Głębokie uczenie i inteligencja obliczeniowa
Automatyka i Robotyka II stopień
2024/2025

Detekcja sentymentów Twettów z użyciem sieci neuronowych oraz transformerów

Skład zespołu: Bartłomiej Barszczak, Janusz Chmiel, Michał Cynarski, Tomasz Mikurda, Damian Ogorzały, Patryk Smajdor, Adam Trybus
Opiekun: dr hab. inż. Joanna Kwiecień

1. Wstęp

Celem projektu było zbadanie i porównanie działania różnych rozwiązań bazujących na sieciach neuronowych na przykładzie analizy sentymentu tweetów. W projekcie wykorzystano sieci ANN, CNN, RNN LSTM i GRU oraz transformery.

Do analizy wybrano dataset sentiment140 zawierający ponad 1.6 miliona tweetów: https://www.kaggle.com/datasets/kazanova/sentiment140

Dataset zawiera kolumny:

> **target**: the polarity of the tweet (*0* = negative, *2* = neutral, *4* = positive) - Tak naprawdę tylko 0 i 4, żadna wartość nie jest oznaczona 2.
>
> **ids**: The id of the tweet (*2087*)
>
> **date**: the date of the tweet (*Sat May 16 23:58:44 UTC 2009*)
>
> **flag**: The query (*lyx*). If there is no query, then this value is NO_QUERY.
>
> **user**: the user that tweeted (*robotickilldozr*)
>
> **text**: the text of the tweet (*Lyx is cool*)

W projekcie wykorzystano jedynie kolumny target oraz text.

2. Pre-processing

Przed rozpoczęciem pracy z sieciami neuronowymi należało odpowiednio przygotować dane. W tym wykorzystano prosty skrypt w Pythonie ładujący dataset do dataframe-u przy pomocy pandas, a następnie wykonano kilka kroków pre-processingu.

Podjęte kroki:

- Zamiana wszystkich wielkich liter do małych.
- Usunięcie linków.
- Usunięcie wspomnień “@”.
- Usunięcie znaków interpunkcyjnych.
- Usunięcie słów “stopwords” przy wykorzystując zestaw z bilbioteki ntl:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself',
'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
```

'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

Po wykonaniu pre-processingu podzielono zestaw danych na zbiory treningowe i testowe. Wykorzystano w tym celu funkcję train_test_split z biblioteki scikit learn. Zbiory zostały podzielone w stosunku 80% treningowy, 20% testowy, do stratyfikacji wykorzystano wartość sentymentu, aby uniknąć sytuacji gdzie np: w zbiorze testowym znajdą się jedynie dane oznaczone jako pozytywne.

3. Sieć ANN

Sztuczna sieć neuronowa (ANN – Artificial Neural Network) to jedna z podstawowych struktur w uczeniu maszynowym, inspirowana sposobem działania ludzkiego mózgu. Składa się z warstw połączonych ze sobą neuronów, które wspólnie przetwarzają dane wejściowe, ucząc się rozpoznawać wzorce i podejmować decyzje.

Jednym z najczęściej stosowanych typów warstw w takich sieciach są **warstwy gęste**, znane także jako **warstwy w pełni połączone (Dense layers)**. W warstwie Dense każdy neuron jest połączony ze wszystkimi neuronami poprzedniej i następnej warstwy, co pozwala na pełne wykorzystanie informacji przepływających przez sieć.

W praktyce, każda warstwa Dense realizuje transformację danych wejściowych poprzez:

1. **Obliczenie iloczynu wag i wejść** – każda cecha wejściowa jest przemnażana przez przypisaną jej wagę.
2. **Dodanie biasu (wartości przesunięcia)** – w celu zwiększenia elastyczności modelu.
3. **Zastosowanie funkcji aktywacji** – dzięki której sieć może modelować nieliniowe zależności między danymi.

Najczęściej wykorzystywaną funkcją aktywacji w warstwach ukrytych jest **ReLU (Rectified Linear Unit)**, która pozostawia tylko wartości dodatnie. W warstwach wyjściowych stosuje się inne funkcje, zależnie od typu zadania – np. **sigmoid** dla klasyfikacji binarnej lub **softmax** dla klasyfikacji wieloklasowej.

Sieci oparte na warstwach Dense znajdują szerokie zastosowanie w zadaniach takich jak:

- klasyfikacja (np. analiza sentymentu, rozpoznawanie obrazów),
- regresja (np. przewidywanie cen, trendów),
- wykrywanie anomalii,
- analiza danych tablicowych.

Są szczególnie skuteczne, gdy dane wejściowe są już wstępnie przetworzone i nie wymagają zachowania struktury sekwencyjnej (jak w przypadku tekstu czy szeregów czasowych). Ich prostota i uniwersalność sprawiają, że są chętnie wykorzystywane zarówno w prostych projektach, jak i w bardziej zaawansowanych systemach, często jako część większych architektur.

Architektura sieci Dense:

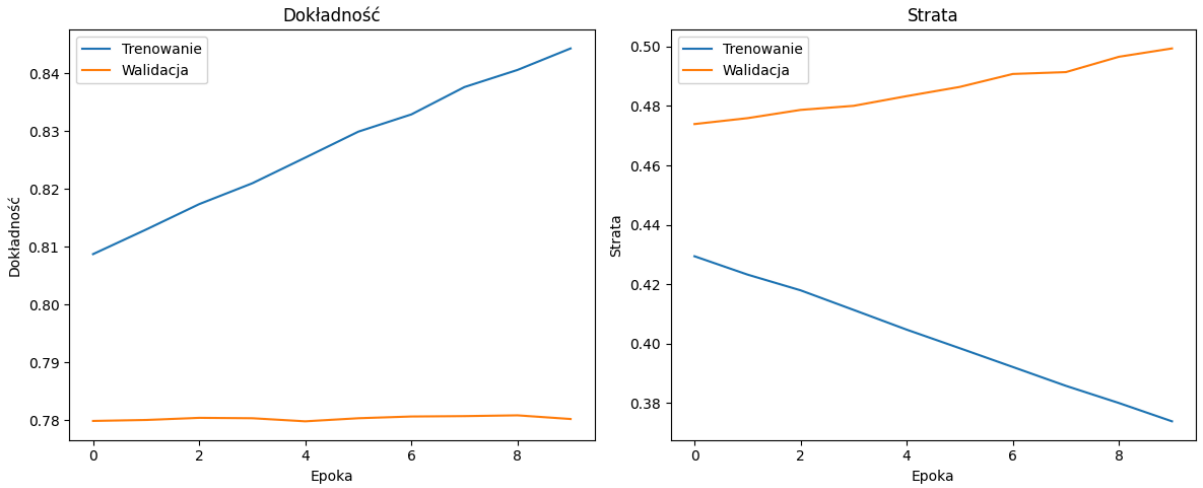
- **Dense (256, ReLU)**
  Pierwsza warstwa gęsta z 256 neuronami.
  Funkcja aktywacji ReLU wprowadza nieliniowość i pozwala sieci uczyć złożonych zależności między cechami wejściowymi.
  Jest to warstwa wejściowa — przyjmuje dane z wektoryzatora TF-IDF (o wymiarze 15 000).
- **BatchNormalization**
  Normalizuje dane przechodzące przez warstwę, co stabilizuje i przyspiesza proces uczenia. Pomaga w utrzymaniu odpowiedniego rozkładu aktywacji w czasie treningu.
- **Dropout (0.5)**
  Losowo wyłącza 50% neuronów podczas uczenia, co zapobiega przeuczeniu i zwiększa zdolność generalizacji modelu.
- **Dense (128, ReLU)**
  Druga warstwa gęsta, zawiera 128 neuronów z funkcją aktywacji ReLU.
  Przetwarza dane z poprzedniej warstwy i wydobywa kolejne poziomy reprezentacji.

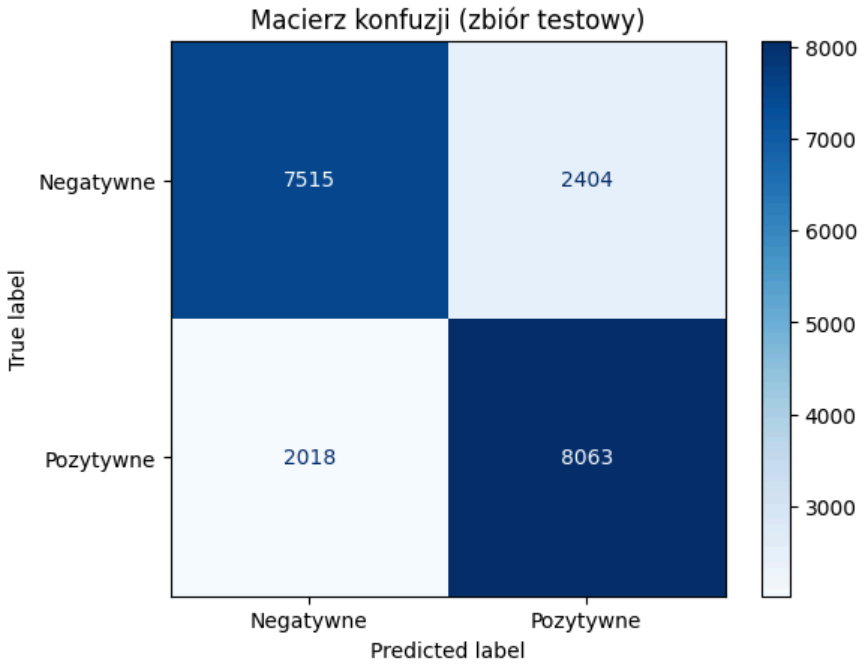
- **BatchNormalization + Dropout (0.3)**
  Normalizacja oraz wyłączenie 30% neuronów w kolejnej warstwie — dodatkowa regularizacja.
- **Dense (64, ReLU)**
  Trzecia warstwa gęsta z 64 neuronami.
  Utrzymuje aktywację ReLU dla ciągłości przekształceń nieliniowych.
- **BatchNormalization + Dropout (0.2)**
  Ostatni etap regularyzacji przed wyjściem — pomaga w dalszej stabilizacji i ograniczeniu przeuczenia.
- **Dense (1, Sigmoid)**
  Warstwa wyjściowa z jednym neuronem.
  Aktywacja sigmoid przekształca wynik do wartości z przedziału [0, 1], reprezentując prawdopodobieństwo przynależności do klasy pozytywnej — wykorzystywana do **klasyfikacji binarnej** (np. analiza sentymentu: pozytywny/negatywny).

