Politechnika Wrocławska

Projekt - Podstawy sieci neuronowych

Temat: Klasyfikacja binarna rezygnacji klientów za pomocą wielowarstwowej sieci neuronowej MLP

Grupa numer: 3

Poniedziałek 17:05 – 18:45

Wykonał:

Michał Białek (264285)

Termin oddania projektu: 19.01.2025

Wprowadzenie do projektu

Celem projektu jest zaprojektowanie, implementacja i wytrenowanie sztucznej sieci neuronowej do binarnej klasyfikacji rezygnacji klientów banku na podstawie dostępnego zestawu danych z platformy Kaggle.com. Projekt obejmuje pełny proces analizy danych, przygotowania modelu, dobór odpowiednich parametrów, i ich optymalizacja, oraz ewaluacji wyników badań

Analiza problemu

Współczesne banki borykają się z problemem rezygnacji klientów z kont bankowych i jednoczesną rezygnacją z usług bankowych (eng. clients churn). Taka sytuacja jest niekorzystana dla banku, a nawet w skrajnym przypadku może skutkować upadłością. Rozwiązaniem tego problemu jest identyfikacja klientów, u których występuje wysokie prawdopodobieństwo rezygnacji z usług, oraz wdrożenie działań zapobiegawczych, takich jak spersonalizowane oferty, promocje, czy lepsza obsługa klienta.

Cel projektu

Celem poniższego projektu jest zbudowanie na bazie dostępnych klienckich, modelu klasyfikacyjnego, który przewiduje rezygnację klienta z usług bankowych.

Analiza danych wejściowych

Poniższy model bazuje na pliku Churn Modelling z strony Kagge.com – link

Dane wejściowe są w formacie CSV, i zawierają 14 kolumn, które można skategoryzować na podstawie typu danych, oraz ich użyteczności w modelu:

Klasyfikacja na podstawie przydatności danych

Z wszystkich 14 kolumn, jedynie 3 kolumny (RowNumber,CustomerId,Surname), nie przenoszą żadnych wartości potrzebnych do analizy odejść klientów, a pełnią jedynie rolę identyfikacyjną.

Klasyfikacja na podstawie typu danych

Z podanych 11 kolumn można zaklasyfikować na podstawie typu przechowywanych danych:

6 kolumn o wartościach numerycznych: CreditScore (zdolność kredytowa) , Age (wiek posiadacza konta), Tenure (czas posiadania konta przez klienta), Balance (aktualne saldo na koncie bankowym) , NumberOfProducts (ilość produktów bankowych) , EstimatedSalary (Szacowana wartość wpłat).

2 kolumny o wartościach kategorycznych, które zostaną przekształcone na wartości binarne, lepsze do analizy, metodą One-Hot Encoding. Są to Geography (dane o kraju pochodzenia posiadacza konta), oraz Gender (płeć)

2 kolumny o wartości binarnej – HasCreditCard (posiadanie karty kredytowej), IsActiveMember (flaga oznajmiająca, czy podany użytkownik jest aktywny)

1 kolumna - zmienna docelowa – Exited (zawiera informację, czy klient opuścił bank)

Analizy problemu i wyboru odpowiedniej architektury sieci neuronowej

Do problemu klasyfikacji binarnej, tego rodzaju najlepszym wyborem jest sieć wielowarstwowa (MLP - Multi-Layer Perceptron) z następujących powodów:

Sieć MLP obsługuje różnorodne dane wejściowe, od numerycznych, jak i również kategoryczne (po przekształceniu przez One-Hot Encoding)

Zdolność do modelowania nieliniowych zależności miedzy cechami danych wejściowych modelu. Wiele cech, takich jak CreditScore, Balance, czy IsActiveMember, może mieć nieliniowy wpływ na odejście klienta. MLP dzięki swoim nieliniowym funkcjom aktywacji (np. ReLU, Sigmoid) jest w stanie uchwycić te zależności.

Jedną z kluczowych cech jest również możliwość dostosowywania ilości neuronów w poszczególnych warstwach, jak i również samą ilość poszczególnych warstw ukrytych.

Opis struktury wielowarstwowej sieci MLP

Sieć MLP składa się z 3 warstw:

1. Warstwa wejściowa - przyjmuje dane wejściowe w postaci wektora cech. Liczba neuronów w tej warstwie jest równa liczbie cech w danych wejściowych.
2. Warstwa ukryta - jest to jedna lub więcej warstw między warstwą wejściową a wyjściową. Każdy neuron w warstwie ukrytej oblicza liniową kombinację wag i przesunięć (biasów) na swoich wejściach, a wynik przepuszczany jest przez funkcję aktywacji (np. ReLU, sigmoid, tanh).

$$Wzór\ dla\ pojedynczego\ neuronu:$$

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

$$a = f(z)$$

$$gdzie:$$

$$f\ -\ funkcja\ aktywacji.$$

$Wagi(w_1, w_2 \ldots.)$ – są to wagi, które określają, jak ważne są poszczególne wejścia w procesie obliczania wyjścia. Wagi są dostosowywane podczas procesu uczenia się.

$Wejścia\ (w_1x_1 + w_2x\ \ )$ – reprezentują dane wejściowe, które mogą pochodzić z poprzedniej warstwy sieci lub bezpośrednio z danych treningowych

$Bias\ (b)$ - to stała wartość dodawana do obliczeń, która pozwala neuronowi lepiej dopasowywać się do danych nawet wtedy, gdy wszystkie wejścia są równe 0. Bias umożliwia translację funkcji aktywacji.

$Suma\ ważona\ (z)$ - To wynik sumowania iloczynów wag i odpowiadających im wejść oraz biasu.

Oraz oznaczenia takie jak:

Wyjście neuronu (a) - To wynik działania funkcji aktywacji $f(z)$ na sumę ważoną (z). Funkcja aktywacji wprowadza nieliniowość do modelu, co pozwala sieci neuronowej rozwiązywać złożone problemy.

Funkcja aktywacji $f(z)$- może to być np. funkcja sigmoidalna, ReLU (Rectified Linear Unit) lub inna funkcja nieliniowa, która przekształca wartość z w wartość wyjściową a.

3. **Warstwa wyjściowa - Generuje wyniki na podstawie obliczeń w warstwach ukrytych.**

Do warstwy wejściowej podawane są dane, i następuje proces uczenia:

- Sieć dostosowuje swoje wagi i biasy na podstawie błędu przewidywań w procesie propagacji wstecznej (backpropagation) oraz algorytmu optymalizacji (np. SGD, Adam).
- Celem jest minimalizacja funkcji kosztu cross-entropy dla klasyfikacji

Rodzaj i dobór parametrów sieci, oraz ich wpływ na działanie sieci

Podczas tworzenia sieci neuronowych musimy dobrać odpowiednie parametry sieci, w celu uzyskania jak najlepszych wyników działania.

Do dyspozycji mamy następujące parametry podstawowe:

1. Liczba warstw ukrytych
2. Liczba neuronów w warstwach
3. Funkcja aktywacji

Oraz następujące parametry optymalizacyjne:

1. Szybkość uczenia (Learning rate)- nieska wartość zapewnia stabilne, ale wolne uczenie. Z drugiej strony duża wartość może przyśpieszyć proces uczenia, ale zwięksa ryzyko braku zbieżności.
2. Batch size - Mały batch size prowadzi do szybszej aktualizacji wag, ale z większym szumem. Duży zapewnia bardziej stabilne uczenie, ale jest wolniejszy i wymaga więcej pamięci.
3. Liczba epok - Zbyt mała liczba epok prowadzi do niedouczenia, natomiast zbyt duża liczba epok może prowadzić do przetrenowania.

