Akademia Łomżyńska

Wydział nauk Informatyczno-Technologicznych



**Wydziałowy projekt zespołowy**

Rozpoznawanie odzieży za pomocą bazy danych Fashion MNIST.

**Zespół autorski**

Łada Agata

Łaszczych Agata

**Prowadzący ćwiczenia**

dr Janusz Rafałko

Informatyka

Studia stacjonarne I stopnia, rok IV, semestr VII

Rok akademicki: 2024/2025

**Spis treści**

[1. Temat 2](#_Toc183466572)

[2. Opis i wstępne założenia projektu 2](#_Toc183466573)

[3. Podział zadań 2](#_Toc183466574)

[4. Język programowania 2](#_Toc183466575)

[5. Przegląd literatury 3](#_Toc183466576)

[6. Trenowanie kodu 3](#_Toc183466577)

[7. Implementacja kodu 9](#_Toc183466578)

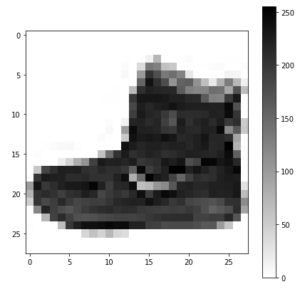
[BIBLIOGRAFIA 13](#_Toc183466579)

### Temat

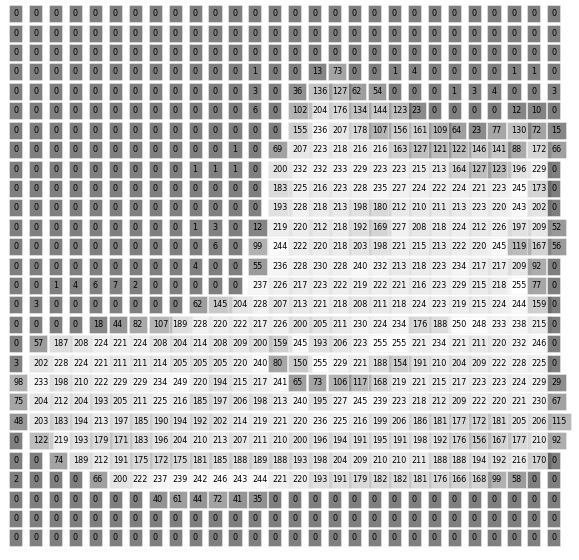
Rozpoznawanie odzieży za pomocą bazy danych Fashion MNIST.

### Opis i wstępne założenia projektu

Projekt ma na celu opracowanie systemu rozpoznawania różnych rodzajów odzieży przy użyciu technik głębokiego uczenia, z wykorzystaniem zestawu danych Fashion MNIST. Zbiór danych zawiera obrazy w skali szarości od 0 do 255 o rozdzielczości 28x28 pikseli, sklasyfikowane w 10 różnych kategoriach odzieży.



Zdj.1. Skala zdjęcia 28x28 pikseli



Zdj.2. Skala szarości od 0 do 255

### Podział zadań

Analiza i obróbka danych – Agata Łada oraz Agata Łaszczych

Zaznajomienie się z literaturą – Agata Łada oraz Agata Łaszczych.

Implementacja modelu – Agata Łada.

Trenowanie modelu – Agata Łada.

Testowanie modelu – Agata Łaszczych.

Dokumentacja i raport – Agata Łaszczych.

### Język programowania

Projekt zostanie zrealizowany w języku Python, z wykorzystaniem bibliotek takich jak:

* **TensorFlow** – służy ona do implementacji i korzystania z innowacyjnych metod uczenia maszynowego. Nazwa **TensorFlow** jest nawiązaniem do sposobu działania sieci neuronowych, czyli rozmaitych operacji matematycznych realizowanych za pośrednictwem wielowymiarowych tablic danych, które w terminologii matematycznej są nazywane tensorami.
* **Keras** - pozwala na szybkie i proste tworzenie modeli uczenia maszynowego
* **NumPy** - Umożliwia tworzenie obiektów reprezentujących wielowymiarowe tablice oraz dostarcza funkcje do szybkich operacji na tablicach wielowymiarowych. oraz
* **Matplotlib** -  najpopularniejsza biblioteka do tworzenia wykresów w Pythonie. Pozwala tworzyć wykresy liniowe, słupkowe, kołowe, a także bardziej zaawansowane wizualizacje.