Parametry trenowania sieci:

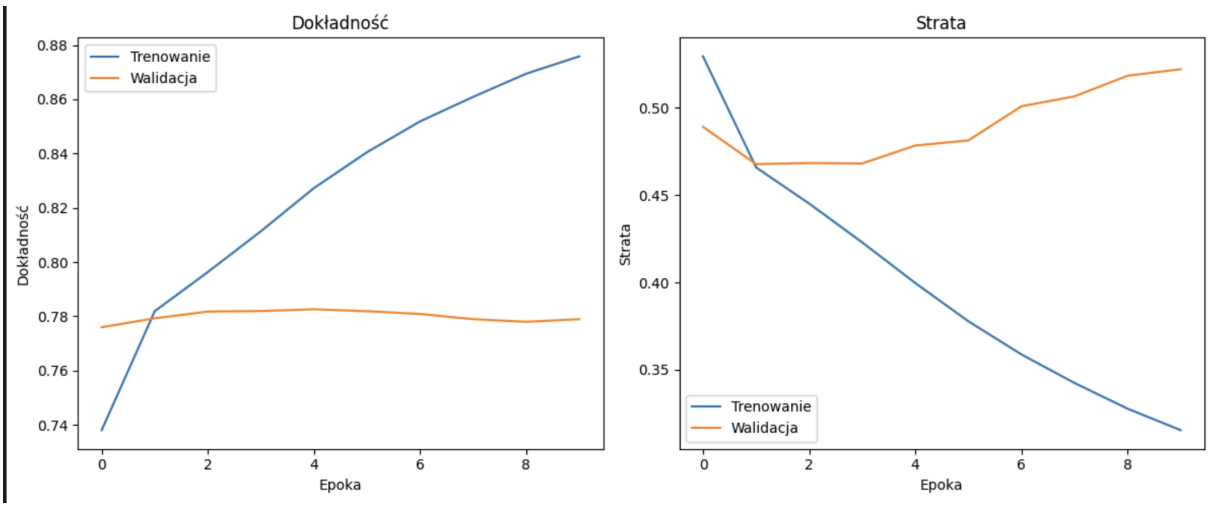
- **Optymalizator:** RMSprop
  Użyto funkcji optymalizacji RMSprop, która dobrze radzi sobie z nierównomiernie skalowanymi gradientami. Kluczowym parametrem był:
- **Learning rate (szybkość uczenia):** 0.0001
  Jest to parametr decydujący o tym, jak duże kroki wykonuje model podczas aktualizacji wag. Zbyt wysoki learning rate może spowodować niestabilność, natomiast zbyt niski — bardzo powolną konwergencję.
- **Funkcja straty:** binary_crossentropy
  Używana do klasyfikacji binarnej — porównuje przewidziane prawdopodobieństwa z rzeczywistymi etykietami (0 lub 1).
- **Liczba epok:** 10
  Oznacza, że model przeszedł 10 razy przez cały zbiór treningowy. Podczas treningu zauważono oznaki przeuczenia po kilku epokach.
- **Batch size:** 512
  Dane były dzielone na partie po 512 przykładów. Mniejszy batch size zwiększa dokładność wag, ale spowalnia obliczenia; większy przyspiesza trening, kosztem możliwego spłaszczenia lokalnych minimów.
- **Validation split:** 0.1
  10% danych treningowych zostało automatycznie odłożone jako zbiór walidacyjny, służący do monitorowania przetrenowania.

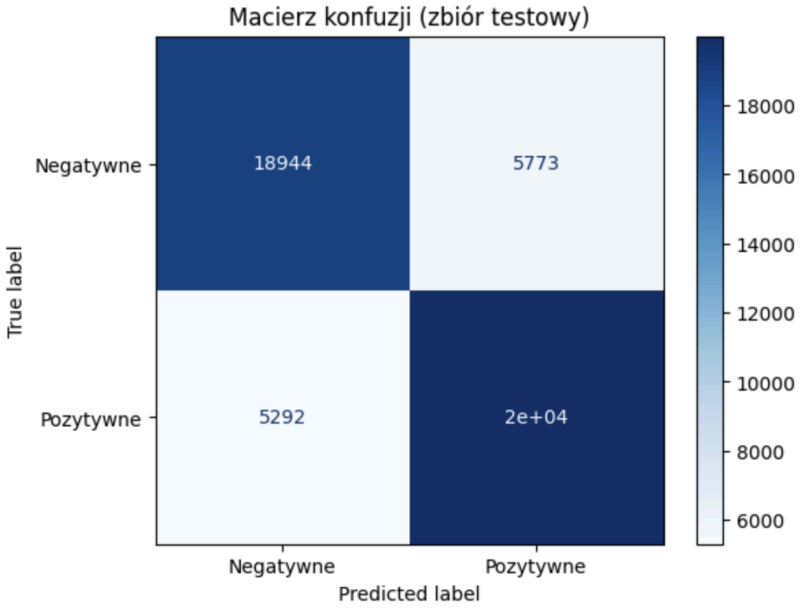
Na poniższym wykresie widać przebieg dokładności i straty dla zbioru treningowego oraz walidacyjnego. Ostateczna dokładność na zbiorze testowym wynosi około 78%, a stratę oszacowano na 50%.



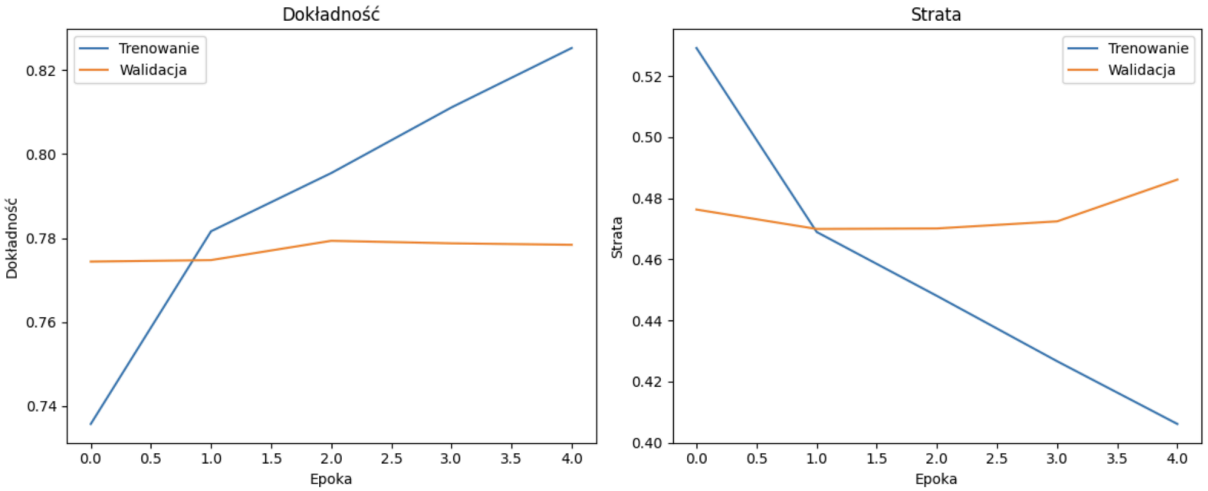


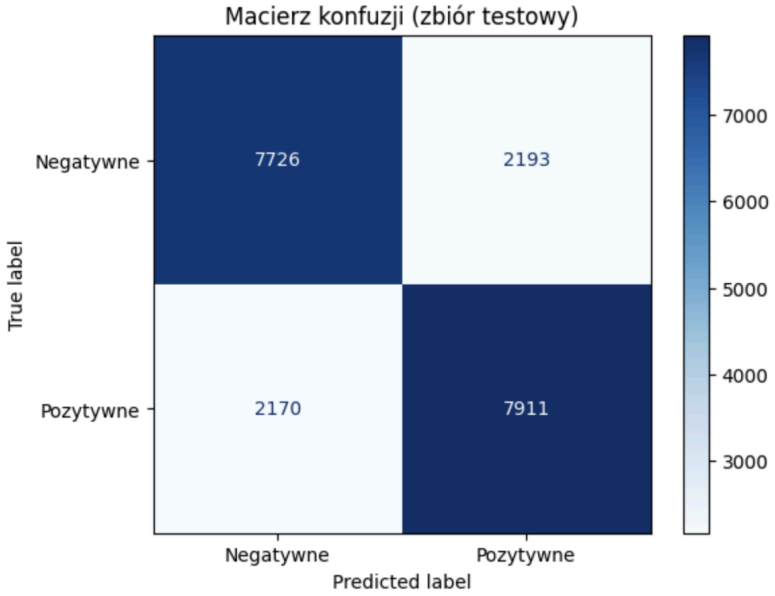
Po zwiększeniu learning rate do wartości wyniki są identyczne





Zmniejszając liczbę epok, batch size oraz wielkość zbiorów treningowych i testowych wyniki dalej nie znacząco się różnią, accuracy dla 1.6 miliona tweetów i 10 epok jest praktycznie taka sama jak ta dla 200 tysięcy tweetów i 5 epok.





Na wykresach można zauważyć, że sieć szybko się uczy i praktycznie od razu osiąga maksymalną skuteczność, efekt ten jak i wyniki są dokładnie takie same dla znacznie zmniejszonych zbiorów testowych i treningowych około 10% wszystkich, jak i dla całego zbioru. Zmiana parametrów takich jak batch size, learning rate czy liczba epok w znikomym stopniu wpływa na wyniki. Skuteczność mimo zmiany tych parametrów waha się w okolicach 77-79%, jest to prawdopodobnie spowodowane samą baza tweetów.

4. Sieć CNN

Sieć CNN (ang. Convolutional Neural Network) to rodzaj sztucznej sieci neuronowej, która jest szczególnie skuteczna w przetwarzaniu obrazów. Charakterystyczne dla niej warstwy to:

- **Warstwy splotowe (konwolucyjne)** – CNN stosuje operacje splotu, które umożliwiają wykrywanie cech charakterystycznych
- **Warstwy poolingowe** – redukują liczbę parametrów w sieci, zmniejszając rozmiar reprezentacji danych przy zachowaniu istotnych informacji. Najczęściej stosowane są operacje max pooling lub average pooling.

Aby móc korzystać ze splotu, dane muszą być przekonwertowane na wartości liczbowe. Do tego celu wykorzystano embedder sentence-transformers/multi-qa-mpnet-base-cos-v1. Zamienił on komentarze na wektory o długości 768.

