

**Uniwersytet Jana Kochanowskiego w Kielcach**  
**Wydział Nauk Ścisłych i Przyrodniczych**  
**Kierunek: Inżynieria Danych**

**Mikołaj Ryba**  
Numer albumu 141249

Praca inżynierska

**Programowanie z wykorzystaniem wielkich modeli  
językowych**

*Pracę w formie drukowanej i elektronicznej  
przyjmuję i dopuszczam do obrony*

.....  
*Data* *podpis*

Promotor:  
Prof. dr hab. Wojciech Broniowski

Kielce 2024



## Streszczenie

Niniejsza praca inżynierska ilustruje wykorzystanie modelu ChatGPT-4 w tworzeniu programów opartych na uczeniu maszynowym. Zbadano i zademonstrowano potencjał ChatGPT-4 w zakresie uczenia nadzorowanego, nienadzorowanego i wzmacnianego. Przeprowadzona analiza ma na celu nie tylko zaprezentowanie możliwości ChatGPT-4, ale również ukazanie, jak te zaawansowane technologie mogą wpływać na rozwój i optymalizację rozwiązań w dziedzinie uczenia maszynowego, a także ułatwić i przyspieszyć pracę programisty.

## Spis treści

1. Wstęp	5
1.1. Cel pracy	5
1.2. Wprowadzenie	5
2. Podstawy teoretyczne	6
2.1. Uczenie maszynowe i sztuczne sieci neuronowe	6
2.2. Model ChatGPT-4	8
3. Metodologia	10
3.1. Narzędzia i technologie użyte w projektach	10
3.2. Opis realizowanych projektów	11
4. Analiza przypadków	15
4.1. Budowa modelu do klasyfikacji kamieni szlachetnych z pomocą ChatGPT-4	15
4.1.1. Konwersacja z ChatGPT-4	15
4.1.2. Ocena skuteczności i dokładności ChatGPT-4	29
4.2. Budowa modelu do regresji cen samochodów z pomocą ChatGPT-4	30
4.2.1. Konwersacja z ChatGPT-4	30
4.2.2. Ocena skuteczności i dokładności ChatGPT-4	40
4.3. Budowa modelu do rekomendacji filmów z pomocą ChatGPT-4	41
4.3.1 Konwersacja z ChatGPT-4	41
4.3.2 Ocena skuteczności i dokładności ChatGPT-4	52
4.4. Trenowanie agenta w środowisku KungFuMaster z pomocą ChatGPT-4	53
4.4.1. Konwersacja z ChatGPT-4	53
4.4.2. Ocena skuteczności i dokładności ChatGPT-4	69
5. Podsumowanie i wnioski	70
6. Bibliografia	72

## Spis Rysunków, Wykresów i Tabel

<i>Rys. 1: Architektura Transformera</i>	5
<i>Rys. 2: Liczba parametrów modeli językowych na przestrzeni lat</i>	7
<i>Rys. 3: Wykresy dokładności i straty zbudowanego modelu konwolucyjnego</i>	16
<i>Rys. 4: Wykresy dokładności i straty modelu VGG16</i>	23
<i>Rys. 5: Raport klasyfikacji modelu VGG16</i>	24
<i>Rys. 6: Macierz pomyłek modelu VGG16</i>	25
<i>Rys. 7: Środowisko KungFuMaster</i>	51
<i>Rys. 8: Wykres historii przeszukiwania najlepszych hiperparametrów dla modelu DQN</i>	62
<i>Rys. 9: Wykres ważności hiperparametrów dla modelu DQN</i>	62
<i>Rys. 10: Średnia nagroda agenta na epizod w trakcie treningu</i>	65
<i>Rys. 11: Średnia długość gry na epizod w treningu agenta</i>	65
<i>Rys. 12: Zmiana współczynnika eksploracji w czasie treningu</i>	65

# 1. Wstęp

## 1.1. Cel pracy

Celem niniejszej pracy inżynierskiej jest zbadanie i ocena wykorzystania modelu ChatGPT-4<sup>[1]</sup> w rozwiązywaniu typowych problemów uczenia maszynowego<sup>[2]</sup>. Praca koncentruje się na analizie efektywności ChatGPT-4 w różnych kontekstach programistycznych, w tym jego roli jako zaawansowanego asystenta w procesie programowania i tworzenia systemów uczenia maszynowego. Dodatkowo, badanie obejmuje identyfikację kluczowych wyzwań i ograniczeń związanych z zastosowaniem ChatGPT-4, podkreślając przy tym przyszłą rolę informatyków w adaptacji i wykorzystaniu nowych technologii w branży IT.

## 1.2. Wprowadzenie

W ostatnich latach sektor sztucznej inteligencji doświadcza spektakularnego postępu, szczególnie w obszarze dużych modeli językowych, takich jak ChatGPT-4. Ten postęp jest nie tylko kwantyfikowalny w zakresie zdolności technologicznych tych modeli, ale także w ich wpływie na różne dziedziny – od edukacji po przemysł. ChatGPT-4, jako jeden z najbardziej zaawansowanych modeli, reprezentuje jak dotąd szczytowe osiągnięcie w dziedzinie przetwarzania języka naturalnego, przekraczając granice poprzednich modeli zarówno pod względem skali, jak i złożoności zadań, które może realizować.

Jego wprowadzenie na rynek oznacza znaczący krok naprzód w zakresie interakcji człowieka z maszyną, oferując możliwości, które jeszcze niedawno wydawały się odległe. Dzięki swojej zdolności do generowania koherentnego i kontekstowego języka, ChatGPT-4 otwiera nowe ścieżki w automatyzacji, edukacji, programowaniu i wielu innych dziedzinach. Wpływ tych technologii na przyszłość pracy, kreatywności i codziennego życia jest ogromny i ciągle ewoluuje.

W kontekście tej dynamicznej ewolucji, niniejsza praca inżynierska ma na celu zgłębienie możliwości, które ChatGPT-4 oferuje w specyficznym kontekście programowania jakim jest uczenie maszynowego. Będzie ona badać zarówno nowe horyzonty, które otwiera ChatGPT-4, jak i wyzwania, które stoją przed tym modelem oraz jego użytkownikami. Praca ta dąży do zrozumienia, w jaki sposób ChatGPT-4 może służyć jako narzędzie wspierające i transformujące pracę programistów, edukatorów i badaczy, a także jakie są potencjalne ograniczenia i przeszkody w jego aplikacji.

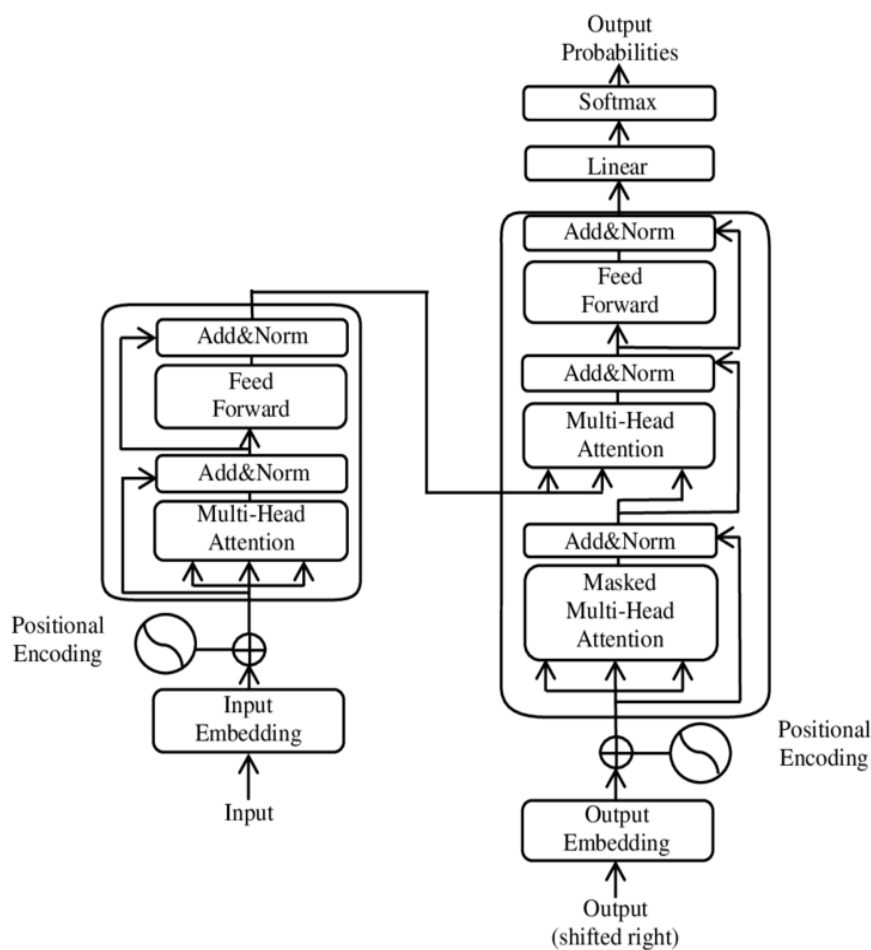
## 2. Podstawy teoretyczne

### 2.1. Uczenie maszynowe i sztuczne sieci neuronowe

Uczenie maszynowe to dział sztucznej inteligencji, który polega na trenowaniu modeli, które umożliwiają komputerom uczenie się i dokonywanie przewidywań lub decyzji na podstawie danych. W odróżnieniu od tradycyjnych programów, które działają na sztywno zdefiniowanych zasadach, systemy uczenia maszynowego dostosowują swoje działanie na podstawie doświadczeń, czyli analizy i przetwarzania danych. Uczenie maszynowe może przyjmować różne formy, takie jak uczenie nadzorowane, gdzie model uczy się na podstawie danych z etykietami, uczenie nienadzorowane, gdzie model znajduje wzorce w danych bez etykiet oraz uczenie ze wzmocnieniem, gdzie model uczy się poprzez interakcję ze środowiskiem i otrzymywanie nagród za dobre decyzje.

W ostatnich latach, dzięki przełomom w sile obliczeniowej, sztuczne sieci neuronowe stały się znaczącym narzędziem w systemach uczenia maszynowego. Sieci te, składające się z wielowarstwowych struktur algorytmicznych, naśladują sposób, w jaki ludzki mózg przetwarza informacje. Każdy sztuczny neuron w sieci działa jako punkt przetwarzania, symulując funkcje biologicznych neuronów poprzez ważenie i przetwarzanie sygnałów wejściowych, a następnie przekazywanie przetworzonej informacji do kolejnych warstw. Sieci neuronowe mogą być trenowane za pomocą różnych algorytmów, co pozwala im adaptować się i poprawiać swoje działanie na podstawie dostarczanych danych. Ich wyjątkowa zdolność do samodzielnego 'uczenia się' z danych, bez konieczności programowania bezpośrednich reguł, sprawia, że są one bardzo skuteczne w wykrywaniu skomplikowanych wzorców i relacji, które mogą być niewidoczne dla tradycyjnych algorytmów.

Sztuczne sieci neuronowe znalazły zastosowanie w architekturze transformatorów<sup>[3]</sup>, stanowiących kolejny przełom w dziedzinie uczenia maszynowego. Transformery pierwszy raz zostały zaprezentowane w pracy naukowej pt. „Attention Is All You Need”<sup>[4]</sup> napisanej przez naukowców Google Brain, w 2017 roku. Jest to rodzaj modelu używanego do przetwarzania języka naturalnego, charakteryzujący się zdolnością do efektywnego analizowania sekwencji danych. Transformery, choć z początku skupiały się na przetwarzaniu języka, szybko znalazły zastosowanie w innych obszarach, takich jak przetwarzanie obrazów i rozpoznawanie mowy. Ta wszechstronność sprawia, że transformery są coraz częściej wykorzystywane także w zadaniach takich jak generowanie muzyki, prognozowanie w finansach i zaawansowane analizy w medycynie.



Rys. 1: Architektura Transformera

Architektura transformera, przedstawiona na rys. 1, specjalizuje się w obsłudze sekwencji danych, wykorzystując kontekst i wiedzę nabytą z poprzednich doświadczeń. Klucowym elementem tej architektury jest innowacyjny mechanizm uwagi, który pozwala modelowi koncentrować się na różnych częściach sekwencji danych, co zwiększa dokładność i skuteczność przetwarzania informacji. Dzięki temu, modele oparte na transformatorach mogą efektywnie analizować i interpretować duże zbiory danych, wykorzystując zdobytą wiedzę do rozpoznawania wzorców, relacji i znaczeń.

Zastosowanie architektury transformera było kluczowe w rozwoju dzisiejszych zaawansowanych modeli językowych<sup>[5]</sup>, takich jak ChatGPT-4. Transformery, dzięki swojej zdolności do efektywnego przetwarzania sekwencji językowych, znacznie przyczyniły się do poprawy dokładności i złożoności przetwarzania języka naturalnego. Oprócz aplikacji w języku naturalnym, te modele znajdują zastosowanie również w innych dziedzinach, takich jak rozpoznawanie wzorców czy automatyczne wnioskowanie, demonstrując ich wszechstronność.



## 2.2. Model ChatGPT-4

Modele językowe, takie jak ChatGPT-4, są zaawansowanymi systemami uczenia maszynowego specjalizującymi się w przetwarzaniu i generowaniu języka naturalnego. ChatGPT-4, będący ewolucją poprzednich modeli GPT, opiera się na architekturze Transformer, która wykorzystuje mechanizm uwagi do lepszego modelowania zależności (tj. korelacji) w danych tekstowych. Kluczowym elementem ChatGPT-4 jest jego zdolność do generowania koherentnych i kontekstualnych tekstów, co znajduje zastosowanie w wielu obszarach, od automatycznego generowania treści po interaktywnych asystentów.

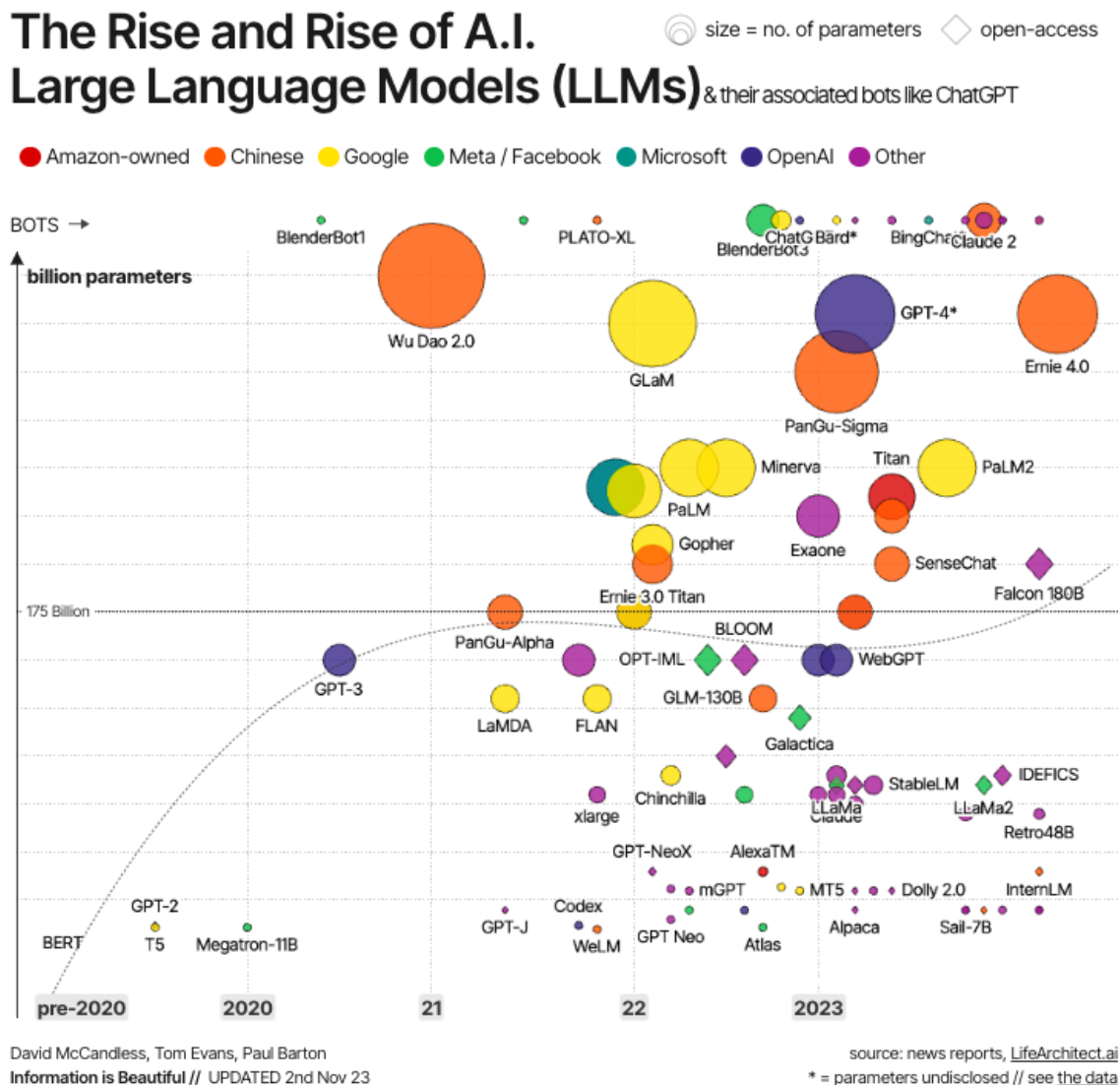
ChatGPT, będąc aplikacją GPT-4, wychodzi poza standardowe zastosowania modeli językowych, umożliwiając nie tylko przetwarzanie języka naturalnego, ale również interpretację i reagowanie na kod programistyczny. To otwiera nowe możliwości w dziedzinie automatyzacji zadań programistycznych i edukacji w zakresie programowania. Ponadto, wielomodalność GPT-4, daje mu zdolność do przetwarzania różnych typów danych wejściowych – od tekstu, przez grafikę, aż po mowę – znacznie rozszerza zakres jego zastosowań.

Niezbędne elementy dla efektywnego trenowania ChatGPT-4 to:

- Olbrzymia moc obliczeniowa – trenowanie modelu tej skali wymaga ogromnych zasobów i może trwać wiele dni, generując przy tym znaczne koszty.
- Wielki korpus danych – ChatGPT-4 jest trenowany na gigantycznym zbiorze danych tekstowych, który może osiągać rozmiary wielu terabajtów.
- Odpowiednia architektura sieci, umożliwiająca wydajne kodowanie kontekstu i zależności w danych.

Efektem tych działań jest zaawansowana forma Sztucznej Inteligencji, zdolna do realizacji zadań o złożoności dotąd nieosiągalnej dla wcześniejszych modeli. Ilustracją tego wzrostu jest rys. 2, przedstawiający ewolucję wielkości modeli językowych na przestrzeni lat. W szczególności, widać jak znaczący postęp nastąpił między GPT-2, GPT-3.5 a GPT-4.

# The Rise and Rise of A.I. Large Language Models (LLMs) & their associated bots like ChatGPT



Rys. 2: Liczba parametrów modeli językowych na przestrzeni lat<sup>[6]</sup>

Należy wspomnieć że, modele te takie jak ChatGPT-4 posiadają również ograniczenia, takie jak tendencja do generowania stronniczych lub nieprecyzyjnych informacji, co wynika częściowo z charakteru danych, na których zostały wytrenowane. Istnieją również wyzwania związane z interpretowalnością ich decyzji i odpowiedzi, co jest krytyczne zwłaszcza w zastosowaniach wymagających wysokiego stopnia odpowiedzialności i przejrzystości, jak np. w medycynie czy prawie. Zrozumienie, jak model dochodzi do konkretnych wniosków, pozostaje jednym z kluczowych wyzwań w dziedzinie AI, mającym znaczący wpływ na dalszy rozwój i akceptację tych technologii w społeczeństwie.

