Uniwersytet Bielsko-Bialski

Prognozowanie parametrów zdrowotnych pacjentów

Zadanie ARCH

Sprawozdanie z ćwiczeń
Nauka o Danych II

Data wykonania:
28.06.2025

Autor:
Bartosz Bieniek 058085

1. Cel ćwiczenia

Zadania umieścić na Github.

2. Przebieg ćwiczenia

1. Importowanie bibliotek

Zaimportowano niezbędne biblioteki do implementacji modelu autoenkodera oraz przetwarzania i wizualizacji danych. Wykorzystano TensorFlow z API Keras, bibliotekę NumPy do obliczeń oraz Matplotlib do tworzenia wykresów.

```python
# 1. Importowanie bibliotek
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
```

Rys. 1. Importowanie bibliotek

2. Załadowanie zbioru danych MNIST

Pobrano zbiór danych MNIST, zawierający obrazy cyfr od 0 do 9. Następnie przeskalowano wartości pikseli do zakresu [0, 1] i spłaszczono obrazy do wektorów o długości 784 cech, przygotowując dane do treningu autoenkodera w formacie tablicowym.

```python
# 2. Załaduj dane MNIST
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.
x_test = x_test.astype("float32") / 255.
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)
```

Rys. 2. Załadowanie zbioru danych MNIST

3. Filtrowanie danych – wybór klas 0–4

Z danych treningowych wybrano tylko próbki należące do klas od 0 do 4. Umożliwiło to zbudowanie modelu uczonego wyłącznie na danych „normalnych”, co jest istotne w zastosowaniach detekcji anomalii.

```python
# 3. Filtruj tylko klasy 0-4 do treningu
train_mask = y_train <= 4
x_train_filtered = x_train[train_mask]
```

Rys. 3. Filtrowanie danych – wybór klas 0–4

4. Budowa architektury autoenkodera

Zaprojektowano symetryczny autoenkoder składający się z dwóch warstw kodujących (128 i 64 neurony) oraz dwóch warstw dekodujących (128 i 784 neurony). Zastosowano funkcję aktywacji ReLU oraz na wyjściu funkcję sigmoidalną w celu uzyskania wartości rekonstrukcji w przedziale [0, 1]. Model skompilowano z funkcją straty mse i optymalizatorem Adam.

```
# 4. Budowa autoenkodera
input_dim = 784
encoding_dim = 64

input_img = tf.keras.Input(shape=(input_dim,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

decoded = layers.Dense(128, activation='relu')(encoded)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

Rys. 4. Budowa architektury autoenkodera

5. Trening autoenkodera

Model autoenkodera został wytrenowany na danych klas 0–4 w ciągu 20 epok, z użyciem batchy po 256 próbek i walidacją na 20% zbioru treningowego. Trening pozwolił modelowi nauczyć się kompresji i odtwarzania danych należących wyłącznie do klas „normalnych”.

```
# 5. Trening autoenkodera
autoencoder.fit(x_train_filtered, x_train_filtered,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_split=0.2)
```

Rys. 5. Trening autoenkodera

6. Ewaluacja na zbiorze testowym (klasy 0–9)

Przeprowadzono rekonstrukcję obrazów z pełnego zbioru testowego, obejmującego wszystkie cyfry 0–9. Obliczono błędy rekonstrukcji jako średni błąd kwadratowy dla każdej próbki, co umożliwiło późniejsze wyznaczenie odchyleń od danych uczonych.

```
# 6. Ewaluacja na pełnym zbiorze testowym (klasy 0–9)
reconstructions = autoencoder.predict(x_test)
reconstruction_errors = np.mean(np.square(x_test - reconstructions), axis=1)
```

Rys. 6. Ewaluacja na zbiorze testowym (klasy 0–9)

7. Wyznaczenie progu detekcji anomalii

Na podstawie błędów rekonstrukcji dla klas 0–4 wyznaczono próg detekcji anomalii. Próg ustalono jako średni błąd rekonstrukcji plus dwa odchylenia standardowe, co miało na celu wychwycenie próbek wyraźnie odbiegających od danych „normalnych”.

```
# 7. Ustal próg detekcji na podstawie klas 0–4
test_mask_0_4 = y_test <= 4
threshold = np.mean(reconstruction_errors[test_mask_0_4]) + 2 * np.std(reconstruction_errors[test_mask_0_4])
```

Rys. 7. Wyznaczenie progu detekcji anomalii

8. Wykrywanie anomalii

Zidentyfikowano próbki testowe jako anomalie, jeśli ich błąd rekonstrukcji przekroczył wyznaczony próg. Jako dane anomalne potraktowano próbki należące do klas 5–9, które nie występowały w zbiorze treningowym.