Sieć stworzono wzorując się na sieci przeznaczonej do identyfikacji cyfr 0-9 ze zbioru danych MNIST przy użyciu frameworku PyTorch. Warstwy konwolucyjne dwuwymiarowe zastąpiono warstwami jednowymiarowymi. Zmieniono rozmiar ostatniej warstwy z 10 neuronów do 1 neurona. Sieć realizuje zadanie klasyfikacji binarnej więc jako funkcję straty użyto BCEWithLogitsLoss() czyli funkcję łączącą dwie operacje: Sigmoidę która automatycznie przekształca wyjście sieci na przedział [0, 1], interpretując je jako prawdopodobieństwo przynależności do klasy pozytywnej oraz Binary Cross Entropy (BCE) która oblicza stratę dla klasyfikacji binarnej na podstawie tych prawdopodobieństw.

Szczegółowy opis sieci:

1. Wejście:

   Oczekuje tensora o kształcie (rozmiar partii, 768).

2. Pierwsza warstwa konwolucyjna:

   Wyjście: 16 kanałów, długość taka sama (padding). Jądro o rozmiarze 5

3. MaxPooling

   Redukuje długość sekwencji o połowę.

4. Druga warstwa konwolucyjna:

   Wyjście: 32 kanały, długość taka sama (padding). Jądro o rozmiarze 5

5. MaxPooling

   Kolejne zmniejszenie długości sekwencji o połowę.

6. Spłaszczenie

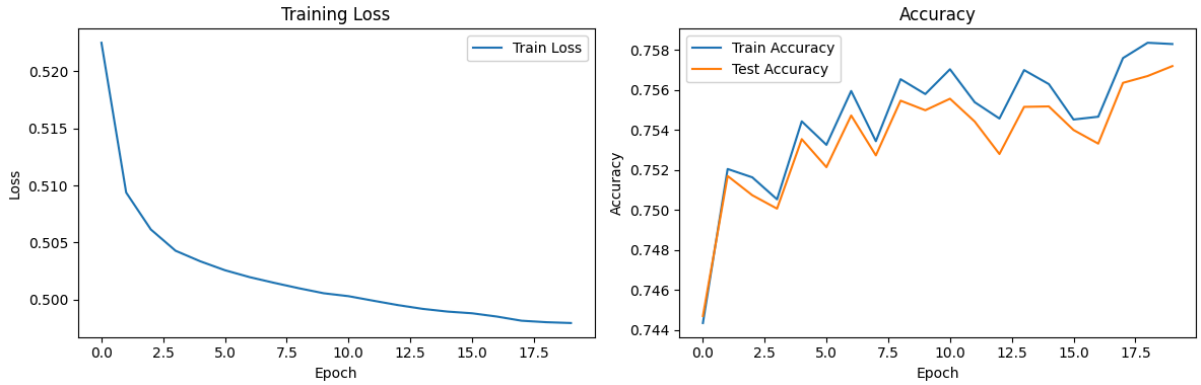
Wyjście z konwolucji i poolingów jest spłaszczane do wektora.

7. Warstwa w pełni połączona (fully connected)

   Wyjście: pojedyncza wartość

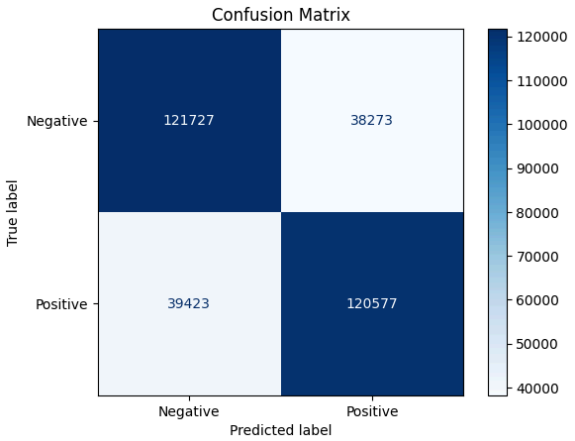
Funkcja aktywacji: Po każdej konwolucji używane jest ReLU.

Do wytrenowania sieci użyto 20 epok:



Po pierwszej epoce sieć osiągnęła dokładność 0.752. Po 19 kolejnych dokładność ta wzrosła do 0.756

Uzyskano następującą macierz konfuzji na zbiorze testowym:



5. Sieć LSTM

LSTM (Long Short-Term Memory) to specjalny typ rekurencyjnej sieci neuronowej (RNN), zaprojektowany do przetwarzania i przewidywania sekwencji danych. Główną zaletą LSTM jest zdolność do długoterminowego zapamiętywania

informacji, co rozwiązuje problem tak zwanego zanikającego gradientu, często występujący w klasycznych RNN.

LSTM składa się z jednostek (komórek), z których każda zawiera:

- Stan komórki (cell state) – główny nośnik długoterminowej pamięci, przepływa wzdłuż całej sekwencji z minimalnymi zmianami.
- Bramkę zapominania (forget gate) – decyduje, które informacje zostaną usunięte ze stanu komórki.
- Bramkę wejściową (input gate) – kontroluje, jakie nowe informacje zostaną dodane do stanu komórki.
- Bramkę wyjściową (output gate) – określa, jaka część stanu komórki zostanie przekazana jako wyjście jednostki.

Dzięki tym trzem bramkom LSTM może selektywnie zapamiętywać, zapominać i przekazywać informacje, co czyni tę sieć szczególnie skuteczną w zadaniach takich jak rozpoznawanie mowy, tłumaczenie językowe, prognozowanie szeregów czasowych czy dokonana w tej pracy analiza tekstu.

Na potrzeby projektu zaimplementowano trzy modele sieci o różnych warstwach LSTM z wykorzystaniem biblioteki tensorflow.

Pierwszy z nich o następującej architekturze:

Embedding - Warstwa zamienia indeksy słów z tekstu na gęste wektory o długości 200, input_dim=100000 – rozmiar słownika (maksymalna liczba unikalnych słów), output_dim=200 – wymiar przestrzeni osadzeń (embedding space).

LSTM (128) - Warstwa LSTM z 128 jednostkami, return_sequences=True – zwraca całą sekwencję (czyli wyjście dla każdego kroku czasowego), co umożliwia przekazanie sekwencji do kolejnej warstwy LSTM.

LSTM (64) - Kolejna warstwa LSTM z 64 jednostkami. Ponownie zwraca sekwencję, umożliwiając dalsze przetwarzanie czasowe.

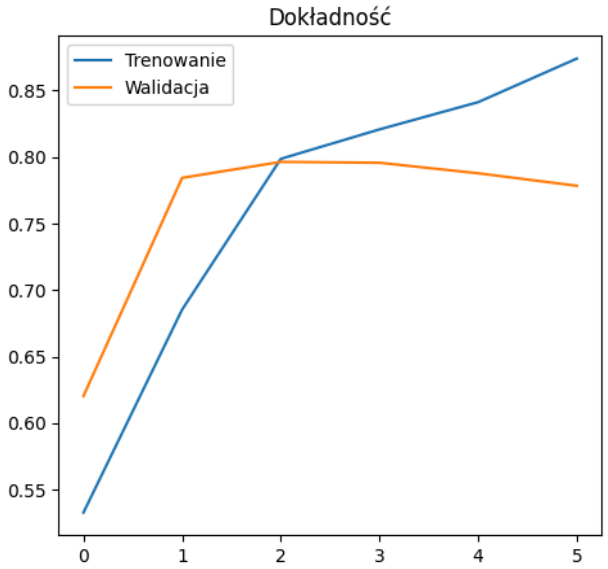
Dropout - Warstwa regularyzacyjna, losowo "wyłącza" 50% neuronów podczas uczenia, co zapobiega przeuczeniu.

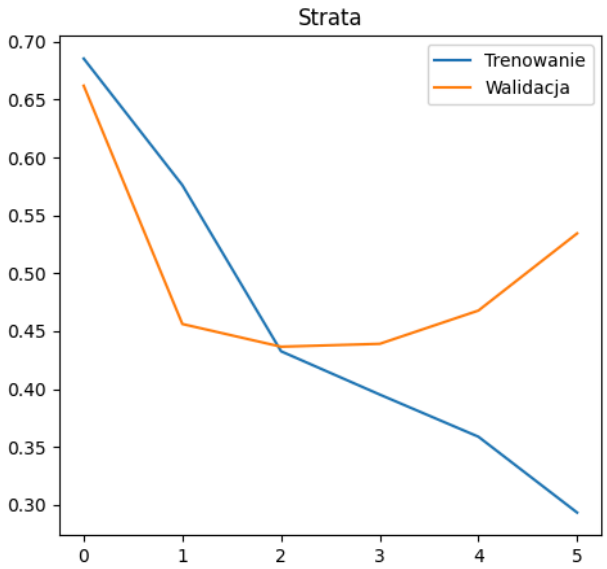
LSTM (64) - Trzecia warstwa LSTM z 64 jednostkami, return_sequences=False, więc zwraca tylko ostatnie wyjście z sekwencji (reprezentację całego ciągu).

Dense (64, ReLU) - Gęsta (fully connected) warstwa z 64 neuronami i funkcją aktywacji ReLU, przetwarza reprezentację z LSTM.

Dense (1, Sigmoid) - Warstwa wyjściowa z jednym neuronem i aktywacją sigmoid. Służy do klasyfikacji binarnej (np. pozytywna/negatywna opinia, tak/nie itp.).

Proces uczenia sieci został zakończony po piątej epoce, kiedy narastały widoczne skutki przetrenowania sieci - dokładność na zbiorze trenującym ciągle rosła, natomiast wyniki na zbiorach testowych zaczęły się pogarszać.





Na całym zbiorze testowym sieć osiągnęła dokładność na poziomie 79,7%.

Kolejny, mniejszy model zbudowano z dwukierunkowych bloków LSTM:

Embedding - warstwa jednakowa jak poprzednio.

Bidirectional LSTM (64) - dwukierunkowa warstwa LSTM z 64 jednostkami.