https://www.tensorflow.org/datasets/catalog/fashion\_mnist?hl=pl

https://matplotlib.org/

### Przegląd literatury

**Artykuły pomocnicze:**

https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/

„Deep Learning CNN for Fashion MNIST Classification” Jason Brownlee, 2020

– omawia zastosowanie konwolucyjnych sieci neuronowych (CNN) do klasyfikacji obrazów, koncentrując się na zestawie danych Fashion MNIST. Kluczowe wnioski obejmują skuteczność CNN w przetwarzaniu obrazów oraz wpływ różnych architektur na wydajność modelu.

https://ieeexplore.ieee.org/abstract/document/8388338

„Data augmentation for improving deep learning in image classification problem” Agnieszka Mikołajczyk, Michał Grochowski, 2018

– opisuje różne techniki augmentacji danych, takie jak odwracanie, rotacja i skalowanie, oraz ich korzyści w zapobieganiu nadmiernemu dopasowaniu modelu i poprawie dokładności.

https://ieeexplore.ieee.org/abstract/document/8474802

„Transfer Learning in Image Classification” Manali Shaha, Meenakshi Pawar, 2018

– bada koncepcję uczenia transferowego, w którym wcześniej wytrenowane modele na dużych zbiorach danych (np. ImageNet) są dostosowywane do specyficznych zadań, co może poprawić wydajność na mniejszych zbiorach danych, takich jak Fashion MNIST.

