SPRAWOZDANIE

Zajęcia: Uczenie Maszynowe

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 4 Data 11.01.2025 Temat: "5. Implementacja algorytm´ow optymalizacji gradientowej do trenowania modeli. 6. Projektowanie i trening prostych sieci neuronowych w TensorFlow lub PyTorch. 7. Zastosowanie konwolucyjnych sieci neuronowych (CNN) do analizy obrazu" Wariant 11	Szymon Nycz Informatyka II stopień, niestacjonarne, 1 semestr, gr.1b

1. Polecenie:

   Powikłania zawału mięśnia sercowego:
   https://www.kaggle.com/datasets/rafatashrafjoy/myocardial-infarction-complications

2. Link do repozytorium:

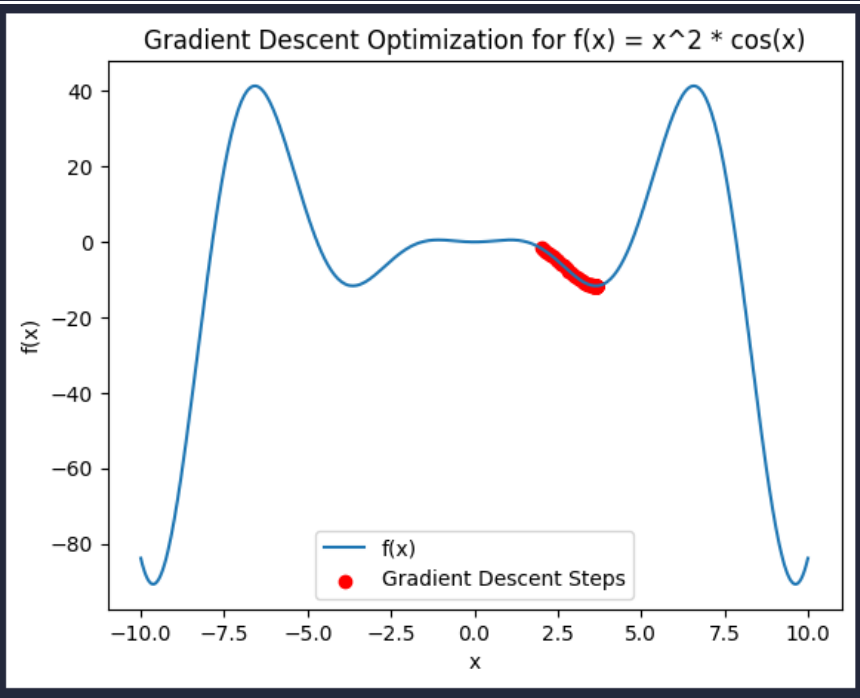
   Link: https://github.com/Maciek332/Semestr_1_Nycz/tree/master/UM

3. Opis programu opracowanego

1) Zaimplementuj w Pythonie optymalizację funkcji metodą spadku gradientu wraz z wizualizacją.

```python
import numpy as np
import matplotlib.pyplot as plt

def gradient_descent(f, grad_f, theta_init, learning_rate, iterations):
    theta = theta_init
    history = [theta]
    for i in range(iterations):
        theta -= learning_rate * grad_f(theta)
        history.append(theta)
    return theta, history

# Definicja funkcji i jej pochodnej
f = lambda x: x**2 * np.cos(x)
grad_f = lambda x: 2*x*np.cos(x) - x**2*np.sin(x)

# Parametry
theta_init = 2.0
learning_rate = 0.01
iterations = 100

# Optymalizacja
optimal_theta, history = gradient_descent(f, grad_f, theta_init, learning_rate, iterations)

# Wizualizacja
x = np.linspace(-10, 10, 400)
y = f(x)

plt.plot(x, y, label='f(x)')
plt.scatter(history, f(np.array(history)), color='red', label='Gradient Descent Steps')
plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent Optimization for f(x) = x^2 * cos(x)')
plt.show()

print("Optymalne theta:", optimal_theta)
```

