



Maciej Bujalski

Kierunek: informatyka

Specjalność: informatyka stosowana

Specjalizacja: bazy danych

Numer albumu: 386012

Validacja wyników pomiarów meteorologicznych z wykorzystaniem rekurencyjnych sieci neuronowych

Praca inżynierska

wykonana pod kierunkiem
dr Krzysztofa Podlaskiego
w Katedrze Systemów Inteligentnych,
WFiIS UŁ

Łódź 2023

Spis treści

WSTĘP	3
CEL PRACY	3
ROZDZIAŁ 1. PODSTAWOWE ZAGADNIENIA W ZAKRESIE KLASYFIKACJI I SIECI NEURONOWYCH ...	4
1.1. KLASYFIKACJA	4
1.2. ISTNIEJĄCE NARZĘDZIA KLASYFIKACJI	4
1.3. WYBÓR NARZĘDZIA NA PODSTAWIE ANALIZY	4
1.4. SIECI NEURONOWE.....	5
1.5. REKURENCYJNE SIECI NEURONOWE.....	5
1.6. KOMÓRKI LTSM	6
1.7. KOMÓRKI GRU	8
1.8. SIECI GĘSTE	9
ROZDZIAŁ 2. ŚRODOWISKO PRACY	10
2.1. OMÓWIENIE ŚRODOWISKA PRACY.....	10
2.2. WYMAGANIA SPRZĘTOWE I PROGRAMOWE.....	10
2.3. PYTHON	10
2.4. OMÓWIENIE BIBLIOTEK UŻYTYCH DO UCZENIA MASZYNNEGO.....	11
2.5. JUPYTER NOTEBOOK	11
2.6. NVIDIA CUDA TOOLKIT	11
2.7. NVIDIA CUDNN.....	11
ROZDZIAŁ 3. DANE UŻYTE DO UCZENIA SIECI	12
3.1. OMÓWIENIE DANYCH.....	12
3.2. POZYSKANIE DANYCH.....	14
3.3. PRZYGOTOWANIE DANYCH.....	16
3.4. PRZETWORZENIE DANYCH	17
3.5. ZBIÓR TRENINGOWY	17
3.6. ZBIÓR WALIDACYJNY	17
ROZDZIAŁ 4. SIEĆ NEURONOWA SŁUŻĄCA DO KLASYFIKACJI Z UŻYCIEM KOMÓREK GRU	18
4.1. ZASTOSOWANIE SIECI	18
4.2. ARCHITEKTURA SIECI	18
4.3. PORÓWNANIE MODELI SIECI.....	20
4.3.1. Parametry modelu nr 1	20
4.3.2. Wyniki dla modelu nr 1	21
4.3.3. Parametry modelu nr 2	22
4.3.4. Wyniki dla modelu nr 2	23
4.3.5. Parametry modelu nr 3	24
4.3.6. Wyniki dla modelu nr 3	25
4.3.7. Podsumowanie modeli i ich wyników	26
4.4. PORÓWNANIE OPTYMALIZATORÓW SIECI	27
4.4.1. Optymalizator nr 1 – Optymalizator Adam.....	27
4.4.2. Wyniki dla optymalizatora nr 1 - Optymalizator Adam	28
4.4.1.1. Optymalizator nr 1.1 - Optymalizator Adam z użyciem zmienionego modelu.....	29
4.4.1.2. Wyniki dla optymalizatora nr 1.1 - Optymalizator Adam z użyciem zmienionego modelu.....	30
4.4.2. Optymalizator nr 2	31