### Trenowanie kodu

* **Kod treningu**

**import tensorflow as tf**

**import numpy as np**

**# Załaduj zestaw danych Fashion MNIST**

**fashion\_mnist = tf.keras.datasets.fashion\_mnist**

**(X\_train, y\_train), (X\_val, y\_val) = fashion\_mnist.load\_data()**

**# Normalizacja danych (zmiana zakresu pikseli z 0-255 na 0-1)**

**X\_train = X\_train.astype('float32') / 255.0**

**X\_val = X\_val.astype('float32') / 255.0**

**# Konwersja etykiet na one-hot encoding**

**y\_train\_one\_hot = tf.keras.utils.to\_categorical(y\_train, 10)**

**y\_val\_one\_hot = tf.keras.utils.to\_categorical(y\_val, 10)**

**# Budowanie modelu MLP**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Flatten, Dense**

**model = Sequential([**

**Flatten(input\_shape=(28, 28)),  # Spłaszczenie obrazu 28x28 pikseli do wektora**

**Dense(128, activation='relu'),  # Warstwa ukryta z 128 neuronami**

**Dense(10, activation='softmax')  # Warstwa wyjściowa dla 10 klas ])**

**# Kompilowanie modelu**

**model.compile(optimizer='adam',**

**loss='categorical\_crossentropy',**

**metrics=['accuracy'])**

**# Trening modelu**

**history = model.fit(X\_train, y\_train\_one\_hot,  # Dane treningowe i etykiety**

**epochs=10,                # Liczba epok**

**batch\_size=256,            # Rozmiar wsadu**

**validation\_data=(X\_val, y\_val\_one\_hot),  # Dane walidacyjne**

**verbose=1)                # Wyświetlanie postępu**

**# Wyświetlanie wyników treningu**

**import matplotlib.pyplot as plt**

**# Wykres dokładności**

**plt.plot(history.history['accuracy'], label='Dokładność treningu')**

**plt.plot(history.history['val\_accuracy'], label='Dokładność walidacji')**

**plt.title('Dokładność treningu i walidacji')**

**plt.xlabel('Epoka')**

**plt.ylabel('Dokładność')**

**plt.legend()**

**plt.show()**

**# Wykres straty**

**plt.plot(history.history['loss'], label='Strata treningu')**

**plt.plot(history.history['val\_loss'], label='Strata walidacji')**

**plt.title('Strata treningu i walidacji')**

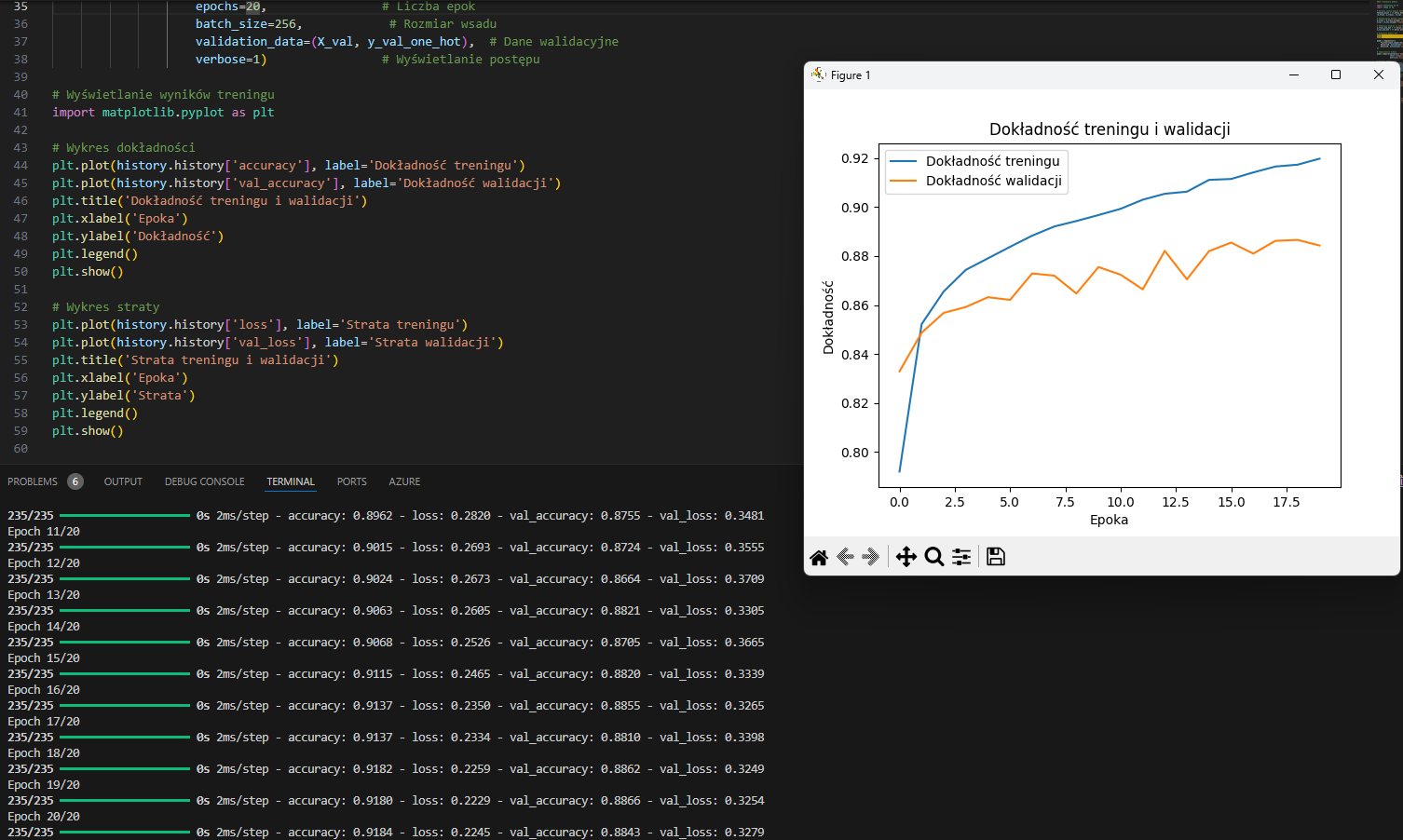