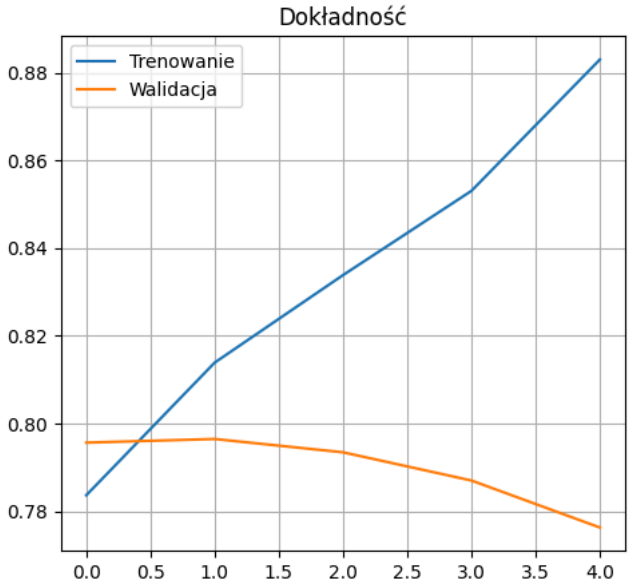
Dropout - Warstwa regularyzacyjna, losowo "wyłącza" 50% neuronów podczas uczenia.

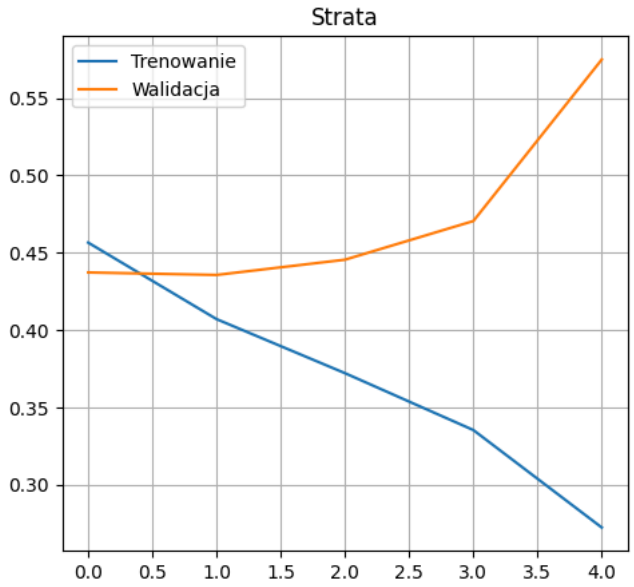
Bidirectional LSTM (32) - dwukierunkowa warstwa LSTM z 64 jednostkami.

Dense (32, ReLU) - Gęsta warstwa z 64 neuronami i funkcją aktywacji ReLU.

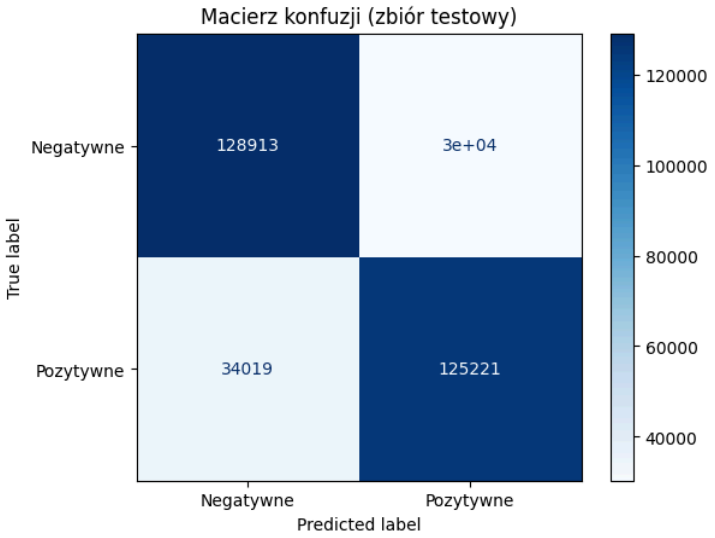
Dense (1, Sigmoid) - Warstwa wyjściowa z jednym neuronem i aktywacją sigmoid.

W takiej konfiguracji, z użyciem warstw dwukierunkowych, przetrenowanie następowało już po drugiej epoce trenowania sieci, tj. zadowalające wyniki otrzymano po pierwszej epoce uczenia.





Sieć najlepiej radziła sobie po jednej epoce uczenia osiągając dokładność na zbiorze testowym równą 79,8%. Utworzono również macierz konfuzji na podstawie zbioru testowego i przewidywanych wyników:



z macierzy wynika, że równowaga między czułością i precyzją jest dobra.

Następnie przetestowano działanie modelu wzbogaconego o kolejne bloki LSTM zbudowanego z warstw:

Embedding - Warstwa zamienia indeksy słów z tekstu na gęste wektory o długości 256, input_dim=100000 – rozmiar słownika (maksymalna liczba unikalnych słów).

Bidirectional LSTM (256) - dwukierunkowa warstwa LSTM z 256 jednostkami.

Bidirectional LSTM (128) - dwukierunkowa warstwa LSTM z 128 jednostkami.

Bidirectional LSTM (64) - dwukierunkowa warstwa LSTM z 64 jednostkami.

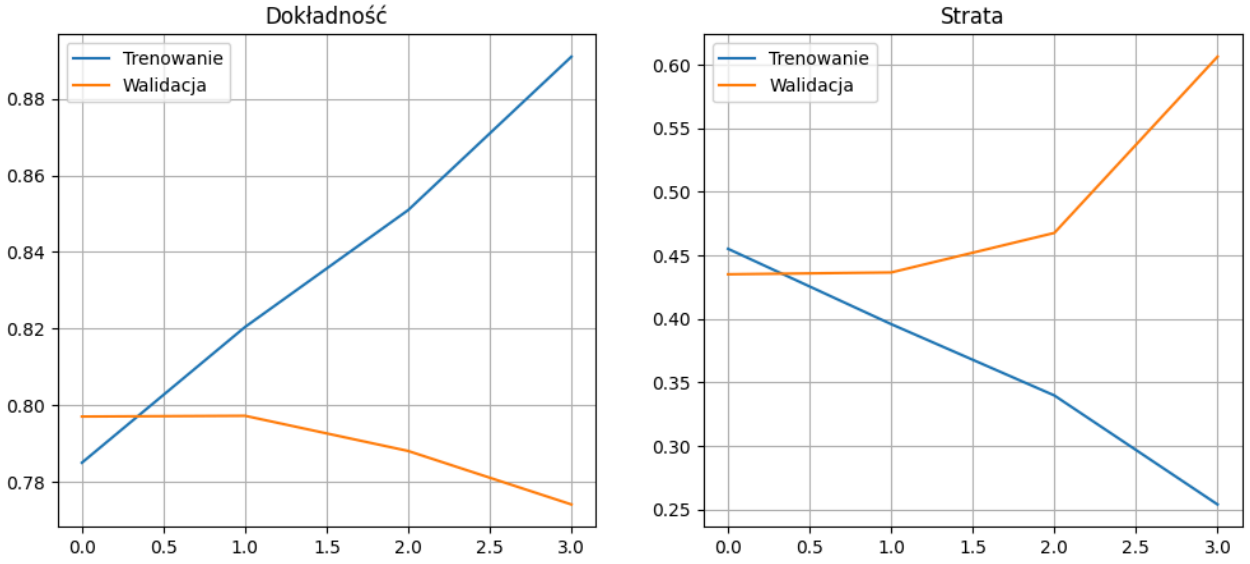
Dropout - Warstwa regularyzacyjna, losowo "wyłącza" 50% neuronów podczas uczenia.

Bidirectional LSTM (32) - dwukierunkowa warstwa LSTM z 64 jednostkami.

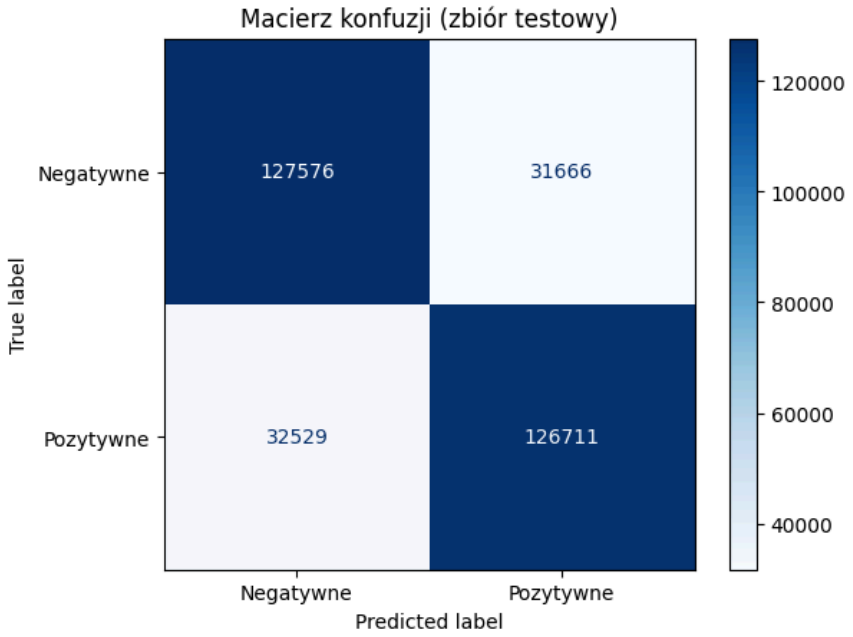
Dense (32, ReLU) - Gęsta warstwa z 64 neuronami i funkcją aktywacji ReLU.

Dense (1, Sigmoid) - Warstwa wyjściowa z jednym neuronem i aktywacją sigmoid.

Dodatkowe warstwy znacznie wydłużyły czas trenowania sieci, jednak wcale nie poprawiły skuteczności jej działania Przebieg uczenia sieci wyglądał podobnie. Dokładność na zbiorze testowym również wyniosła 79,8%.



Macierz konfuzji dla danych testowych także przedstawia zbliżone do poprzednich wartości czułości i precyzji.