Ważnym aspektem jest również rekuraryzacja, czyli zapobieganie przeuczeniu. Do rozwiązania problemu przeuczenia, stosujemy techniki takie jak:

Dropout – czyli wyłączanie losowe 20-50% neuronów w warstwach ukrytych

Regularyzacja wag – czyli karanie modelu, za dawanie dużych wartości wag

Podczas trenowania modelu jest również możliwość monitorowania działania procesu uczenia, oraz bieżące reagowanie na zmiany parametrów modelu. Przykładem takiego mechanizmu może być early stopping, czyli mechanizm zatrzymujący trening, w przypadku wykrycia braku poprawy metryk

Implementacja modelu sieci MLP, oraz obserwacyjny dobór optymalnych parametrów

Poniżej przedstawiam domyślną implementację sieci neuronowej wykonanej w języku Python, z wykorzystaniem następujący bibliotek:

1. Numpy - Biblioteka do obliczeń numerycznych. Wykorzystujemy funkcję seed, do ustawienia ziarna losowości.
2. Pandas – bibliteoka do analizy i manipulacji danymi. W naszym przypadku wykorzystujemy do odczytywania danych z pliku csv, manipulowanie danymi, oraz do kodowania zmiennych kategorycznych metodą one-hot encoding
3. Seaborn – bibliteka do wizualizacji danych (Wizualizacja macierzy konfuzji w formie mapy cieplnej)
4. Matplotlib – wizualizacja danych
5. Scikit-learn - Biblioteka do uczenia maszynowego, zawiera narzędzia do preprocessingu danych, podziału zbiorów danych, trenowania modeli i ewaluacji. Dzięki tej bibliotece dzielimy dane na zbiory treningowe, walidacyjne i testowe.
6. TensorFlow – tworzenie i trenowanie sieci neuronowych. Używamy w niej funkcji do tworzenia i definowania parametrów sieci neuronowej.

Pierwotna implementacja posiada następujące parametry:

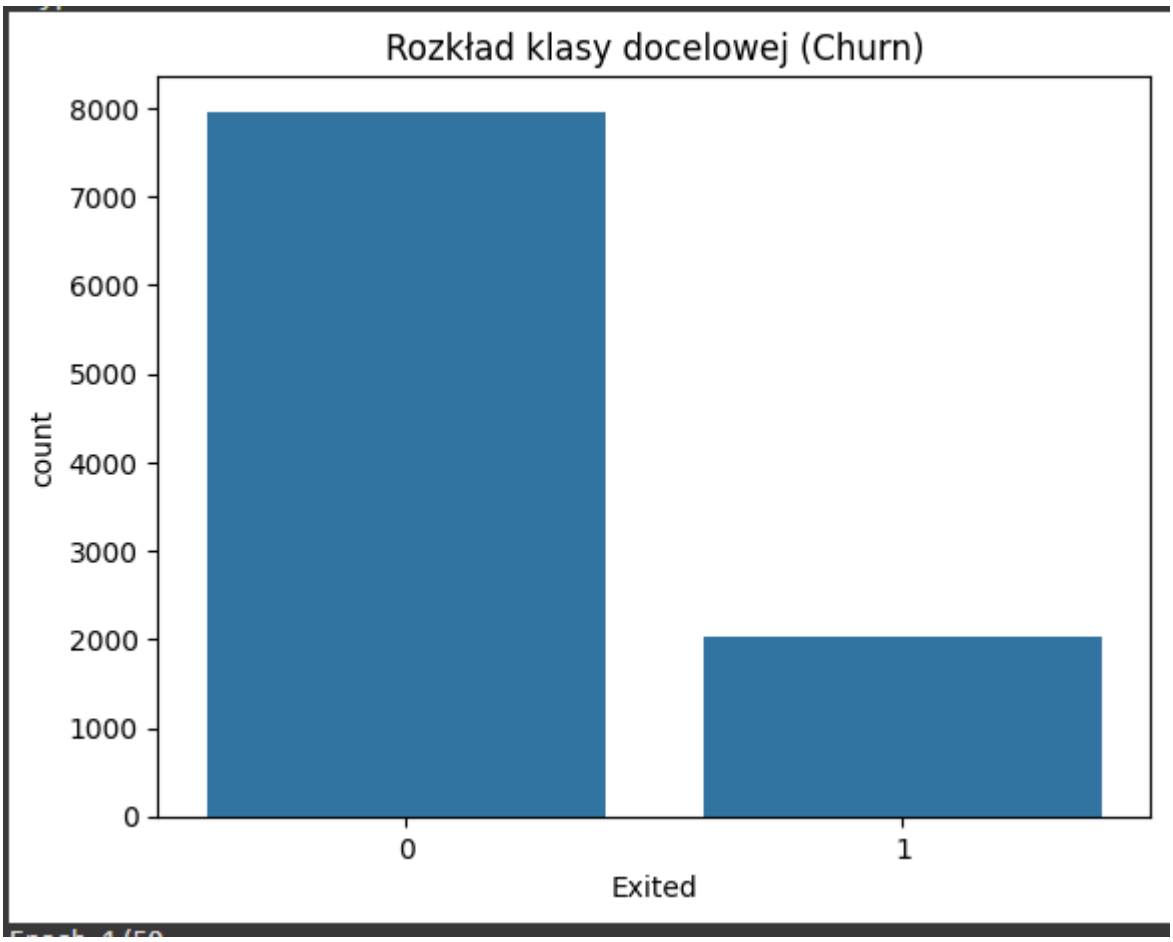
- 3 warstwy ukryte, o ilości neuronów odpowiednio 128,64,32 neurony, wraz z funkcją aktywacji ReLU.
- Wartość Drouput ustawioną na 0,5 oraz Regularyzacji Lambda = 0.001
- Wartość wyjściowa to neuron z aktywacją sigmoid
- Dodatkowo zastosowano mechanizm, binary_crossentropy, który nagradza i karze za dobre/złe predykcje

Podstawowa, jak i również różne implementacje podanej sieci neuronowe jest dostępna na repozytorium github:

https://github.com/MichalBialek01/neural-networks-churn-detection

Kroki opisujące implementację bazową:

1. Wczytanie pliku „Churn_Modelling.csv” do obiektu typu DataFrame
2. Wykonujemy podstawową walidację danych, czyli wyświetlamy informację na temat sieci neuronowej, oraz sprawdzamy, czy pobrany dataset posiada wartości puste. W przypadku, gdyby wystąpiły, potrzebna byłaby walidacja danych i odrzucanie przypadków, gdzie dane byłby niepoprawne. W tym kroku wykonujemy również obserwacje zbilansowania wartości Exited poprez sns.countplot. W przypadku, gdy wartości słupków są na podobnym poziomie, oznacza to, że dane są zbilansowane. W przypadku danych nie zbilansowanych, należałoby dane poddać preprocesowaniu
   W naszym przypadku mamy niezbilansowany rozkład:



W tym przypadku, z tego powodu idzie zauważyć dysproporcje. Z tego powodu należy dodać technikę nadpróbkowania (oversampling) dla klasy mniejszościowej (dla nas 1) na zbiorze treningowym. Pozwala to zwiększyć sztucznie ilość próbek. Do tego można zastosować RandomOverSampler z imblearn.over_sampling

3. Usunięcie niepotrzebnych kolumn ('RowNumber', 'CustomerId', 'Surname')
4. Kodowanie zmiennych kategorycznych

Kolumnę Gender mapujemy, na wartości binarne, żeby w przypadku wartości Male, wynikiem była wartość 1, oraz gdy wartość jest równa Female wartość wynosiła 0.
Wykonujemy również One-Hot Encoding kolumny Geography, przy pomocy biblioteki pandas

5. Dokonujemy podziału na cechy X (dane wejściowe) , oraz etykiety Y (zmienna docelowa)
6. Dzielimy zbiory na zbiór: treningowy, walidacyjny i testowy w proporcjach: 70% dane treningowe, 15% dane walidacyjne oraz 15% dane testowe
7. Dokonujemy standaryzacji danych.

   Standaryzacja danych jest stosowana, aby przyspieszyć proces uczenia i poprawić jego stabilność. Ujednolica ona skalę cech (średnia 0, odchylenie standardowe 1), dzięki czemu żadna z nich nie dominuje nad innymi. Dodatkowo zapobiega problemom z gradientami (eksplozją lub zanikiem) oraz poprawia działanie funkcji aktywacji, takich jak sigmoid czy ReLU. Dzięki standaryzacji algorytm uczy się szybciej i dokładniej.

   W kodzie wykorzystujemy StandardScaler do standaryzacji.

8. Tworzymy uproszczony model sieci neuronowej, o określonych parametrach: W podstawowym przypadku 3 warstwy o liczbie neuronów 128,64,32. Podajemy liczbę cech wejściowych, wartość regularayzacji L2, rodzaj funkcji aktywacji, oraz dopout – wszystkie te parametry, dla każdej warstwy. Ostatnim krokiem jest dodanie warstwy ostatniej z jednym neuronem z aktywacją signoid.
9. Kompilujemy model, podając optymalizator Adam, z ustalonym learing_rate na wartość 0.001, oraz binary_crossentropy.
10. Trenowanie modelu – w naszym przypadku stosujemy mechanizm nadzorowania EarlyStopping, który śledzi wynik błędu na zbiorze walidacyjnym, z ustawioną wartością patience=10, co oznacza, że jeżeli wynik walidacyjny nie ulegnie poprawie, trenowanie zatrzyma się. Trenowanie odbywa się poprzez model.fix, w której podajemy parametry takie jak: liczba epok=50, rozmiar paczki danych na jedną iterację -32
11. Następuje trenowanie modelu

Ewaluacja modelu

Po procesie trenowania modelu liczymy metryki takie jak:

1. **test_loss** – finalna wartość funkcji kosztu na zbiorze testowym.
2. **test_accuracy** – końcowa dokładność na zbiorze testowym.

