```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten,
BatchNormalization, Dropout, GRU, LayerNormalization, Input
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np

# ===========================
# Zadanie 1: Klasyfikacja IRIS z Siecią Gęstą i BatchNormalization
# ===========================

# Załadowanie danych IRIS
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

iris_data = load_iris()
X = iris_data.data
y = iris_data.target

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

# Normalizacja danych
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model sieci gęstej z BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Input

# Stosujemy warstwę Input na początku modelu
model_iris = Sequential()
model_iris.add(Input(shape=(4,)))  # Warstwa wejściowa (4 cechy IRIS)
model_iris.add(Dense(64, activation='relu'))  # Warstwa gęsta
model_iris.add(BatchNormalization())  # Warstwa BatchNormalization
model_iris.add(Dense(32, activation='relu'))
model_iris.add(Dense(3, activation='softmax'))  # Warstwa wyjściowa (3
klasy)

# Kompilacja i trenowanie modelu
model_iris.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_iris.fit(X_train, y_train, epochs=50, batch_size=8,
```

```
validation_data=(X_test, y_test))

# Ewaluacja modelu
loss, accuracy = model_iris.evaluate(X_test, y_test)
print(f"Test accuracy (IRIS): {accuracy * 100:.2f}%")
```

```
Epoch 1/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7069 - loss:
0.7375 - val_accuracy: 0.8333 - val_loss: 0.8066
Epoch 2/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8584 - loss:
0.3653 - val_accuracy: 0.9000 - val_loss: 0.7012
Epoch 3/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8929 - loss:
0.2847 - val_accuracy: 0.9667 - val_loss: 0.6297
Epoch 4/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9215 - loss:
0.2406 - val_accuracy: 1.0000 - val_loss: 0.5703
Epoch 5/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9104 - loss:
0.2582 - val_accuracy: 1.0000 - val_loss: 0.5112
Epoch 6/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9157 - loss:
0.2335 - val_accuracy: 1.0000 - val_loss: 0.4550
Epoch 7/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9564 - loss:
0.1630 - val_accuracy: 1.0000 - val_loss: 0.4012
Epoch 8/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9054 - loss:
0.1835 - val_accuracy: 1.0000 - val_loss: 0.3656
Epoch 9/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9257 - loss:
0.2220 - val_accuracy: 1.0000 - val_loss: 0.3491
Epoch 10/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9530 - loss:
0.1250 - val_accuracy: 1.0000 - val_loss: 0.3005
Epoch 11/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9723 - loss:
0.1337 - val_accuracy: 1.0000 - val_loss: 0.2717
Epoch 12/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9590 - loss:
0.0998 - val_accuracy: 1.0000 - val_loss: 0.2327
Epoch 13/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9739 - loss:
0.1040 - val_accuracy: 0.9667 - val_loss: 0.2126
Epoch 14/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9482 - loss:
0.1079 - val_accuracy: 0.9333 - val_loss: 0.2236
Epoch 15/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9725 - loss:
```

```
0.0728 - val_accuracy: 0.9333 - val_loss: 0.1995
Epoch 16/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9592 - loss:
0.0861 - val_accuracy: 0.9333 - val_loss: 0.1547
Epoch 17/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9678 - loss:
0.0830 - val_accuracy: 0.9667 - val_loss: 0.1321
Epoch 18/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9601 - loss:
0.1086 - val_accuracy: 0.9667 - val_loss: 0.1405
Epoch 19/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9516 - loss:
0.1379 - val_accuracy: 0.9333 - val_loss: 0.1485
Epoch 20/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9766 - loss:
0.1135 - val_accuracy: 0.9333 - val_loss: 0.1875
Epoch 21/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9777 - loss:
0.0941 - val_accuracy: 0.9667 - val_loss: 0.1110
Epoch 22/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9628 - loss:
0.0810 - val_accuracy: 0.9667 - val_loss: 0.0846
Epoch 23/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9683 - loss:
0.1849 - val_accuracy: 0.9667 - val_loss: 0.0937
Epoch 24/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9538 - loss:
0.1247 - val_accuracy: 0.9667 - val_loss: 0.0831
Epoch 25/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9304 - loss:
0.1487 - val_accuracy: 0.9667 - val_loss: 0.0912
Epoch 26/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9712 - loss:
0.0875 - val_accuracy: 0.9667 - val_loss: 0.0730
Epoch 27/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9876 - loss:
0.0583 - val_accuracy: 0.9667 - val_loss: 0.0730
Epoch 28/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9974 - loss:
0.0414 - val_accuracy: 0.9667 - val_loss: 0.0772
Epoch 29/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9611 - loss:
0.0898 - val_accuracy: 0.9667 - val_loss: 0.0890
Epoch 30/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss:
0.0377 - val_accuracy: 0.9667 - val_loss: 0.0810
Epoch 31/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss:
0.0367 - val_accuracy: 0.9667 - val_loss: 0.0569
```

```
Epoch 32/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9875 - loss:
0.0456 - val_accuracy: 1.0000 - val_loss: 0.0478
Epoch 33/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9613 - loss:
0.1058 - val_accuracy: 0.9667 - val_loss: 0.0707
Epoch 34/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9914 - loss:
0.1019 - val_accuracy: 0.9667 - val_loss: 0.0605
Epoch 35/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9808 - loss:
0.0558 - val_accuracy: 1.0000 - val_loss: 0.0453
Epoch 36/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss:
0.0252 - val_accuracy: 1.0000 - val_loss: 0.0453
Epoch 37/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9568 - loss:
0.1317 - val_accuracy: 0.9667 - val_loss: 0.0705
Epoch 38/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9898 - loss:
0.0638 - val_accuracy: 1.0000 - val_loss: 0.0441
Epoch 39/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9297 - loss:
0.1514 - val_accuracy: 0.9333 - val_loss: 0.0869
Epoch 40/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9498 - loss:
0.0880 - val_accuracy: 0.9667 - val_loss: 0.0471
Epoch 41/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9803 - loss:
0.0757 - val_accuracy: 0.9667 - val_loss: 0.0508
Epoch 42/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9524 - loss:
0.1218 - val_accuracy: 1.0000 - val_loss: 0.0392
Epoch 43/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9926 - loss:
0.0446 - val_accuracy: 1.0000 - val_loss: 0.0362
Epoch 44/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9955 - loss:
0.0617 - val_accuracy: 1.0000 - val_loss: 0.0303
Epoch 45/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9319 - loss:
0.1116 - val_accuracy: 1.0000 - val_loss: 0.0429
Epoch 46/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9855 - loss:
0.0562 - val_accuracy: 1.0000 - val_loss: 0.0502
Epoch 47/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9874 - loss:
0.0427 - val_accuracy: 1.0000 - val_loss: 0.0424
Epoch 48/50
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9900 - loss:
0.0548 - val_accuracy: 1.0000 - val_loss: 0.0328
Epoch 49/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9696 - loss:
0.0638 - val_accuracy: 1.0000 - val_loss: 0.0300
Epoch 50/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9894 - loss:
0.0419 - val_accuracy: 1.0000 - val_loss: 0.0205
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 1.0000 - loss:
0.0205
Test accuracy (IRIS): 100.00%

# ===========================
# Zadanie 2: Klasyfikacja cyfr MNIST z większymi jądrami
konwolucyjnymi
# ===========================

# Załadowanie danych MNIST
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D,
Dense, Input

# Wczytanie danych
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Zmiana kształtu danych wejściowych (dodanie wymiaru kanału 1 dla
obrazów szaro-skalowych)
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# Normalizacja danych
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# One-hot encoding etykiet
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Model z warstwą konwolucyjną
model_mnist = Sequential()
model_mnist.add(Input(shape=(28, 28, 1)))  # Warstwa wejściowa
(rozmiar obrazu 28x28, 1 kanał)
model_mnist.add(Conv2D(32, (5, 5), activation='relu'))  # Warstwa
konwolucyjna z jądrem 5x5
model_mnist.add(MaxPooling2D(pool_size=(2, 2)))  # Max pooling
model_mnist.add(Flatten())  # Spłaszczanie wyników
model_mnist.add(Dense(128, activation='relu'))  # Warstwa gęsta
model_mnist.add(Dense(10, activation='softmax'))  # Warstwa wyjściowa
(10 klas)
```

```
# Kompilacja i trenowanie modelu
model_mnist.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model_mnist.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test))

# Ewaluacja modelu
loss, accuracy = model_mnist.evaluate(X_test, y_test)
print(f"Test accuracy (MNIST): {accuracy * 100:.2f}%")

Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9036 - loss:
0.3301 - val_accuracy: 0.9801 - val_loss: 0.0619
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9822 - loss:
0.0573 - val_accuracy: 0.9861 - val_loss: 0.0427
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9886 - loss:
0.0365 - val_accuracy: 0.9860 - val_loss: 0.0417
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9922 - loss:
0.0244 - val_accuracy: 0.9835 - val_loss: 0.0490
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9941 - loss:
0.0179 - val_accuracy: 0.9878 - val_loss: 0.0383
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9948 - loss:
0.0149 - val_accuracy: 0.9890 - val_loss: 0.0369
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9973 - loss:
0.0093 - val_accuracy: 0.9881 - val_loss: 0.0395
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9982 - loss:
0.0062 - val_accuracy: 0.9888 - val_loss: 0.0374
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9981 - loss:
0.0061 - val_accuracy: 0.9873 - val_loss: 0.0490
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9975 - loss:
0.0069 - val_accuracy: 0.9887 - val_loss: 0.0453
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9842 - loss:
0.0597
Test accuracy (MNIST): 98.87%

# ===========================
# Zadanie 3: Prognozowanie sekwencji przy użyciu GRU z warstwą Dropout
# ===========================

from tensorflow.keras.models import Sequential
```

```python
from tensorflow.keras.layers import GRU, Dropout, Dense, Input