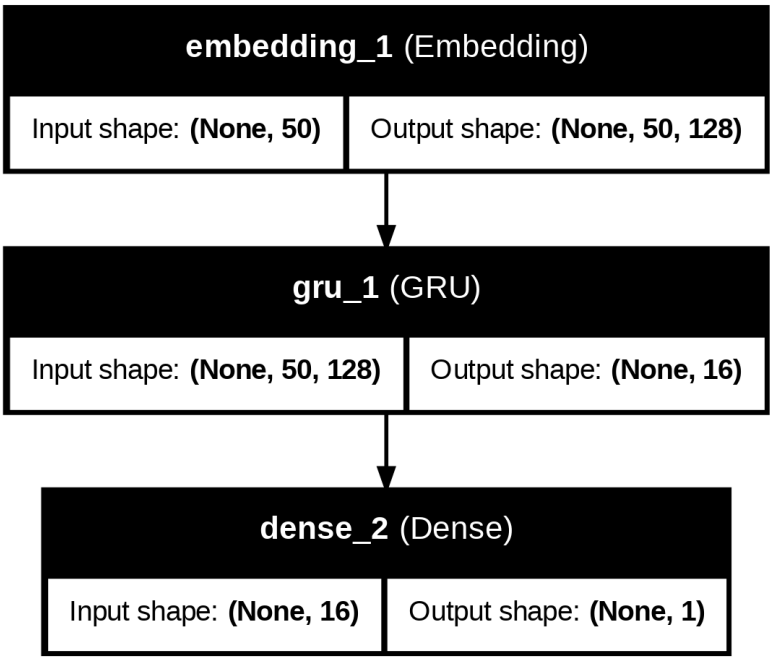


Dodawanie kolejnych warstw sieci zatem niekoniecznie zawsze poprawia działanie systemu, oraz może bez potrzeby wymagać większego zużycia zasobów.

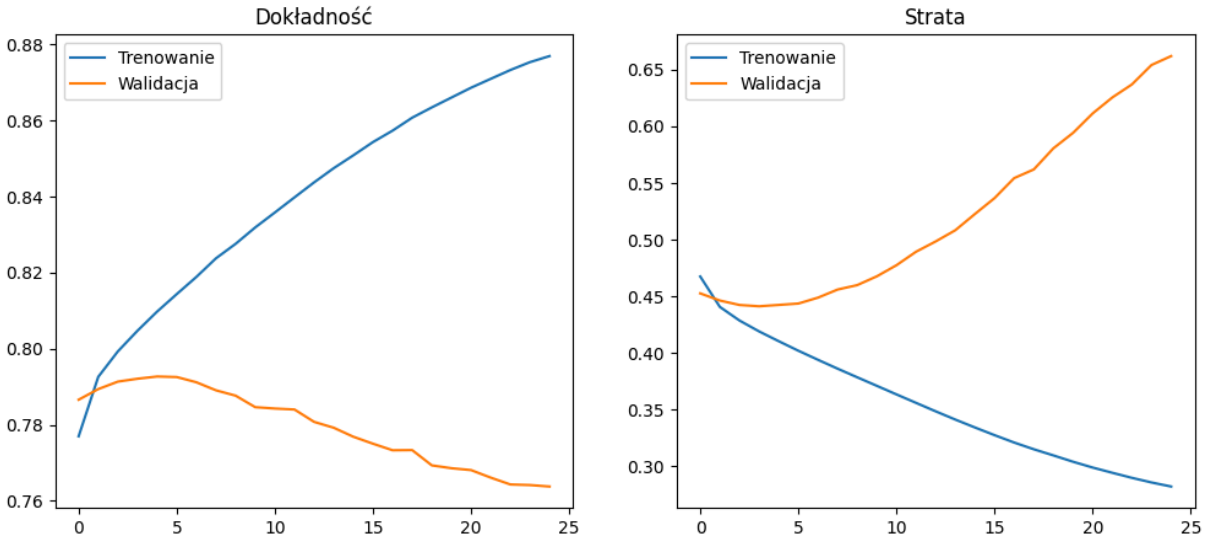
6. Sieć GRU

Sieć GRU (ang. Gated Recurrent Unit - bramkowana jednostka rekurencyjna). Sieć ta podobnie do sieci LSTM jest zbudowana z komórek. Wewnątrz każdej komórki znajdują się bramki. W tym przypadku są to bramka aktualizująca (odpowiednik wejściowej z LSTM) oraz bramka resetujaca (zapominająca w LSTM).

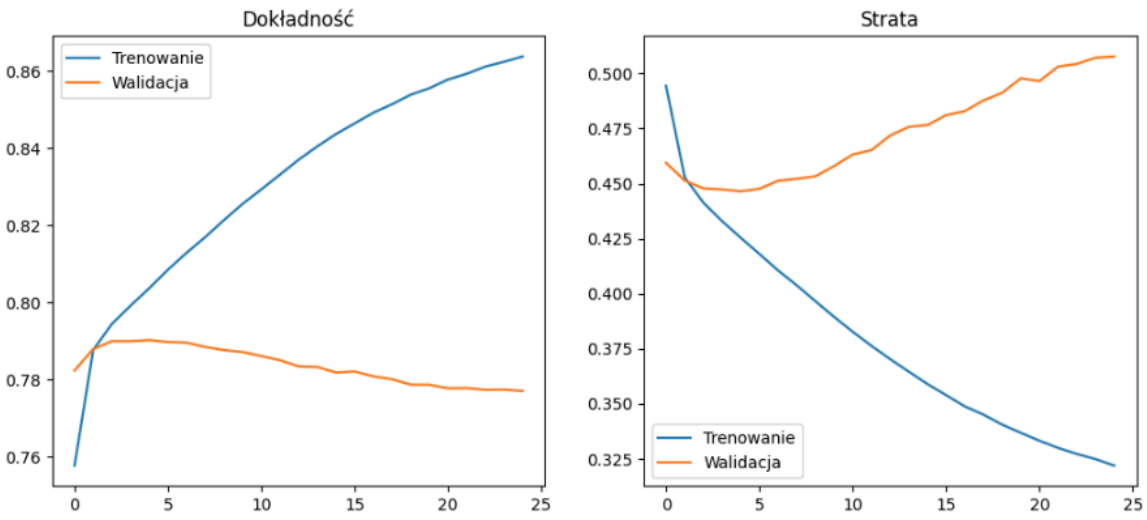
W projekcie powstała prosta sieć GRU zaimplementowana za pomocą dwóch różnych architektur: Tensorflow oraz PyTorch.



Na powyższym diagramie został przedstawiony diagram blokowy stworzonej sieci neuronowej. Jeśli chodzi o warstwę GRU to składa się ona z 16 neuronów. Dodatkowo mamy warstwę embedding oraz dense.



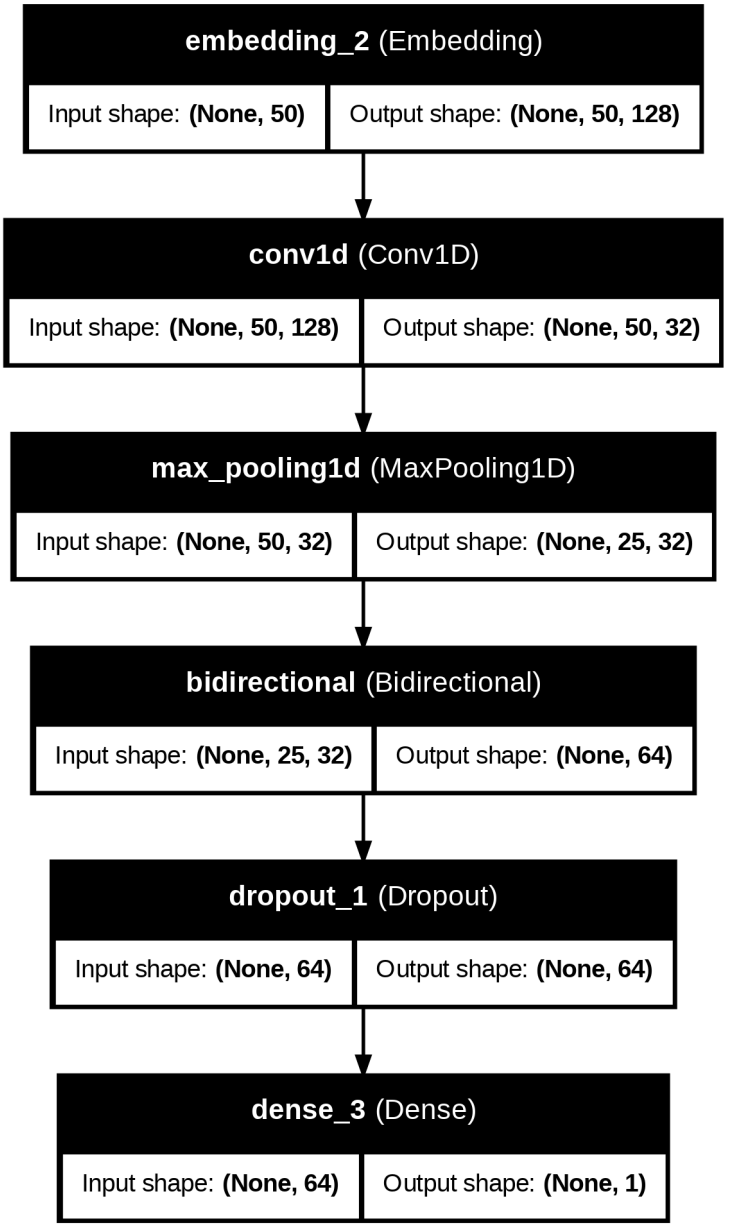
Rys 1. Wyniki trenowania sieci stworzonej w architekturze Tensorflow.



Rys 2. Wyniki trenowania sieci stworzonej w architekturze PyTorch.

Powyższe rysunki przedstawiają wyniki trenowania naszej sieci. Jak możemy zaobserwować to obie architektury poradziły sobie bardzo podobnie z tym zadanie. Dokładność trenowania na zbiorze treningowym to około 85% a na zbiorze walidacyjnym to około 79%. Można tutaj zaobserwować zjawisko owerfittingu, czyli zbyt dokładnego uczenia się danych treningowych.