3. **roc_auc_score** – miara AUC-ROC (Area Under Curve – Receiver Operating Characteristic). Im bliżej 1, tym lepszy model.
4. **f1_score** – łączy precyzję i czułość w jedną liczbę. Szczególnie użyteczne w przypadku niezbalansowanych danych.

Ostatnim krokiem, który pozwoli nam zwizualizować działanie sieci jest macierz konfuzji. Jest to macierz 2x2, która służy do oceny wyników klasyfikacji. Każdy element macierzy ma odpowiednie znaczenie:

True Positives (TP): Liczba przypadków, w których model poprawnie przewidział klasę pozytywną (np. przewidziano 1, a rzeczywista etykieta to 1)

True Negatives (TN): Liczba przypadków, w których model poprawnie przewidział klasę negatywną (np. przewidziano 0, a rzeczywista etykieta to 0).

False Positives (FP): Liczba przypadków, w których model błędnie przewidział klasę pozytywną (np. przewidziano 1, ale rzeczywista etykieta to 0).

False Negatives (FN): Liczba przypadków, w których model błędnie przewidział klasę negatywną (np. przewidziano 0, ale rzeczywista etykieta to 1).

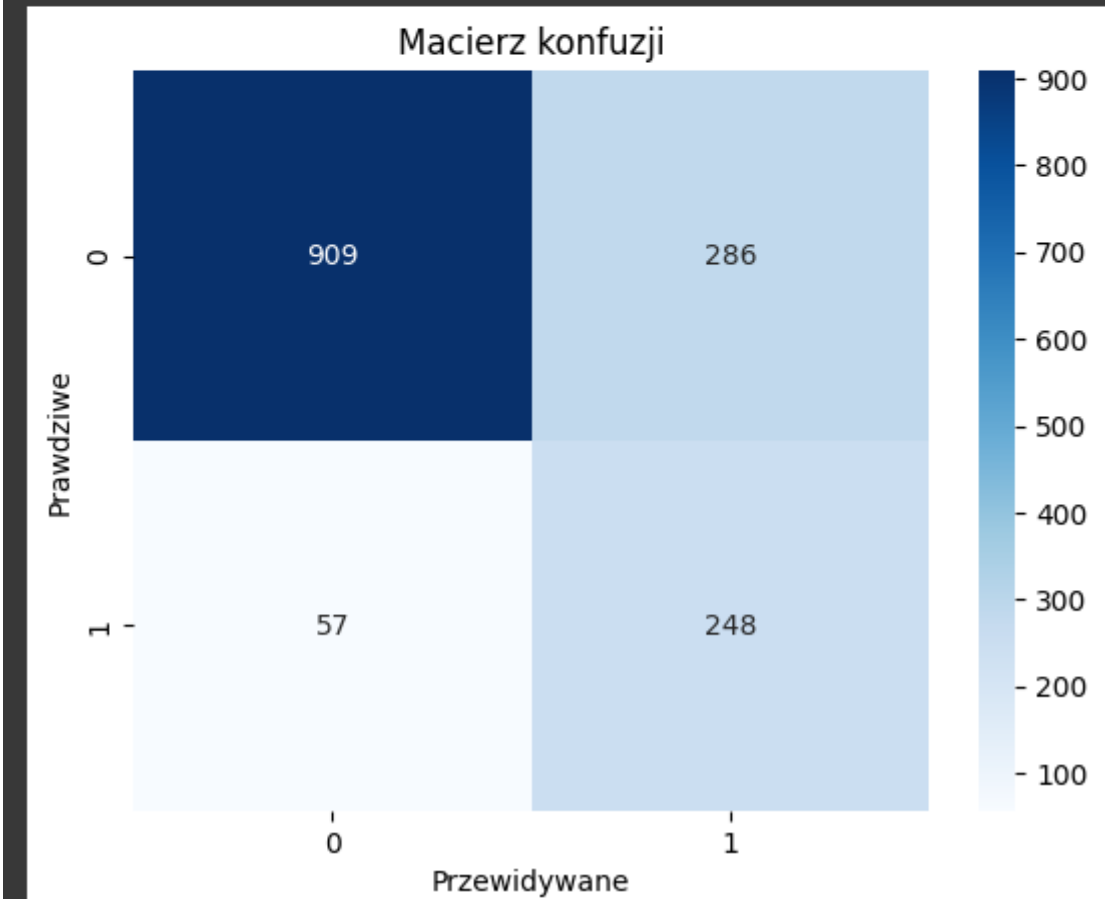
W naszym przypadku bazowym, wyniki wychodzą następujące:

```
Test Loss: 0.4947
Test Accuracy: 0.7713
AUC-ROC: 0.8696
F1 Score: 0.5912

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.76      0.84      1195
           1       0.46      0.81      0.59       305

    accuracy                           0.77      1500
   macro avg       0.70      0.79      0.72      1500
weighted avg       0.84      0.77      0.79      1500
```



Testy w zależności od parametrów:

Parametry, które warto zmienić, aby zbadać zachowanie modelu, w zależności od parametru, włączenia/wyłączenia optymalizatora/nadzorowania.

- **Liczba warstw**: od 1 do 3 (lub nawet 4).
- **Liczba neuronów**: 16, 32, 64, 128 (ewentualnie 256 dla dużych sieci).
- **Funkcje aktywacji**: ReLU, tanh, softplus.
- **Optymalizator**: Adam (z domyślnymi parametrami) oraz np. SGD (z momentum).
- **Liczba epok**: wybrane poziomy (np. 20, 50, 100).
- **Batch_size**: 32, 64 (opcjonalnie 16, 128).
- **EarlyStopping**: włączone / wyłączone.

Test1:

- 1 warstwa ukryta (32 neurony)

- Aktywacja: ReLU

- Optymalizator: Adam

- Liczba epok: 20

- batch_size: 32

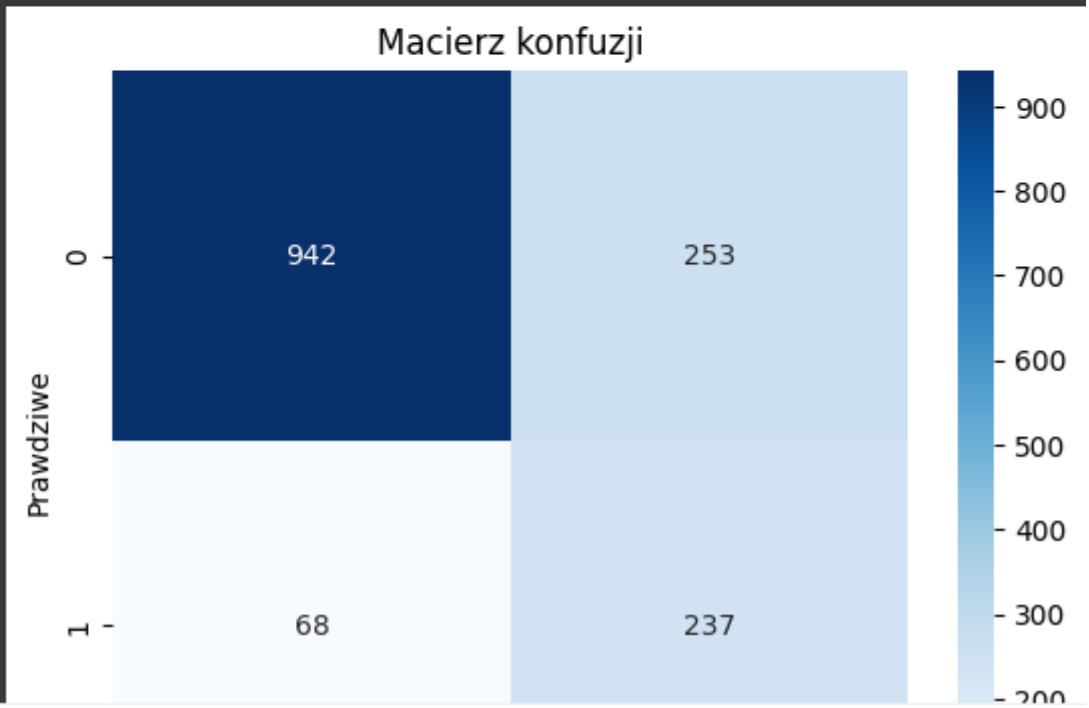
- EarlyStopping: wył

```
Test Loss: 0.4911293685436249
Test Accuracy: 0.7860000133514404
AUC-ROC: 0.8628822278619933
F1 Score: 0.5962264150943396

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      1195
           1       0.48      0.78      0.60       305

    accuracy                           0.79      1500
   macro avg       0.71      0.78      0.73      1500
weighted avg       0.84      0.79      0.80      1500
```