## 3. Metodologia

### 3.1. Narzędzia i technologie użyte w projektach

W projekcie wykorzystano zaawansowane narzędzia i technologie, aby maksymalnie wykorzystać możliwości modelu ChatGPT-4:

- **ChatGPT-4:** Używany jako pomocnicze narzędzie do trenowania modeli, oferujący interaktywne wsparcie, porady i kod.
- **Środowisko programistyczne:** Jupyter Notebook<sup>[7]</sup>, zapewniający elastyczne i interaktywne środowisko do eksperymentowania z kodem i wizualizacji wyników.
- **Język programowania:** Python<sup>[8]</sup>, ze względu na jego wszechstronność i popularność w dziedzinie uczenia maszynowego i analizy danych.
- **Karta graficzna:** Nvidia Geforce RTX 2060<sup>[9]</sup> do przyspieszenia lokalnych obliczeń, co jest kluczowe w intensywnych obliczeniowo zadaniach, takich jak uczenie maszynowe.

## 3.2. Opis realizowanych projektów

### A. Budowa modelu do klasyfikacji kamieni szlachetnych z pomocą ChatGPT-4:

#### 1. Cel projektu:

Celem tego projektu jest ocena zdolności ChatGPT-4 do rozwiązywania problemów klasyfikacyjnych. Analizowana jest jakość skryptów wygenerowanych przez model, oceniając ich efektywność i adekwatność w rozwiązywaniu postawionych problemów. Dodatkowo, badane jest, w jaki sposób narzędzie udziela trafnych porad i wskazówek.

#### 2. Metodyka:

- **Zbieranie i Przygotowanie Danych:** Gromadzenie danych, w tym wypadku zdjęć kamieni szlachetnych<sup>[10]</sup>.
- **Generowanie Kodu:** Wykorzystanie ChatGPT-4 do pomocy w generowaniu skryptów.
- **Implementacja i Trenowanie Modelu:** Budowa modelu klasyfikacyjnego na podstawie wygenerowanych skryptów i trenowanie go na zebranych danych.
- **Ocena i Optymalizacja Modelu:** Testowanie modelu, analiza wyników i dokonywanie potrzebnych modyfikacji przez ChatGPT-4 w celu poprawy dokładności i skuteczności.
- **Analiza i Interpretacja Wyników:** Używanie ChatGPT-4 do interpretacji wyników i wyciągania wniosków.

#### 3. Wyniki:

- **Ocena Dokładności Stworzonego Modelu:** Prezentacja wyników testów modelu, w tym dokładności, precyzji i innych kluczowych metryk.
- **Ocena Skryptów Generowanych przez ChatGPT-4:** Analiza skuteczności i efektywności kodu wygenerowanego przez ChatGPT-4. Ocena, jak kod ten wpłynął na wydajność modelu, oraz jakie były jego mocne i słabe strony w kontekście specyfikacji projektu.
- **Wnioski z Analizy ChatGPT-4:** Omówienie, jak sugestie i analizy przeprowadzone przez ChatGPT-4 wpłynęły na proces budowy i optymalizacji modelu.

## B. Budowa modelu do regresji cen samochodów z pomocą ChatGPT-4:

### 1. Cel projektu:

Celem projektu jest stworzenie modelu regresji do przewidywania cen samochodów przy pomocy ChatGPT-4 gdzie asystent jest celem badań. Model językowy wykorzystywany jest jako narzędzie tworzące model we własnym środowisku, wspomagające proces, oferujące pomoc w generowaniu kodu, analizie danych, i interpretacji wyników.

### 2. Metodyka:

- **Zbieranie i Przygotowanie Danych:** Gromadzenie danych o samochodach<sup>[11]</sup>, obejmujących cenę, przebieg, rok produkcji, i inne istotne cechy.
- **Przekazanie danych:** Przekazanie danych bezpośrednio do ChatGPT-4 w postaci załącznika.
- **Generowanie Kodu:** Wykorzystanie ChatGPT-4 do tworzenia skryptów które wykonuje do analizy danych i budowy modelu.
- **Implementacja Modelu:** Stosowanie wygenerowanych skryptów do budowy modelu regresji.
- **Ocena i Optymalizacja Modelu:** Testowanie modelu, analiza wyników i dokonywanie potrzebnych modyfikacji przez ChatGPT-4 w celu poprawy dokładności i skuteczności.
- **Analiza i Interpretacja Wyników z ChatGPT-4:** Wykorzystanie ChatGPT-4 do analizy wyników modelu i wyciągania wniosków.

### 3. Wyniki:

- **Ocena Dokładności Modelu Regresji:** Prezentacja dokładności modelu, błędu średniokwadratowego (MSE), i współczynnika determinacji ( $R^2$ ).
- **Ocena Skryptów Generowanych przez ChatGPT-4:** Analiza skuteczności kodu wygenerowanego przy pomocy ChatGPT-4.
- **Wnioski z Analizy ChatGPT-4:** Omówienie, w jaki sposób wsparcie ChatGPT-4 przyczyniło się do optymalizacji modelu i interpretacji wyników.

## C. Budowa modelu do rekomendacji filmów z pomocą ChatGPT-4:

### 1. Cel projektu:

Celem projektu jest stworzenie modelu uczenia nienadzorowanego do rekomendacji filmów, korzystając z danych filmowych Netflix<sup>[12]</sup>. ChatGPT-4 służy jako wsparcie w procesie analizy danych, przetwarzania tekstu i wyboru odpowiednich metod modelowania.

### 2. Metodyka:

- **Zbieranie i Przygotowanie Danych:** Gromadzenie danych filmowych filmów Netflix, zawierających tytuł, reżysera, obsadę, kraj produkcji, opis itd.
- **Przekazanie danych:** Przekazanie danych filmowych do ChatGPT-4 w celu analizy i przetwarzania.
- **Generowanie Kodu:** Wykorzystanie ChatGPT-4 do tworzenia skryptów do analizy tekstów i danych filmowych oraz do proponowania metod budowy systemu rekomendacji.
- **Implementacja Modelu:** Stosowanie wygenerowanych skryptów i metod do budowy modelu rekomendacji filmów.
- **Ocena i Optymalizacja Modelu:** Analiza wyników rekomendacji i dokonywanie modyfikacji w modelu z pomocą ChatGPT-4, aby poprawić trafność i jakość rekomendacji.
- **Analiza i Interpretacja Wyników z ChatGPT-4:** Wykorzystanie ChatGPT-4 do analizy wyników systemu rekomendacji i wyciągania wniosków.

### 3. Wyniki:

- **Ocena Jakości Rekomendacji:** Analiza skuteczności rekomendacji na podstawie wybranych filmów, np. „Terminator 3: Rise of the Machines”, i ocena trafności sugerowanych tytułów.
- **Ocena Skryptów Generowanych przez ChatGPT-4:** Analiza kodu wygenerowanego przy pomocy ChatGPT-4.
- **Wnioski z Analizy ChatGPT-4:** Szczegółowa ocena, w jaki sposób wskazówki i analizy dostarczone przez ChatGPT-4 przyczyniły się do rozwoju i doskonalenia modelu.

## D. Trenowanie agenta w środowisku KungFuMaster z pomocą ChatGPT-4:

### 1. Cel projektu:

Celem projektu jest stworzenie i wytrenowanie agenta AI w środowisku OpenAI Gym o nazwie KungFuMaster<sup>[13]</sup>. ChatGPT-4 służy jako narzędzie wspomagające w zrozumieniu środowiska, wyborze odpowiednich algorytmów oraz w dostosowaniu i optymalizacji procesu uczenia.

### 2. Metodyka:

- **Zainstalowanie i Konfiguracja Środowiska:** Wykorzystanie ChatGPT-4 do zrozumienia procesu instalacji OpenAI Gym i środowiska KungFuMaster. Inicjalizacja środowiska i przygotowanie do interakcji z agentem.
- **Wsparcie ChatGPT-4 w Rozpoczęciu Procesu Uczenia:** Użycie ChatGPT-4 do wyboru odpowiednich algorytmów uczenia ze wzmocnieniem. Implementacja prostego algorytmu uczenia ze wzmocnieniem z wykorzystaniem wskazówek ChatGPT-4.
- **Optymalizacja i Dostosowanie Hiperparametrów:** Przeprowadzenie procesu przeszukiwania hiperparametrów z poradami ChatGPT-4.
- **Trening i Ocena Agentu przy Wsparciu ChatGPT-4:** Długotrwałe trenowanie agenta przy użyciu zoptymalizowanych hiperparametrów, z równoczesnym wykorzystaniem ChatGPT-4 do analizy postępów, interpretacji wyników i identyfikacji obszarów wymagających poprawy.

### 3. Wyniki:

- **Analiza wyników Treningu Agentu:** Ocena wydajności agenta na podstawie ich zgodności do radzenia sobie z wyzwaniami w grze KungFuMaster.
- **Ocena Wkładu ChatGPT-4:** Szczegółowa analiza, w jaki sposób wskazówki i analizy przeprowadzone przez ChatGPT-4 przyczyniły się do rozwoju i doskonalenia agenta. Ocena skuteczności pomocy modelu językowego w wyborze algorytmów, optymalizacji hiperparametrów i interpretacji wyników treningu.

## 4. Analiza przypadków

W niniejszym rozdziale przedstawione są kolejno poszczególne przykłady. Zacytowana jest konwersacja z ChatGPT-4 (czcionka wytłuszczona – zapytania, kursywa – odpowiedzi programu). Wygenerowane przez model językowy kody w Pythonie są zawarte w ramkach. Komentarze i uwagi zawarte są na końcu poszczególnych podrozdziałów. Podczas lektury należy zwrócić uwagę na bardzo inteligentne i pomocne zachowania ChatGPT-4. Chociaż rozwiązywane problemy są typowe dla uczenia maszynowego, ich sposób i trafność rozwiązania robią wrażenie.

### 4.1. Budowa modelu do klasyfikacji kamieni szlachetnych z pomocą ChatGPT-4

#### 4.1.1. Konwersacja z ChatGPT-4



Ja:

**Cześć, chciałbym, abyś wytrenował model uczenia maszynowego w Pythonie do przewidywania kategorii kamieni szlachetnych. Mój zbiór danych zawiera ponad 3 200 obrazów kamieni szlachetnych. Obrazy są podzielone na 87 klas, które zostały już przetworzone na dane treningowe i testowe. Obrazy mają różne rozmiary i są w formacie .jpeg.**

**Ten zbiór danych składa się z dwóch folderów:**

**train:** Ten folder zawiera 87 podfolderów i łącznie około 2 800 plików. Każdy podfolder zawiera obrazy .jpeg różnych kamieni szlachetnych.

**test:** Ten folder zawiera 87 podfolderów i łącznie około 400 plików. Każdy podfolder zawiera obrazy .jpeg różnych kamieni szlachetnych.

**Ścieżka pliku do moich danych treningowych: "C:\Users\mikoł\Desktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja\data\train"**

**Ścieżka pliku do moich danych testowych: "C:\Users\mikoł\Desktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja\data\test"**



## ChatGPT:

*Dzięki za informacje. Aby zbudować model klasyfikacji obrazów kamieni szlachetnych, wykorzystamy bibliotekę TensorFlow wraz z Keras. Poniżej znajduje się krok po kroku, jak to zrobić:*

### 1. **Wczytywanie i Przetwarzanie Danych**

*Będziemy korzystać z `ImageDataGenerator` do wczytywania obrazów, a także do zastosowania augmentacji danych<sup>[14]</sup> (wzmacnianie danych poprzez modyfikację oryginalnych obrazów).*

### 2. **Budowanie Modelu**

*Do klasyfikacji obrazów użyjemy sieci konwolucyjnej<sup>[15]</sup> (Convolutional Neural Network, CNN).*

### 3. **Trening Modelu**

*Po zbudowaniu modelu przeprowadzimy proces treningu.*

### 4. **Ewaluacja Modelu**

*Oceniemy model na danych testowych.*

*Oto kod:*

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

# Stałe
IMAGE_SIZE = 150
BATCH_SIZE = 32
EPOCHS = 10

# 1. Wczytywanie i Przetwarzanie Danych
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255) # Tylko skalowanie dla
danych testowych

train_generator = train_datagen.flow_from_directory(
    "C:\Users\mikor\desktop\Pulpit\Praca Inżynierska\Uczenie
Maszynowe\Klasyfikacja\data\train",
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
```

```

"C:\Users\mikoł\Desktop\Pulpit\Praca Inżynierska\Uczenie
Maszynowe\Klasyfikacja\data\test",
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical')

# 2. Budowanie Modelu
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE,
3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(87, activation='softmax')
])

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

# 3. Trening Modelu
history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=test_generator)

# 4. Ewaluacja Modelu
loss, accuracy = model.evaluate(test_generator)
print(f"Strata testu: {loss:.4f}")
print(f"Dokładność testu: {accuracy:.4f}")

```

*Kilka uwag:*

- Aby dostosować ten kod do Twojego przypadku, możesz zmienić wartości, takie jak **`IMAGE_SIZE`**, **`BATCH_SIZE`** czy **`EPOCHS`** w zależności od Twoich potrzeb.
- Jeśli chodzi o architekturę sieci, możesz ją dostosować w zależności od wyników, jakie uzyskasz.
- Możesz również dodać checkpointy, zapisywanie modelu itp., aby ulepszyć proces treningu.



Ja:

Czy mogę jakoś wizualnie wyświetlić wyniki trenowania?



ChatGPT:

Oczywiście, możesz wykorzystać bibliotekę Matplotlib do wizualizacji procesu uczenia się modelu. Wizualizacja pomaga zrozumieć, jak model działał w trakcie treningu, a także może wskazać ewentualne problemy, takie jak przetrenowanie (overfitting).

Poniżej znajduje się kod, który pozwala wyświetlić wykresy dokładności oraz funkcji straty dla danych treningowych i walidacyjnych:

```
import matplotlib.pyplot as plt

# Wizualizacja historii treningu
def plot_training_history(history):
    # Wykres dokładności
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Dokładność treningowa')
    plt.plot(history.history['val_accuracy'], label='Dokładność walidacyjna')
    plt.legend(loc='lower right')
    plt.title('Dokładność treningu i walidacji')

    # Wykres funkcji straty
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Strata treningowa')
    plt.plot(history.history['val_loss'], label='Strata walidacyjna')
    plt.legend(loc='upper right')
    plt.title('Strata treningu i walidacji')

    plt.show()

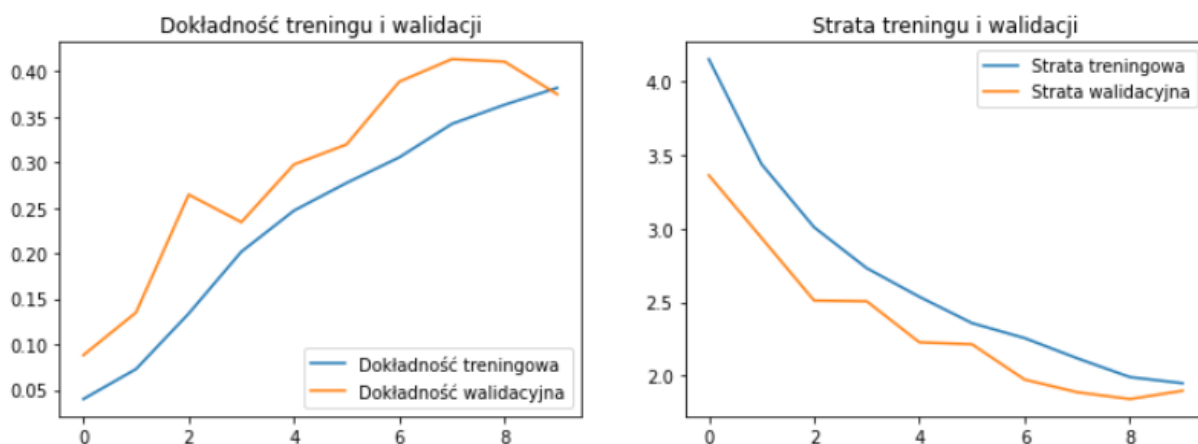
# Wywołanie funkcji
plot_training_history(history)
```

Po treningu modelu możesz wywołać funkcję `plot_training_history`, przekazując jej historię treningu (zmienną `history`), aby wyświetlić wykresy.

Jeśli zauważysz, że dokładność na zbiorze treningowym jest znacznie wyższa niż na zbiorze walidacyjnym, a funkcja straty na zbiorze treningowym jest znacznie niższa niż na zbiorze walidacyjnym, może to wskazywać na problem przetrenowania modelu. W takim przypadku można rozważyć zastosowanie technik regularyzacji, takich jak dropout (który już dodaliśmy do modelu), czy też wcześniejsze zakończenie treningu (early stopping).

Ja:

Czy te wyniki są satysfakcjonujące?



Rys. 3: Wykresy dokładności i straty zbudowanego modelu konwolucyjnego

Epoch 1/10 90/90 [=====] - 16s 145ms/step - loss: 4.1506 - accuracy: 0.0399 - val\_loss: 3.3618 - val\_accuracy: 0.0882 Epoch 2/10 90/90 [=====] - 13s 141ms/step - loss: 3.4382 - accuracy: 0.0728 - val\_loss: 2.9372 - val\_accuracy: 0.1350 Epoch 3/10 90/90 [=====] - 13s 140ms/step - loss: 3.0089 - accuracy: 0.1338 - val\_loss: 2.5113 - val\_accuracy: 0.2645 Epoch 4/10 90/90 [=====] - 13s 141ms/step - loss: 2.7310 - accuracy: 0.2017 - val\_loss: 2.5062 - val\_accuracy: 0.2342 Epoch 5/10 90/90 [=====] - 13s 140ms/step - loss: 2.5366 - accuracy: 0.2468 - val\_loss: 2.2277 - val\_accuracy: 0.2975 Epoch 6/10 90/90 [=====] - 13s 141ms/step - loss: 2.3579 - accuracy: 0.2773 - val\_loss: 2.2142 - val\_accuracy: 0.3196 Epoch 7/10 90/90 [=====] - 13s 141ms/step - loss: 2.2561 - accuracy: 0.3053 - val\_loss: 1.9744 - val\_accuracy: 0.3884 Epoch 8/10 90/90 [=====] - 13s 141ms/step - loss: 2.1189 - accuracy: 0.3421 - val\_loss: 1.8886 - val\_accuracy: 0.4132 Epoch 9/10 90/90 [=====] - 13s 141ms/step - loss: 1.9913 - accuracy: 0.3631 - val\_loss: 1.8424 - val\_accuracy: 0.4105 Epoch 10/10 90/90 [=====] - 13s 140ms/step - loss: 1.9497 - accuracy: 0.3817 - val\_loss: 1.8988 - val\_accuracy: 0.3747 12/12 [=====] - 0s 36ms/step - loss: 1.8988 - accuracy: 0.3747

Strata testu: 1.8988

Dokładność testu: 0.3747



*Analizując wykresy oraz podane statystyki:*

1. **Dokładność (Accuracy):** Model osiągnął dokładność na poziomie około 38% na zbiorze walidacyjnym po 10 epokach. To jest stosunkowo niska wartość, zwłaszcza biorąc pod uwagę, że klasyfikujesz obrazy do 87 różnych klas.
2. **Strata (Loss):** Strata na zbiorze treningowym maleje w miarę postępu treningu, co jest pożądaną tendencją. Jednakże strata na zbiorze walidacyjnym, choć generalnie maleje, zaczyna się nieznacznie zwiększać w ostatnich epokach, co może wskazywać na początek przetrenowania.
3. **Wykresy:** Z wykresów można zauważyć, że dokładność walidacyjna jest generalnie niższa niż dokładność treningowa, co jest typowe, ale różnica nie jest drastyczna. Wskaźniki straty wydają się być bardziej stabilne.

*Podsumowując, wyniki nie są jeszcze satysfakcjonujące. Istnieje kilka kroków, które można podjąć w celu poprawy wyników:*

1. **Zwiększenie liczby epok:** Model może potrzebować więcej czasu na naukę, zwłaszcza przy takiej liczbie klas. Możesz spróbować trenować przez więcej epok, obserwując, czy dokładność walidacyjna się poprawia.
2. **Zmiana architektury modelu:** Możesz eksperymentować z liczbą i rozmiarem warstw w modelu.
3. **Data Augmentation:** Zastosowanie technik augmentacji danych może pomóc modelowi w generalizacji i zapobieganiu przetrenowaniu.
4. **Regularyzacja:** Oprócz dropoutu, możesz rozważyć dodanie regularyzacji L1 lub L2 do modelu.
5. **Zastosowanie wczesnego zatrzymywania (Early Stopping):** Możesz zastosować wczesne zatrzymywanie, aby automatycznie przerwać trening, gdy dokładność walidacyjna przestaje się poprawiać.
6. **Transfer Learning<sup>[13]</sup>:** Wykorzystaj wytrenowane modele, takie jak VGG16 lub ResNet, do ekstrakcji cech, a następnie dodaj kilka warstw własnych, które zostaną wytrenowane na Twoim zbiorze danych.

*Ostatecznie, trenowanie modeli głębokiego uczenia, zwłaszcza dla dużych zbiorów danych i wielu klas, jest bardziej sztuką niż nauką. Wymaga eksperymentowania i cierpliwości.*



Ja:

Dokonaj zmian w podanym przez ciebie kodzie które twoim zdaniem najlepiej wpłynęłyby na wyniki.



ChatGPT:

Okej, skupmy się na kilku kluczowych aspektach, które mogą poprawić wyniki:

1. **Transfer Learning:** Wykorzystamy model VGG16 jako bazę, co pozwoli nam na skorzystanie z wstępnie wytrenowanych wag dla ekstrakcji cech. Następnie dodamy kilka własnych warstw do klasyfikacji.
2. **Data Augmentation:** Aby zwiększyć różnorodność danych treningowych i zapobiec przetrenowaniu.
3. **Early Stopping:** Automatyczne zatrzymywanie treningu, gdy wyniki walidacji zaczynają się pogarszać.

Oto zmodyfikowany kod:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping

# Wczytanie modelu VGG16 bez górnych warstw
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(150,
150, 3))
# Zamrożenie wag w modelu bazowym
for layer in base_model.layers:
    layer.trainable = False

# Dodanie własnych warstw
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dense(87, activation='softmax')(x)

model = Model(base_model.input, x)

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

# Augmentacja danych
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