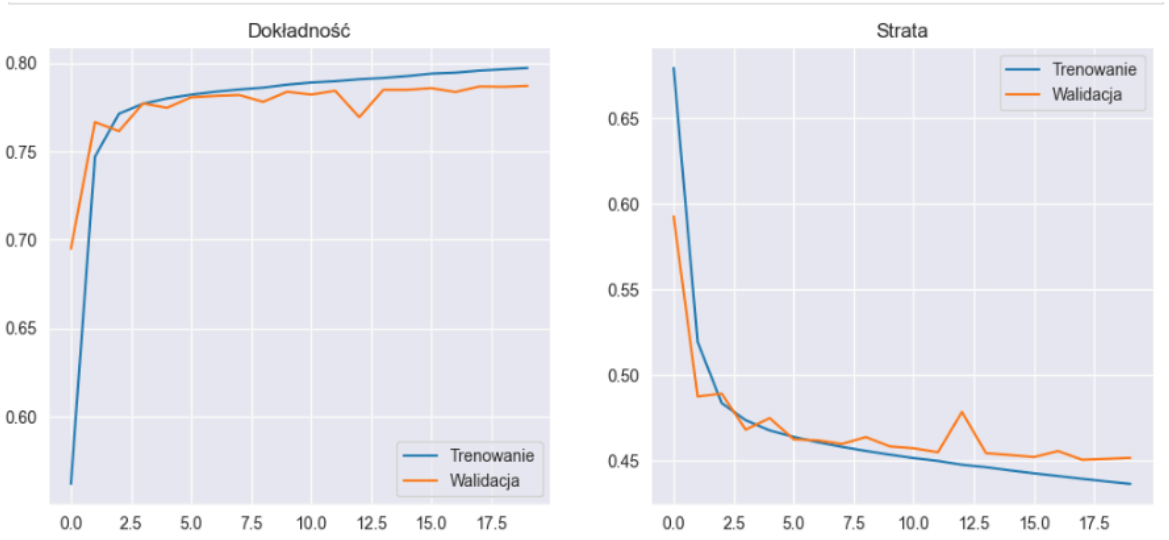
Powstała również jeszcze jedna nieco bardziej zaawansowana sieć której architektura została przedstawiona na poniższym schemacie. Ta sieć została zaimplementowana z użyciem biblioteki Tensorflow.



W odróżnieniu do wcześniejszej sieci ta zawiera:

- **Warstwę konwolucyjną (Conv1D)** która pomaga wykrywać lokalne wzorce
- **Warstwę MaxPooling1D** która redukuje wymiar danych i wybiera najistotniejsze cechy

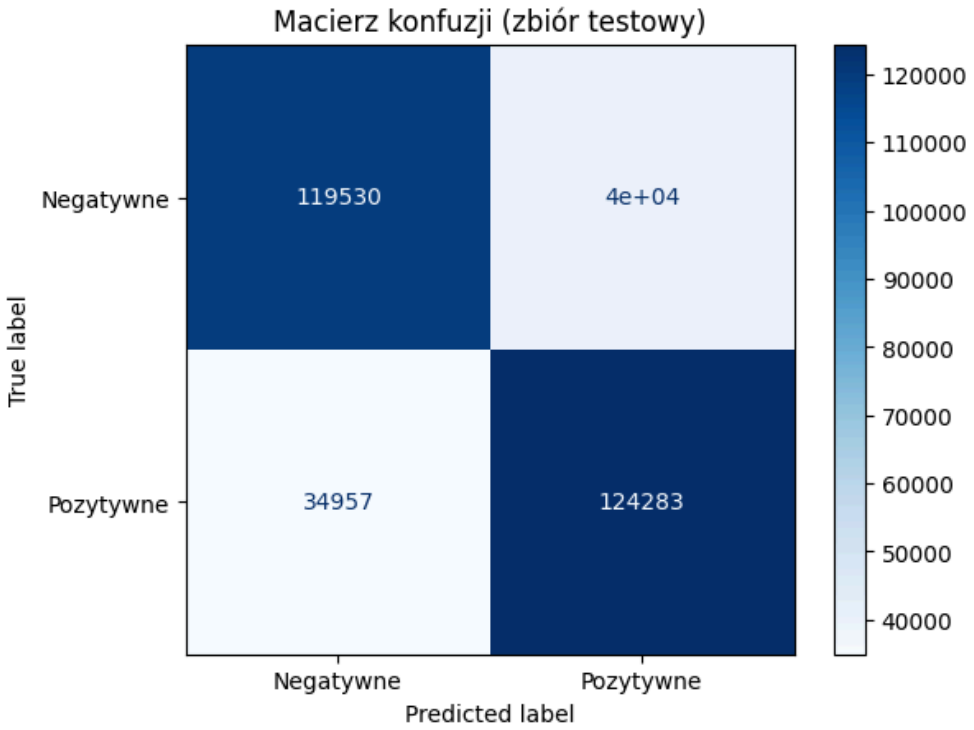
- **Bidirectional GRU** pozwala sieci brać pod uwagę zarówno poprzedni, jak i przyszły kontekst słów – co jest **bardzo istotne w języku naturalnym**, gdzie kolejność słów ma znaczenie.



Rys 3. Wyniki trenowania zaawansowanej architektury sieci

Na poniższym wykresie możemy zaobserwować, że w tym przypadku nie mamy już zjawiska owerfittingu, jednak dokładność sieci na zbiorze treningowym nieco spadła, wynosi ona teraz około 80%. Za to dokładność na zbiorze walidacyjnym jest nieco większa niż wcześniej. Jeśli chodzi o stratę to wygląda ona bardzo podobnie do wcześniejszej architektury sieci. Zaawansowana architektura wymaga jednak więcej czasu do wytrenowania co może być niekorzystne w kontekście uczenia sieci na bardzo dużych zbiorach danych.

Jeśli chodzi o dokładność działania sieci to wszystkie dokładnie przewidują między 60% a 75% przypadków. Tak wygląda przykładowa macierz konfuzji dla naszych danych testowych.



7. Transformery

Transformer to rodzaj modelu sieci neuronowej, który został przedstawiony w pracy „Attention is All You Need” przez Google w 2017 roku. Zamiast przetwarzać dane sekwencyjnie (jak np. modele typu RNN czy LSTM), transformer działa równolegle i wykorzystuje mechanizm uwagi.

Kluczowymi elementami transformerów są:

- Mechanizm uwagi - pozwala modelowi skupić się na istotnych częściach wejścia, niezależnie od ich pozycji dzięki temu model może lepiej rozumieć kontekst i zależności, np. kto co zrobił w długim zdaniu

- Pozycjonalne kodowanie - transformer przetwarza dane równolegle (a nie sekwencyjnie jak RNN), dlatego potrzebuje informacji o kolejności słów. Stosuje się specjalne wektory dodawane do wejścia, które kodują pozycję słowa w zdaniu.

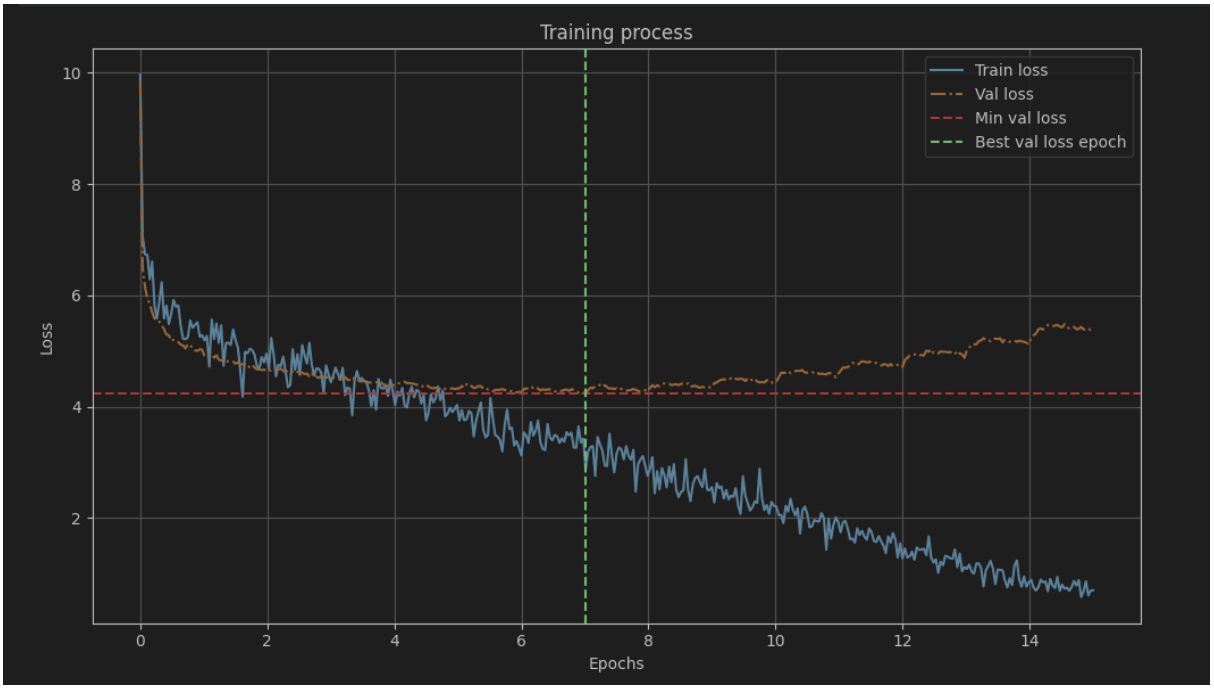
- Transformer może działać jako:
    - Encoder (lepsze rozumienie tekstu)
    - Decoder (lepsze generowanie tekstu)
    - Encoder-Decoder

- Transformer składa się z wielu identycznych warstw. Umożliwia to głębokie rozumienie kontekstu i tworzenie złożonych reprezentacji językowych.

W projekcie została zaimplementowana architektura transformera wykorzystująca wyłącznie bloki dekodera. Wykorzystana została architektura taka jak w modelu GPT-2, ale z mocno okrojonymi parametrami z przyczyn sprzętowych (mniejszy model był w stanie trenować się z wykorzystaniem karty graficznej w rozsądnym czasie).
Pierwszy model posiadał 12 warstw transformera z czego każdy blok korzystał z mechanizmu uwagi MultiHead Attention których również było 12 na warstwę. Pełna konfiguracja została przedstawiona w tabeli poniżej:

Parametr	Wartość	Opis
`vocab_size`	50257	Rozmiar słownika tokenów (liczba unikalnych tokenów)
`max_length`	300	Maksymalna długość wejściowej sekwencji (w tokenach)
`context_length`	300	Liczba tokenów kontekstu, które model bierze pod uwagę
`embedding_dim`	768	Wymiar przestrzeni embeddingów
`n_layers`	12	Liczba warstw transformera (bloków encoder/decoder)
`n_heads`	16	Liczba głów w mechanizmie multi-head attention
`dropout`	0.15	Współczynnik dropout (regularizacja)
`qkv_bias`	False	Czy dodawać bias (przesunięcie) w warstwach Q, K i V

Model ten następnie został wytrenowany na zbiorze 10 książek by móc nauczyć kontekstu zdań oraz słów. Poniżej został przedstawiony wykres przebiegu funkcji Loss, a także próbka wygenerowanego tekstu.