Przypadek 2:

- 1 warstwa ukryta (32 neurony)

- Aktywacja: ReLU

- Optymalizator: Adam

- Liczba epok: 50

- batch_size: 32

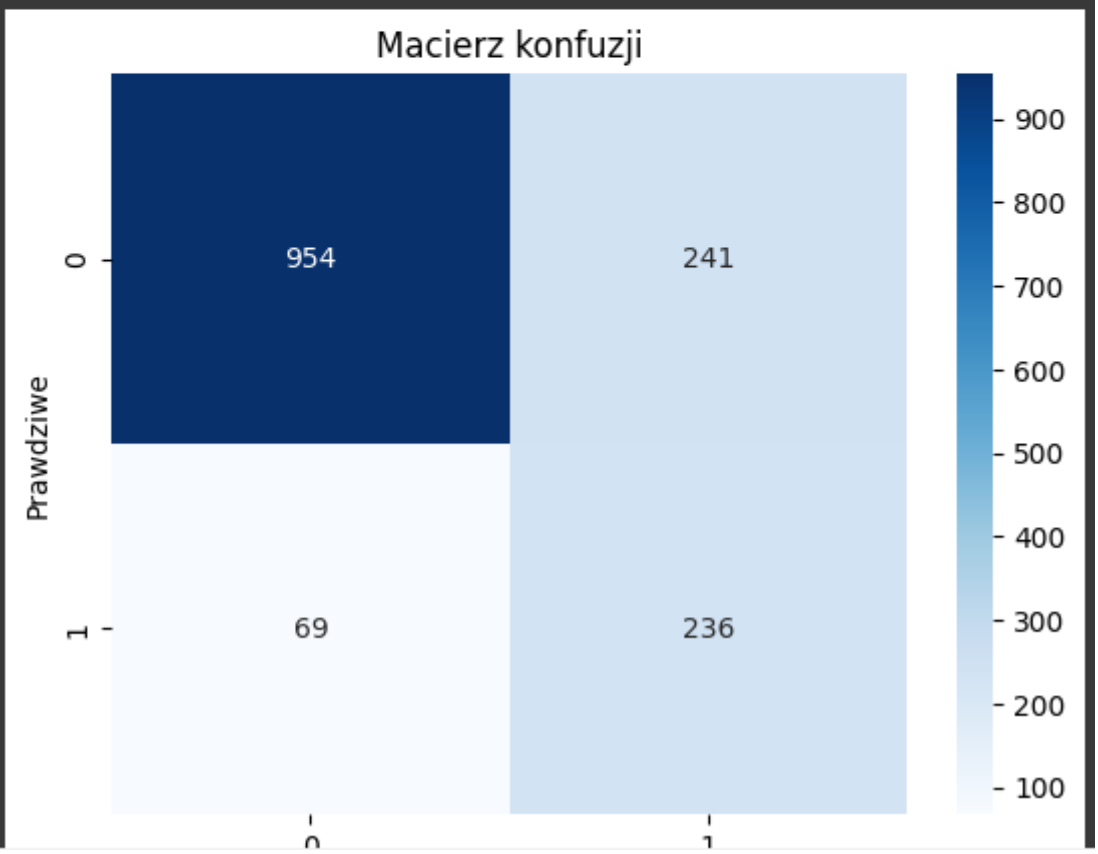
- EarlyStopping: włączone

```
Test Loss: 0.46661239862442017
Test Accuracy: 0.7933333516120911
AUC-ROC: 0.8677220659853214
F1 Score: 0.6035805626598465

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.93      0.80      0.86      1195
           1       0.49      0.77      0.60       305

    accuracy                           0.79      1500
   macro avg       0.71      0.79      0.73      1500
weighted avg       0.84      0.79      0.81      1500
```



"""

Przypadek 3:

- 2 warstwy ukryte: (64, 32)

- Aktywacja: ReLU

- Optymalizator: Adam

- Liczba epok: 50

- batch_size: 32

- EarlyStopping: włączone

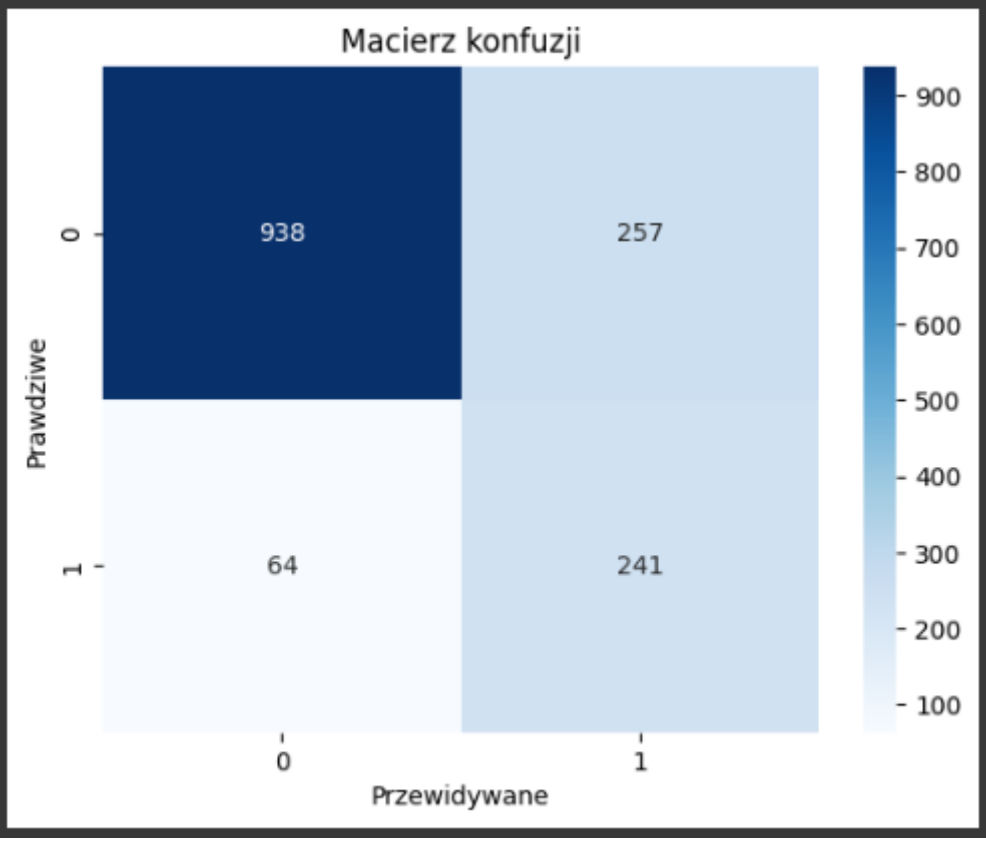
"""

```
Test Loss: 0.4690035879611969
Test Accuracy: 0.7860000133514404
AUC-ROC: 0.869494478359284
F1 Score: 0.6002490660024906

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.78      0.85      1195
           1       0.48      0.79      0.60       305

    accuracy                           0.79      1500
   macro avg       0.71      0.79      0.73      1500
weighted avg       0.84      0.79      0.80      1500
```



Przypadek 4:

"""

Przypadek 4:

- 2 warstwy (64, 32)

- Aktywacja: ReLU

- Optymalizator: SGD (momentum=0.9)