```

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'C:\\Users\\mikoł\\Desktop\\Pulpit\\Praca Inżynierska\\Uczenie
Maszynowe\\Klasyfikacja\\data\\train`,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    'C:\\Users\\mikoł\\Desktop\\Pulpit\\Praca Inżynierska\\Uczenie
Maszynowe\\Klasyfikacja\\data\\test`,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

# Wczesne zatrzymywanie
early_stop = EarlyStopping(monitor='val_loss`, patience=3,
restore_best_weights=True)

history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stop])

```

*W tym kodzie wykorzystujemy model VGG16 do ekstrakcji cech i dodajemy nasze własne warstwy do klasyfikacji. Używamy augmentacji danych, aby zwiększyć różnorodność danych treningowych. Wczesne zatrzymanie (early stopping) pozwoli nam uniknąć przetrenowania poprzez zatrzymanie procesu uczenia, gdy wyniki walidacji zaczynają się pogarszać.*

*Mam nadzieję, że te zmiany pomogą poprawić wyniki Twojego modelu!*

 Ja:

Korzystając z twojego kodu otrzymałem taki błąd:

```

-----
AttributeError Traceback (most recent call last)
<ipython-input-19-fb8dc23c8a7c> in <module>
2 base_model.trainable = False
3
----> 4 model.compile(loss='categorical_crossentropy`,
optimizer=optimizers.Adam(lr=1e-4), metrics=['accuracy`])
5
6 # Wczesne zatrzymanie
AttributeError: module `keras.optimizers` has no attribute `Adam`

```

 ChatGPT:

Wygląda na to, że masz problem z importem optymalizatora `Adam`. W zależności od tego, czy korzystasz z `tensorflow.keras` czy z samego `keras`, import może się nieco różnić.

Jeżeli korzystasz z `tensorflow.keras`, użyj następującego importu:

```
from tensorflow.keras import optimizers
```

Jeżeli korzystasz z `keras`:

```
from keras import optimizers
```

Jednakże, w poprzednim kodzie, którego dostarczyłem, używałem po prostu `optimizer='adam'`, co powinno działać bezpośrednio.

Dlatego proponuję zamienić:

```
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(lr=1e-4), metrics=['accuracy'])
```

na:

```
from tensorflow.keras.optimizers import Adam

model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=1e-4), metrics=['accuracy'])
```

Upewnij się, że posiadasz odpowiednie importy na początku pliku i że korzystasz z odpowiedniej biblioteki (`tensorflow.keras` lub `keras`). Jeśli nadal występują problemy, daj znać!

 Ja:

Kod działa już poprawnie. Jakie znasz techniki do testowania wydajności modelu? Podaj mi kod dla jednej z nich.

 ChatGPT:

Testowanie wydajności modelu klasyfikacji obrazów można przeprowadzić na wiele sposobów, zależnie od tego, które aspekty wydajności są najbardziej interesujące. Oto kilka technik:

1. **Confusion Matrix**<sup>[14]</sup>: Macierz błędów pozwala zrozumieć, które klasy są mylone przez model.
2. **Classification Report**: Zawiera podsumowanie miar wydajności, takich jak precyzja, pełność (recall), f1-score dla każdej klasy.
3. **ROC Curves i AUC**: Są używane głównie w klasyfikacji binarnej do oceny wydajności modelu.
4. **Precision-Recall Curves**: Podobnie jak ROC, ale często używane w przypadkach, gdy klasy są niezbalansowane.



Oto przykład jak użyć **Classification Report** i **Confusion Matrix**:

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Przewidzenia dla danych testowych
test_generator.reset() # Resetowanie generatora przed przewidywaniami
predictions = model.predict(test_generator, steps=test_generator.n //
test_generator.batch_size+1)
predicted_classes = np.argmax(predictions, axis=1)

# Prawdziwe etykiety
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification Report
report = classification_report(true_classes, predicted_classes,
target_names=class_labels)
print(report)

# Confusion Matrix
cm = confusion_matrix(true_classes, predicted_classes)
print(cm)
```

Pamiętaj, że aby użyć powyższego kodu, liczba kroków w `model.predict` powinna odpowiadać liczbie obrazów w zbiorze testowym podzielonej przez rozmiar partii (`batch_size`), z tego powodu używamy `test_generator.n // test_generator.batch_size+1` by upewnić się, że wszystkie obrazy zostaną uwzględnione.

Dodatkowo, jeśli liczba próbek nie jest wielokrotnością `batch_size`, ostatni batch może być mniejszy lub może zawierać próbki, które były już użyte. Aby tego uniknąć, możesz ustawić `shuffle=False` w `test_generator` i dokładnie znać liczbę próbek, co pozwoli na właściwe ustawienie parametru `steps` w funkcji `predict`.

Aby lepiej zrozumieć macierz błędów, często wyświetla się ją w formie graficznej za pomocą biblioteki `matplotlib` lub `seaborn`:

```
import seaborn as sns
import matplotlib.pyplot as plt

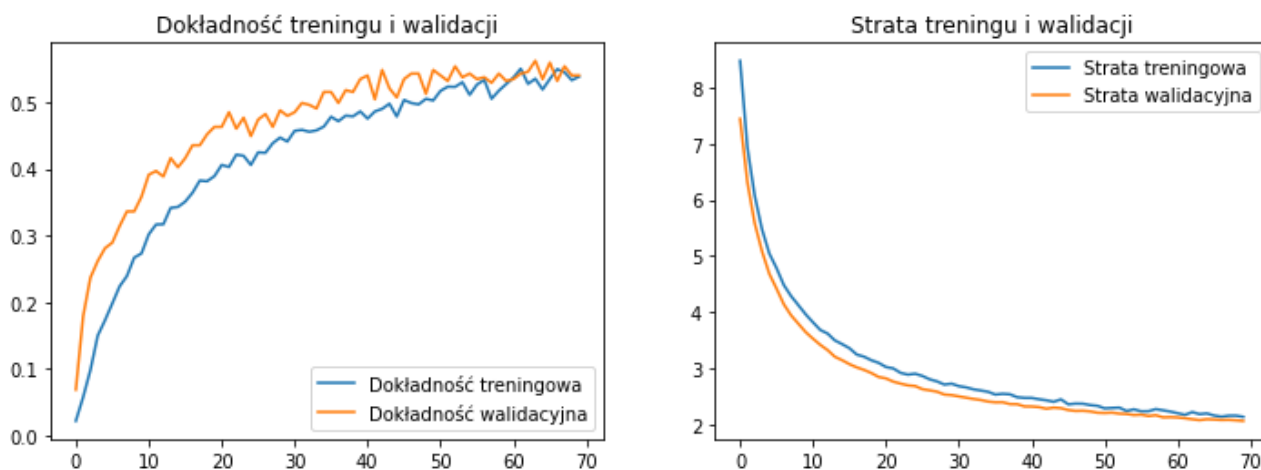
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap =
'Blues_r', xticklabels=class_labels, yticklabels=class_labels)
plt.ylabel('Rzeczywista etykieta')
plt.xlabel('Przewidywana etykieta')
title = 'Dokładność: {0}'.format(accuracy_score(true_classes,
predicted_classes))
plt.title(title, size = 12)
plt.show()
```

Wykorzystanie tych technik pomoże Ci dokładnie ocenić, gdzie model działa dobrze, a gdzie są błędy, co może dostarczyć wskazówek, jak poprawić model.

Ja:

Dziękuję za pomoc w kwestii kodu. Na zakończenie naszej rozmowy chciałbym poprosić Cię o ocenę wykresów oraz wyników, które dotyczą dokładności mojego finalnego modelu.

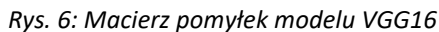
Załączam wykresy dokładności i straty modelu z okresu treningu, raport klasyfikacji oraz macierz pomyłek.



Rys. 4: Wykresy dokładności i straty modelu VGG16

	precision	recall	f1-score	support
Alexandrite	0.33	1.00	0.50	4
Almandine	0.14	0.25	0.18	4
Amazonite	1.00	0.50	0.67	4
Amber	0.43	0.75	0.55	4
Amethyst	0.14	0.25	0.18	4
...	...	...	...	...
Tsavorite	0.75	0.75	0.75	4
Turquoise	1.00	1.00	1.00	4
Variscite	1.00	0.25	0.40	4
Zircon	0.33	0.25	0.29	4
Zoisite	1.00	1.00	1.00	4
accuracy			0.54	363
macro avg	0.59	0.54	0.53	363
weighted avg	0.59	0.54	0.53	363

Rys. 5: Raport klasyfikacji modelu VGG16



Analizując przedstawione wyniki oraz wykresy, oto kilka kluczowych obserwacji:

### 1. Wykresy dokładności i straty:

- **Dokładność:** Wykres pokazuje wzrost dokładności zarówno treningowej, jak i walidacyjnej, co jest pozytywnym znakiem. Jednakże, istnieje przestrzeń na poprawę, ponieważ dokładność walidacyjna wydaje się być stabilna poniżej 60%. To sugeruje, że model może nie być w pełni zdolny do generalizacji na nowych danych.
- **Strata:** Strata dla zestawu treningowego i walidacyjnego maleje w miarę treningu, co jest dobre. Jednakże, widać, że strata walidacyjna przestaje spadać i zaczyna się stabilizować, co może wskazywać na to, że dalsze treningi nie przynoszą znaczącej poprawy.

### 2. Macierz pomyłek:

- Macierz pomyłek jest bardzo gęsta i trudna do interpretacji ze względu na dużą liczbę klas. Dla bardziej szczegółowej analizy warto by było przyjrzeć się szczególnym klasom, które są źle klasyfikowane, i zastanowić się nad strategiami ich poprawy.
- Niektóre klasy są dobrze klasyfikowane (diagonale macierzy), ale występuje wiele pomyłek między różnymi klasami.

### 3. Raport klasyfikacji:

- Istnieją znaczne różnice w miarach precyzji, pełności i wyniku F1 między klasami. Niektóre klasy mają wysoką precyzję i pełność (np. "Larimar", "Tsavorite"), podczas gdy inne mają bardzo niskie wartości (np. "Beryl Golden", "Garnet Red").
- Średnia wartość F1-score (macro average) wynosi 0.53, co jest poniżej oczekiwań w kontekście zrównoważonego zbioru danych. Niska średnia F1 wskazuje na to, że model ma trudności z klasyfikacją wielu klas.

Podsumowując, chociaż model wykazuje pewien potencjał, wyniki sugerują, że jest jeszcze miejsce na poprawę. Możliwe kroki, które możesz podjąć, to:

- **Przeanalizować błędnie klasyfikowane przypadki:** Zrozumienie, dlaczego niektóre klasy są mylone, może pomóc w poprawie przetwarzania danych lub cech ekstrakcyjnych.
- **Zbalansowanie zbioru danych:** Jeśli niektóre klasy są nadreprezentowane lub niedostatecznie reprezentowane, może to wpłynąć na wyniki.
- **Tuning hiperparametrów:** Dostosowanie stopy uczenia się, rozmiaru batcha lub innych hiperparametrów może przynieść poprawę.
- **Zwiększenie złożoności modelu:** Jeśli obecny model jest zbyt prosty, może nie być w stanie uchwycić złożoności danych.
- **Wykorzystanie zaawansowanych technik:** Takich jak ensembling modeli, bardziej skomplikowane architektury sieci neuronowych lub specjalistyczne techniki obróbki obrazów dla kamieni szlachetnych.

Ostateczny cel to nie tylko zwiększenie dokładności, ale także zapewnienie, że model jest w stanie generalizować i poprawnie klasyfikować obrazy w różnorodnych i niewidocznych wcześniej scenariuszach.

#### 4.1.2. Ocena skuteczności i dokładności ChatGPT-4

##### 1. Jakość Skryptów Generowanych przez ChatGPT-4:

- ChatGPT-4 skutecznie służył jako asystent w generowaniu kodu, dostarczając kompleksowe skrypty wykorzystujące TensorFlow i Keras do budowy modelu klasyfikacyjnego.
- Model zaproponowany przez ChatGPT-4 obejmował kluczowe aspekty, takie jak augmentacja danych, architektura CNN, oraz techniki przeciwdziałania przetrenowaniu (np. dropout, early stopping).
- W odpowiedzi na problematyczne wyniki, ChatGPT-4 zasugerował modyfikacje, w tym wykorzystanie Transfer Learning (VGG16), co wskazuje na zdolność adaptacji i ulepszania propozycji w oparciu o feedback.

##### 2. Efektywność i Adekwatność w Rozwiązywaniu Problemów Klasyfikacyjnych:

- ChatGPT-4 wykazał się umiejętnością analizy i interpretacji wyników, identyfikując kluczowe obszary do poprawy, takie jak niska dokładność i potencjalne przetrenowanie.
- Sugestie dotyczące zwiększenia liczby epok, eksperymentowania z architekturą modelu, oraz zastosowania różnych technik regularyzacji wykazały głębokie zrozumienie problematyki uczenia maszynowego.

##### 3. Wsparcie w Procesie Budowy i Optymalizacji Modelu:

- ChatGPT-4 służył jako narzędzie wspierające w procesie iteracyjnym, pomagając w dostosowywaniu modelu do specyfikacji projektu.
- Zaproponowane zmiany, jak wykorzystanie VGG16 i Early Stopping, miały na celu poprawę wydajności modelu, co jest zgodne z najlepszymi praktykami w dziedzinie uczenia maszynowego.

##### 4. Ocena Wyników i Wnioski:

- Analizując wyniki finalnego modelu, ChatGPT-4 skupił się na szczegółowej ocenie wykresów dokładności i straty, raportu klasyfikacji oraz macierzy pomyłek, dostarczając wartościowych wniosków.
- Zidentyfikowano miejsca do poprawy, takie jak nierównomierne wyniki między klasami i potencjalne przetrenowanie. ChatGPT-4 przedstawił praktyczne sugestie dotyczące dalszej optymalizacji modelu, co świadczy o jego przydatności w procesie ewaluacji i iteracyjnego doskonalenia modeli uczenia maszynowego.

Podsumowując, ChatGPT-4 wykazał skuteczność w generowaniu i dostosowywaniu kodu, analizie wyników oraz w udzielaniu praktycznych porad, co czyni go wartościowym narzędziem w projektach klasyfikacyjnych, szczególnie w początkowych fazach rozwoju modelu i jego iteracyjnej optymalizacji.

## 4.2. Budowa modelu do regresji cen samochodów z pomocą ChatGPT-4

### 4.2.1. Konwersacja z ChatGPT-4



Ja:

Cześć, chciałbym, abyś wytrenował model uczenia maszynowego do przewidywania zmiennej `Price`. Moje dane są w załączniku.



ChatGPT:

```
import pandas as pd

# Re-loading the dataset as the previous attempt was interrupted
file_path = '/mnt/data/used_cars_UK.csv'
used_cars_df = pd.read_csv(file_path)

# Displaying the first few rows of the dataset to understand its structure
used_cars_df.head()
```

*Twoja baza danych używanych samochodów z Wielkiej Brytanii zawiera następujące kolumny:*

1. **`Unnamed: 0`**: Wygląda na indeks wierszy.
2. **`title`**: Nazwa/model samochodu.
4. **`Price`**: Cena samochodu (zmienna, którą chcesz przewidzieć).
5. **`Mileage(miles)`**: Przebieg w milach.
6. **`Registration\_Year`**: Rok rejestracji.
7. **`Previous Owners`**: Liczba poprzednich właścicieli.
8. **`Fuel type`**: Typ paliwa.
9. **`Body type`**: Typ nadwozia.
10. **`Engine`**: Pojemność silnika.
11. **`Gearbox`**: Typ skrzyni biegów.
12. **`Doors`**: Liczba drzwi.
13. **`Seats`**: Liczba siedzeń.
14. **`Emission Class`**: Klasa emisji.
15. **`Service history`**: Historia serwisowa.

Aby wytrenować model uczenia maszynowego do przewidywania ceny (**`Price`**), należy wykonać następujące kroki:

1. **Czyszczenie Danych:** Usunięcie lub uzupełnienie brakujących danych, przekształcenie tekstowych zmiennych kategoryalnych na numeryczne.
2. **Eksploracja Danych:** Analiza rozkładów zmiennych, sprawdzenie korelacji między zmiennymi.
3. **Inżynieria Cech<sup>[15]</sup> (Feature Engineering):** Utworzenie nowych cech, które mogą lepiej reprezentować dane dla modelu.
4. **Podział Danych:** Podzielenie danych na zestawy treningowe i testowe.
5. **Wybór Modelu:** Wybór odpowiedniego modelu uczenia maszynowego.
6. **Trenowanie Modelu:** Uczenie modelu na danych treningowych.
7. **Ocena Modelu:** Testowanie modelu na danych testowych i ocena jego dokładności.
8. **Dostrojenie Modelu:** Dostosowanie parametrów modelu dla lepszej wydajności.