```
[12] 2h 7m
Ep 15/15 (Step 724/1334 [54.27%]): Train loss 0.733, Val loss 5.484
Ep 15/15 (Step 774/1334 [58.02%]): Train loss 0.753, Val loss 5.427
Ep 15/15 (Step 824/1334 [61.77%]): Train loss 0.695, Val loss 5.412
Ep 15/15 (Step 874/1334 [65.52%]): Train loss 0.762, Val loss 5.438
Ep 15/15 (Step 924/1334 [69.27%]): Train loss 0.870, Val loss 5.435
Ep 15/15 (Step 974/1334 [73.01%]): Train loss 0.789, Val loss 5.392
Ep 15/15 (Step 1024/1334 [76.76%]): Train loss 0.888, Val loss 5.399
Ep 15/15 (Step 1074/1334 [80.51%]): Train loss 0.582, Val loss 5.433
Ep 15/15 (Step 1124/1334 [84.26%]): Train loss 0.685, Val loss 5.391
Ep 15/15 (Step 1174/1334 [88.01%]): Train loss 0.868, Val loss 5.384
Ep 15/15 (Step 1224/1334 [91.75%]): Train loss 0.612, Val loss 5.407
Ep 15/15 (Step 1274/1334 [95.50%]): Train loss 0.701, Val loss 5.379
Ep 15/15 (Step 1324/1334 [99.25%]): Train loss 0.705, Val loss 5.408
Hello, I suppose you know, that the time I told he would have a right good captain to the guidance of a woman of a daughters."
```

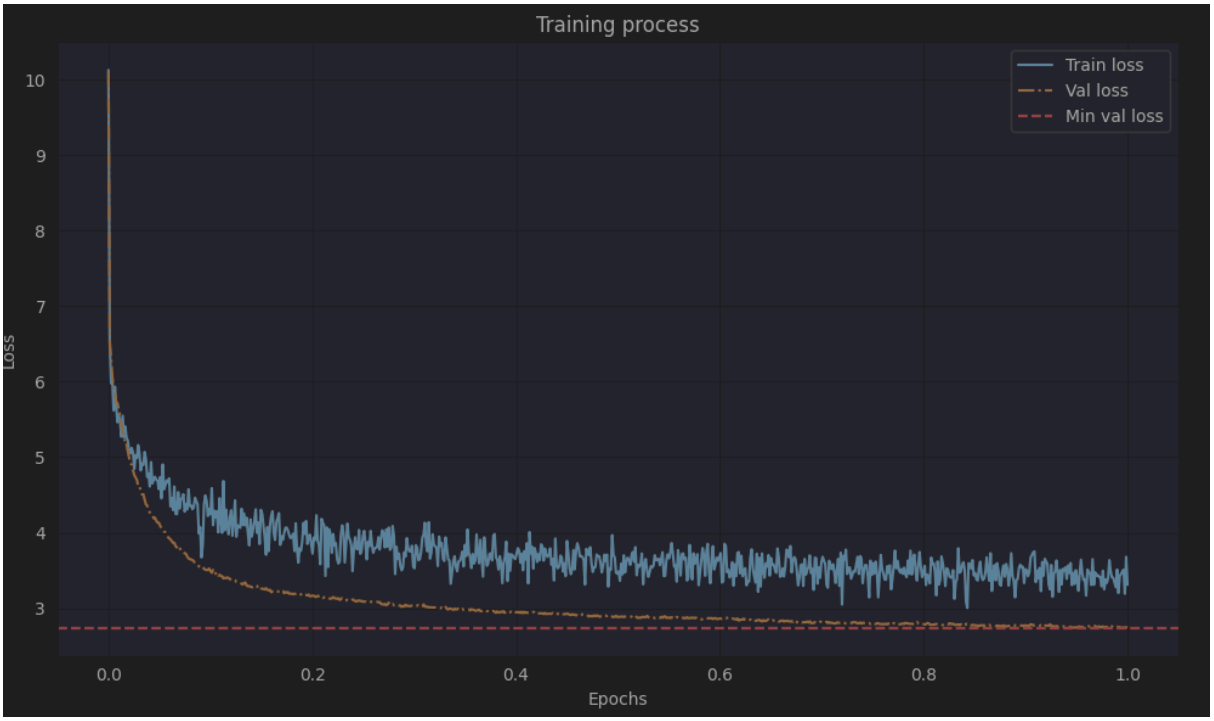
Model ten następnie został wykorzystany do stworzenia nowego modelu tym razem już do analizy sentymentu jako klasyfikator za pomocą techniki Fine-Tuning. Wszystkie warstwy modelu podstawowego zostały zamrożone a następne ostatni blok z warstwy transformera oraz ostatnia warstwa normalizacji została odmrożona. Na koniec warstwa wyjściowa została zmieniona w taki sposób żeby model zwracał wektor prawdopodobieństwa danej klasy (w tym przypadku wektor 2 elementowy bo były 2 klasy od przewidzenia - możliwe było również wykorzystanie wektora 1 elementowego). Taki model następnie został wytrenowany na naszym przygotowanym zbiorze sentymentów. Pomimo początkowo obiecujących wynikach generowanego tekstu podstawowego modelu, model do analizy sentyment nie radził sobie w żadnym stopniu z klasyfikowaniem wiadomości. Dokładność tego klasyfikatora na zbiorze uczącym, walidacyjnym oraz testowym wynosiła około 50%

co pokazuje że model mógł równie dobrze losowo zgadywać i osiągnął by zbliżoną dokładność niż wytrenowany.

Z tego powodu został wytrenowany nowy model podstawowy którego parametry zostały jeszcze bardziej okrojone ale był uczony na znacznie większym zbiorze. Początkowo zakładane było uczenie modelu na zbiorze 10000 książek jednak czas uczenia tego modelu wynosiłby około 30h. Dlatego zbiór książek został ograniczony do 1000 pozycji przez co model uczył się przez około 3.5h. Parametry nowego modelu są następujące:

Parametr	Wartość	Opis
`vocab_size`	50257	Rozmiar słownika tokenów (liczba unikalnych tokenów)
`max_length`	300	Maksymalna długość wejściowej sekwencji (w tokenach)
`context_length`	300	Liczba tokenów kontekstu, które model bierze pod uwagę
`embedding_dim`	384	Wymiar przestrzeni embeddingów
`n_layers`	8	Liczba warstw transformera (bloków encoder/decoder)
`n_heads`	8	Liczba głów w mechanizmie multi-head attention
`dropout`	0.1	Współczynnik dropout (regularizacja)
`qkv_bias`	False	Czy dodawać bias (przesunięcie) w warstwach Q, K i V

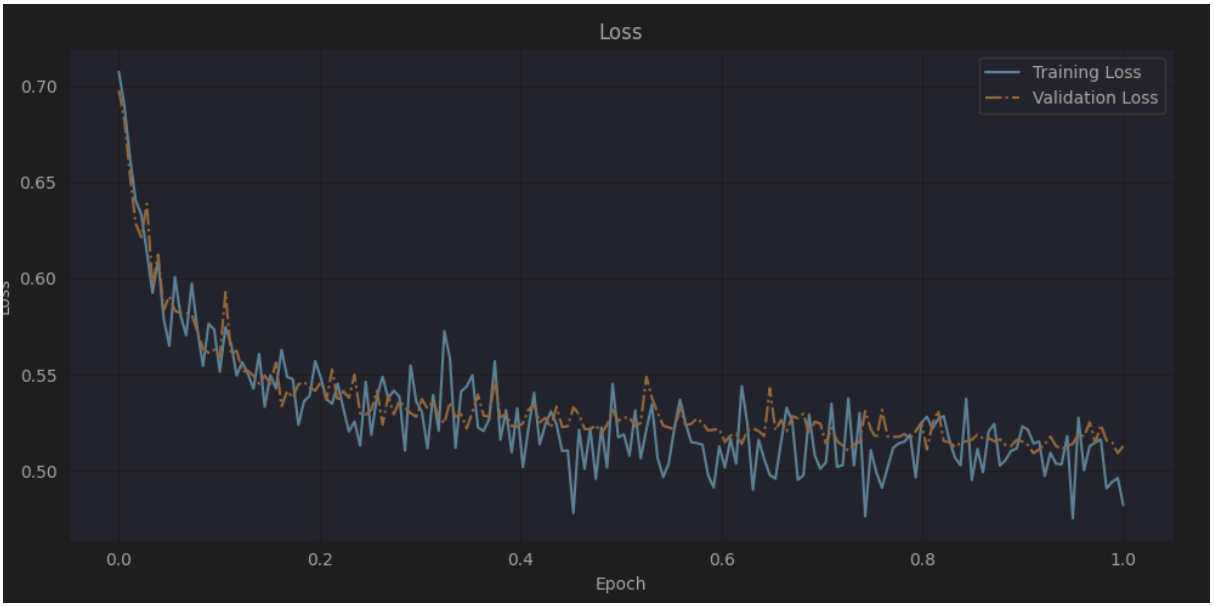
Model był trenowany przez jedną epokę, a przebieg funkcji loss wygląda następująco:

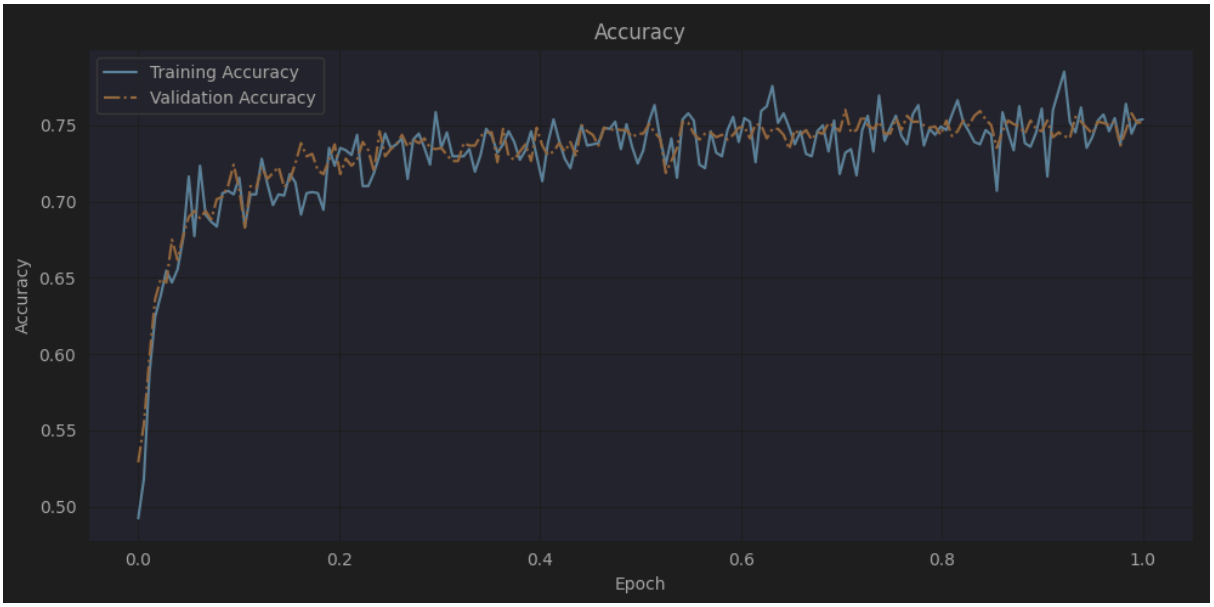


Można zauważyć że w porównaniu do pierwszego bazowego modelu funkcja straty osiąga niższe wartości, co oznacza że model jest pewniejszy swoich decyzji.