**plt.xlabel('Epoka')**

**plt.ylabel('Strata')**

**plt.legend()**

**plt.show()**

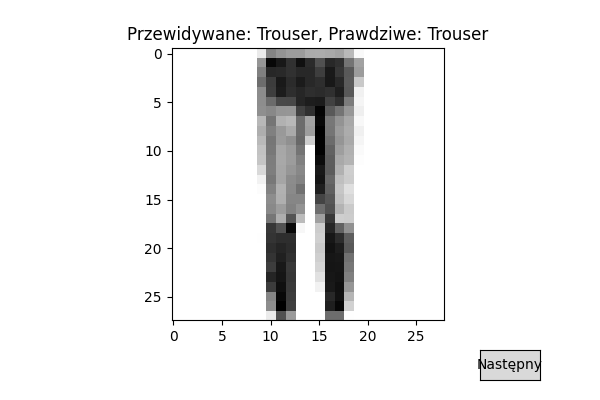
**Trening dla 20 epok**

****

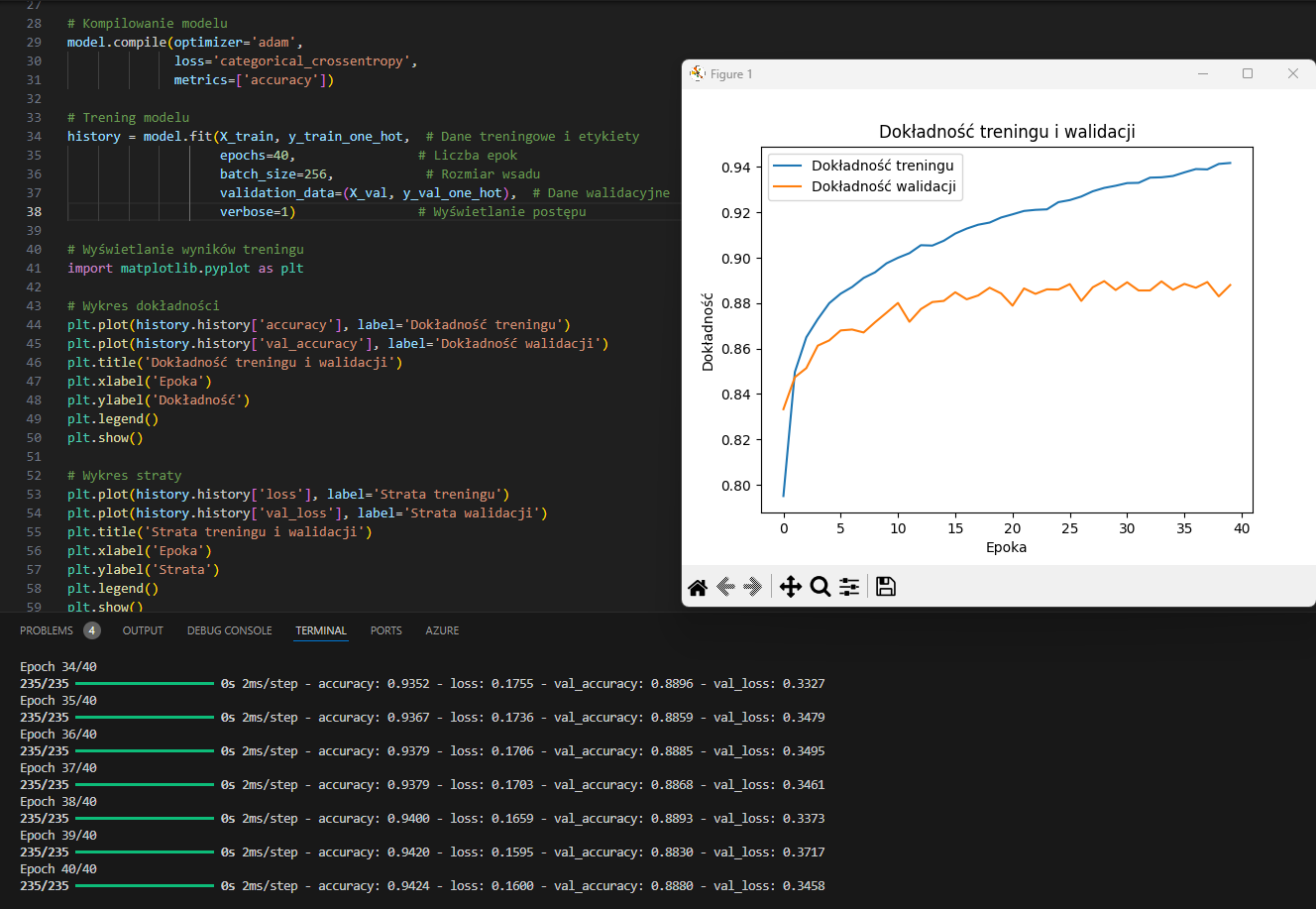
**Zdj.1.** Wykres dokładności treningu

****

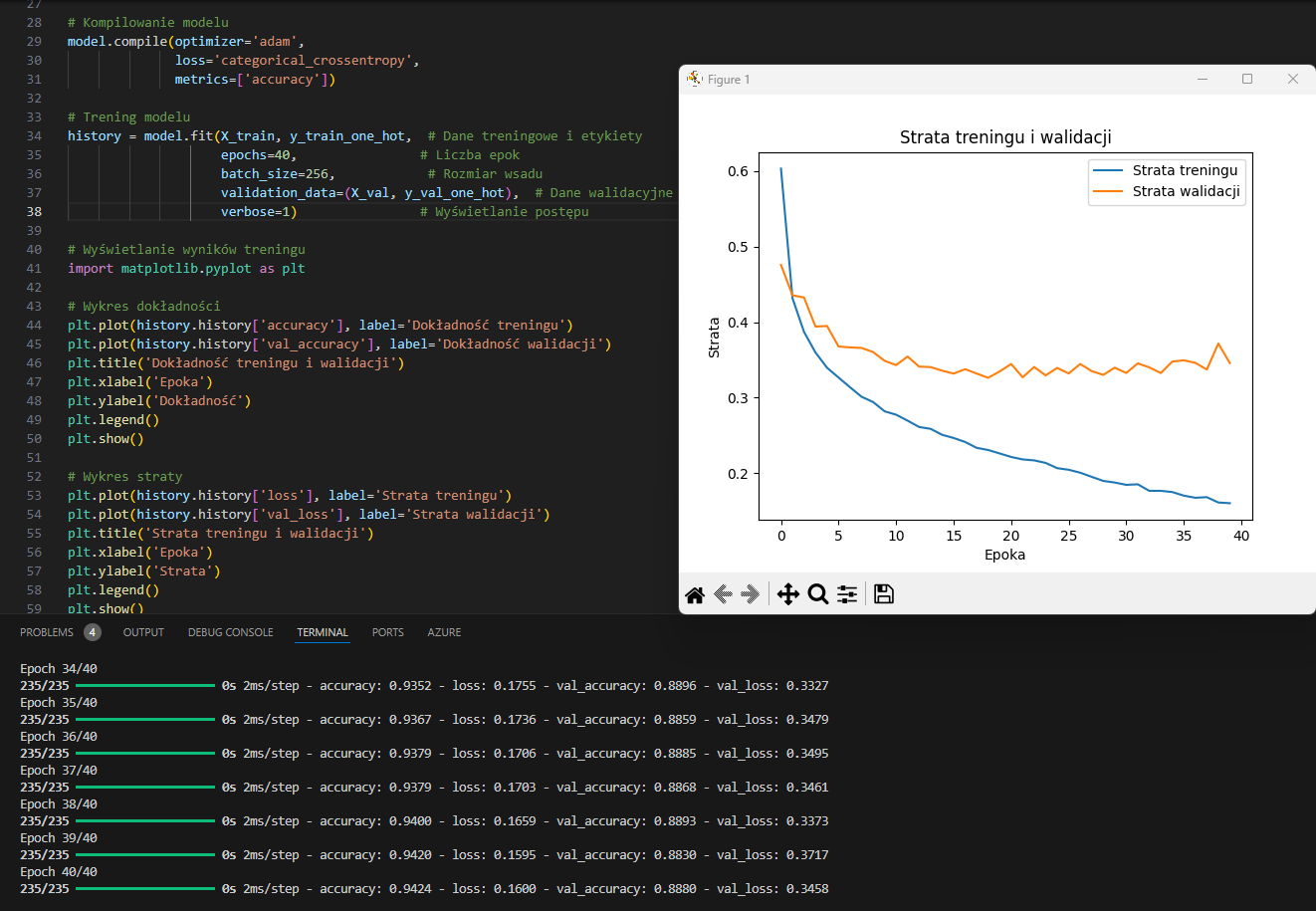
**Zdj.2.** Wykres strat treningu

****

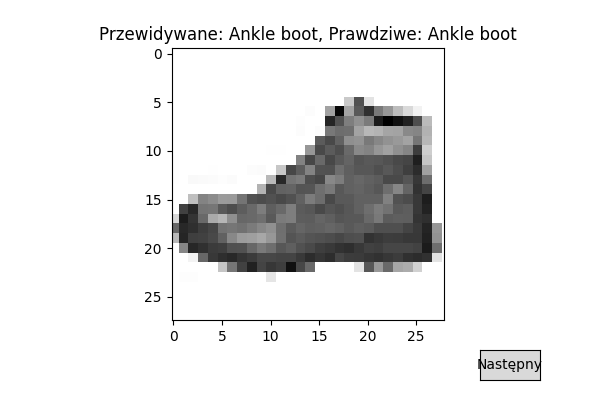
**Zdj.3.** Wynik treningu

**Trening dla 40 epok**

**Zdj.4.** Wykres dokładności treningu

****

**Zdj.5.** Wykres strat treningu

****

**Zdj.6.** Wynik treningu

### Implementacja kodu

**#import bibliotek**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.widgets import Button

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageOps

**# Załaduj zestaw danych Fashion MNIST**

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_val, y\_val) = fashion\_mnist.load\_data()