Przystąpię teraz do przetworzenia danych i przygotowania ich do modelowania. Zacznę od czyszczenia danych.

```
# Checking for missing values and data types in the dataset
missing_values = used_cars_df.isnull().sum()
data_types = used_cars_df.dtypes

missing_values, data_types
```

Analiza brakujących danych i typów danych w Twoim zbiorze danych wykazała:

1. **Brakujące dane:**
  - **`Previous Owners`:** Brakuje 1409 wartości.
  - **`Engine`:** Brakuje 45 wartości.
  - **`Doors`:** Brakuje 25 wartości.
  - **`Seats`:** Brakuje 35 wartości.
  - **`Emission Class`:** Brakuje 87 wartości.
  - **`Service history`:** Brakuje 3145 wartości.
2. **Typy danych:**
  - Większość kolumn ma odpowiednie typy danych, takie jak liczby całkowite (**`int64`**) dla ceny, przebiegu, roku rejestracji itd.
  - Kolumny z danymi tekstowymi, takie jak **`title`**, **`Fuel type`**, **`Body type`**, **`Engine`**, **`Gearbox`**, **`Emission Class`** i **`Service history`**, są typu **`object`** i mogą wymagać przekształcenia do formatu numerycznego dla modelowania.

Należy rozważyć następujące kroki w celu przygotowania danych:

1. **Usunięcie lub uzupełnienie brakujących wartości:** Możemy usunąć wiersze lub kolumny z dużą liczbą brakujących danych lub uzupełnić brakujące wartości, na przykład używając średniej, mediany lub najczęściej występującej wartości.
2. **Kodowanie zmiennych kategoryalnych:** Zmienne tekstowe (kategorialne) należy przekształcić na numeryczne przy użyciu technik takich jak kodowanie gorąco-jedynkowe (one-hot encoding) lub kodowanie etykiet (label encoding).
3. **Normalizacja danych:** Normalizacja cech liczbowych, takich jak przebieg czy rok rejestracji, może być pomocna w niektórych modelach.

Zacznę od czyszczenia danych, skupiając się na uzupełnianiu brakujących wartości i przekształcaniu zmiennych kategoryalnych.

```
from sklearn.impute import SimpleImputer
import numpy as np

# Dropping the 'Unnamed: 0' column as it's just an index
used_cars_df.drop(columns='Unnamed: 0', inplace=True)

# Imputing missing values
# For numerical columns, using median to impute missing values
num_imputer = SimpleImputer(strategy='median')
used_cars_df['Previous Owners'] =
num_imputer.fit_transform(used_cars_df[['Previous Owners']])
used_cars_df['Doors'] = num_imputer.fit_transform(used_cars_df[['Doors']])
used_cars_df['Seats'] = num_imputer.fit_transform(used_cars_df[['Seats']])

# For categorical columns, using the mode (most frequent) to impute missing
values
cat_imputer = SimpleImputer(strategy='most_frequent')
used_cars_df['Engine'] = cat_imputer.fit_transform(used_cars_df[['Engine']])
used_cars_df['Emission Class'] =
cat_imputer.fit_transform(used_cars_df[['Emission Class']])
used_cars_df['Service history'] =
cat_imputer.fit_transform(used_cars_df[['Service history']])

# Checking the dataset after imputation
used_cars_df.isnull().sum()
```

Brakujące dane w zbiorze zostały pomyślnie uzupełnione. Teraz przystąpię do kodowania zmiennych kategoryalnych, aby umożliwić ich użycie w modelu uczenia maszynowego.

Użyję techniki kodowania gorąco-jedynkowego (one-hot encoding) dla zmiennych kategoryalnych. Ta technika tworzy nowe kolumny wskazujące obecność (lub brak) każdej możliwej wartości w oryginalnej kolumnie danych. Jest to przydatne dla modeli, które wymagają danych numerycznych, takich jak większość algorytmów uczenia maszynowego.



```

from sklearn.preprocessing import OneHotEncoder

# Selecting categorical columns for one-hot encoding
categorical_columns = ['Fuel type', 'Body type', 'Engine', 'Gearbox', 'Emission
Class', 'Service history']

# Applying OneHotEncoder
encoder = OneHotEncoder(sparse=False, drop='first') # 'drop=first' to avoid
multicollinearity
encoded_columns = encoder.fit_transform(used_cars_df[categorical_columns])
encoded_columns_df = pd.DataFrame(encoded_columns,
columns=encoder.get_feature_names_out(categorical_columns))

# Adding the new encoded columns to the original dataframe
used_cars_encoded_df = used_cars_df.join(encoded_columns_df)

# Dropping the original categorical columns as they are now encoded
used_cars_encoded_df.drop(columns=categorical_columns, inplace=True)

# Displaying the modified dataframe
used_cars_encoded_df.head()

```

*Zmienna kategorialna została pomyślnie zakodowana za pomocą kodowania gorąco-jedynkowego. Teraz mamy dodatkowe kolumny, które reprezentują różne kategorie dla każdej z oryginalnych zmiennych kategorialnych.*

*Następnym krokiem jest podział danych na zestawy treningowe i testowe. Zwykle stosuje się stosunek podziału, na przykład 80% danych do treningu i 20% do testowania. Następnie będziemy mogli wybrać i wytrenować model uczenia maszynowego, wykorzystując te dane.*

*Przystępuję teraz do podziału danych.*

```

from sklearn.model_selection import train_test_split

# Defining the features and the target variable
X = used_cars_encoded_df.drop(columns=['title', 'Price']) # Dropping the title
column as it's not useful for prediction
y = used_cars_encoded_df['Price']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Checking the shape of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape

```

*Dane zostały podzielone na zestawy treningowe i testowe. Zestaw treningowy składa się z 2948 próbek, a zestaw testowy z 737 próbek.*

Teraz wybiorę model uczenia maszynowego do przewidywania ceny samochodu (**Price**). Biorąc pod uwagę charakter problemu (regresja) oraz rodzaj i rozmiar danych, dobrze nadają się następujące modele:

1. **Regresja Liniowa**: Prosty i efektywny dla problemów regresji liniowej.
2. **Drzewo Decyzyjne**: Dobrze radzi sobie z nieliniowościami i interakcjami między cechami.
3. **Las Losowy (Random Forest)**: Bardziej zaawansowana wersja drzew decyzyjnych, zazwyczaj oferująca lepszą wydajność.
4. **Gradient Boosting**: Potężny model, który iteracyjnie poprawia swoje prognozy.

Rozpocznę od wytrenowania modelu regresji liniowej, a następnie ocenię jego wydajność. W zależności od wyników, możemy rozważyć użycie bardziej zaawansowanych modeli.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Training the Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predicting on the test set
y_pred = lin_reg.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

Model regresji liniowej został wytrenowany i oceniony na zestawie testowym. Wyniki są następujące:

- **Średni Błąd Kwadratowy (MSE)**: 4,891,755.08
- **Współczynnik determinacji ( $R^2$ )**: 0.775

Współczynnik  $R^2$  wynosi około 0.775, co oznacza, że model wyjaśnia około 77.5% zmienności ceny samochodu. Jest to dość dobra wydajność, ale możliwe, że inne modele, takie jak las losowy czy gradient boosting, mogłyby dać lepsze wyniki.

Jeśli chcesz, mogę przetestować inne modele, aby sprawdzić, czy mogą one zapewnić lepsze przewidywania. Czy chciałbyś, żebym kontynuował z innymi modelami, czy też skupić się na dalszym dostosowywaniu i analizie modelu regresji liniowej?



Ja:

Dziękuję za wytrenowanie modelu regresji liniowej na tych danych. Co do twojego pytania dotyczącego innych modeli to tak. Jeśli masz możliwość to przetestuj też bardziej zaawansowane modele które mogą dać lepsze wyniki.



Wygłąda na to, że wystąpił błąd techniczny podczas próby trenowania bardziej zaawansowanych modeli. Możliwe przyczyny to ograniczenia sprzętowe lub zasoby systemowe, które nie są wystarczające do obsługi tych modeli. Zamiast próbować ponownie, mogę podać ogólne informacje na temat tych modeli i jak zazwyczaj są używane do poprawy wyników w podobnych zadaniach:

1. **Random Forest (Las Losowy):** Jest to model ensemble, który działa przez trenowanie wielu drzew decyzyjnych na różnych podzbiorach danych i uśrednianie ich prognoz. Zazwyczaj radzi sobie lepiej niż pojedyncze drzewo decyzyjne, ponieważ redukuje ryzyko przeuczenia i zwiększa dokładność prognoz.
2. **Gradient Boosting:** To kolejny model ensemble, który iteracyjnie trenuje drzewa decyzyjne w taki sposób, że każde kolejne drzewo stara się poprawić błędy popełnione przez poprzednie. Jest często uważany za jeden z najbardziej efektywnych dostępnych algorytmów uczenia maszynowego dla zadań regresyjnych i klasyfikacyjnych.

W typowych scenariuszach, zarówno Random Forest, jak i Gradient Boosting, mają potencjał do osiągnięcia lepszych wyników niż prosta regresja liniowa, zwłaszcza w przypadkach, gdy zależności w danych są bardziej złożone i nieliniowe. Jednakże wymagają one również bardziej starannej regulacji parametrów i mogą być bardziej podatne na przeuczenie, szczególnie w przypadku niewielkich zestawów danych.

Jeśli zdecydujesz się na użycie tych modeli, zalecam eksperymentowanie z różnymi ustawieniami parametrów, takimi jak liczba drzew w Random Forest czy prędkość uczenia w Gradient Boosting, aby znaleźć optymalną konfigurację dla Twoich danych.

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Training a Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Training a Gradient Boosting Regressor
gb_reg = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_reg.fit(X_train, y_train)
y_pred_gb = gb_reg.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

mse_rf, r2_rf, mse_gb, r2_gb
```

 Ja:

Używając twojego kodu dotyczącego testowania innych modeli, otrzymałem dane wyniki:

(3019364.3054279084,  
0.8609211706301507,  
3192981.2067854283,  
0.8529239788516639)

 ChatGPT:

Wyniki, które uzyskałeś dla modeli *Random Forest* i *Gradient Boosting*, wskazują na ich wyższą skuteczność w przewidywaniu ceny samochodów w porównaniu z wcześniejszym modelem regresji liniowej. Oto szczegóły:

1. **RandomForestRegressor:**

- Średni Błąd Kwadratowy (MSE): 3019364.3054279084,
- Współczynnik determinacji ( $R^2$ ): 0.8609211706301507

2. **GradientBoostingRegressor:**

- Średni Błąd Kwadratowy (MSE): 3192981.2067854283,
- Współczynnik determinacji ( $R^2$ ): 0.8529239788516639)

## Analiza Wyników

- **Random Forest** osiągnął najwyższy współczynnik  $R^2$  (0.861), co oznacza, że model ten jest w stanie wyjaśnić około 86.1% zmienności w danych dotyczących ceny samochodu. Jest to znacząca poprawa w porównaniu do regresji liniowej (około 77.5%).
- **Gradient Boosting** również osiągnął dobre wyniki, z  $R^2$  równym 0.853, co oznacza, że wyjaśnia on około 85.3% zmienności w cenach samochodów.

MSE dla obu modeli jest niższy niż w przypadku regresji liniowej, co wskazuje na mniejsze błędy w prognozach cen.

## Rekomendacje

- **Random Forest** wydaje się być najlepszym wyborem na podstawie tych wyników, oferując najwyższą dokładność.
- Mimo to, warto zwrócić uwagę na potencjalne ryzyko przeuczenia (*overfitting*), szczególnie w modelach takich jak *Random Forest* i *Gradient Boosting*. Możesz to sprawdzić, porównując wyniki na zestawie treningowym i testowym.
- Możesz również eksperymentować z dostrojeniem hiperparametrów dla obu modeli, aby sprawdzić, czy da się uzyskać jeszcze lepsze wyniki. Na przykład, możesz zmieniać liczbę drzew w *Random Forest* lub szybkość uczenia się w *Gradient Boosting*.

Jeśli potrzebujesz pomocy w dostrojeniu hiperparametrów lub masz inne pytania dotyczące tych modeli, daj mi znać!



Ja:

Odnosnie twojej rekomendacji może podaj mi kod na przeszukiwanie hiperparametrów wykorzystując model Random Forest Regressor.



ChatGPT:

Oczywiście! Do przeszukiwania hiperparametrów w modelu Random Forest Regressor możemy użyć narzędzia takiego jak GridSearchCV z biblioteki scikit-learn. GridSearchCV przeszukuje siatkę określonych wartości parametrów i znajduje kombinację, która daje najlepsze wyniki. Poniżej znajdziesz przykładowy kod wykorzystujący GridSearchCV do dostrojenia hiperparametrów dla Random Forest Regressor.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Definiowanie parametrów do przetestowania
param_grid = {
    'n_estimators': [50, 100, 200], # Liczba drzew
    'max_features': ['auto', 'sqrt', 'log2'], # Maksymalna liczba cech
    'max_depth': [4, 6, 8, 10], # Maksymalna głębokość drzewa
    'min_samples_split': [2, 5, 10], # Minimalna liczba próbek wymagana do
    'min_samples_leaf': [1, 2, 4]} # Minimalna liczba próbek wymagana w liściu

# Tworzenie modelu Random Forest
rf = RandomForestRegressor(random_state=42)

# Tworzenie instancji GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')

# Przeszukiwanie siatki parametrów
grid_search.fit(X_train, y_train)

# Wyświetlenie najlepszych parametrów
print("Najlepsze parametry:", grid_search.best_params_)

# Używanie najlepszego modelu do dokonania prognoz
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)

# Ocena modelu
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("R2:", r2)
```

## Wskazówki do Użycia Kodu:

1. **Parametry w `param\_grid`:** Zdefiniowane parametry są tylko przykładowe. Możesz je dostosować w zależności od swoich potrzeb i ograniczeń obliczeniowych.
2. **Liczba Foldów w Cross-Validation (cv):** Użyłem `cv=3` dla prostoty, ale możesz zwiększyć tę liczbę dla dokładniejszej walidacji krzyżowej.
3. **Scoring:** Użyłem `neg\_mean\_squared\_error` jako metryki, ale możesz wybrać inną, która lepiej pasuje do Twojego zadania.
4. **N\_jobs:** Ustawienie `n\_jobs=-1` pozwala na używanie wszystkich dostępnych rdzeni procesora, co przyspiesza przeszukiwanie, ale wymaga więcej zasobów.

Ten kod pomoże Ci znaleźć najlepsze ustawienia dla modelu Random Forest na Twoich danych. Po dostrojeniu hiperparametrów model powinien oferować jeszcze lepszą wydajność.



Ja:

Używając twojego kodu dotyczącego przeszukiwania hiperparametrów w modelu RandomForestRegressor otrzymałem wyniki:

Fitting 3 folds for each of 324 candidates, totalling 972 fits

Najlepsze parametry: {'max\_depth': 10, 'max\_features': 'auto', 'min\_samples\_leaf': 1, 'min\_samples\_split': 5, 'n\_estimators': 200}

MSE: 3078007.4350967756

R2: 0.8582199338796638



ChatGPT:

Wyniki, które uzyskałeś z GridSearchCV dla modelu RandomForestRegressor, wskazują na skuteczne dostrojenie hiperparametrów. Oto szczegółowa analiza:

### Najlepsze Hiperparametry:

- `max\_depth`: 10
- `max\_features`: 'auto'
- `min\_samples\_leaf`: 1
- `min\_samples\_split`: 5
- `n\_estimators`: 200

Te parametry zostały wybrane jako najlepsze w wyniku przeszukiwania siatki, co oznacza, że powinny zapewnić optymalną równowagę między wydajnością a ryzykiem przeuczenia dla Twojego modelu.

### **Wyniki Modelu po Dostrojeniu:**

- **Średni Błąd Kwadratowy (MSE):** 3,078,007.44
- **Współczynnik determinacji ( $R^2$ ):** 0.858

### **Analiza Wyników:**

- **MSE** po dostrojeniu jest nieco wyższy niż wynik uzyskany wcześniej bez dostrojenia hiperparametrów (3,019,364.31). Może to wynikać z różnic w podziale danych treningowych i testowych lub z zastosowania walidacji krzyżowej w procesie GridSearch.
- **Współczynnik  $R^2$**  pozostaje wysoki (0.858), co wskazuje na to, że model dobrze wyjaśnia zmienność ceny samochodów.

### **Dalsze Kierunki:**

1. **Ewaluacja na Różnych Zestawach Danych:** Sprawdź, jak model radzi sobie na różnych zestawach danych, aby ocenić jego ogólną stabilność i zdolność do generalizacji.
2. **Analiza Ważności Cech:** Możesz zbadać, które cechy najbardziej przyczyniają się do prognoz ceny, co może dostarczyć cennych informacji biznesowych.
3. **Eksperymentowanie z Inne Modelami:** Jeśli czas i zasoby na to pozwalają, warto przetestować inne modele, takie jak Gradient Boosting czy nawet głębokie sieci neuronowe, aby zobaczyć, czy mogą one zapewnić lepszą wydajność.

Pamiętaj, że dostrojenie hiperparametrów to proces iteracyjny i czasami wymaga wielu prób, aby znaleźć optymalne ustawienia. Jeśli masz jakiegokolwiek pytania lub potrzebujesz dalszej pomocy, daj mi znać!

#### 4.2.2. Ocena skuteczności i dokładności ChatGPT-4

##### 1. Wsparcie w Przygotowaniu Danych i Generowaniu Kodu:

- ChatGPT-4 skutecznie wspierał programistę w procesie czyszczenia danych, uzupełniania brakujących wartości, kodowania zmiennych kategoryalnych oraz normalizacji danych. Jego wiedza o biblioteki Pythona takich jak pandas, sklearn, była niezbędna w przygotowaniu tych danych do analizy.
- Możliwość ChatGPT-4 do budowy modeli i analizy danych w swoim własnym środowisku okazała się pomocna. Dzięki temu model językowy mógł samodzielnie uruchamiać utworzone skrypty i oceniać otrzymane wyniki. Jedynym problemem była budowa bardziej zaawansowanych modeli regresyjnych, wymagających użycia większej ilości pamięci, na co system ChatGPT-4 nie pozwalał.

##### 2. Wybór i Implementacja Modeli:

- ChatGPT-4 zaproponował i zaimplementował różne modele regresyjne, w tym Regresję Liniową, Random Forest i Gradient Boosting, dostosowane do specyfiki problemu.
- Zaproponowane modele były odpowiednie dla zadania regresji cen samochodów, co uwidaczniało się w wydajności tych modeli.

##### 3. Ocena i Optymalizacja Modelu:

- Model regresji liniowej zaprezentowany przez ChatGPT-4 wykazał dobrą wydajność, co zostało potwierdzone przez analizę MSE i R2. Dalsze modele, jak Random Forest i Gradient Boosting, pokazały jeszcze lepsze wyniki, co wskazuje na skuteczną ocenę i selekcję modelu przez ChatGPT-4.
- Dostrojenie hiperparametrów modelu Random Forest przy użyciu GridSearchCV wykazało umiejętność ChatGPT-4 w fine-tuning modeli, co jest istotne w osiągnięciu optymalnej wydajności.

##### 4. Interpretacja Wyników i Wnioski:

- ChatGPT-4 skutecznie interpretował wyniki, wskazując na znaczące poprawy w dokładności modelu po dostosowaniu hiperparametrów.
- Analiza i rekomendacje dotyczące dalszych kroków, takie jak ewaluacja modelu na różnych zestawach danych i analiza ważności cech, były wartościowe i wskazywały na głębokie zrozumienie procesu modelowania.

Podsumowując, ChatGPT-4 wykazał się dużą skutecznością i dokładnością w projektowaniu, implementacji i optymalizacji modeli regresji cen samochodów. Jego zdolność do analizy danych, generowania efektywnego kodu, wyboru odpowiednich modeli, a także interpretacji wyników pokazują, że jest bardzo biegły w tworzeniu modeli regresji.



## 4.3. Budowa modelu do rekomendacji filmów z pomocą ChatGPT-4

### 4.3.1 Konwersacja z ChatGPT-4



Ja:

Cześć, chciałbym, abyś wytrenował model uczenia nienadzorowanego do rekomendowania filmów, wykorzystując dane filmów Netflix'a. Dane znajdują się w załączniku.