✓ 0.9s

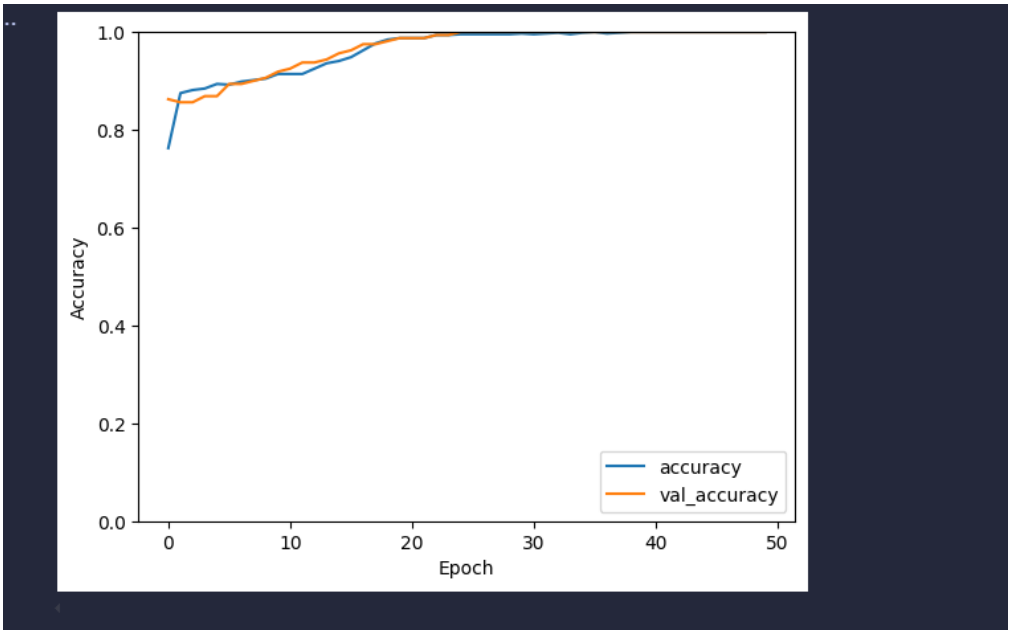



```
Optymalne theta: 3.6435970380919547
```

2) Stwórz w Pythonie najprostszą sieć neuronową wraz z ewaluacją i prognozowaniem.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.layers import Dense    Import "tensorflow.keras.layers" could not be resolved

# Wygenerowanie danych make_moons
X, y = make_moons(n_samples=1000, noise=0.1, random_state=42)

# Standaryzacja danych
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Budowa modelu
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')  # Warstwa wyjściowa do klasyfikacji binarnej
])

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

# Trening
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Ocena modelu
loss, accuracy = model.evaluate(X_test, y_test)
print("Strata na danych testowych:", loss)
print("Dokładność na danych testowych:", accuracy)

# Wizualizacja
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/50
c:\Users\szymo\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.6322 - loss: 0.6372 - val_accuracy: 0.8625 - val_loss: 0.4851
Epoch 2/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8549 - loss: 0.4458 - val_accuracy: 0.8562 - val_loss: 0.3400
Epoch 3/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9015 - loss: 0.2973 - val_accuracy: 0.8562 - val_loss: 0.2732
Epoch 4/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8902 - loss: 0.2603 - val_accuracy: 0.8687 - val_loss: 0.2511
Epoch 5/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9045 - loss: 0.2328 - val_accuracy: 0.8687 - val_loss: 0.2404
Epoch 6/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9091 - loss: 0.2215 - val_accuracy: 0.8938 - val_loss: 0.2308
Epoch 7/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8837 - loss: 0.2418 - val_accuracy: 0.8938 - val_loss: 0.2218
Epoch 8/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8859 - loss: 0.2356 - val_accuracy: 0.9000 - val_loss: 0.2112
Epoch 9/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9052 - loss: 0.1916 - val_accuracy: 0.9062 - val_loss: 0.1995
Epoch 10/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9101 - loss: 0.2046 - val_accuracy: 0.9187 - val_loss: 0.1872
Epoch 11/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9204 - loss: 0.1809 - val_accuracy: 0.9250 - val_loss: 0.1734
Epoch 12/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9168 - loss: 0.1699 - val_accuracy: 0.9375 - val_loss: 0.1593
Epoch 13/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9146 - loss: 0.1728 - val_accuracy: 0.9375 - val_loss: 0.1435
...
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0066 - val_accuracy: 1.0000 - val_loss: 0.0023
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0022
Strata na danych testowych: 0.0028550871647894382
Dokładność na danych testowych: 1.0
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