- Liczba epok: 50

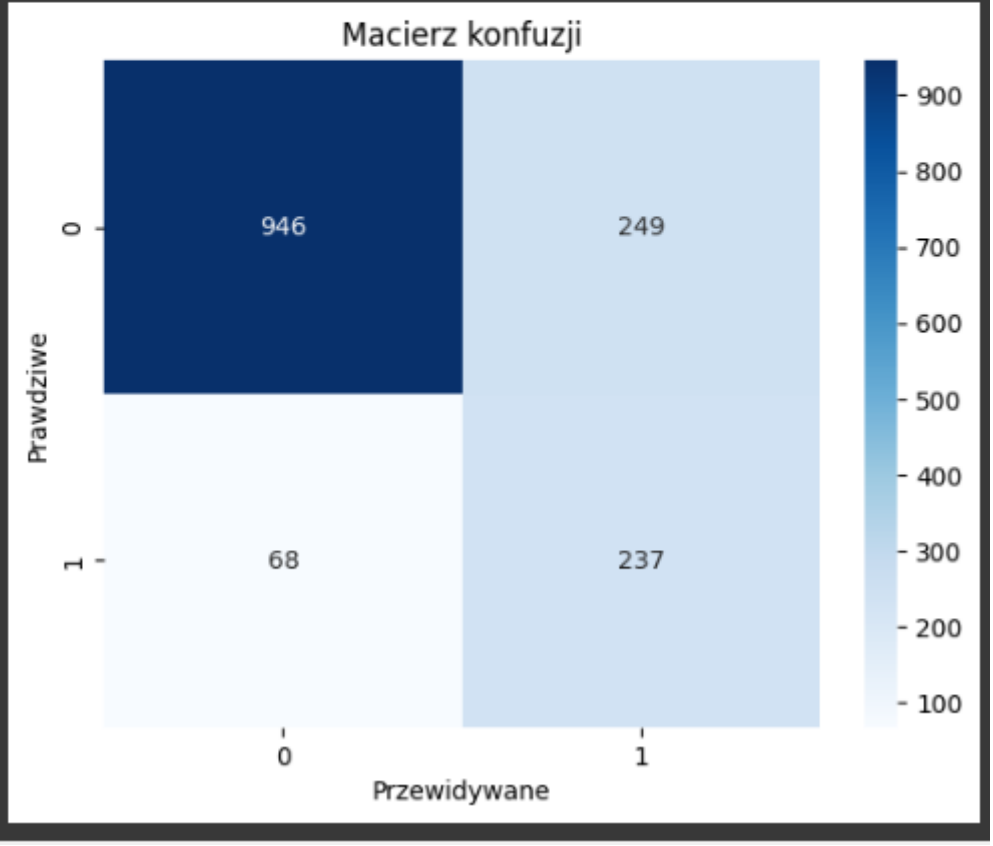
- batch_size: 32

- EarlyStopping: włączone

"""

Raport klasyfikacji:

	precision	recall	f1-score	support
0	0.93	0.79	0.86	1195
1	0.49	0.78	0.60	305
accuracy			0.79	1500
macro avg	0.71	0.78	0.73	1500
weighted avg	0.84	0.79	0.80	1500



"""

Przypadek 5:

- 2 warstwy (64, 32)

- Aktywacja: tanh

- Optymalizator: Adam

- Liczba epok: 50

- batch_size: 32

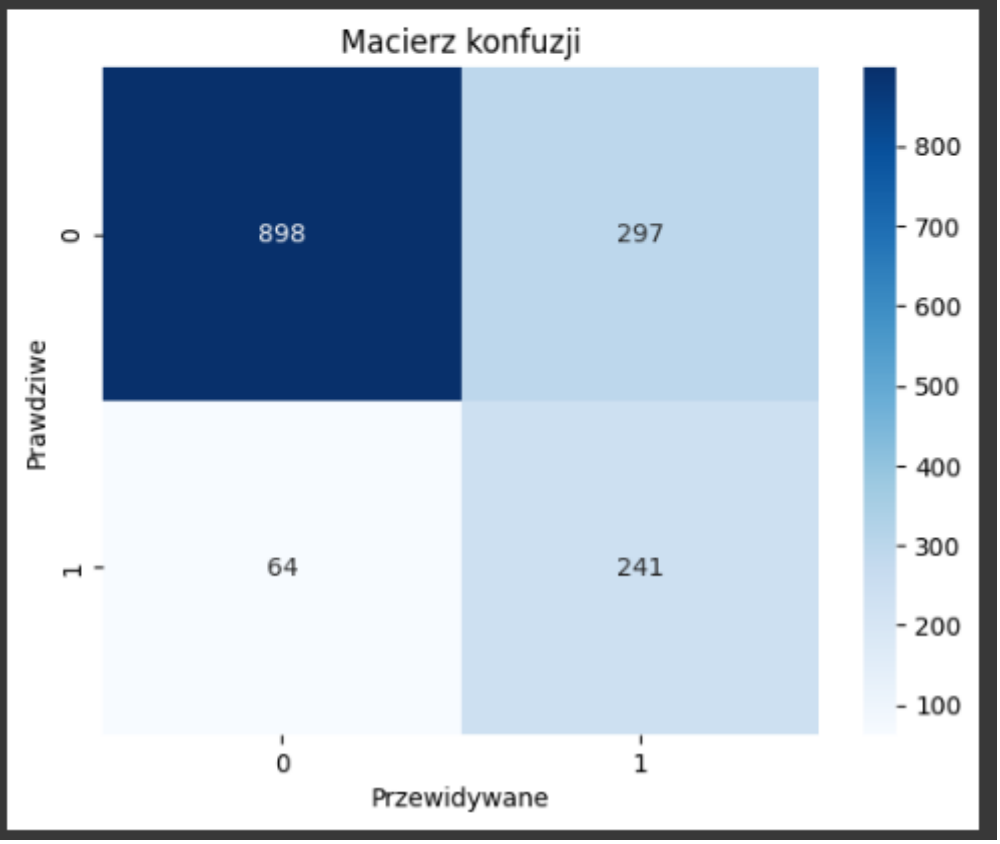
- EarlyStopping: włączone

```
Test Loss: 0.5179463028907776
Test Accuracy: 0.7593333125114441
AUC-ROC: 0.8583469373756774
F1 Score: 0.571767497034401

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.93      0.75      0.83      1195
           1       0.45      0.79      0.57       305

    accuracy                           0.76      1500
   macro avg       0.69      0.77      0.70      1500
weighted avg       0.83      0.76      0.78      1500
```



"""

Przypadek 6:

- 3 warstwy (128, 64, 32)
- Aktywacja: ReLU
- Optymalizator: Adam
- Liczba epok: 50
- batch_size: 32

- EarlyStopping: włączone

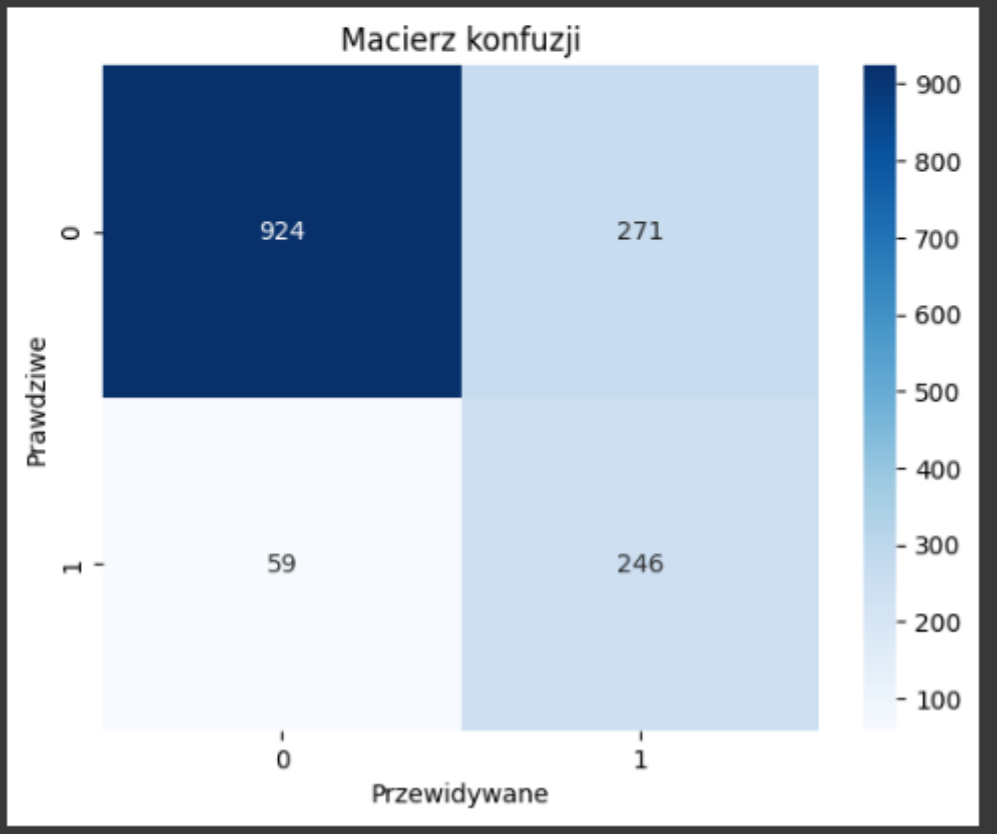
"""

```
Test Loss: 0.48771804571151733
Test Accuracy: 0.7799999713897705
AUC-ROC: 0.8696097126003154
F1 Score: 0.5985401459854015

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.77      0.85      1195
           1       0.48      0.81      0.60       305

    accuracy                           0.78      1500
   macro avg       0.71      0.79      0.72      1500
weighted avg       0.85      0.78      0.80      1500
```