ChatGPT:

Świetnie! Zaczniemy od wytrenowania modelu uczenia nienadzorowanego do rekomendacji filmów na podstawie danych z Netflix. Najpierw przyjrzymy się strukturze dostarczonego pliku CSV, aby zrozumieć, jakie informacje zawiera i jak możemy je wykorzystać. Następnie omówimy możliwe podejścia do budowy systemu rekomendacji. Przejdę teraz do analizy pliku.

Plik danych z Netflix zawiera następujące kolumny:

- ``show_id``: Unikalny identyfikator programu.
- ``type``: Typ programu (film lub serial telewizyjny).
- ``title``: Tytuł programu.
- ``director``: Reżyser.
- ``cast``: Obsada.
- ``country``: Kraj produkcji.
- ``date_added``: Data dodania do Netflix.
- ``release_year``: Rok wydania.
- ``rating``: Ocena wiekowa.
- ``duration``: Czas trwania.
- ``listed_in``: Kategorie, do których program jest zaliczany.
- ``description``: Opis programu.

Do stworzenia systemu rekomendacji<sup>[16]</sup>, możemy skorzystać z różnych podejść. Jednym z nich jest wykorzystanie informacji tekstowych, takich jak opis, tytuł, obsada, i kategoria, do stworzenia cech (ang. *features*), które będą podstawą do znalezienia podobieństw między filmami. Możemy użyć technik przetwarzania języka naturalnego (NLP<sup>[17]</sup>) do przekształcenia tych danych tekstowych w wektory, a następnie zastosować algorytmy takie jak K-means lub podobieństwo cosinusowe do grupowania lub porównywania filmów.

Przed przystąpieniem do modelowania, warto również przeprowadzić wstępną obróbkę danych, taką jak usunięcie brakujących wartości, zamiana tekstów na małe litery, usunięcie znaków interpunkcyjnych i podobnych.

Chciałbyś, abym poszedł dalej i zaczął przygotowywać dane, a następnie wybrał odpowiednią metodę do budowy systemu rekomendacji?



Ja:

Tak. Jeśli napotkasz jakiś problem techniczny podaj mi kod abym mógł zastosować go w swoim środowisku.



ChatGPT:

W porządku, zaczniemy od przygotowania danych. Skoncentrujemy się na kilku kolumnach, które mogą być najbardziej pomocne w budowaniu systemu rekomendacji: ``title``, ``cast``, ``country``, ``listed_in`` (kategorie), i ``description``. Wykonamy następujące kroki:

1. **Czyszczenie danych:** Usuniemy brakujące wartości i przetworzymy tekst.
2. **Przetwarzanie tekstu:** Zamienimy tekst na małe litery, usuniemy znaki interpunkcyjne i podzielimy tekst na słowa (tokenizacja).
3. **Wektoryzacja tekstu:** Użyjemy metody, takiej jak TF-IDF<sup>[18]</sup> (Term Frequency-Inverse Document Frequency), aby przekształcić tekst w wektory liczbowe.
4. **Budowa modelu rekomendacji:** Możemy użyć metody podobieństwa cosinusowego lub innych technik uczenia maszynowego do oceny podobieństwa między filmami.

Zacznijmy od kroków 1 i 2, czyli od czyszczenia i przetwarzania tekstu. Po zakończeniu tego etapu, możemy przejść do wektoryzacji tekstu i budowy modelu rekomendacji. Przygotuję teraz odpowiedni kod.

Dane zostały przygotowane poprzez wstępne przetworzenie tekstu. Każdy rekord teraz zawiera połączone i oczyszczone informacje z kolumn ``title``, ``cast``, ``country``, ``listed_in``, i ``description``. Teraz przejdziemy do kroku 3, czyli do wektoryzacji tekstu.

Do wektoryzacji użyjemy metody TF-IDF (Term Frequency-Inverse Document Frequency), która przekształci tekst w wektory liczbowe, biorąc pod uwagę nie tylko częstotliwość występowania słów, ale także ich ważność w całym zbiorze danych. Następnie, w kroku 4, użyjemy tych wektorów do oceny podobieństwa między filmami przy użyciu podobieństwa cosinusowego.

Zaimplementuję teraz wektoryzację TF-IDF i pokażę, jak można wykorzystać wynikowe wektory do tworzenia rekomendacji.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Data Preprocessing
# Filling missing values and combining text columns
netflix_data.fillna('', inplace=True)
netflix_data['combined_features'] = netflix_data['title'] + ' ' +
netflix_data['cast'] + ' ' + netflix_data['country'] + ' ' +
netflix_data['listed_in'] + ' ' + netflix_data['description']

# Function to clean and preprocess text
def clean_text(text):
```

```

    text = text.lower() # convert to lowercase
    text = re.sub(r'\s+', ' ', text) # replace multiple spaces with single
space
    text = re.sub(r'[^w\s]', '', text) # remove punctuation
    return text

# Applying the cleaning function to combined features
netflix_data['combined_features'] =
netflix_data['combined_features'].apply(clean_text)

# Display the cleaned combined features of the first few rows
netflix_data['combined_features'].head()

```

*Wektoryzacja TF-IDF została zakończona, co dało nam macierz o wymiarach 8807 x 53785. Oznacza to, że mamy 8807 filmów reprezentowanych za pomocą 53785 unikalnych cech (słów).*

*Teraz możemy użyć tej macierzy do znalezienia podobieństw między filmami. Użyjemy podobieństwa cosinusowego, które jest popularną miarą w systemach rekomendacji do oceny podobieństwa między różnymi dokumentami (w naszym przypadku filmami).*

*Załóżmy, że chcesz znaleźć filmy podobne do konkretnego filmu. Wybierzmy na przykład film z indeksem 0 (pierwszy film w naszym zbiorze danych) i znajdziemy filmy, które są do niego najbardziej podobne. Poniżej przedstawię kod, który to umożliwia. Następnie możesz zastosować ten sam schemat w swoim środowisku, wybierając inne filmy jako punkty odniesienia.*

```

# Vectorization using TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix =
tfidf_vectorizer.fit_transform(netflix_data['combined_features'])

# Example: Displaying the shape of the TF-IDF matrix
tfidf_matrix.shape

```

*Dla filmu "Dick Johnson Is Dead" (pierwszy film w naszym zbiorze danych), model rekomendacyjny na podstawie podobieństwa cosinusowego zaproponował następujące filmy jako najbardziej podobne:*

1. "The Death and Life of Marsha P. Johnson"
2. "End Game"
3. "How to Be a Player"
4. "New Girl"
5. "Win It All"

Możesz użyć tej metody, aby znaleźć rekomendacje dla innych filmów w zbiorze danych. Aby to zrobić, wystarczy podać tytuł filmu jako argument funkcji `find_similar_movies`. Poniżej znajdziesz kod, który możesz wykorzystać w swoim środowisku:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Wczytywanie danych
netflix_data = pd.read_csv('ścieżka_do_pliku_csv') # Podmień na odpowiednią
ścieżkę do pliku

# Przetwarzanie i czyszczenie danych
netflix_data.fillna('', inplace=True)
netflix_data['combined_features'] = netflix_data['title'] + ' ' +
netflix_data['cast'] + ' ' + netflix_data['country'] + ' ' +
netflix_data['listed_in'] + ' ' + netflix_data['description']
netflix_data['combined_features'] =
netflix_data['combined_features'].apply(lambda x: re.sub(r'^\w\s', '',
x.lower()).replace('\s+', ' '))

# Wektoryzacja TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix =
tfidf_vectorizer.fit_transform(netflix_data['combined_features'])

# Funkcje pomocnicze
def get_title_from_index(index):
    return netflix_data[netflix_data.index == index]["title"].values[0]

def get_index_from_title(title):
    return netflix_data[netflix_data.title == title].index.values[0]

def find_similar_movies(movie_title, num_recommendations=5):
    movie_index = get_index_from_title(movie_title)
    cosine_similarities = cosine_similarity(tfidf_matrix[movie_index],
tfidf_matrix)
    similar_movies = list(enumerate(cosine_similarities[0]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x: x[1],
reverse=True)[1:num_recommendations+1]

    recommended_movies = []
    for i in sorted_similar_movies:
        movie_title = get_title_from_index(i[0])
        recommended_movies.append(movie_title)

    return recommended_movies

# Przykład użycia
test_movie_title = 'Podaj tytuł filmu'
recommended_movies = find_similar_movies(test_movie_title)
print(recommended_movies)
```

Zastąp ``ścieżka_do_pliku_csv`` odpowiednią ścieżką do twojego pliku CSV i ``test_movie_title`` tytułem filmu, dla którego chcesz uzyskać rekomendacje. Ten kod pomoże Ci zbudować podstawowy system rekomendacji<sup>[19]</sup> filmów.



Ja:

**Czy, korzystając ze stworzonego przez ciebie systemu rekomendacji, możesz polecić mi inne filmy podobne do "Terminator 3: Rise of the Machines"?**



ChatGPT:

*Filmy rekomendowane jako podobne do "Terminator 3: Rise of the Machines" według stworzonego systemu rekomendacji to:*

1. "Terminator Salvation"
2. "Fat Ballerina - David A. Arnold"
3. "9"
4. "Total Recall"
5. "Stardust"

*Mam nadzieję, że te rekomendacje będą dla Ciebie interesujące!*

Kod:

```
# Using the recommendation system to find movies similar to "Terminator 3: Rise of the Machines"
recommended_movies_for_terminator_3 = find_similar_movies("Terminator 3: Rise of the Machines")
recommended_movies_for_terminator_3
```



Ja:

**Twoim zdaniem, czy podane przez system filmy są dobrą rekomendacją? Co można by zrobić aby poprawić przedstawiony przez ciebie system rekomendacji.**



ChatGPT:

*Ocena jakości rekomendacji zależy od wielu czynników, w tym od osobistych preferencji widza i kontekstu wyboru filmów. System rekomendacji oparty na podobieństwie cosinusowym i wektoryzacji TF-IDF, który zaimplementowałem, jest podstawowym podejściem i ma swoje ograniczenia. Oto kilka uwag na temat rekomendacji i sugestii dotyczących ulepszeń:*

## Ocena Rekomendacji

1. **Znaczenie Kontekstowe:** Filmy takie jak "Terminator Salvation" i "Total Recall" wydają się być dobrymi rekomendacjami ze względu na podobny gatunek (science fiction, akcja). Jednakże inne tytuły, jak "Fat Ballerina - David A. Arnold", mogą wydawać się mniej trafne, co sugeruje, że algorytm mógł skupić się na innych cechach niż te, które są najważniejsze dla konkretnego gatunku lub stylu filmu.

## Sposoby Poprawy Systemu

1. **Lepsze Przetwarzanie Tekstowe:** Udoskonalenie przetwarzania tekstu, w tym lepsza tokenizacja i lematyzacja<sup>[23]</sup>, mogłoby poprawić jakość wektoryzacji.
2. **Zastosowanie Metod NLP:** Użycie bardziej zaawansowanych metod NLP, takich jak osadzenia słów (word embeddings<sup>[24]</sup>) z wykorzystaniem modeli takich jak Word2Vec lub BERT, mogłoby poprawić sposób reprezentacji znaczeń w opisach filmów.
3. **Uwzględnienie Oceny Użytkowników:** Integracja ocen użytkowników mogłaby pomóc w dostosowaniu rekomendacji do indywidualnych preferencji.
4. **Klasteryzacja Gatunków:** Wykorzystanie klasteryzacji do identyfikacji i rekomendacji filmów wewnątrz konkretnych gatunków mogłoby zwiększyć trafność sugestii.
5. **Uczenie Maszynowe i Głębokie Uczenie:** Zastosowanie zaawansowanych technik uczenia maszynowego i głębokiego uczenia, takich jak sieci neuronowe, mogłoby znacząco poprawić dokładność systemu rekomendacji.
6. **Feedback Użytkowników:** Włączenie mechanizmu feedbacku, który pozwala użytkownikom oceniać rekomendacje, może pomóc w ciągłym ulepszaniu systemu.
7. **Hybrydowy System Rekomendacji:** Połączenie kilku podejść, takich jak rekomendacje oparte na treści i filtrowanie kolaboratywne, może dać bardziej zróżnicowane i trafne wyniki.

Wdrożenie tych usprawnień może znacząco zwiększyć skuteczność systemu rekomendacji. Warto jednak pamiętać, że każdy system rekomendacji będzie miał swoje ograniczenia i nie zawsze będzie w stanie idealnie dopasować się do indywidualnych preferencji każdego użytkownika.



Ja:

Spróbuj ulepszyć system, wykorzystując uczenie maszynowe i głębokie uczenie. Jeśli nie będziesz miał wystarczających zasobów do trenowania tego systemu, prześlij mi kod, a ja przeprowadzę trening w moim środowisku.

## ChatGPT:

*Ulepszając system rekomendacji poprzez wykorzystanie uczenia maszynowego i głębokiego uczenia, możemy skorzystać z bardziej zaawansowanych technik, takich jak osadzenia słów (word embeddings) i sieci neuronowe. Jednym z popularnych podejść jest zastosowanie sieci neuronowych do przetwarzania sekwencji, takich jak LSTM<sup>[22]</sup> (Long Short-Term Memory) lub GRU (Gated Recurrent Units), które są skuteczne w analizie danych sekwencyjnych, takich jak tekst.*

*Poniżej przedstawiam podstawowy zarys kodu, który możesz użyć do zbudowania i wytrenowania takiego modelu. Ze względu na ograniczenia zasobów w tym środowisku, nie będę mógł przeprowadzić treningu, ale dostarczę Ci kod, który możesz uruchomić w swoim środowisku.*

### **Krok 1: Przygotowanie Danych**

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# Tokenizacja i sekwencyjne kodowanie tekstu
tokenizer = Tokenizer()
tokenizer.fit_on_texts(netflix_data['combined_features'])

# Przekształcanie tekstów na sekwencje
sequences = tokenizer.texts_to_sequences(netflix_data['combined_features'])

# Ustalanie maksymalnej długości sekwencji
max_len = max(len(x) for x in sequences)

# Wyrównywanie sekwencji
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post')
```

### **Krok 2: Budowa Modelu**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Parametry modelu
vocab_size = len(tokenizer.word_index) + 1 # Dodanie 1 dla słowa zerowego
embedding_dim = 50 # Możesz dostosować rozmiar osadzeń

# Budowa modelu
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(LSTM(100, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dense(100, activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
```

## Krok 3: Trening Modelu

```
# Przygotowanie danych wyjściowych
y = np.array([x for x in padded_sequences]).reshape(-1, 1)

# Trening modelu
model.fit(padded_sequences, y, epochs=5) # Ustal odpowiednią liczbę epok
```

### Uwagi

1. **Optymalizacja i Dostosowanie:** Parametry modelu, takie jak rozmiar osadzeń i architektura sieci, mogą wymagać dostosowania w zależności od specyfikacji i zasobów dostępnych w twoim środowisku.
2. **Obliczenie Podobieństwa:** Po wytrenowaniu modelu, możesz użyć osadzeń z warstwy `Embedding` do obliczenia podobieństwa między filmami, podobnie jak w przypadku TF-IDF i podobieństwa cosinusowego.
3. **Ograniczenia:** Trening takiego modelu może być zasobożerny i wymagać dużych zbiorów danych do osiągnięcia dobrej jakości.

Możesz dostosować ten kod do swoich potrzeb i dostępnych zasobów. To tylko punkt wyjścia, a ulepszenie modelu może wymagać dodatkowych eksperymentów i dostrojenia.

 Ja:

Dziękuję. Trening utworzonego modelu zakończył się pomyślnie. Czy wiesz jak teraz wykorzystać ten model w naszym systemie dla rekomendacji filmów? Będę chciał wiedzieć jak wykorzystać go dla sugerowania filmów podobnych do 'Terminator 3: Rise of the Machines'

 ChatGPT:

Wspaniale, że trening modelu zakończył się pomyślnie! Teraz, aby wykorzystać wytrenowany model w systemie rekomendacji filmów, skupimy się na wykorzystaniu osadzeń słów (word embeddings) generowanych przez model. Oto kroki, jakie należy podjąć:

## Krok 1: Ekstrakcja Osadzeń

Pierwszym krokiem jest wyekstrahowanie osadzeń z modelu. Każdy film będzie reprezentowany przez wektor osadzeń, który jest średnią wektorów osadzeń wszystkich słów w jego opisie.

```
# Ekstrakcja warstwy osadzeń
embeddings_layer = model.get_layer('embedding')
embeddings = embeddings_layer.get_weights()[0]

# Funkcja do obliczania średniego wektora osadzeń dla filmu
```



```
def get_embedding_vector_for_movie(movie_title):
    # Konwersja tytułu na sekwencję
    sequence = tokenizer.texts_to_sequences([movie_title])
    padded_sequence = pad_sequences(sequence, maxlen=max_len, padding='post')

    # Obliczanie średniego wektora osadzeń
    movie_embedding = np.mean([embeddings[i] for i in padded_sequence[0] if i
    != 0], axis=0)
    return movie_embedding

# Przykład: Wektor osadzeń dla "Terminator 3: Rise of the Machines"
terminator_embedding = get_embedding_vector_for_movie("Terminator 3: Rise of
the Machines")
```

## Krok 2: Obliczanie Podobieństwa

Następnie, użyjemy miary podobieństwa, takiej jak podobieństwo kosinusowe, do porównania wektorów osadzeń filmu referencyjnego z wektorami innych filmów.

```
from sklearn.metrics.pairwise import cosine_similarity

# Funkcja do znalezienia podobnych filmów
def find_similar_movies_using_embeddings(movie_title, num_recommendations=5):
    movie_embedding = get_embedding_vector_for_movie(movie_title)
    all_movies_embeddings = [get_embedding_vector_for_movie(title) for title in
    netflix_data['title']]

    # Obliczanie podobieństwa
    similarities = cosine_similarity([movie_embedding],
    all_movies_embeddings)[0]
    similar_movies_indices = np.argsort(similarities)[::-
    1][1:num_recommendations+1]

    # Zwracanie rekomendowanych tytułów
    recommended_movies = [netflix_data.iloc[i]['title'] for i in
    similar_movies_indices]
    return recommended_movies

# Znajdowanie filmów podobnych do "Terminator 3: Rise of the Machines"
recommended_movies = find_similar_movies_using_embeddings("Terminator 3: Rise
of the Machines")
```

## Uwagi

- **Normalizacja:** Upewnij się, że wektory osadzeń są znormalizowane przed obliczaniem podobieństwa.
- **Wydajność:** Proces ten może być dość czasochłonny, ponieważ wymaga obliczenia osadzeń dla każdego filmu. Rozważ zapisanie wyników w celu przyspieszenia przyszłych rekomendacji.
- **Jakość Osadzeń:** Jakość rekomendacji będzie zależała od jakości osadzeń słów. Jeśli osadzenia nie są wystarczająco dobre, rozważ użycie bardziej zaawansowanych modeli NLP.

Teraz możesz użyć tej metody, aby znaleźć filmy podobne do dowolnego filmu w Twojej bazie danych Netflix.

 Ja:

Dziękuję za podany kod. Na podstawie filmu 'Terminator 3: Rise of the Machines' zasugerowane filmy przez system to:

- 'The Phantom of the Opera',
- 'The One',
- 'The Other One: The Long Strange Trip of Bob Weir',
- 'The Crying Game',
- 'Gargantia on the Verdurous Planet'

Co sądzisz o tych sugestjach?