**# Normalizacja danych (zmiana zakresu pikseli z 0-255 na 0-1)**

X\_val = X\_val.astype('float32') / 255.0

**# Etykiety klas**

class\_names = ['T-shirt/top', 'Spodnie', 'Sweter', 'Sukienka', 'Płaszcz',

               'Sandał', 'Koszula', 'Sneaker', 'Torebka', 'Kozak/botki]

**# Konwersja etykiet na one-hot encoding**

y\_val\_one\_hot = tf.keras.utils.to\_categorical(y\_val, len(class\_names))

**# Budowanie modelu MLP**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense

model = Sequential()

model.add(Flatten(input\_shape=(28, 28)))  # Warstwa wejściowa

model.add(Dense(128, activation='relu'))  # Ukryta warstwa z 128 neuronami

model.add(Dense(10, activation='softmax'))  # Warstwa wyjściowa (10 klas)

**# Kompilowanie modelu**

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

**# Trening modelu**

model.fit(X\_train.astype('float32') / 255.0,

          tf.keras.utils.to\_categorical(y\_train),

          epochs=30, batch\_size=256, verbose=1)

**# Klasyfikacja obrazów ze zbioru walidacyjnego**

y\_val\_pred = model.predict(X\_val)

**# Interaktywny system do przewijania zdjęć**

class ImageViewer:

    def \_\_init\_\_(self, predictions, true\_labels, images):

        self.predictions = predictions

        self.true\_labels = true\_labels

        self.images = images

        self.index = 0

ax\_upload = plt.axes([0.1, 0.05, 0.1, 0.075])

        self.btn\_upload = Button(ax\_upload, 'Wgraj obraz')

        self.btn\_upload.on\_clicked(self.upload\_image)

**# Rysowanie pierwszego obrazu**

        self.fig, self.ax = plt.subplots(figsize=(6, 4))

        plt.subplots\_adjust(bottom=0.2)

        self.im = self.ax.imshow(self.images[self.index], cmap=plt.cm.binary)

        self.pred\_text = self.ax.set\_title(self.\_get\_image\_title(self.index))

**# Przyciski "Następny"**

        ax\_next = plt.axes([0.8, 0.05, 0.1, 0.075])

        self.btn\_next = Button(ax\_next, 'Następny')

        self.btn\_next.on\_clicked(self.next\_image)

        plt.show()

 def \_get\_image\_title(self, i):

        predicted\_label = np.argmax(self.predictions[i])

        confidence = np.max(self.predictions[i]) \* **100  # Wyciągnij maksymalne prawdopodobieństwo i zamień na %**

        true\_value = self.true\_labels[i]

        return (f"Przewidywane: {class\_names[predicted\_label]} ({confidence:.2f}%), **#miejsca po przecinku**

                f"Prawdziwe: {class\_names[true\_value]}")

    def update\_image(self):

        self.im.set\_data(self.images[self.index])

        self.pred\_text.set\_text(self.\_get\_image\_title(self.index))

        self.fig.canvas.draw\_idle()  **# Odświeżanie interfejsu**

    def next\_image(self, event):

        self.index = (self.index + 1) % len(self.images)

        self.update\_image()

    def upload\_image(self, event):

        file\_path = filedialog.askopenfilename(

            title="Wybierz obraz",

            filetypes=[("Obrazy", "\*.png;\*.jpg;\*.jpeg;\*.bmp")]

        )

        if file\_path:

**# Wczytanie obrazu**

            img = Image.open(file\_path)

**# Przeskalowanie do rozmiaru 28x28 i zamiana na skalę szarości**

            img = ImageOps.grayscale(img.resize((28, 28)))

