Laboratorium Analizy Procesów Uczenia.

Data wykonania ćwiczenia:	**10.05.2024**
Rok studiów:	**1**
Semestr:	1
Grupa studencka:	**1b**
Grupa laboratoryjna:	**-**

Ćwiczenie nr	5

Temat: Uczenie głębokie w R i Python. Klasyfikator obrazów za pomocą Keras.

Osoby wykonujące ćwiczenia:

1. Gracjan Wackermann

Katedra Informatyki i Automatyki

1. Cel ćwiczenia:

Celem jest uczenie głębokie za pomocą R i pakietu Keras.

2. Zadanie do wykonania:

Zadanie dotyczy sieci głębokiej w celu klasyfikacji obrazów pobranych ze zbioru danych.

3. MNIST database of handwritten digits

- Wariant nr. 3 -

Uzyskany kod:

```
import numpy as np
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt

# Wczytanie danych MNIST
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

# Sprawdzenie kształtu danych
print(f"x_train shape: {x_train.shape}")
print(f"{x_train.shape[0]} train samples")
print(f"{x_test.shape[0]} test samples")

# Skalowanie obrazów do zakresu [0, 1]
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# Upewnienie się, że obrazy mają kształt (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Konwersja etykiet na macierze binarne klas
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Architektura modelu
model = keras.Sequential([
    keras.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
```

```
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation="softmax")
])

# Wyświetlenie podsumowania modelu
model.summary()

# Kompilowanie modelu
model.compile(
    loss="categorical_crossentropy",
    optimizer="adam",
    metrics=["accuracy"]
)

# Trenowanie modelu
batch_size = 128
epochs = 15

history = model.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.1
)

# Ocena modelu na zbiorze testowym
score = model.evaluate(x_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")

# Prognozowanie nowych danych
predictions = model.predict(x_test)

# Wyświetlenie przykładowego obrazu z jego przewidywaną klasą
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {np.argmax(y_test[i])}\nPred:
{np.argmax(predictions[i])}")
    plt.axis('off')
plt.show()
```

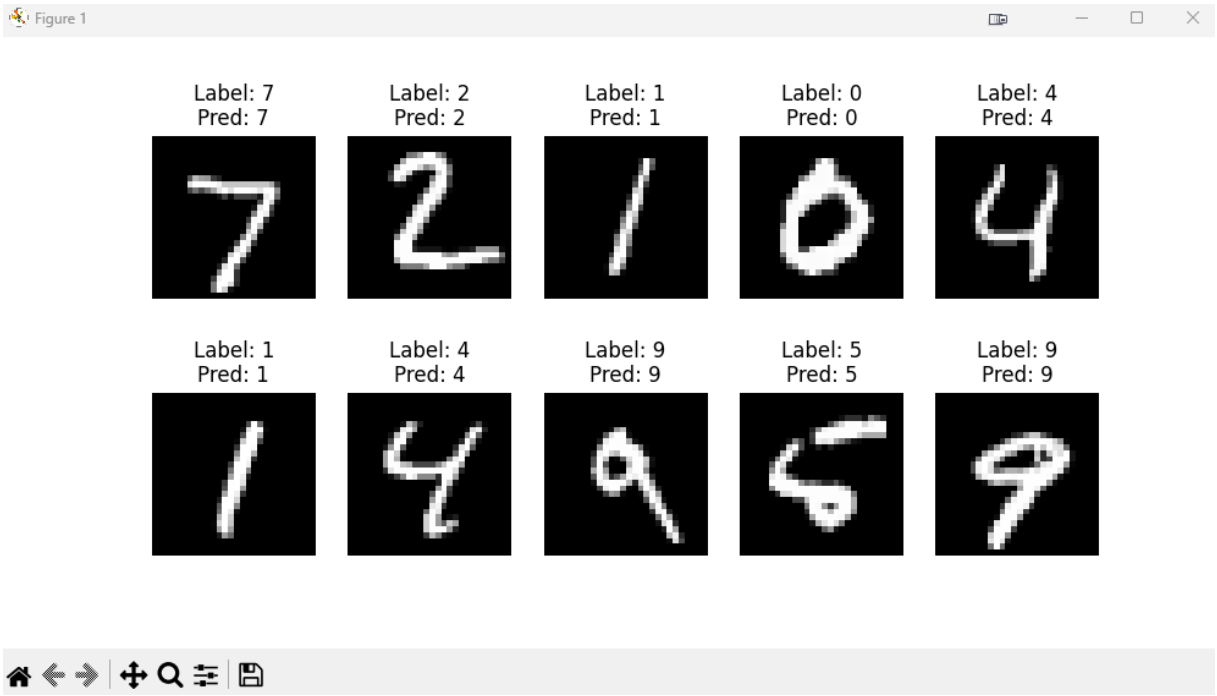
Wyniki z konsol:

```
x_train shape: (60000, 28, 28)
60000 train samples
10000 test samples
2024-06-13 11:11:44.997973: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16,010

```
Total params: 34,826 (136.04 KB)
Trainable params: 34,826 (136.04 KB)
Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.7695 - loss: 0.7543 - val_accuracy: 0.9775 - val_loss: 0.0823
Epoch 2/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9614 - loss: 0.1229 - val_accuracy: 0.9858 - val_loss: 0.0567
Epoch 3/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9742 - loss: 0.0866 - val_accuracy: 0.9867 - val_loss: 0.0466
Epoch 4/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9780 - loss: 0.0739 - val_accuracy: 0.9880 - val_loss: 0.0440
Epoch 5/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9811 - loss: 0.0599 - val_accuracy: 0.9908 - val_loss: 0.0371
Epoch 6/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9830 - loss: 0.0557 - val_accuracy: 0.9910 - val_loss: 0.0373
Epoch 7/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9849 - loss: 0.0489 - val_accuracy: 0.9900 - val_loss: 0.0346
Epoch 8/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9855 - loss: 0.0475 - val_accuracy: 0.9913 - val_loss: 0.0315
Epoch 9/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9862 - loss: 0.0440 - val_accuracy: 0.9928 - val_loss: 0.0291
Epoch 10/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9853 - loss: 0.0451 - val_accuracy: 0.9925 - val_loss: 0.0307
Epoch 11/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9878 - loss: 0.0383 - val_accuracy: 0.9920 - val_loss: 0.0319
Epoch 12/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9868 - loss: 0.0411 - val_accuracy: 0.9932 - val_loss: 0.0283
Epoch 13/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9893 - loss: 0.0345 - val_accuracy: 0.9917 - val_loss: 0.0321
Epoch 14/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9895 - loss: 0.0310 - val_accuracy: 0.9923 - val_loss: 0.0290
Epoch 15/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9888 - loss: 0.0330 - val_accuracy: 0.9900 - val_loss: 0.0320
Test loss: 0.027467403560876846
Test accuracy: 0.9918000102043152
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



3. **Wnioski:**

- Na zajęciach poznaliśmy sposoby na konstruowanie sieci głębokiej w celu klasyfikacji obrazów pobranych ze zbioru danych. Uzyskany model trafnie rozróżnia ręcznie napisane liczby i przypisuje do nich poprawne etykiety. Dowodzi to poprawności działania kodu.

Link do repozytorium: https://github.com/fireinx/apu