 ChatGPT:

*Sugerowane filmy, takie jak 'The Phantom of the Opera', 'The One', 'The Other One: The Long Strange Trip of Bob Weir', 'The Crying Game', i 'Gargantia on the Verdurous Planet', wydają się być dość zróżnicowane pod względem gatunku i tematyki w porównaniu do 'Terminator 3: Rise of the Machines'. Oto moje przemyślenia na temat tych sugestii:*

1. **Zróżnicowanie Gatunków:** Wydaje się, że rekomendacje nie koncentrują się wyłącznie na gatunku science fiction lub akcji, co jest charakterystyczne dla serii 'Terminator'. Na przykład 'The Phantom of the Opera' to musical, a 'The Crying Game' to thriller/dramat.
2. **Podobieństwo Tematyczne:** Niektóre z tych filmów mogą mieć pewne wspólne elementy tematyczne (np. tajemnice, złożone postacie), ale mogą znacznie odbiegać od tonu i stylu 'Terminator 3'.
3. **Wpływ Modelu:** Rekomendacje mogą odzwierciedlać ograniczenia lub specyficzne cechy modelu, który został użyty. Na przykład, jeśli model skupiał się na określonych słowach w opisach, mógł wyłapać nietypowe skojarzenia.
4. **Subiektywność Rekomendacji:** Ostatecznie ocena tych rekomendacji zależy od indywidualnych preferencji widza. Dla niektórych osób taka różnorodność może być interesująca, podczas gdy inni mogą preferować bardziej spójne sugestie związane z gatunkiem lub stylem.

## **Dalsze Ulepszenia**

- **Analiza Gatunku:** Możesz rozważyć dodanie analizy gatunku do modelu, aby zapewnić, że rekomendacje są bardziej spójne pod względem gatunkowym.
- **Udoskonalenie Osadzeń:** Eksperymentowanie z bardziej zaawansowanymi osadzeniami słów lub technikami NLP może poprawić jakość rekomendacji.
- **Feedback Użytkowników:** Włączenie opinii użytkowników może pomóc w dostosowywaniu i poprawie rekomendacji.

*Pamiętaj, że systemy rekomendacji są często procesem iteracyjnym, gdzie ciągłe ulepszenia i dostosowania prowadzą do lepszych wyników.*

##### 1. Wsparcie w Przygotowaniu Danych i Generowaniu Kodu:

- ChatGPT-4 efektywnie wspierał proces przygotowania danych, w tym czyszczenie i przetwarzanie tekstu. Użycie technik NLP do przetworzenia danych tekstowych było właściwym kierunkiem.
- Generowanie kodu dla przetwarzania danych, wektoryzacji TF-IDF i obliczanie podobieństwa cosinusowego wskazuje na rozumienie przez model językowy kroków jakie muszą być podjęte dla stworzenia systemu rekomendacji.

##### 2. Budowa i Optimalizacja Modelu:

- ChatGPT-4 zaproponował stosowanie podstawowych technik NLP i algorytmów uczenia maszynowego do budowy systemu rekomendacji. Wykorzystanie TF-IDF i podobieństwa cosinusowego było efektywnym podejściem do znajdowania podobieństw między filmami.
- W odpowiedzi na prośbę o ulepszenie systemu, ChatGPT-4 przedstawił zaawansowany model wykorzystujący sieci neuronowe, co wskazuje na elastyczność w adaptacji do bardziej złożonych rozwiązań.

##### 3. Ocena Jakości Rekomendacji:

- Rekomendacje filmów, choć różnorodne, mogły nie być w pełni spójne z oczekiwaniami dotyczącymi konkretnego gatunku filmu, jak w przypadku "Terminator 3: Rise of the Machines". To sugeruje, że model mógł skupiać się na innych aspektach niż te, które są najważniejsze dla użytkownika.
- Rekomendacje, takie jak "The Phantom of the Opera" i "The Crying Game", wskazują na szeroki zakres analizowanych cech, ale mogą nie być wystarczająco ukierunkowane na konkretny gatunek czy temat.

##### 4. Sugestie dotyczące Ulepszeń:

- ChatGPT-4 zasugerował kilka sposobów na ulepszenie systemu rekomendacji, w tym lepsze przetwarzanie tekstu, zastosowanie bardziej zaawansowanych metod NLP i uwzględnienie feedbacku użytkowników. To pokazuje głębokie zrozumienie wyzwań i możliwości rozwoju w dziedzinie systemów rekomendacji.
- Propozycje ulepszenia modelu poprzez zaawansowane techniki uczenia maszynowego i głębokiego uczenia są wartościowe i mogą przyczynić się do znaczącego ulepszenia dokładności rekomendacji.

Podsumowując, ChatGPT-4 wykazał się umiejętnościami w zakresie analizy danych, przetwarzania języka naturalnego i budowy podstawowych systemów rekomendacji. Pomimo pewnych ograniczeń w trafności rekomendacji zbudowanego modelu, ChatGPT-4 dostarczył solidnych podstaw dla dalszego rozwoju i udoskonalenia systemu. Ulepszenia zaproponowane przez model wskazują na potencjał do znaczącego zwiększenia skuteczności rekomendacji przy dalszym rozwoju i iteracyjnym dostosowywaniu systemu.

## 4.4. Trenowanie agenta w środowisku KungFuMaster z pomocą ChatGPT-4

### 4.4.1. Konwersacja z ChatGPT-4

 Ja:

Witaj, Chat GPT. Zainteresowałem się środowiskiem OpenAI Gym o nazwie KungFuMaster i chciałbym stworzyć dla niego agenta AI. Mógłbyś mi wyjaśnić, jak pobrać to środowisko do Pythona i następnie jak rozpocząć proces uczenia agenta gry?



Rys. 7: Środowisko KungFuMaster

 ChatGPT:

Oczywiście, mogę Ci w tym pomóc. OpenAI Gym to biblioteka, która dostarcza szeroki zbiór środowisk testowych, które pomagają w rozwijaniu i porównywaniu algorytmów uczenia ze wzmocnieniem. Oto kroki, które musisz wykonać, aby rozpocząć pracę ze środowiskiem KungFuMaster w OpenAI Gym:

#### 1. Instalacja OpenAI Gym:

Musisz zainstalować pakiet OpenAI Gym. Możesz to zrobić za pomocą `pip`, narzędzia do instalacji pakietów dla Pythona. Uruchom następujące polecenie w terminalu lub wierszu poleceń:

```
pip install gym
```

## 2. Instalacja środowisk Atari:

Ponieważ KungFuMaster jest częścią środowisk Atari w OpenAI Gym, musisz również zainstalować `gym[atari]`. Możesz to zrobić następującym poleceniem:

```
pip install gym[atari]
```

## 3. Importowanie Gym i inicjalizacja środowiska:

Po zainstalowaniu pakietów możesz zaimportować Gym do swojego skryptu Pythona i zainicjować środowisko KungFuMaster:

```
import gym
env = gym.make('KungFuMaster-v0')
```

## 4. Rozpoczęcie procesu uczenia:

Aby rozpocząć proces uczenia, musisz zaimplementować algorytm uczenia ze wzmocnieniem, który będzie interaktywał ze środowiskiem. Oto bardzo prosty przykład, w którym agent podejmuje losowe działania:

```
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        action = env.action_space.sample() # Wybierz losowe działanie
        observation, reward, done, info = env.step(action)
        if done:
            print("Epizod zakończony po {} krokach".format(t+1))
            break
    env.close()
```

## 5. Implementacja algorytmu uczenia ze wzmocnieniem:

Proces uczenia wymaga bardziej skomplikowanego podejścia, takiego jak Q-learning, Deep Q Network (DQN), Policy Gradients, itp. Musisz zaimplementować taki algorytm, który będzie mógł nauczyć się strategii maksymalizującej nagrody w grze.

## 6. Trenowanie i ocena agenta:

Po zaimplementowaniu algorytmu będziesz musiał trenować swojego agenta na wielu epizodach gry, monitorując jego postępy i dostosowując parametry algorytmu w razie potrzeby.

To jest tylko bardzo ogólne wprowadzenie do procesu. Każdy z tych kroków wymaga głębszego zrozumienia szczegółów i potencjalnie znacznej ilości kodu, zwłaszcza jeśli chodzi o implementację i dostrojenie algorytmów uczenia ze wzmocnieniem. Jeśli potrzebujesz bardziej szczegółowych informacji na każdym z tych etapów, daj mi znać!



Ja:

Skorzystałem z twojego kodu i zostało wyświetlone środowisko w którym agent wykonywał losowe działania. Co muszę teraz zrobić aby wytrenować agenta na tym środowisku?



ChatGPT:

Wytrenowanie agenta w środowisku KungFuMaster z OpenAI Gym wymaga zaimplementowania algorytmu uczenia ze wzmocnieniem. Istnieje wiele różnych algorytmów, które możesz użyć, a ich wybór zależy od wielu czynników, takich jak złożoność środowiska, wymagania dotyczące wydajności, dostępne zasoby obliczeniowe itp. Oto kroki, które musisz wykonać, aby wytrenować agenta:

**1. Wybierz algorytm uczenia ze wzmocnieniem:**

- *Q-Learning*: Dla prostych problemów.
- *Deep Q-Network (DQN)*<sup>[25]</sup>: Łączy Q-learning z głębokimi sieciami neuronowymi.
- *Policy Gradients*<sup>[26]</sup>: Bezpośrednie uczenie polityk działania.
- *Actor-Critic*<sup>[27]</sup>: Kombinuje zalety Q-learningu i gradientów polityk.
- *Proximal Policy Optimization (PPO)* lub *Trust Region Policy Optimization (TRPO)*: Dla bardziej zaawansowanych zastosowań.

**2. Zaprojektuj sieć neuronową:**

Jeśli wybierzesz algorytm oparty na sieciach neuronowych, takie jak DQN lub PPO, będziesz musiał zbudować sieć neuronową, która będzie służyła jako aproksymator funkcji wartości lub polityki. W Pythonie zazwyczaj używa się do tego biblioteki TensorFlow lub PyTorch.

**3. Zbieraj doświadczenie:**

Interakcja z środowiskiem jest kluczowa. Agent musi wykonywać działania, obserwować nagrody i nowe stany, a następnie używać tych informacji do aktualizacji swojej polityki.

**4. Implementuj proces uczenia:**

Używając zebranych danych (stan, akcja, nagroda, nowy stan), musisz zaimplementować logikę aktualizacji dla wybranego algorytmu. W przypadku DQN będzie to aktualizacja sieci Q za pomocą mini-batchów z pamięci powtórek (experience replay), podczas gdy w przypadku metod gradientu polityki będzie to aktualizacja parametrów polityki w kierunku większej nagrody.

**5. Optymalizacja i testowanie:**

Podczas gdy agent się uczy, będziesz chciał monitorować jego wydajność i dostosowywać hiperparametry, takie jak wskaźnik uczenia się, rozmiar mini-batchy, czynnik dyskontowy (gamma) i inne. Następnie testuj swojego agenta, aby zobaczyć, jak dobrze radzi sobie na nowych instancjach środowiska.

**6. Debugowanie i dopracowywanie:**

Uczenie ze wzmocnieniem często wymaga wielu iteracji i dopracowywania. Możesz potrzebować debugować swojego agenta i dostosować podejście, jeśli nie osiąga oczekiwanej wydajności.

**7. Użyj odpowiednich narzędzi:**

Istnieje wiele bibliotek, które mogą ułatwić implementację algorytmów RL, takich jak Stable Baselines, RLlib, TensorForce, itp. Te biblioteki zapewniają gotowe do użycia implementacje standardowych algorytmów uczenia ze wzmocnieniem.

Oto przykładowy szkielet kodu, który pokazuje, jak można zainicjować proces uczenia (zakładając, że korzystamy z metody DQN):

```
import gym
from stable_baselines3 import DQN

# Inicjalizacja środowiska
env = gym.make('ALE/KungFuMaster-v5')

# Inicjalizacja agenta DQN
model = DQN('CnnPolicy', env, verbose=1)

# Trenowanie agenta
model.learn(total_timesteps=100000)

# Testowanie agenta
obs = env.reset()
for i in range(1000):
    action, _states = model.predict(obs, deterministic=True)
    obs, rewards, done, info = env.step(action)
    env.render()
    if done:
        break
env.close()
```

 Ja:

Podczas użycia twojego kodu do trenowania agenta pojawił się ten błąd.

**MemoryError: Unable to allocate 93.9 GiB for an array with shape (1000000, 1, 3, 210, 160) and data type uint8**

 ChatGPT:

Błąd `MemoryError` wskazuje, że proces próbuje zaalokować więcej pamięci RAM, niż jest dostępne w systemie. W kontekście uczenia ze wzmocnieniem i używania sieci DQN, taki błąd jest często spowodowany przez próbę załadowania zbyt wielkiej ilości danych do pamięci, np. w przypadku pamięci powtórek (*experience replay buffer*).

Oto kilka kroków, które możesz podjąć, aby rozwiązać ten problem:

1. **Zmniejsz rozmiar bufora pamięci powtórek:** W DQN, bufor pamięci powtórek przechowuje poprzednie przejścia, które są wykorzystywane do uczenia sieci. Zmniejszenie liczby przechowywanych przejść może pomóc zmniejszyć zużycie pamięci.
2. **Użyj mniejszych wsadów (*batch size*):** Podczas treningu sieci, każda aktualizacja jest wykonywana na podstawie próbki (*batch*) z bufora pamięci powtórek. Mniejsze wsady zmniejszą wymagania pamięciowe.
3. **Zmniejsz częstotliwość aktualizacji:** Możesz również zaktualizować sieć rzadziej, co może zmniejszyć ilość pamięci potrzebnej na jednorazowe przechowywanie danych do aktualizacji.
4. **Zmniejsz rozmiar lub złożoność modelu:** Użycie prostszej sieci neuronowej z mniejszą liczbą parametrów może zmniejszyć ilość pamięci potrzebną do przechowywania i aktualizacji sieci.

5. **Zmiana reprezentacji danych:** Jeśli używasz obrazów, możliwe, że przetwarzasz je w pełnej rozdzielczości. Możesz spróbować zmniejszyć rozdzielczość lub użyć innej metody przetwarzania wstępnego, aby zmniejszyć rozmiar danych wejściowych.
6. **Wykorzystanie pamięci dyskowej:** Jeśli żadne z powyższych rozwiązań nie jest możliwe, możesz skorzystać z technik takich jak pamięć wirtualna, przenosząc część danych z pamięci RAM na dysk twardy, co może być wolniejsze, ale pozwoli na przetwarzanie większych danych.
7. **Sprawdź wycieki pamięci:** Upewnij się, że nie masz wycieków pamięci w swoim kodzie, które mogłyby powodować, że z biegiem czasu zużycie pamięci stale rośnie.
8. **Upgrade sprzętu:** Jeśli żadne z powyższych rozwiązań nie jest wystarczające, możliwe, że będziesz musiał rozważyć ulepszenie swojego sprzętu komputerowego, aby miał więcej pamięci RAM.

Jeśli jesteś ograniczony sprzętowo, polecałbym rozpocząć od pierwszych czterech kroków, ponieważ są to zmiany, które możesz wprowadzić w kodzie bez potrzeby inwestowania w nowy sprzęt.

Oto jak można zmodyfikować rozmiar bufora i rozmiar partii w kodzie:

```
from stable_baselines3.common.buffer import ReplayBuffer

buffer_size = 10000 # Zmniejsz tę liczbę, jeśli zabraknie ci pamięci
batch_size = 32 # Zmniejsz tę liczbę, aby zmniejszyć zużycie pamięci na
partię

model = DQN('MlpPolicy', env, buffer_size=buffer_size, batch_size=batch_size,
verbose=1)
```

Pamiętaj, że zmniejszenie rozmiaru bufora i wielkości partii może wpłynąć na uczenie się agenta, więc musisz znaleźć równowagę między zużyciem pamięci a wydajnością algorytmu.

 Ja:

Po obejrzeniu gry wytrenowanego agenta stwierdzam, że nie radzi sobie z podstawowymi rzeczami takimi jak atakowanie czy bronienie. Agent robi wszystko, żeby przetrwać jak najdłużej, nie zabijając przeciwników. Jak mogę poprawić wyniki modelu? Jak mogę wyświetlić wyniki trenowania?

Mój aktualny kod to:

```
from stable_baselines3.common.buffer import ReplayBuffer

buffer_size = 10000 # Decrease this number if you're running out of memory
batch_size = 32 # Decrease this number to reduce memory consumption per batch

model = DQN('MlpPolicy', env, buffer_size=buffer_size, batch_size=batch_size,
verbose=1)

#Trenowanie agenta
model.learn(total_timesteps=10_000);
```



## ChatGPT:

Jeśli wytrenowany agent nie radzi sobie z podstawowymi elementami gry, jak atakowanie czy bronienie, może to oznaczać, że wymaga on dalszego trenowania, dostrojenia hiperparametrów, lub też że sama architektura modelu nie jest wystarczająco dobra do nauki odpowiednich strategii. Oto kilka sugestii, jak można poprawić wyniki modelu:

1. **Zwiększ liczbę kroków uczenia:** 10 000 kroków to bardzo mało dla skomplikowanego zadania, jakim jest gra w KungFuMaster. Zwiększenie `total_timesteps` do kilkuset tysięcy lub nawet milionów kroków może znacznie poprawić wyniki.
2. **Dostosuj hiperparametry:** Zmiana hiperparametrów takich jak współczynnik uczenia się (`learning rate`), współczynnik dyskontowania (`gamma`), rozmiar sieci neuronowej, czy polityka eksploracji może mieć duży wpływ na wydajność agenta.
3. **Dodaj nagrody kształtujące:** Możesz zmodyfikować funkcję nagrody, aby zachęcić agenta do pożądanых działań, takich jak atakowanie przeciwników. Nazywa się to "reward shaping".
4. **Użycie zaawansowanych algorytmów:** DQN jest dość podstawowym algorytmem, a środowiska takie jak KungFuMaster mogą wymagać zaawansowanych technik jak PPO, A3C, lub Rainbow.
5. **Wykorzystanie preprocesingu:** Przetwarzanie wstępne obserwacji, takie jak zmiana rozdzielczości, konwersja do skali szarości, wycinanie niepotrzebnych części obrazu itp., może pomóc w skupieniu się na istotnych elementach gry.
6. **Zmniejsz wariancję:** Użyj technik takich jak użycie większej liczby kroków aktualizacji polityki lub zastosowanie technik redukcji wariancji, jak Generalized Advantage Estimation (GAE).
7. **Analiza wyników:** Użyj narzędzi do wizualizacji, takich jak TensorBoard, aby analizować, jak agent uczy się w czasie. Może to dać wgląd w to, co może być poprawione.
8. **Regularizacja i normalizacja:** Zastosowanie technik regularizacji, takich jak dropout czy batch normalization, może pomóc w stabilizacji procesu uczenia.

Co do wyświetlania postępu treningu, możesz użyć TensorBoard, narzędzia do wizualizacji, które integruje się z TensorFlow (używany przez Stable Baselines). Aby zintegrować TensorBoard z Twoim środowiskiem, możesz dodać dodatkowy parametr do swojego kodu. Oto przykładowy sposób, jak to zrobić:

1. **Konfiguracja TensorBoard w Stable Baselines3:**  
W przypadku Stable Baselines3, możesz skonfigurować zapisywanie logów do TensorBoard bezpośrednio w konstruktorze modelu. Na przykład:

```
from stable_baselines3 import DQN

model = DQN('MlpPolicy', env, tensorboard_log="./tensorboard_logs/", verbose=1)
```

W tym przypadku, logi będą zapisywane w folderze `./tensorboard_logs/`.

## 2. **Trenowanie modelu:**

Podczas trenowania modelu, wszystkie istotne informacje (takie jak nagrody, strata, itp.) będą automatycznie zapisywane do określonego folderu logów.

```
model.learn(total_timesteps=10000, log_interval=4)
```

'log\_interval' określa, jak często informacje są logowane do TensorBoard.

## 3. **Uruchomienie TensorBoard:**

Uruchom TensorBoard, wskazując folder, w którym zapisywane są logi:

```
tensorboard --logdir=./tensorboard_logs/
```

## 4. **Analiza w TensorBoard:**

Po uruchomieniu TensorBoard, otwórz <http://localhost:6006> w przeglądarce, aby przejrzeć zapisane dane. Będziesz mógł śledzić różne metryki, takie jak nagrody, straty, i inne, w zależności od tego, co zapisuje Twój model RL.