"""

Przypadek 7:

- 3 warstwy (128, 64, 32)

- Aktywacja: tanh

- Optymalizator: Adam

- Liczba epok: 100

- batch_size: 32

- EarlyStopping: włączone

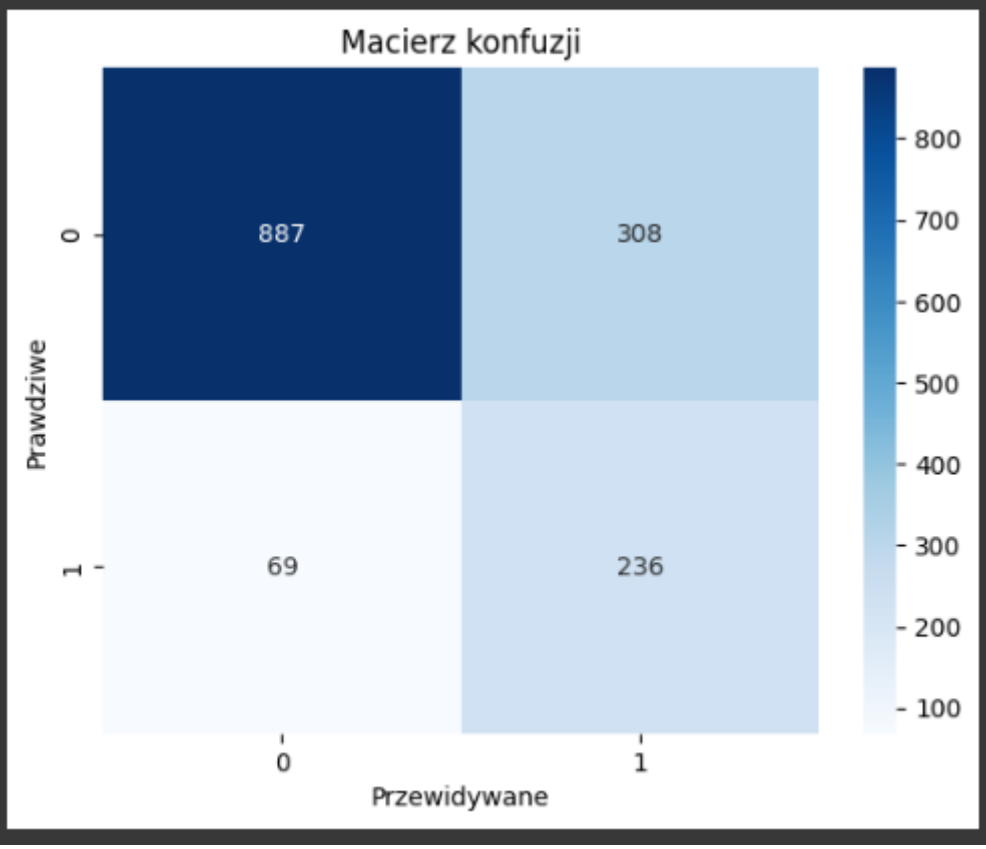
"""

```
Test Loss: 0.5418013334274292
Test Accuracy: 0.7486666440963745
AUC-ROC: 0.8495424926263803
F1 Score: 0.5559481743227326

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.93      0.74      0.82      1195
           1       0.43      0.77      0.56       305

    accuracy                           0.75      1500
   macro avg       0.68      0.76      0.69      1500
weighted avg       0.83      0.75      0.77      1500
```



"""

Przypadek 8:

- 2 warstwy (64, 32)

- Aktywacja: softplus

- Optymalizator: Adam

- Liczba epok: 50

- batch_size: 64

- EarlyStopping: włączone

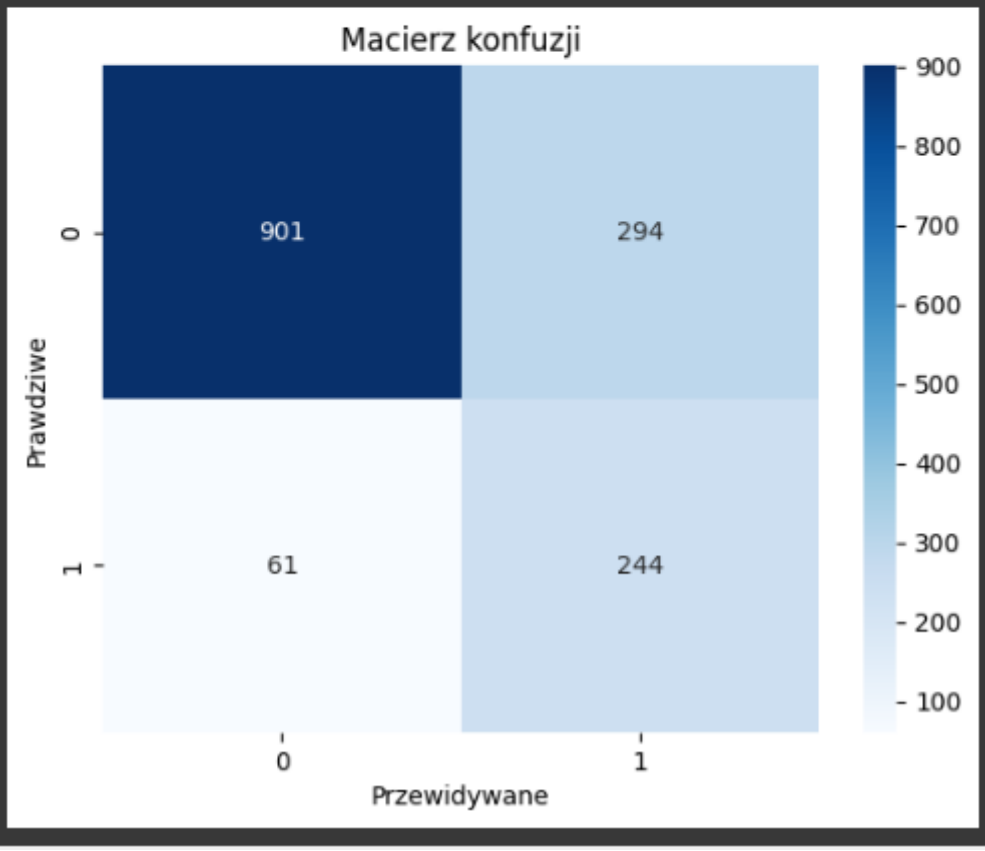
"""

```
Test Loss: 0.5680938959121704
Test Accuracy: 0.7633333206176758
AUC-ROC: 0.8585033267027917
F1 Score: 0.5788849347568209

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.75      0.84      1195
           1       0.45      0.80      0.58       305

    accuracy                           0.76      1500
   macro avg       0.70      0.78      0.71      1500
weighted avg       0.84      0.76      0.78      1500
```



"""

Przypadek 9:

- 3 warstwy (64, 32, 16)

- Aktywacja: ReLU

- Optymalizator: SGD (momentum=0.9)