**# Zamiana na tablicę numpy**

            img\_array = np.array(img) / 255.0  **# Normalizacja do zakresu [0, 1]**

**# Przewidywanie klasy**

            pred = model.predict(img\_array.reshape(1, 28, 28))

            predicted\_label = np.argmax(pred)

            confidence = np.max(pred) \* 100

        # Wyświetl wgrany obraz

            self.images = [img\_array]  # Zamień aktualną listę obrazów na wgrany obraz

            self.predictions = pred  # Przewidywanie dla wgranego obrazu

            self.true\_labels = [predicted\_label]  # Symulacja poprawnej etykiety

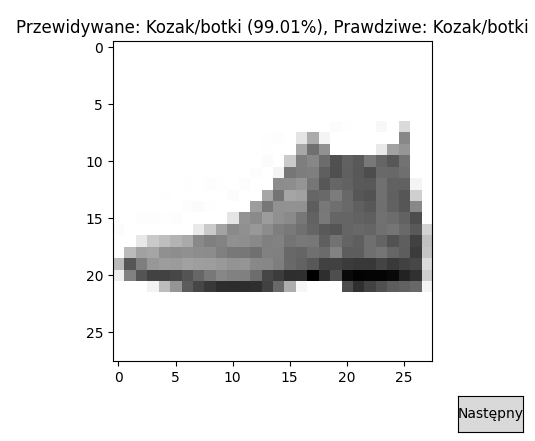
            self.index = 0  # Zresetuj indeks

            self.update\_image()  # Zaktualizuj interfejs

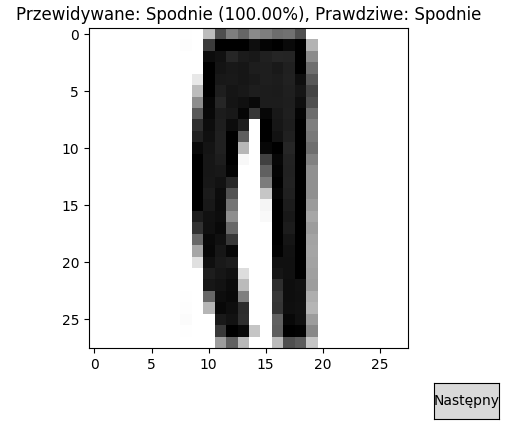
**# Uruchomienie ImageViewer**

viewer = ImageViewer(y\_val\_pred, y\_val, X\_val)

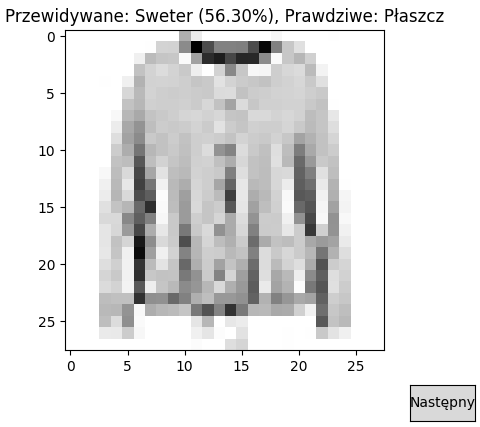
**Wyniki końcowe programu**



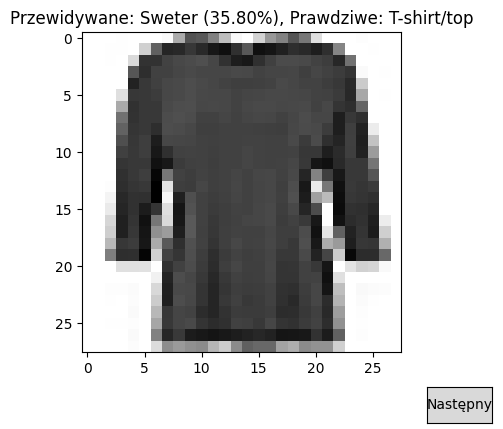
**Zdj.7.** Wynik pozytywny



**Zdj.8.** Wynik pozytywny 2



**Zdj.9.** Wynik negatywny 1



**Zdj.10.** Wynik negatywny

### BIBLIOGRAFIA

„Deep Learning CNN for Fashion MNIST Classification” Jason Brownlee, 2020

„Data augmentation for improving deep learning in image classification problem” Agnieszka Mikołajczyk, Michał Grochowski, 2018

„Transfer Learning in Image Classification” Manali Shaha, Meenakshi Pawar, 2018