Pamiętaj, że szczegółowość i rodzaj danych dostępnych w TensorBoard będą zależeć od konkretnego algorytmu i jego implementacji w używanej bibliotece. Stable Baselines3 zwykle oferuje dobre wsparcie dla integracji z TensorBoard, co pozwala na dość szczegółowe śledzenie postępów w uczeniu.



Ja:

Dziękuję za podany kod na pewno zastosuję w swojej pracy. Co do sugestii dotyczących poprawy wyników agenta, na pewno zwiększę liczbę wykonywanych kroków podczas uczenia, ale również podoba mi się pomysł dostosowania hiperparametrów. Czy wiesz może jak przeszukać przestrzeń hiperparametrów dla modelu ?



ChatGPT:

Przeszukiwanie przestrzeni hiperparametrów to kluczowy element w procesie strojenia modeli uczenia maszynowego, w tym algorytmów uczenia ze wzmocnieniem. Oto kilka podejść, które możesz wykorzystać:

1. **Ręczne dostosowywanie:** Najprostsze podejście, polegające na ręcznym eksperymentowaniu z różnymi wartościami hiperparametrów i obserwowaniu wyników. Jest to czasochłonne, ale może dać intuicję na temat działania poszczególnych hiperparametrów.
2. **Grid Search:** Systematyczne przeszukiwanie przez określony zestaw wartości dla każdego hiperparametru. Jest to proste, ale może być bardzo czasochłonne, szczególnie gdy liczba hiperparametrów jest duża.
3. **Random Search:** Losowe próbkowanie przestrzeni hiperparametrów. Często jest bardziej efektywne niż grid search, ponieważ nie wszystkie hiperparametry mają równy wpływ na wyniki.

4. **Bayesian Optimization:** Używa modeli probabilistycznych do przewidywania wyników uczenia na podstawie poprzednich prób i skupia się na obszarach, które najbardziej obiecują poprawę. Jest to bardziej zaawansowana metoda, ale często prowadzi do lepszych wyników w krótszym czasie.
5. **Automated Machine Learning (AutoML) narzędzia:** Istnieją narzędzia i biblioteki, takie jak Hyperopt, Optuna, Ray Tune, które mogą automatyzować proces poszukiwania hiperparametrów.
6. **Genetic Algorithms:** Używa strategii ewolucyjnych do iteracyjnego wybierania, łączenia i modyfikowania zestawu hiperparametrów, aby znaleźć optymalne rozwiązanie.

Oto przykład, jak można użyć Optuna do przeszukiwania przestrzeni hiperparametrów dla modelu DQN:

1. Zainstaluj 'Optuna':

```
pip install optuna
```

2. Przykładowy skrypt z Optuna:

```
import optuna
from stable_baselines3 import DQN
from stable_baselines3.common.env_util import make_atari_env
from stable_baselines3.common.vec_env import VecFrameStack

def optimize_agent(trial):
    """ Funkcja optymalizacji dla Optuna """
    #Definiowanie przestrzeni hiperparametrów
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-2, log=True)
    gamma = trial.suggest_float("gamma", 0.8, 0.9999)
    exploration_fraction = trial.suggest_float("exploration_fraction", 0.1,
0.5)
    exploration_final_eps = trial.suggest_float("exploration_final_eps", 0.01,
0.1)
    target_update_interval =
trial.suggest_categorical("target_update_interval", [100, 500, 1000])
    learning_starts = trial.suggest_categorical("learning_starts", [0, 1000,
5000])

    #Utworzenie środowiska i modelu
    env = make_atari_env('ALE/KungFuMaster-v5', n_envs=1)
    env = VecFrameStack(env, n_stack=4)
    model = DQN('CnnPolicy', env, buffer_size=10_000, batch_size=32,
        learning_rate=learning_rate,
        gamma=gamma,
        exploration_fraction=exploration_fraction,
        exploration_final_eps=exploration_final_eps,
        target_update_interval=target_update_interval,
        learning_starts=learning_starts, train_freq=train_freq,
        verbose=0)

    #Trenowanie modelu
    model.learn(total_timesteps=10000)

    #Ocena modelu
```

```

rewards = []
for _ in range(5):
    obs = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(obs)
        obs, reward, done, _ = env.step(action)
        total_reward += reward
    rewards.append(total_reward)

mean_reward = sum(rewards) / len(rewards)
return mean_reward

study = optuna.create_study(direction='maximize')
study.optimize(optimize_agent, n_trials=100)

print('Najlepsze hiperparametry:', study.best_trial.params)

```

W tym przykładzie używamy Optuna do iteracyjnego uruchamiania funkcji `optimize_agent`, która tworzy, trenuje i ocenia agenta DQN z różnymi zestawami hiperparametrów. `study.optimize` jest używane do przeprowadzenia określonej liczby prób (w tym przypadku 100). Na końcu wypisywane są najlepsze znalezione hiperparametry.

Pamiętaj, że przeszukiwanie hiperparametrów, szczególnie w środowiskach uczenia ze wzmocnieniem, może wymagać dużych zasobów obliczeniowych i czasu.



Ja:

Czy wiesz może jak zwizualizować wyniki wykorzystując zmienną `study`?



ChatGPT:

Optuna oferuje kilka narzędzi do wizualizacji wyników optymalizacji hiperparametrów. Oto kilka popularnych metod wizualizacji, które możesz wykorzystać, korzystając ze zmiennej `'study'`:

1. **Plot Optimization History:** Wizualizuje zmianę wartości obiektywnej w kolejnych próbach.

```
optuna.visualization.plot_optimization_history(study)
```

2. **Plot Parallel Coordinates:** Wyświetla relacje między hiperparametrami i wynikami finalowymi.

```
optuna.visualization.plot_parallel_coordinate(study)
```

3. **Plot Contour:** Wyświetla kontur interakcji między parametrami.

```
optuna.visualization.plot_contour(study)
```

4. **Plot Slice:** Wyświetla relacje między poszczególnymi hiperparametrami a wynikiem.

```
optuna.visualization.plot_slice(study)
```

5. **Plot Param Importances:** Wskazuje, jak ważne są poszczególne hiperparametry dla wyników optymalizacji.

```
optuna.visualization.plot_param_importances(study)
```

6. **Plot EDF (Empirical Distribution Function):** Wyświetla rozkład wartości obiektywnej wśród prób.

```
optuna.visualization.plot_edf(study)
```

Aby użyć tych funkcji, musisz mieć zainstalowaną bibliotekę `matplotlib` w swoim środowisku. Przedstawione metody wizualizacji pomogą Ci zrozumieć, jak poszczególne hiperparametry wpływają na wyniki modelu DQN i jakie kombinacje są najbardziej efektywne.

Pamiętaj, że każda z tych metod graficznych dostarcza różnych informacji, więc wybierz te, które najlepiej odpowiadają Twoim potrzebom w zakresie analizy i prezentacji wyników.

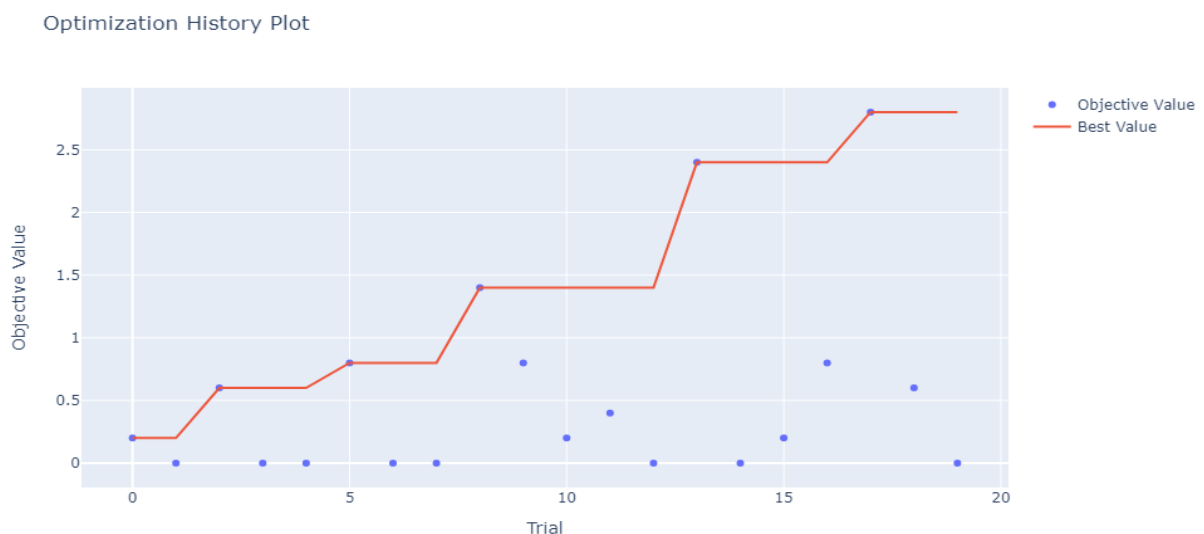


Ja:

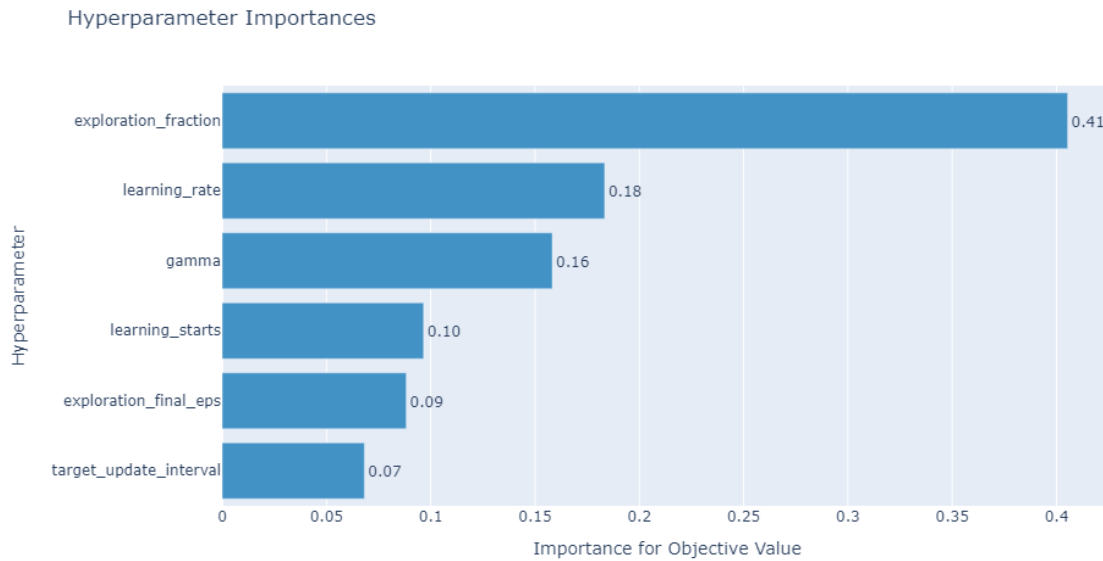
Wykorzystałem kod `optuna` do przeszukiwania hiperparametrów który mi zwrócił najlepsze hiperparametry dla mojego agenta.

Najlepsze hiperparametry: {'learning\_rate': 0.00019058466404356072, 'gamma': 0.8550473227193727, 'exploration\_fraction': 0.29674020789856287, 'exploration\_final\_eps': 0.06011559762977968, 'target\_update\_interval': 100, 'learning\_starts': 5000}

Dzięki twoim rozwiązaniom wykorzystałem też zmienną `study` i wyświetliłem wyniki tego przeszukiwania które podsyłam w załączniku.



Rys. 8: Wykres historii przeszukiwania najlepszych hiperparametrów dla modelu DQN



Rys. 9: Wykres ważności hiperparametrów dla modelu DQN

 ChatGPT:

Wspaniale, że udało Ci się wykorzystać Optuna do optymalizacji hiperparametrów Twojego agenta i stworzyć wizualizacje wyników. Wygląda na to, że masz już wyniki, które wskazują najlepsze znalezione hiperparametry:

- *learning\_rate*: 0.00019058466404356072
- *gamma*: 0.8550473227193727
- *exploration\_fraction*: 0.29674020789856287
- *exploration\_final\_eps*: 0.06011559762977968
- *target\_update\_interval*: 100
- *learning\_starts*: 5000

Te hiperparametry są rezultatem przeszukiwania i pokazują, które kombinacje parametrów przyniosły najlepsze wyniki podczas procesu uczenia. Teraz możesz użyć tych hiperparametrów do dalszego trenowania swojego agenta i zobaczyć, jak wpłyną one na jego wydajność w środowisku KungFuMaster.

Podsumowując wizualizacje:

1. **Optimization History Plot:** Pokazuje, jak wartość funkcji celu zmieniała się z każdym eksperymentem. Linia reprezentuje najlepsze znalezione rozwiązanie, a punkty oznaczają wyniki każdego pojedynczego eksperymentu. Wyraźnie widać, że wartość funkcji celu generalnie wzrosła wraz z liczbą prób.

2. **Hyperparameter Importances:** Wskazuje, które hiperparametry mają największy wpływ na wyniki. W twoim przypadku wydaje się, że `exploration_fraction` ma największy wpływ na wyniki, a `target_update_interval` najmniejszy.

Mając te informacje, możesz dalej dostosowywać swojego agenta. Jeśli chcesz dalej eksperymentować z hiperparametrami, możesz również rozważyć przeprowadzenie dodatkowych iteracji z Optuna, potencjalnie zmieniając zakres poszukiwanych wartości, aby zobaczyć, czy można uzyskać jeszcze lepsze wyniki.

Czy jest coś konkretnego, co chciałbyś teraz zrobić z tymi wynikami, lub czy masz jakieś pytania dotyczące dalszych kroków w Twoim projekcie?



Ja:

Dziękuję. Korzystając z kodów wcześniej przez ciebie dostarczonych wytrenowałem finalnego agenta. Jednakże tym razem wykorzystałem najlepsze hiperparametry znalezione przez bibliotekę optuna. Wykonałem to w taki sposób:

```
# Wyciągnięcie najlepszych parametrów z obiektu 'study'
best_params = study.best_trial.params

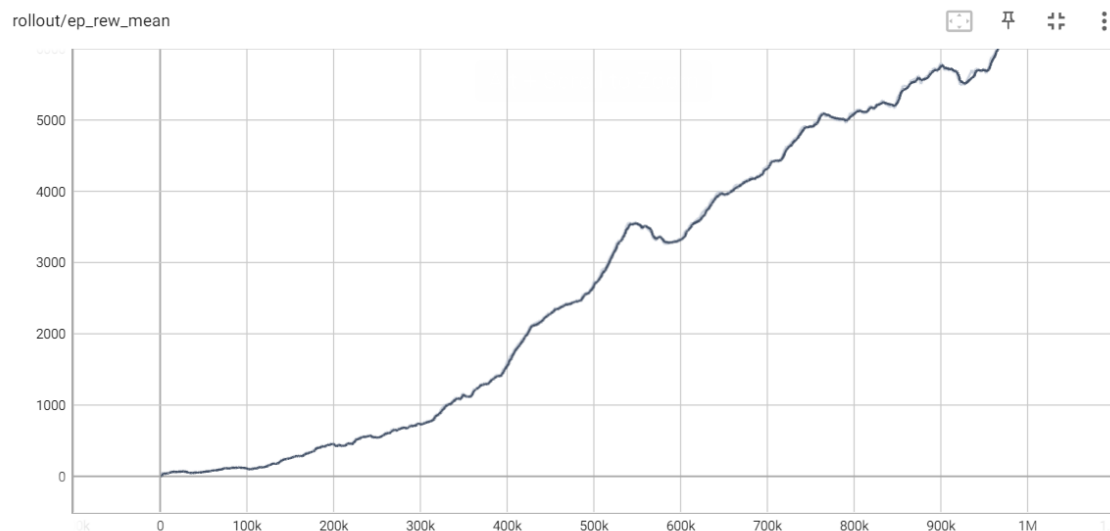
# Stworzenie środowiska, możesz użyć tego samego środowiska co wcześniej
env = make_atari_env('ALE/KungFuMaster-v5', n_envs=1)
env = VecFrameStack(env, n_stack=4)

# Tworzenie modelu z najlepszymi hiperparametrami
model = DQN('CnnPolicy', env, buffer_size=10_000, batch_size=32,
            **best_params,
            tensorboard_log="./tensorboard_logs/",
            verbose=1)

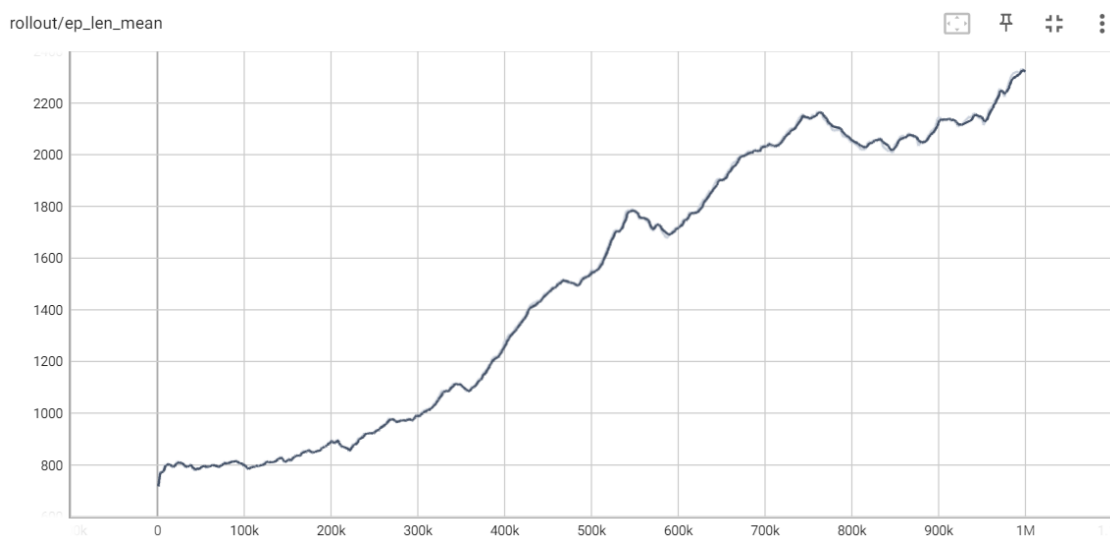
# Trenowanie modelu
model.learn(total_timesteps=1_000_000, log_interval=4);
```

Dzięki bibliotece TensorBoard uzyskałem wykresy, które pokazują osiągnięcia agenta w trakcie treningu. Wykresy te prześlę ci w załączniku abyś mógł je skomentować.