4.4.3.	<i>Wyniki dla optymalizatora nr 2</i>	32
4.4.4.	<i>Optymalizator nr 3</i>	33
4.4.5.	<i>Wyniki dla optymalizatora nr 3</i>	34
4.4.6.	<i>Podsumowanie optymalizatorów i ich wyników</i>	35
4.5.	PORÓWNANIE POZOSTAŁYCH HIPERPARAMETRÓW SIECI.....	36
4.5.1.	<i>Hiperparametr nr 1 - Funkcja Aktywacji Komórek GRU</i>	36
4.5.1.	<i>Hiperparametr nr 2 – Metryka</i>	40
4.5.2.	<i>Hiperparametr nr 3 - Parametr shuffle</i>	44
4.5.3.	<i>Hiperparametr nr 4 - Funkcja błędu</i>	48
4.5.4.	<i>Hiperparametr nr 5 - Wielkość batcha</i>	53
4.5.5.	<i>Hiperparametr nr 6 - Liczba epok</i>	63
4.5.6.	<i>Podsumowanie pozostałych hiperparametrów i ich wyników</i>	69
4.6.	PORÓWNANIE RÓŻNYCH WIELKOŚCI OKNA CZASOWEGO	69
4.6.1.	<i>Okno czasowe nr 1</i>	69
4.6.2.	<i>Wyniki dla okna czasowego nr 1</i>	71
4.6.3.	<i>Okno czasowe nr 2</i>	72
4.6.4.	<i>Wyniki dla okna czasowego nr 2</i>	74
4.6.5.	<i>Okno czasowe nr 3</i>	75
4.6.6.	<i>Wyniki dla okna czasowego nr 3</i>	76
4.6.7.	<i>Podsumowanie okien czasowych i ich wyników</i>	77
4.7.	POZOSTAŁE PRZETESTOWANE MODELE SIECI.....	77
4.8.	ZASTOSOWANE NAZEWNICTWO PLIKÓW	78
4.9.	WYBÓR NAJLEPSZEGO MODELU SIECI Z NAJLEPSZYM JEJ HIPERPARAMETRAMI I ROZMIAREM OKNA	79
4.10.	PODSUMOWANIE KLASYFIKACJI BINARNEJ WYNIKÓW POMIARÓW EMISJI CO ₂ NA TERENACH BAGIENNYCH Z UŻYCIEM KOMÓREK GRU	81
ROZDZIAŁ 5. WPŁYW GPU NA WYDAJNOŚĆ UCZENIA MODELU		82
5.1.	PROCEDURA TESTOWA.....	82
5.2.	KONFIGURACJE TESTOWE	83
5.3.	UŻYCIE GPU	83
5.4.	WYNIKI DLA KONFIGURACJI NR 1	84
5.5.	WYNIKI DLA KONFIGURACJI NR 1.1	85
5.6.	WYNIKI DLA KONFIGURACJI NR 2	86
5.7.	WYNIKI DLA KONFIGURACJI NR 2.1	88
5.8.	WYNIKI DLA KONFIGURACJI NR 3	88
5.9.	PODSUMOWANIE I WNIOSKI	90
ROZDZIAŁ 6. PODSUMOWANIE		93
6.1.	ZBIÓR UCZĄCY ORAZ WALIDACYJNY	93
6.2.	OSIĄGNIĘTA SKUTECZNOŚĆ MODELU ORAZ WYBRANY MODEL	93
6.3.	OSIĄGNIĘTA SZYBKOSĆ UCZENIA MODELU ORAZ WYBRANY MODEL	93
6.4.	NAPOTKANE TRUDNOŚCI	94
6.5.	DALSZY ROZWÓJ.....	94
BIBLIOGRAFIA		95