```
# 8. Wykrywanie anomalii
y_pred_anomaly = reconstruction_errors > threshold
y_true_anomaly = y_test > 4  # klasy 5-9 to anomalie
```

Rys. 8. Wykrywanie anomalii

9. Ocena skuteczności detekcji anomalii

Wykorzystano metryki klasyfikacyjne do oceny skuteczności wykrywania anomalii. Obliczono dokładność, precyzję i recall oraz zaprezentowano macierz pomyłek, pozwalającą ocenić skuteczność klasyfikacji binarnej na poziomie wykrywania nietypowych danych.

```
# 9. Ocena skuteczności
from sklearn.metrics import classification_report, confusion_matrix

print("Raport klasyfikacji (anomalia = 1):")
print(classification_report(y_true_anomaly, y_pred_anomaly))
print("Macierz pomyłek:")
print(confusion_matrix(y_true_anomaly, y_pred_anomaly))
```

Rys. 9. Ocena skuteczności detekcji anomalii

10. Wizualizacja błędów rekonstrukcji

Wygenerowano histogramy błędów rekonstrukcji osobno dla danych „normalnych” (klasy 0–4) i anomalii (klasy 5–9). Na wykresie zaznaczono również próg detekcji. Wizualizacja pozwoliła dostrzec wyraźne różnice w rozkładzie błędów między dwiema grupami, co potwierdziło skuteczność podejścia autoenkodera do detekcji anomalii.

```python
# 10. Przykładowe błędy rekonstrukcji
plt.hist(reconstruction_errors[test_mask_0_4], bins=50, alpha=0.6, label='Klasy 0–4')
plt.hist(reconstruction_errors[~test_mask_0_4], bins=50, alpha=0.6, label='Klasy 5–9 (anomalie)')
plt.axvline(threshold, color='red', linestyle='--', label='Próg detekcji')
plt.xlabel("Błąd rekonstrukcji")
plt.ylabel("Liczba próbek")
plt.legend()
plt.title("Histogram błędu rekonstrukcji")
plt.show()
```

Rys. 10. Wizualizacja błędów rekonstrukcji

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 128)	8,320
dense_3 (Dense)	(None, 784)	101,136

Total params: 218,192 (852.31 KB)

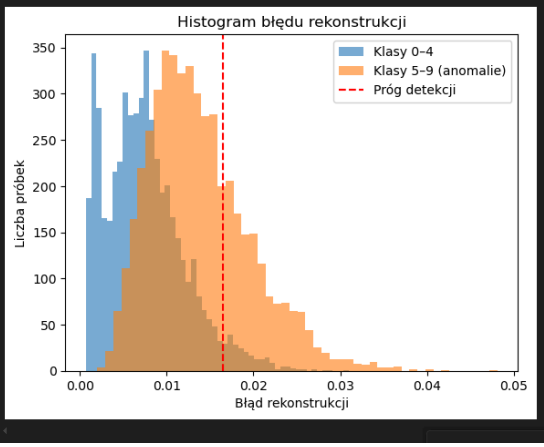
Trainable params: 218,192 (852.31 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1210 - val_loss: 0.0446
Epoch 2/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0407 - val_loss: 0.0307
Epoch 3/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0288 - val_loss: 0.0237
Epoch 4/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0226 - val_loss: 0.0190
Epoch 5/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0182 - val_loss: 0.0160
Epoch 6/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0154 - val_loss: 0.0145
Epoch 7/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0139 - val_loss: 0.0131
Epoch 8/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0129 - val_loss: 0.0123
Epoch 9/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0118 - val_loss: 0.0115
Epoch 10/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0112 - val_loss: 0.0109
Epoch 11/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0106 - val_loss: 0.0103
Epoch 12/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0100 - val_loss: 0.0099
Epoch 13/20
...

Macierz pomyłek:
[[4913  226]
 [3503 1358]]
```

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



Rys. 11 Wynik wykonywania programu

3. Wnioski

Na podstawie przeprowadzonego eksperymentu można sformułować następujące wnioski. Autoenkoder wytrenowany wyłącznie na danych reprezentujących klasy 0–4 nauczył się efektywnej reprezentacji i rekonstrukcji

tych wzorców, co umożliwiło skuteczne wykrywanie anomalii w postaci cyfr z klas 5–9.