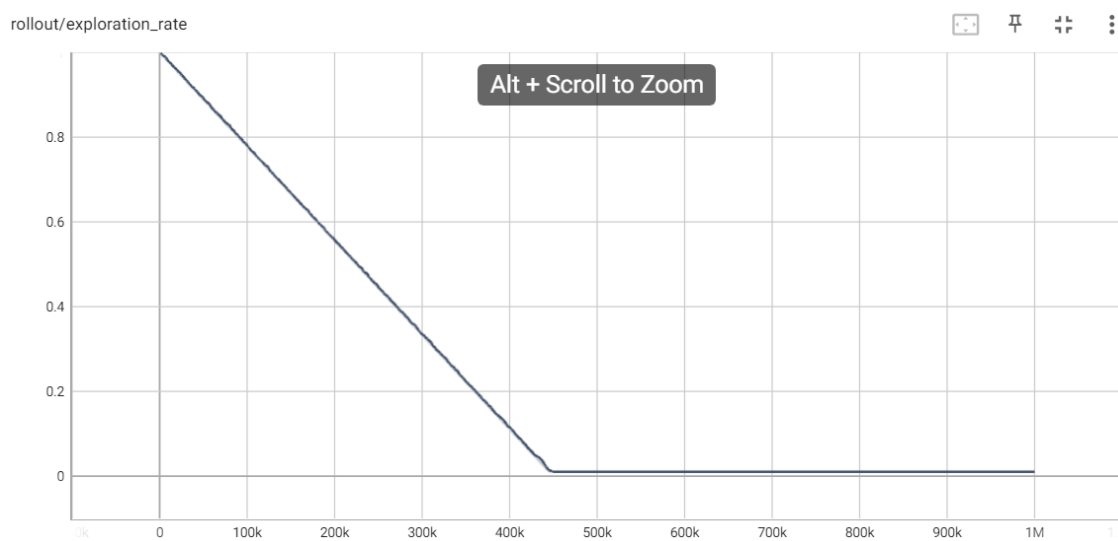
Co do moich spostrzeżeń/obserwacji gry wytrenowanego agenta, stwierdzam że radzi on sobie o wiele lepiej niż poprzednio wytrenowany model. Przeciwnicy specjalizujący się w walce bezpośredniej, którzy zbliżają się do gracza, są łatwo eliminowani przez model, a przeciwnicy rzucający szurikenami, szczególnie na poziomie głowy, są z łatwością unikani przez agenta. Natomiast szurikeny rzucone na poziomie nóg często trafiają w cel, co sugeruje, że agent nie nauczył się jeszcze efektywnie unikać ataków skierowanych na niższe partie ciała.



Rys. 10: Średnia nagroda agenta na epizod w trakcie treningu



Rys. 11: Średnia długość gry na epizod w treningu agenta



Rys. 12: Zmiana współczynnika eksploracji w czasie treningu





Gratuluję postępów z Twoim agentem! Na podstawie przesłanych wykresów TensorBoard, można zauważyć następujące informacje:

1. **Średnia nagroda za epizod (`ep_rew_mean`):** Wykres pokazuje stabilny wzrost średniej nagrody, co wskazuje na to, że Twój agent uczy się i poprawia swoje działania w środowisku. Regularność wzrostu świadczy o tym, że proces uczenia jest stabilny i skuteczny.
2. **Średnia długość epizodu (`ep_len_mean`):** Analogicznie do średniej nagrody, średnia długość epizodu również wzrasta, co wskazuje, że agent przetrwał dłużej w grze. Jest to kolejny dobry znak, sugerujący, że agent staje się lepszy w unikaniu końca gry (np. unikaniu przeciwników, trudnych sytuacji).
3. **Wskaźnik eksploracji (`exploration_rate`):** Ten wykres pokazuje, jak zmniejsza się eksploracja wraz z postępami w uczeniu. Początkowo wysoki wskaźnik eksploracji jest pożądany, aby agent mógł nauczyć się różnorodnych strategii. Z czasem jednak eksploracja maleje, co pozwala agentowi wykorzystywać nauczone już strategie do maksymalizacji nagród.

Twoje obserwacje dotyczące wydajności agenta są bardzo cenne i sugerują, że agent nauczył się niektórych kluczowych aspektów gry, takich jak walka bliska i unikanie. To, że agent nie unika skutecznie szurikenów rzuconych na poziomie nóg, może wskazywać na konieczność dalszego uczenia lub potrzebę wprowadzenia specyficznych nagród kształtujących, które zachęcałyby do unikania ataków nisko celowanych.

Możliwe, że dodatkowe dostosowanie funkcji nagrody lub dalsze trenowanie z większą liczbą epizodów mogłoby pomóc agentowi lepiej rozumieć i reagować na te niższe ataki. Można też rozważyć modyfikację architektury sieci neuronowej, aby lepiej przetwarzała informacje wizualne dotyczące dolnych regionów ekranu, gdzie pojawiają się te szurikeny.

Jeśli masz możliwość przekazywania bardziej szczegółowych informacji zwrotnych agentowi, może to także pomóc w nauce unikania ataków nisko celowanych. Inną strategią może być zastosowanie uczenia przez demonstrację (*learning from demonstration*), gdzie agent mógłby uczyć się z gry eksperta ludzkiego, który efektywnie unika takich ataków.

Jeśli masz jakieś dodatkowe pytania lub potrzebujesz dalszej pomocy, daj mi znać!

#### 4.4.2. Ocena skuteczności i dokładności ChatGPT-4

##### 1. Wsparcie w Konfiguracji i Inicjalizacji Środowiska:

- ChatGPT-4 skutecznie przekazał instrukcje dotyczące instalacji i konfiguracji środowiska OpenAI Gym.

##### 2. Wybór Algorytmów Uczenia ze Wzmocnieniem:

- Model językowy zaproponował różne algorytmy uczenia ze wzmocnieniem, takie jak Q-learning, DQN, Policy Gradients, co wskazuje na szerokie zrozumienie różnych metod dostępnych w uczeniu ze wzmocnieniem. Sugerowanie różnych algorytmów pomaga w dostosowaniu podejścia do specyfiki problemu i dostępnych zasobów.

##### 3. Optymalizacja i Dostosowywanie Hiperparametrów:

- ChatGPT-4 efektywnie wyjaśnił proces optymalizacji hiperparametrów, co jest kluczowe dla doskonalenia wydajności agenta AI. Sugerował użycia popularnej biblioteki Optuna dla przeszukiwania hiperparametrów oraz wyjaśnił, jak przeprowadzić ten proces.

##### 4. Analiza i Rozwiązywanie Problemów Technicznych:

- ChatGPT-4 skutecznie zdiagnozował i zaproponował rozwiązania dla problemów technicznych, takich jak błąd MemoryError, co było pomocne w pozbyciu się owego błędu.

##### 5. Wsparcie w Trenowaniu i Oceny Agentów:

- Model językowy skutecznie doradzał w procesie trenowania agenta, dając wskazówki dotyczące ilości kroków uczenia, dostosowywania hiperparametrów i debugowania agenta. Wsparcie w interpretacji wyników trenowania, analizie postępów i identyfikacji obszarów do poprawy było przydatne w doskonaleniu agenta.

##### 6. Interpretacja Wyników Treningu:

- ChatGPT-4 efektywnie zinterpretował wyniki treningu przedstawione w TensorBoard, oferując wgląd w wydajność agenta i sugerując możliwe obszary do dalszego rozwoju, takie jak unikanie ataków nisko wykonywanych przez przeciwników.

Podsumowując, ChatGPT-4 zapewnił skuteczne wsparcie w całym procesie projektu, od konfiguracji środowiska, przez wybór i implementację algorytmów, po optymalizację hiperparametrów i analizę wyników. Udzielone wskazówki były praktyczne i dostosowane do problemu, co pozwoliło na skuteczne zbudowanie i poprawę agenta AI w środowisku KungFuMaster.

## 5. Podsumowanie i wnioski

W niniejszej pracy inżynierskiej starałem się przetestować model językowy o nazwie ChatGPT-4 we wszystkich popularnych problemach uczenia maszynowego, takich jak uczenie nadzorowane, nienadzorowane i wzmacniane. Dla każdej kategorii wybrałem odpowiedni projekt, który nie powinien być za łatwy, ale też nie za trudny dla asystenta.

Dla uczenia nadzorowanego wybrałem 2 projekty. Budowę modelu klasyfikacyjnego, dla zdjęć kamieni szlachetnych, oraz budowę modelu regresyjnego dla przewidywania ceny używanych samochodów z Wielkiej Brytanii. W ramach uczenia nienadzorowanego skupiłem się na opracowaniu systemu rekomendacyjnego dla filmów Netflix. W kontekście uczenia wzmacnianego postanowiłem skorzystać z jednego ze środowisk dostępnych w popularnej bibliotece o nazwie Gym, znanej również jako Gymnasium. Dla ułatwienia zrozumienia problemu wybrałem środowisko KungFuMaster, w którym zadaniem agenta jest pokonanie przeciwników. Każdy z kolejnych projektów, ze względu na rosnącą trudność rozwiązania problemu, powinien stanowić coraz większe wyzwanie dla sztucznej inteligencji.

### Uczenie nadzorowane

#### Klasyfikacja

W problemie klasyfikacji zdjęć kamieni szlachetnych model językowy otrzymał tylko informacje dotyczące formatu danych i ścieżki, w której pliki się znajdują. To wystarczyło, aby ChatGPT-4 wygenerował działający skrypt do trenowania modelu klasyfikacyjnego. Dopytując go o możliwości wyświetlenia wyników treningu, otrzymałem kolejny działający skrypt do tego przeznaczony. Po ocenie wyników przekazanych przeze mnie, asystent poprawnie wymienił sposoby na poprawę trenowanego modelu, które następnie zastosował w nowo podanym kodzie. Pod koniec pomógł w ocenie i wyświetleniu wyników nowo wytrenowanego modelu, wykorzystując poprawnie raport klasyfikacji i macierz pomyłek.

#### Regresja

Kolejnym wyzwaniem dla ChatGPT-4 było stworzenie modelu, który będzie przewidywał cenę używanych samochodów z Wielkiej Brytanii. Różnica od poprzedniego zadania nie polegała tylko na rodzaju modelu, jaki musiał zbudować, ale też na danych, które otrzymał. Ze względu na to, iż ChatGPT-4 jest zdolny do przyjmowania danych w formie załączników w celu ich analizy i przeprowadzania badań, przesłałem mu je właśnie w ten sposób. Całą pracę od analizy przez czyszczenie i przetwarzanie danych do budowy prostego modelu regresji liniowej, model językowy zdołał wykonać samodzielnie we własnym środowisku, bez jakiegokolwiek pomocy z mojej strony, na końcu podając wyniki owego wytrenowanego modelu. Aby poprawić wyniki modelu regresyjnego, asystent zaproponował zastosowanie

bardziej zaawansowanych algorytmów, takich jak las losowy czy wzmocnienie gradientowe. Okazało się, że nie może on wytrenować tych modeli we własnej platformie programistycznej ze względu na ograniczenia pamięci, więc musiał przekazać skrypty mi, do wykonania jak w poprzednim projekcie klasyfikacji. Po wybraniu ostatecznego modelu regresyjnego, jakim był `RandomForestRegressor`, asystent zaproponował jeszcze dostrojenie hiperparametrów za pomocą `GridSearchCV`, co było dobrym pomysłem. Zakończyłem ten projekt, pozwalając ChatGPT-4 zinterpretować ostateczne wyniki, z czym w pełni się zgodziłem.

## Uczenie nienadzorowane

Następnym projektem do wykonania był system rekomendacji filmów Netflixa. Tu asystent także otrzymał dane w postaci załącznika, które również bardzo dobrze przeanalizował i przetworzył, z wykorzystaniem specjalnej metody `TfidfVectorizer` do zakodowania opisów filmów. Pierwszy program, jaki stworzył, polegał na znajdowaniu najbardziej podobnych filmów przy użyciu metryki podobieństwa kosinusowego. Jest to bardzo proste, ale i bardzo efektywne obliczeniowo rozwiązanie, a filmy, które były polecane przez system, były podobne do oryginału, jakim był 'Terminator Salvation'. Aby poprawić rekomendację filmów, sztuczna inteligencja zaproponowała wiele rzeczy, z czego zostało wybrane głębokie uczenie. Do wcześniej opisanego systemu asystent dołączył stworzony przez siebie skrypt modelu uczenia głębokiego, który zawierał zaawansowane techniki, takie jak osadzenia słów czy warstwy długiej, krótkotrwałej pamięci. Model uczenia głębokiego został wytrenowany we własnym środowisku użytkownika ze względu na ograniczenia ChatGPT-4. Do rekomendacji filmów zostały wykorzystane wyuczone osadzenia słów z modelu głębokiego. Uzyskane rekomendacje trochę odbiegały od oryginału i były bardziej zróżnicowane, co potwierdził asystent projektu, również sugerując możliwe ulepszenia.

## Uczenie wzmacniane

Ostatecznym i najbardziej wymagającym projektem było wytrenowanie agenta sztucznej inteligencji. Agent ten miał za zadanie nauczyć się, jak przetrwać/pokonać przeciwników w grze KungFuMaster. Zadanie to nie było proste, ponieważ jedynym pośrednikiem między postęпами agenta a modelem ChatGPT-4 byłem ja. Promptując ChatGPT-4 do tego zadania, otrzymałem od niego spis zadań od instalacji bibliotek przez wyświetlenie środowiska do ostatecznego trenowania agenta. Dzięki skryptom, które od niego otrzymałem, byłem w stanie bardzo szybko uruchomić środowisko z agentem wykonującym losowe akcje. Od razu przeszliśmy do trenowania agenta, w którym model językowy polecił mi kilka algorytmów uczenia wzmacnianego, podając mi skrypt wykorzystujący metodę DQN, którą zastosowałem. Po uruchomieniu kodu otrzymałem błąd, który został przez ChatGPT-4 wytłumaczony jako problem z alokacją pamięci RAM, podając mi różne rozwiązania od zmniejszenia parametrów modelu po 'upgrade sprzętu'. Wykorzystując poradę o zmniejszeniu wartości rozmiaru bufora, zdołałem wytrenować początkowego agenta, który nie radził sobie zbyt dobrze.

Tłumacząc asystentowi, że aktor wykonuje losowe ataki, podał on wiele sugestii, które mogą poprawić jego zachowanie. Pierwsza z nich była najbardziej trafna, a było to zwiększenie długości treningu, jednocześnie poinformowałem go, że podoba mi się opcja z dostosowaniem hiperparametrów modelu. Na to ChatGPT-4 wygenerował mi skrypt do tworzenia modeli z różnymi parametrami i sprawdzania jak sobie radzą wykorzystując świetną do tego bibliotekę Optuna. Ostatecznie wykorzystałem najlepsze parametry do stworzenia ostatecznego modelu, który uczył się milion kroków czasowych. Wynikowy aktor dobrze radzi sobie z przeciwnikami, choć widać, że jest jeszcze miejsce na poprawę, co sam stwierdził model językowy.

Wykorzystany w tej pracy ChatGPT-4 wykazał się jako wszechstronne narzędzie wspomagające w różnych projektach programowania uczenia maszynowego. Jego zdolność do generowania kodu, analizy danych, i interpretacji wyników okazała się niezwykle przydatna, przyczyniając się do sukcesu projektów. Model wykazał elastyczność w dostosowywaniu się do różnych wymagań projektowych, od klasyfikacji i regresji po uczenie nienadzorowane i uczenie ze wzmocnieniem. Mimo pewnych ograniczeń w specyficznych obszarach, ogólnie ChatGPT-4 stanowi cenne narzędzie wspierające proces rozwoju modeli uczenia maszynowego, dostarczając praktycznych porad, efektywnego kodu i głębokiego zrozumienia różnorodnych aspektów tej dziedziny.

Projekty zrealizowane z wykorzystaniem ChatGPT-4 ukazują znaczący potencjał modeli językowych w długoterminowym rozwoju uczenia maszynowego. Przewiduję, że takie narzędzia będą odgrywać coraz istotniejszą rolę, nie tylko w usprawnianiu istniejących metod, ale również w przełamywaniu barier innowacyjności i eksploracji nowych ścieżek w dziedzinie AI. ChatGPT-4 stanowi przykład zaawansowanego narzędzia AI, które może radykalnie zmienić procesy tworzenia i doskonalenia modeli uczenia maszynowego. Jego zdolność do generowania skutecznych rozwiązań i analiz wskazuje na możliwość kierunku, w którym modele językowe mogą nie tylko wspierać, ale i inicjować procesy 'rekurencyjnej auto-udoskonalenia' sztucznej inteligencji. Ta perspektywa otwiera drogę do scenariuszy, gdzie AI nie tylko asystuje w tworzeniu modeli uczenia maszynowego, ale staje się aktywnym twórcą, zdolnym do samodzielnego iteracyjnego rozwijania i ulepszania technologii AI.

## 6. Bibliografia

1. <https://chat.openai.com/>
2. [https://pl.wikipedia.org/wiki/Uczenie\\_maszynowe](https://pl.wikipedia.org/wiki/Uczenie_maszynowe)
3. [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)
4. [https://en.wikipedia.org/wiki/Transformer\\_\(machine\\_learning\\_model\)](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))
5. <https://kajodata.com/ai/co-to-jest-llm-duze-modele-jezykowe/>
6. <https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt>
7. <https://jupyter.org/>
8. <https://www.python.org/>
9. <https://www.nvidia.com/pl-pl/geforce/news/gfecnt/nvidia-geforce-rtx-2060/>
10. <https://www.kaggle.com/datasets/lsind18/gemstones-images>
11. <https://www.kaggle.com/datasets/muhammadawaistayyab/used-cars-prices-in-uk/>
12. <https://www.kaggle.com/datasets/shivamb/netflix-shows>
13. [https://www.gymnasium.dev/environments/atari/kung fu master/](https://www.gymnasium.dev/environments/atari/kung_fu_master/)
14. <https://pl.wikipedia.org/wiki/Augmentacja>
15. <https://mirosławmamczur.pl/jak-działają-konwolucyjne-sieci-neuronowe-cnn/>
16. [https://en.wikipedia.org/wiki/Transfer\\_learning](https://en.wikipedia.org/wiki/Transfer_learning)
17. [https://en.wikipedia.org/wiki/Confusion\\_matrix](https://en.wikipedia.org/wiki/Confusion_matrix)
18. <https://datascience.eu/pl/uczenie-maszynowe/inzynieria-cech/>
19. <https://blog.prokulski.science/2021/05/18/jak-stworzyc-system-rekomendacji-produktowych/>
20. [https://pl.wikipedia.org/wiki/Programowanie\\_neurolingwistyczne](https://pl.wikipedia.org/wiki/Programowanie_neurolingwistyczne)
21. <https://pl.wikipedia.org/wiki/TFIDF>
22. [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)
23. <https://pl.wikipedia.org/wiki/Lematyzacja>
24. [https://en.wikipedia.org/wiki/Word\\_embedding](https://en.wikipedia.org/wiki/Word_embedding)
25. <https://medium.com/data-science-in-your-pocket/deep-q-networks-dqn-explained-with-examples-and-codes-in-reinforcement-learning-928b97efa792>
26. <https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d>
27. <https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f>

Kielce, dn.....

.....  
imię i nazwisko studenta

.....  
numer albumu

.....  
kierunek, specjalność

.....  
rodzaj studiów, forma studiów

## OŚWIADCZENIE

Przedkładając w roku akademickim ..... pracę licencjacką/inżynierską/magisterską\*  
pod tytułem .....

.....  
oświadczam, że:

- pracę napisałam /napisałem\* samodzielnie,
- praca nie stanowi istotnego fragmentu lub innych elementów cudzego utworu,
- praca nie narusza żadnych innych istniejących praw autorskich,
- wykorzystane w pracy materiały źródłowe zastosowane zostały z zachowaniem zasad prawa cytatu
- wersja elektroniczna (na nośniku elektronicznym i/lub w systemie Wirtualna Uczelnia) pracy jest tożsama z wersją drukowaną.

Równocześnie oświadczam, że jestem świadoma/świadomy\*, iż na podstawie art. 15a ustawy z 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 24 poz. 83 ze zm.) Uniwersytetowi Jana Kochanowskiego w Kielcach przysługuje pierwszeństwo w opublikowaniu mojej pracy licencjackiej/magisterskiej\* w terminie 6 miesięcy od daty jej obrony.

W przypadku nie skorzystania przez Uniwersytet Jana Kochanowskiego w Kielcach z prawa pierwszeństwa publikacji wyrażam zgodę/ nie wyrażam zgody\* na udostępnianie mojej pracy licencjackiej/ magisterskiej\* przez Uniwersytet Jana Kochanowskiego w Kielcach dla celów naukowych i dydaktycznych.

Prawdziwość powyższego oświadczenia potwierdzam własnoręcznym podpisem .

czytelny podpis studenta

\*niewłaściwe skreślić