WSTĘP

Dane otaczają nas ze wszystkich stron, ale nie wszystkie jesteśmy w stanie zbadać i sprawdzić bez błędów czy zafałszowań. Ich użyteczność jest ogromna pozwala sprawdzić m.in. klimat, modele klimatyczne, przewidywać pogodę, warunki na drodze, rozwój populacji fauny i flory na danym terenie itp. W badaniach przyrody na przełomie wieków wiele czasu poświęcano na odczytywanie danych z czujników analogowych np. termometrów rtęciowych lub alkoholowych. Za każdym razem o określonej godzinie trzeba było odczytać jaka jest temperatura, pomimo ciężkich niesprzyjających warunków. W drugiej dekadzie XX i na przełomie XX i XXI wieku do użytku weszły termometry oraz czujniki elektroniczne, satelity. Znacznie ułatwiły one pracę badaczy. Teraz nie trzeba być w danym miejscu by odczytać dane, można to zrobić zdalnie. Często jest tak, że chcąc zmierzyć przykładowo temperaturę, jesteśmy zdani na to czy termometr elektroniczny będzie działał poprawnie bez zafałszowań. Czasami bywa tak, że czujnik odmówi posłuszeństwa lub zacznie podawać zafałszowane dane np. o 2 stopnie Celsiusa. Sam się kiedyś z tym spotkałem używając amatorskiej stacji pogodowej, gdzie w wyniku postępującej korozji na laminacie płytki drukowanej czujnika, pomiar zaczął być czasami zafałszowany o 1-2 stopnie Celsiusa, by po chwili wracać do normy. Jednak w pewnym momencie nagle czujnik zaczął podawać informację o 50 stopniach Celsiusa w jesienny wieczór. Jednakże, nie zawsze jest tak, że pomiar będzie ciągle zafałszowyany, gdyż zdarzają się sytuacje, że jakieś ciało obce zabrudzi czujnik np. w przypadku laserowego czujnika smogu wystarczy mgła by dane były niepoprawne. Na podstawie danych w przeszłości możemy określić czy czujnik się myli czy też jest to anomalia, która nieznacznie odbiega od modelu. Przytoczony powyżej czujnik temperatury pokazuje, iż na podstawie odczucia fizycznego przez moje ciało i pomiary w poprzednich latach byłem w stanie stwierdzić, iż 50 stopni Celsiusa to nie jest poprawny pomiar, ale w przypadku danych zafałszowanych o 1-2 stopnie Celsiusa było już trudniej, gdyż musiałem dokonać porównania pomiarów dwóch czujników obok siebie. W przypadku, gdybym dane porównał z danymi z lat poprzednich również bym dokonał odkrycia, że czujnik czasami zafałszowuje pomiary. Wykrycie tego wymagałoby dużej ilości obliczeń i sprawdzania ręcznie. Na szczęście z pomocą służą Rekurencyjne Sieci Neuronowe.

CEL PRACY

Celem pracy jest zbudowanie systemu opartego o rekurencyjną sieć neuronową klasyfikującą jakość danych pomiarowych. Na potrzeby pracy został wykorzystany zbiór danych związany z pomiarami parametrów meteorologicznych na terenie Biebrzańskiego Parku Narodowego. Wspomniane pomiary obarczone są znaczną niepewnością i kategoryzowane są jako wyniki wysokiej lub niskiej jakości. Zbudowana została rekurencyjna sieć neuronowa pozwalająca na klasyfikację, do której ze wspomnianych kategorii należy konkretny pomiar. Wykorzystane zostały rekurencyjne neuronowe sieci typu GRU. Zbadane zostało, jak zmienia się efektywność klasyfikacji w zależności od wielkości okna czasowego, architektury sieci czy jej hierparametrów. Zbadany został również wpływ karty graficznej na szybkość uczenia oraz wpływ konfiguracji sprzętowej na wyniki. Część praktyczna pracy została przygotowana w języku Python z użyciem biblioteki-tensorflow.

ROZDZIAŁ 1. Podstawowe zagadnienia w zakresie klasyfikacji i sieci neuronowych

1.1. Klasyfikacja

Klasyfikacja danych jest to podzielenie zbioru danych na klasy o określonych cechach. Ilość klas może być z góry zdefiniowana (chcemy podzielić odgórnie zbiór na ileś klas) lub też może być określona na podstawie danego zbioru danych metodą klasteryzacji. Jeżeli ilość klas jest większa, niż dwie jest to klasyfikacja wieloklasowa. Dane można klasyfikować również na dwie klasy o określonych parametrach, jest to wtedy klasyfikacja binarna. Klasyfikacja binarna jest stosowana przykładowo m.in. celem oddzielenia danych wysokiej jakości od tych które mogą być niskiej jakości (zafałszowane lub nie wiarygodne w badanach naukowych). Innym przykładem klasyfikacji binarnej jest podział produktów na linii na spełniające wymagania i te, które wymagań nie spełniają (są wadliwe lub nie spełniają norm jakościowych). Najważniejsze w klasyfikacji binarnej i wieloklasowej są precyza i pełność. Skuteczność danego klasyfikatora jest również sprawdzana przy użyciu tablicy pomyłek, która przedstawia tabela 1. Każdy wiersz w tablicy pomyłek oznacza klasę, do której powinien trafić dany obiekt, zaś kolumna oznacza klasę przewidzianą według klasyfikatora. Mamy 4 informacje na temat przypadków, które zostały poklasyfikowane: prawdziwie pozytywne — PP, prawdziwie negatywne — PN, fałszywie pozytywne — FP, fałszywie negatywne — FN. Precyza jest to iloraz PP przez sumę PP i FP. Pełność jest to iloraz PP przez sumę PP i FN.