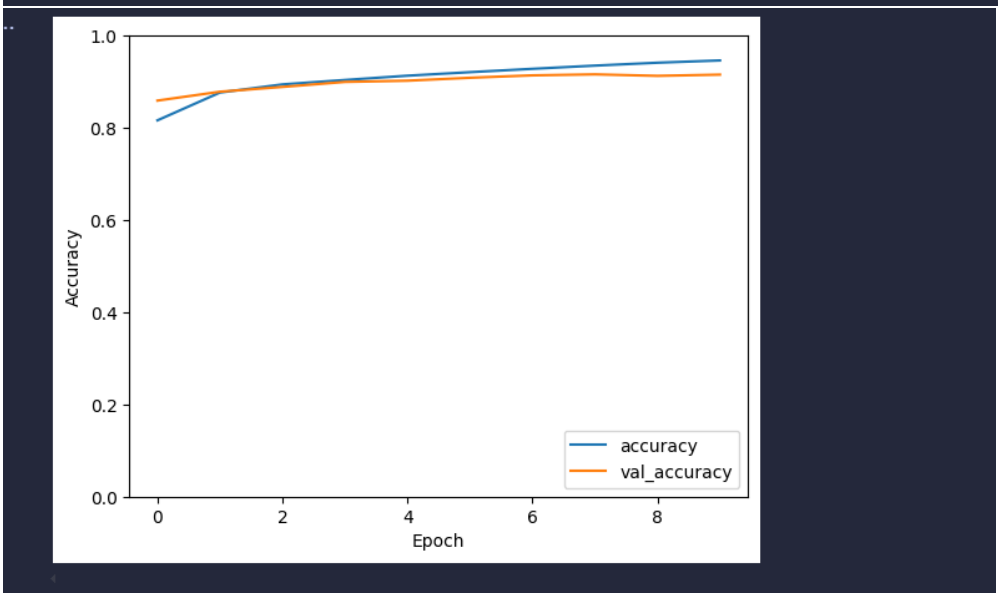


3) Zaprojektuj, wytrenuj i przetestuj sieć konwolucyjną, wykorzystując jeden z dostępnych w Pythonie podstawowych zbiorów danych.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPooling2D    Import "tensorflow.keras.layers" could not be resolved
from tensorflow.keras.datasets import fashion_mnist    Import "tensorflow.keras.datasets" could not be resolved
from tensorflow.keras.utils import to_categorical    Import "tensorflow.keras.utils" could not be resolved

# Wczytanie danych Fashion MNIST
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# Normalizacja danych
X_train = X_train / 255.0
X_test = X_test / 255.0

# Dodanie wymiaru dla kanałów (konieczne dla Conv2D)
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]

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

# Budowa modelu
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Warstwa wyjściowa do klasyfikacji 10 klas
])

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

# Trening
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Ocena modelu
loss, accuracy = model.evaluate(X_test, y_test)
print("Strata na danych testowych:", loss)
print("Dokładność na danych testowych:", accuracy)
```

```python
# Wizualizacja
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 3us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
c:\Users\szymo\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 16s 19ms/step - accuracy: 0.7457 - loss: 0.7229 - val_accuracy: 0.8590 - val_loss: 0.3910
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.8696 - loss: 0.3544 - val_accuracy: 0.8783 - val_loss: 0.3341
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.8909 - loss: 0.3001 - val_accuracy: 0.8887 - val_loss: 0.3034
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.9055 - loss: 0.2582 - val_accuracy: 0.8995 - val_loss: 0.2785
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9129 - loss: 0.2384 - val_accuracy: 0.9022 - val_loss: 0.2740
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.9196 - loss: 0.2176 - val_accuracy: 0.9085 - val_loss: 0.2540
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.9284 - loss: 0.1940 - val_accuracy: 0.9137 - val_loss: 0.2464
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.9343 - loss: 0.1754 - val_accuracy: 0.9159 - val_loss: 0.2417
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 13s 17ms/step - accuracy: 0.9417 - loss: 0.1599 - val_accuracy: 0.9127 - val_loss: 0.2442
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.9474 - loss: 0.1405 - val_accuracy: 0.9153 - val_loss: 0.2464
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9114 - loss: 0.2615
Strata na danych testowych: 0.2572509634304047
Dokładność na danych testowych: 0.9115999937057495
```



4. Wnioski

Metoda optymalizacji gradientowej jest fundamentalnym narzędziem używanym do minimalizowania funkcji kosztu J(θ) w trakcie trenowania modeli. Kategorialna krosentropia to często stosowana funkcja kosztu w zadaniach klasyfikacji wieloklasowej, która skutecznie penalizuje model za niskie prawdopodobieństwo przypisane prawdziwej klasie. Sieci neuronowe przetwarzają dane wejściowe, aby uzyskać wyniki. Struktura modelu obejmuje warstwę wejściową, ukryte warstwy transformujące dane przy pomocy funkcji aktywacji oraz warstwę wyjściową, która generuje końcowe wyniki. Konwolucyjne sieci neuronowe (CNN) przetwarzają obrazy, rozpoznając lokalne wzorce dzięki operacjom konwolucji.