 0.0092 - val_accuracy: 0.9822 - val_loss: 0.0753
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9981 - loss: 0.0066 - val_accuracy: 0.9813 - val_loss: 0.0826
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 958us/step - accuracy: 0.9768 - loss: 0.1003
Accuracy for MNIST dataset: 98.13%
```

Rys. Warstwa konwolucyjna – rozpoznawanie cyfr MNIST

Krok 4: Warstwa rekurencyjna – analiza sentymentu IMDB

W tym kroku zastosowano warstwę SimpleRNN do analizy sentymentu recenzji filmowych z bazy IMDB. Przed trenowaniem dane zostały przycięte do określonej długości, a model zawiera warstwę embedding oraz RNN z aktywacją ReLU. Po pięciu epokach model osiągnął satysfakcjonującą dokładność, pokazując, że nawet prosta rekurencyjna architektura może skutecznie analizować dane sekwencyjne.



```python
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, Embedding, Input
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential

# Załadowanie danych IMDB, ograniczając liczbę słów do 20000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=20000)

# Przygotowanie danych (przycinanie sekwencji do 200 słów)
max_len = 200  # Maksymalna długość sekwencji
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Model RNN
model_rnn = Sequential([
    Input(shape=(max_len,)),  # Wejście
    Embedding(input_dim=20000, output_dim=128),  # Warstwa embedding
    SimpleRNN(128, activation='relu'),  # Warstwa RNN
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Warstwa wyjściowa z aktywacją sigmoidalną
])

# Kompilacja modelu
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Trenowanie modelu
model_rnn.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test))

# Ocena modelu
loss, accuracy = model_rnn.evaluate(x_test, y_test)
print(f'Accuracy for IMDB sentiment analysis: {accuracy * 100:.2f}%')
```

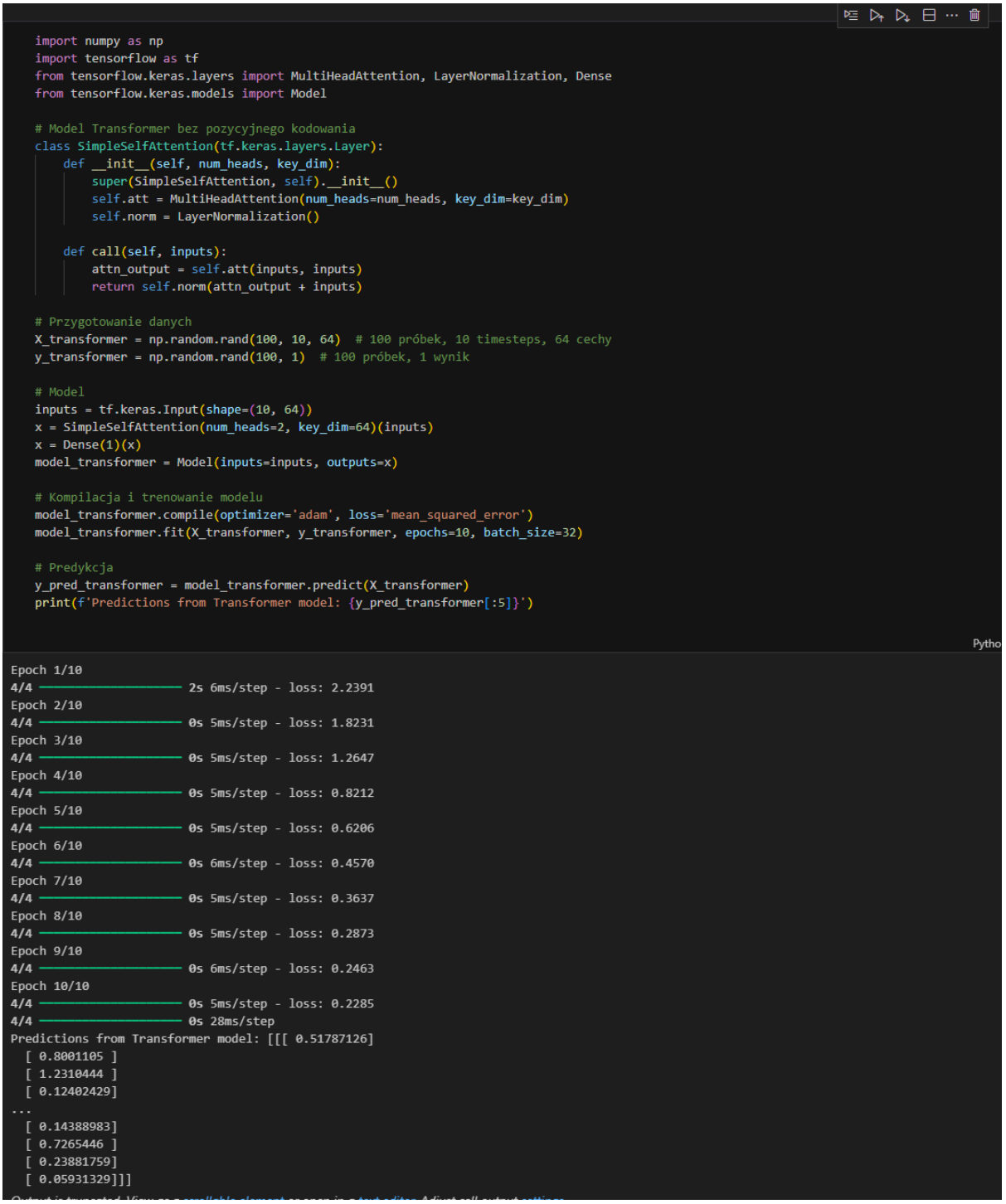
Python

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 18ms/step - accuracy: 0.5814 - loss: 0.7286 - val_accuracy: 0.7285 - val_loss: 0.5396
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.7502 - loss: 0.6388 - val_accuracy: 0.8015 - val_loss: 0.4366
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.8242 - loss: 0.4163 - val_accuracy: 0.7933 - val_loss: 0.4453
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.8852 - loss: 0.2963 - val_accuracy: 0.8109 - val_loss: 0.4471
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.9117 - loss: 0.2385 - val_accuracy: 0.7648 - val_loss: 0.5535
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7609 - loss: 0.5648
Accuracy for IMDB sentiment analysis: 76.48%
```