Także próbka wygenerowanego tekstu jest (subiektywnie) lepszej jakości:
I am ashamed, because we have the secret which had made us to give the old people an hour of age. We shall not believe that we cannot see a new man again,--but our friends will have gone far and a month, and we shall be able to put a dead man down at our own home. When we were thinking what to call him would become old we came together; and as soon as we were going, I came to him for

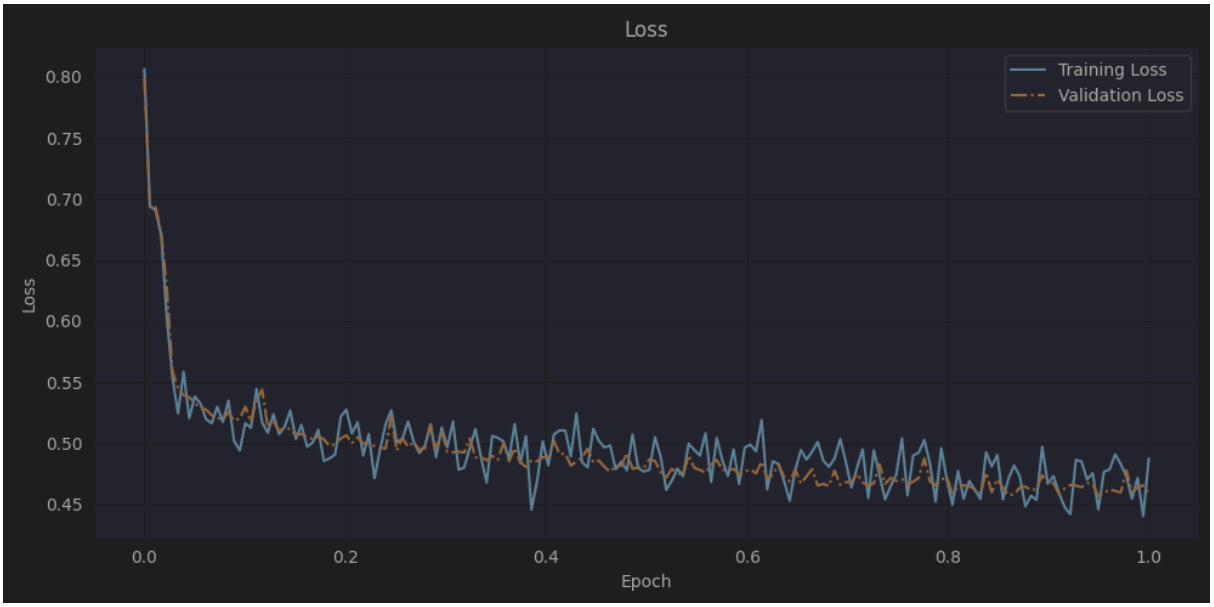
Następnie analogicznie jak dla pierwszego modelu, ten model został przerobiony na klasyfikator. Po wyuczeniu na tym samym zbiorze danych model ten osiągnął znacznie lepsze wyniki, osiągając na zbiorze treningowym oraz walidacyjnym dokładność około 75%.

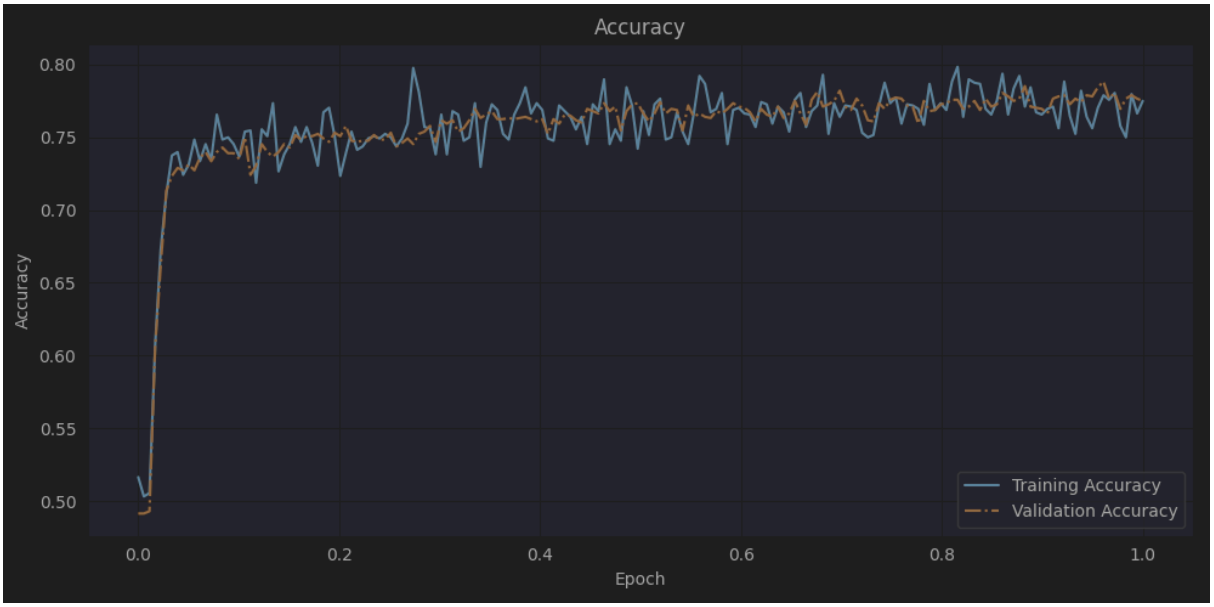




Natomiast na zbiorze testowym model osiągnął dokładność 74.29%. Wynik okazał się być zaskoczeniem ponieważ zakładano że model który został jeszcze bardziej okrojony nie osiągnie dokładności lepszej niż 60%.

Na koniec został dodatkowo stworzony ostatni model. Model ten posiada dokładnie taką samą konfigurację jak model GPT-2 Small (124M parametrów). W tym przypadku również model został przebudowany jako klasyfikator analogicznie jak w 2 poprzednich przypadkach, a następnie nauczony na tym samym zbiorze danych. Ostatecznie wyniki trenowania tego modelu zostały zaprezentowane na wykresach.





Model ten osiągnął dokładność około 78% zarówno na zbiorach treningowym i uczącym jak i również testowym.

Niestety z powodu tego jak długo te modele potrzebowały czasu na uczenie, nie zostało zbadanie, jaki ma wpływ dodanie warstwy encodera na analizę sentymentu. Warte uwagi byłoby również sprawdzenie modelu wyuczonego na zakładanym początkowo zbiorze 10000 książek.

8. Podsumowanie

W ramach projektu przeprowadzono analizę sentymentu tweetów przy użyciu różnych architektur sieci neuronowych: ANN, CNN, LSTM, GRU oraz transformerów. Wykorzystano zestaw danych Sentiment140, zawierający ponad 1,6 miliona tweetów. Przed treningiem dane poddano pre-processingowi – usunięto m.in. linki, znaki interpunkcyjne oraz słowa stop.
Dla każdej architektury przygotowano osobny model i dokonano jego trenowania oraz ewaluacji. Sieć ANN osiągnęła dokładność ok. 78%, jednak była podatna na przeuczenie. Sieć CNN, zaadaptowana z modelu do rozpoznawania cyfr, uzyskała dokładność 75,6%. LSTM w różnych konfiguracjach osiągało najlepsze wyniki – do 79,8% dokładności przy zastosowaniu dwukierunkowych warstw LSTM. Natomiast model transformera stworzony od zera osiągnął skuteczność ok. 75%, a model transformera z wyuczonymi wagami osiągnał skuteczność ok. 78%.
Wyniki wskazują, że sieci rekurencyjne (szczególnie LSTM) najlepiej sprawdzają się w analizie sekwencji tekstowych, podczas gdy prostsze architektury, jak ANN czy CNN, mogą mieć ograniczoną skuteczność w tego typu zadaniach.

9. Udział poszczególnych osób w dany etap projektu

	Literatura	Przygotowanie danych	ANN	CNN	LSTM	GRU	Transformery	Dokumentacja
Bartłomiej Barszczak	25%						100%	14%
Janusz Chmiel				100%				14%
Michał Cynarski	25%					100%		14%
Tomasz Mikurda	25%	100%						14%
Damian Ogorzały			100%					14%
Patryk Smajdor					67%			14%
Adam Trybus	25%				33%			14%

10. Literatura

[1] Kumar, J N V R Swarup & Sri, J. & Manoj, K. & Srividya, D. & Prathyusha, D. & Kumar, M Naga. (2022). Sentiment Analysis on Textual Tweets using Ensemble Classifier (LSTM-GRU). 926-932. 10.1109/ICCES54183.2022.9835850.

[2] Pedipina, Seenaiah & Shanmuganathan, Sankar & Ranganayakulu, Dhanalakshmi. (2020). Sentiment Analysis of Twitter Political Data using GRU Neural Network. 5307-5320.

[3] https://www.kaggle.com/code/kritanjalijain/twitter-sentiment-analysis-lstm

[4] https://www.kaggle.com/code/abanoubyounanh/sentiment-analysis-rnn-lstms-gru

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. arXiv preprint arXiv:1706.03762. https://arxiv.org/abs/1706.03762

[6] Raschka, S. (2024). Build a Large Language Model (From Scratch). Retrieved from https://sebastianraschka.com/books/#build-a-large-language-model-from-scratch

[7]https://javilopezcastillo.medium.com/sentiment-analysis-using-lstm-networks-a-deep-dive-into-textual-data-61cdd2e43dec