- Liczba epok: 50

- batch_size: 32

- EarlyStopping: wyłączone

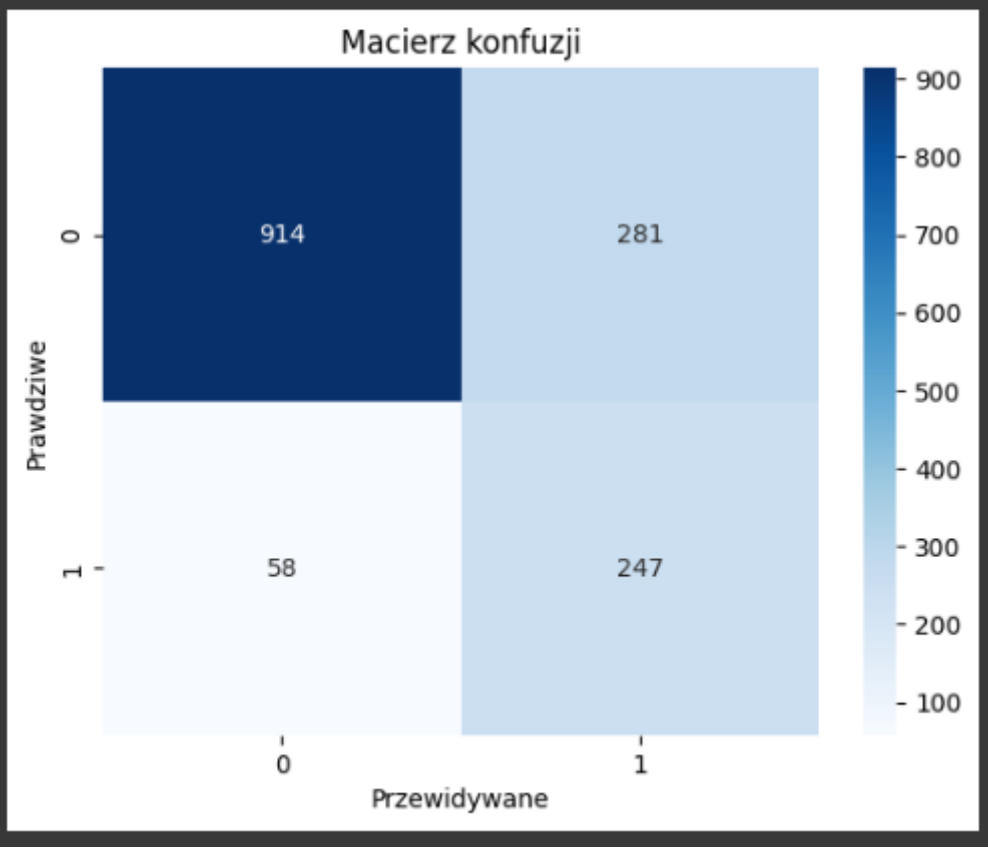
"""

```
Test Loss: 0.5025773644447327
Test Accuracy: 0.7739999890327454
AUC-ROC: 0.8693133959805199
F1 Score: 0.5930372148859544

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.76      0.84      1195
           1       0.47      0.81      0.59       305

    accuracy                           0.77      1500
   macro avg       0.70      0.79      0.72      1500
weighted avg       0.84      0.77      0.79      1500
```



Przypadek 10:

- 3 warstwy (128, 64, 32)

- Aktywacja: softplus

- Optymalizator: Adam

- Liczba epok: 50

- batch_size: 32

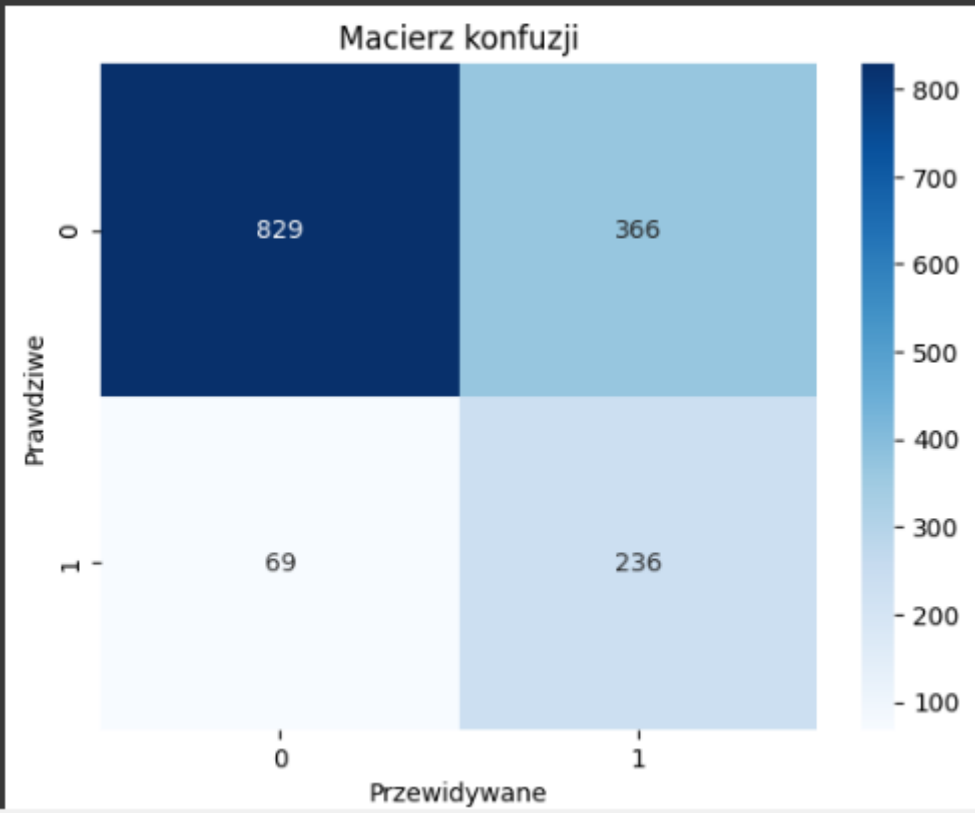
- EarlyStopping: włączone

```
Test Loss: 0.6264234781265259
Test Accuracy: 0.7099999785423279
AUC-ROC: 0.797665134782907
F1 Score: 0.5203969128996693

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.92      0.69      0.79      1195
           1       0.39      0.77      0.52       305

    accuracy                           0.71      1500
   macro avg       0.66      0.73      0.66      1500
weighted avg       0.82      0.71      0.74      1500
```



Przypadek 11

- 3 warstwy (128, 64, 32)

- Aktywacja: softplus

- Optymalizator: SGD (z momentum=0.9)

- Liczba epok: 50

- batch_size: 32

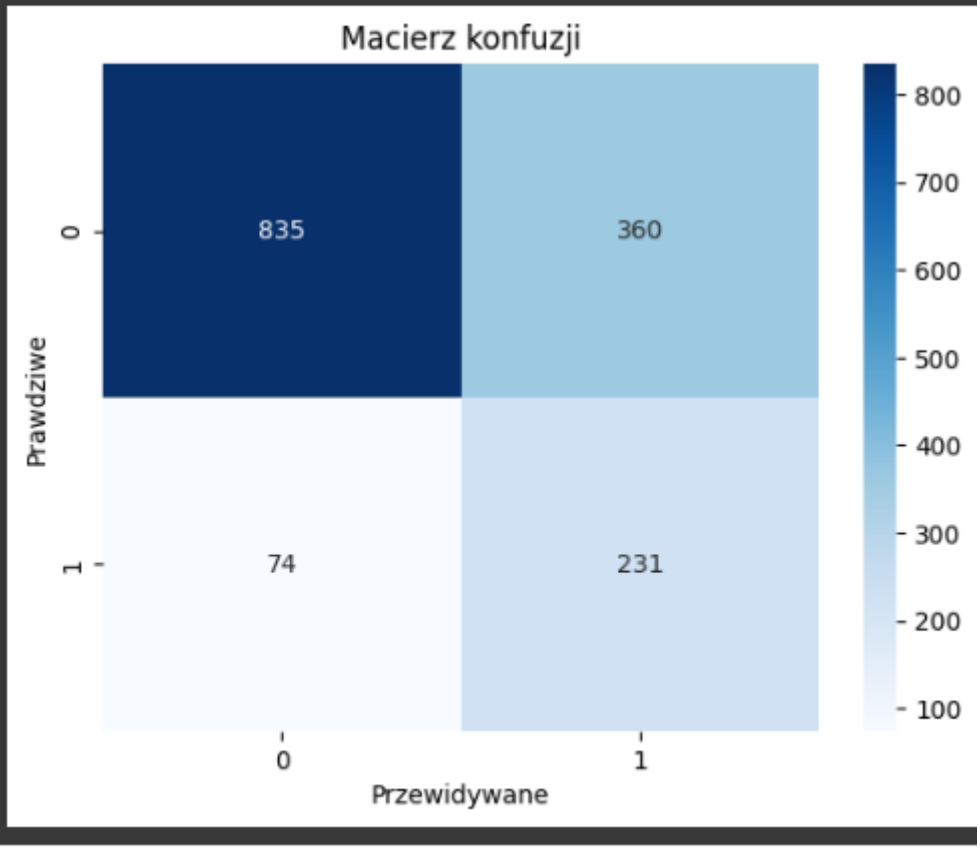
- EarlyStopping: wyłączone

```
Test Loss: 0.6348061561584473
Test Accuracy: 0.7106666564941406
AUC-ROC: 0.798235818643254
F1 Score: 0.515625

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.92      0.70      0.79      1195
           1       0.39      0.76      0.52       305

    accuracy                           0.71      1500
   macro avg       0.65      0.73      0.65      1500
weighted avg       0.81      0.71      0.74      1500
```