Rys. Warstwa rekurencyjna – analiza sentymentu IMDB

Krok 5: Warstwa Transformer – mechanizm self-attention bez pozycyjnego kodowania

Na koniec zbudowano uproszczony model typu Transformer, oparty na warstwie MultiHeadAttention bez zastosowania pozycyjnego kodowania. Dane wejściowe były sztucznie wygenerowanymi sekwencjami numerycznymi. Model został nauczony regresji na danych ciągłych i wykonał predykcję. To podejście demonstruje możliwości samoatencji w kontekście sekwencji, nawet bez dodatkowych informacji o kolejności pozycji.



```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization, Dense
from tensorflow.keras.models import Model

# Model Transformer bez pozycyjnego kodowania
class SimpleSelfAttention(tf.keras.layers.Layer):
    def __init__(self, num_heads, key_dim):
        super(SimpleSelfAttention, self).__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=key_dim)
        self.norm = LayerNormalization()

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)
        return self.norm(attn_output + inputs)

# Przygotowanie danych
X_transformer = np.random.rand(100, 10, 64)  # 100 próbek, 10 timesteps, 64 cechy
y_transformer = np.random.rand(100, 1)  # 100 próbek, 1 wynik

# Model
inputs = tf.keras.Input(shape=(10, 64))
x = SimpleSelfAttention(num_heads=2, key_dim=64)(inputs)
x = Dense(1)(x)
model_transformer = Model(inputs=inputs, outputs=x)

# Kompilacja i trenowanie modelu
model_transformer.compile(optimizer='adam', loss='mean_squared_error')
model_transformer.fit(X_transformer, y_transformer, epochs=10, batch_size=32)

# Predykcja
y_pred_transformer = model_transformer.predict(X_transformer)
print(f'Predictions from Transformer model: {y_pred_transformer[:5]}')
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 2.2391
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.8231
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.2647
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8212
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.6206
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.4570
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.3637
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2873
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2463
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2285
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Predictions from Transformer model: [[[ 0.51787126]
  [ 0.8001105 ]
  [ 1.2310444 ]
  [ 0.12402429]
...
  [ 0.14388983]
  [ 0.7265446 ]
  [ 0.23881759]
  [ 0.05931329]]]
```

Rys. 5. Warstwa rekurencyjna – analiza sentymentu IMDB

3. Wnioski

Na podstawie przeprowadzonych eksperymentów można sformułować następujące wnioski. Nawet proste architektury sieci głębokich, takie jak jedno- lub dwuwarstwowe modele Dense czy Conv2D, pozwalają uzyskać wysoką skuteczność na klasycznych zbiorach danych, takich jak IRIS i MNIST.