import numpy as np
from tensorflow.keras.preprocessing.sequence import
TimeseriesGenerator

# Generowanie sztucznych danych sekwencyjnych
data = np.sin(np.linspace(0, 100, 1000))  # Sztuczne dane - funkcja
sinusoidalna
targets = np.roll(data, -1)  # Prognozowanie kolejnych wartości
targets[-1] = data[-1]  # Ostatnia wartość celu

# Przygotowanie danych do treningu (z pomocą TimeseriesGenerator)
sequence_length = 50  # Długość sekwencji wejściowej
generator = TimeseriesGenerator(data, targets, length=sequence_length,
batch_size=32)

# Model z warstwą GRU
model_gru = Sequential()
model_gru.add(Input(shape=(sequence_length, 1)))  # Warstwa wejściowa
(sekwencja długości 50)
model_gru.add(GRU(64, activation='relu'))  # Warstwa GRU
model_gru.add(Dropout(0.2))  # Warstwa Dropout
model_gru.add(Dense(1))  # Warstwa wyjściowa (prognoza jednej
wartości)

# Kompilacja modelu
model_gru.compile(optimizer='adam', loss='mean_squared_error')

# Trening modelu
model_gru.fit(generator, epochs=10)

# Prognozowanie na danych testowych
test_data = np.sin(np.linspace(100, 120, 100))  # Testowe dane
test_generator = TimeseriesGenerator(test_data, test_data,
length=sequence_length, batch_size=32)

# Wywołanie predict bez argumentów związanych z wieloprocesowością
predictions = model_gru.predict(test_generator)

# Wyświetlanie pierwszych kilku prognoz
print(f"Pierwsze prognozy: {predictions[:5]}")

Epoch 1/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.4422
Epoch 2/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.1706
Epoch 3/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0654
Epoch 4/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0461
```

```
Epoch 5/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0404
Epoch 6/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0229
Epoch 7/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0138
Epoch 8/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0110
Epoch 9/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0117
Epoch 10/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0113
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
Pierwsze prognozy: [[ 0.00698585]
 [-0.24499094]
 [-0.48419213]
 [-0.6633777 ]
 [-0.7823252 ]]
```

```
import tensorflow as tf
from tensorflow.keras.layers import LayerNormalization, Dense, Input
import numpy as np

# Zadanie 4: Wprowadzenie dwóch bloków Transformer Encoder
# ===========================

# Prosty model Transformer Encoder
class TransformerEncoder(tf.keras.layers.Layer):
    def __init__(self, num_heads, key_dim):
        super(TransformerEncoder, self).__init__()
        self.att =
tf.keras.layers.MultiHeadAttention(num_heads=num_heads,
key_dim=key_dim)
        self.norm1 = LayerNormalization()
        self.norm2 = LayerNormalization()
        self.ffn = tf.keras.Sequential([
            Dense(128, activation='relu'),
            Dense(64)
        ])

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)  # Multi-head attention
        out1 = self.norm1(attn_output + inputs)  # Residual connection
and layer normalization
        ffn_output = self.ffn(out1)  # Feedforward network
        out2 = self.norm2(ffn_output + out1)  # Another residual
connection and normalization
        return out2

# Przygotowanie danych (np. dla tekstu lub sekwencji)
```

```python
X_transformer = np.random.rand(100, 10, 64)  # Przykładowe dane: 100 próbek, 10 timesteps, 64 cechy
y_transformer = np.random.rand(100, 1)  # Przykładowe dane: 100 próbek, 1 wynik

# Model z Transformer Encoder
inputs = Input(shape=(10, 64))
x = TransformerEncoder(num_heads=2, key_dim=64)(inputs)
x = Dense(1)(x)  # Warstwa wyjściowa (np. regresja)
model_transformer = tf.keras.models.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)

# Wyświetlanie wyników
print(f"Pierwsze prognozy: {y_pred_transformer[:5]}")
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 1.8402
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 1.2445
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.6319
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.3776
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2069
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1814
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1561
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1483
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1433
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1131
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
Pierwsze prognozy: [[[0.28944317]
  [0.4014288 ]
  [0.49978384]
  [0.42256597]
  [0.50786173]
  [0.3463941 ]
  [0.51711893]
```

```
  [0.6571032 ]
  [0.45677057]
  [0.3832771 ]]

 [[0.540481  ]
  [0.4235727 ]
  [0.13946886]
  [0.6808267 ]
  [0.52350956]
  [0.3749635 ]
  [0.55630755]
  [0.66051346]
  [0.42626962]
  [0.6213175 ]]

 [[0.45441017]
  [0.53836185]
  [0.6687996 ]
  [0.54372895]
  [0.51846945]
  [0.48706612]
  [0.56204987]
  [0.58986545]
  [0.51089317]
  [0.48925003]]

 [[0.414041  ]
  [0.5310236 ]
  [0.7494235 ]
  [0.5361346 ]
  [0.45250866]
  [0.57787263]
  [0.5628777 ]
  [0.4776264 ]
  [0.40017447]
  [0.18394859]]

 [[0.60435855]
  [0.44212607]
  [0.4563798 ]
  [0.2787672 ]
  [0.54331326]
  [0.43808737]
  [0.40974176]
  [0.535267  ]
  [0.4645554 ]
  [0.30843037]]]
```