"""

Przykład 12:

- 3 warstwy (8, 4, 2) - bardzo małe warstwy

- Aktywacja: softplus

- Optymalizator: SGD (z momentum=0.9)

- Liczba epok: 50

- batch_size: 32

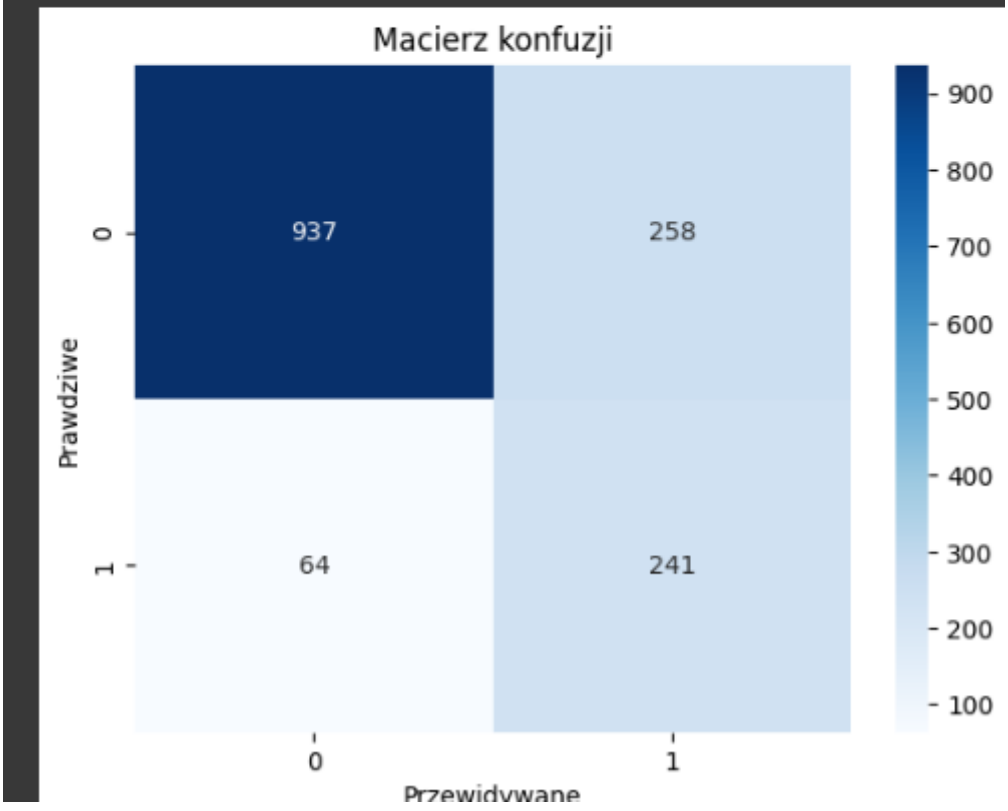
- EarlyStopping: wyłączone

```
Test Loss: 0.4658828675746918
Test Accuracy: 0.7853333353996277
AUC-ROC: 0.870635846079978
F1 Score: 0.599502487562189

Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.94      0.78      0.85      1195
           1       0.48      0.79      0.60       305

    accuracy                           0.79      1500
   macro avg       0.71      0.79      0.73      1500
weighted avg       0.84      0.79      0.80      1500
```



Podsumowanie – wnioski i napotkane problemy

Sieci neuronowe umożliwiają rozwiązywanie skomplikowanych problemów z kategorii identyfikacji i predykcji.

Sieci możemy podzielić na wiele rodzajii, ze względu na charakter rozwiązywanego problemu (klasyfikacyjny, klasyfikacyjny-binarny, regresyjny), jak i również na architekturę.

Ważnymi aspektami pracy z sieciami neuronowymi jest odpowiednie rozpoznanie danych wejściowych, oraz zastosowanie pre-procesowania danych, która obejmuje usunięcie zbędnych wartości, weryfikację i ewentualne usunięcie wartości błędnych, kodowanie zmiennych kategorycznych, oraz standaryzację danych.

Najważniejszym aspektem pracy z sieciami neuronowymi jest dobór odpowiednich parametrów sieci, w celu osiągnięcia możliwie jak najlepszych wyników. Oprócz doboru parametrów podstawowych, nowoczesne biblioteki umożliwiają optymalizację procesu uczenia, i doboru poprzez zastosowanie optymalizatora Adam, funkcję binary_crossentropy, jak i również monitorowanie trenowanego modelu, i odpowiednie działania w przypadku braku poprawy metryk (np. straty walidacyjnej w nadzorowaniu EarlyStopping**)**

Na podstawie obserwacji, można wyciągnąć następujące wnioski:

- Zwiększenie liczby warstw ukrytych (do maksymalnie 3) poprawia zdolność sieci do uchwycenia bardziej złożonych zależności w danych. Jednak powyżej 3 warstw nie zauważono znaczącego wzrostu dokładności, co wskazuje na możliwość przeuczenia modelu.
- Warstwy z większą liczbą neuronów (np. 128 i 64) dają lepsze wyniki w porównaniu z mniejszymi (np. 32 i 16). Jednak zwiększanie liczby neuronów powyżej tego zakresu prowadziło do wzrostu kosztów obliczeniowych, bez znaczącej poprawy jakości klasyfikacji.
- Funkcja aktywacji ReLU sprawdziła się najlepiej w większości przypadków. Funkcje tanh i softplus również dawały dobre wyniki, ale były bardziej podatne na problemy z gradientami (eksplozją lub zanikiem), co wydłużało proces uczenia.
- Optymalizator Adam zapewnił najszybszą konwergencję i stabilne wyniki w porównaniu z optymalizatorem SGD (momentum=0.9). W przypadku SGD zauważono większe wahania w wynikach, co wymagało dokładniejszego dostrajania parametrów, takich jak learning rate.

- Mniejszy batch size (np. 32) pozwalał na szybszą aktualizację wag, ale wprowadzał większy szum w uczeniu. Z kolei większe wartości batch size (np. 64) prowadziły do bardziej stabilnych wyników, choć proces uczenia trwał dłużej.
- Włączenie mechanizmu EarlyStopping pozwoliło uniknąć przeuczenia modelu, szczególnie przy dużej liczbie epok.
- Dodanie Dropout na poziomie 0.5 oraz regularyzacji L2 (lambda=0.001) skutecznie zapobiegało przeuczeniu, szczególnie w bardziej złożonych modelach.
- Wykorzystanie techniki nadpróbkowania dla klasy mniejszościowej (klasa 1) znacznie poprawiło zdolność modelu do identyfikacji klientów z wysokim ryzykiem rezygnacji.
- Najlepsze wyniki uzyskano dla modelu z 3 warstwami ukrytymi (128, 64, 32), funkcją aktywacji ReLU, optymalizatorem Adam i mechanizmem EarlyStopping. Metryki takie jak AUC-ROC oraz F1-score osiągnęły wartości powyżej 0.9, co wskazuje na wysoką jakość klasyfikacji.

Podczas projektu, głównymi problemami były:

- Dobór odpowiedniej sieci sieci neuronowej do rozwiązywanego problemu
- Nauka zasady działania sieci neuronowych, wpływ poszczególnych parametrów na działanie sieci neuronowej, oraz poznanie dodatkowych możliwości optymalizacyjnych działania sieci neuronowych
- Wyszukanie i zastosowanie bibliotek do uczenia maszynowego – wyszukiwanie odpowiednich metod
- Problem doboru parametrów, w celu uniknięcia przetrenowania, na początkowym etapie prac nad projektem