Tabela 1. Tablica pomyłek

		Stan faktyczny	
		Prawda	Fałsz
Stan przewidziany	Prawda	PP	FP
	Fałsz	FP	PN

1.2. Istniejące narzędzia klasyfikacji

W klasyfikacji danych najczęściej stosowane są metody: metoda K-najbliższych sąsiadów[1], Drzewa decyzyjne [2], Las Losowy [3], naiwna klasyfikacja Bayesa[4], Liniowa Analiza Dyskryminacyjna [5], Maszyny Wektorów Nośnych[6], klasyfikacja z użyciem sieci neuronowych, np. wykorzystując komórki LSTM[7].

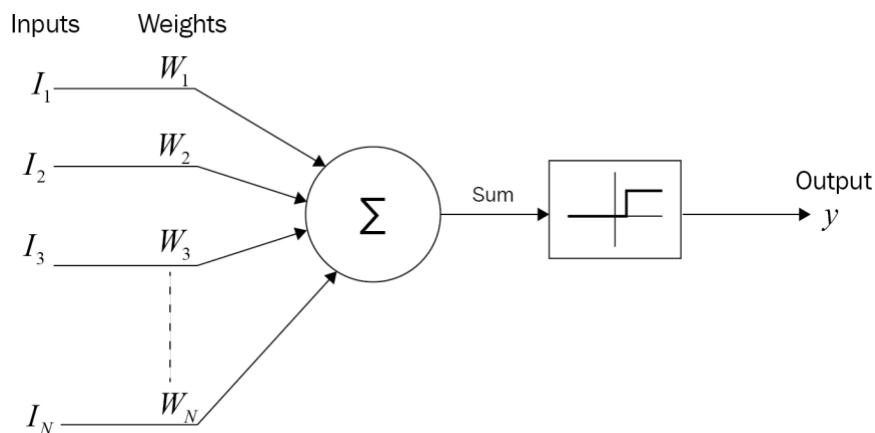
1.3. Wybór narzędzia na podstawie analizy

Sieci neuronowe są ostatnio bardzo często opisywane w Internecie i pracach naukowych, w szczególności rekurencyjne sieci neuronowe, stąd wybór padł właśnie na nie zamiast np. naiwnej klasyfikacji Bayesa czy na drzewa decyzyjne. Wybrana została klasyfikacja z użyciem komórek GRU, gdyż można jej użyć w wypadku, gdy nie posiadamy etykiet i na podstawie nauczonych wcześniej wyników jesteśmy w stanie sklasyfikować dane. Ponadto komórki GRU są szybsze niż komórki LSTM. Posiadają dwie bramki zaś LSTM jedną. Dodatkowo wybrane komórki GRU dobrze się skalują z kartami graficznymi, dzięki czemu jest możliwe szybsze nauczenie modeli i ich dopracowanie celem uzyskania lepszych rezultatów. Ponadto klasyfikacja tych wyników z użyciem sieci LSTM została już wykonana [8].

1.4. Sieci neuronowe

Sieci neuronowe są to połączone warstwy sztucznych neuronów stosowane w uczeniu maszynowym. Sztuczne neurony składają się z wejść, na które otrzymują sygnały wejściowe, sumatora, elementu z funkcją aktywacji i jednego wyjścia. Modelem matematycznym, który najczęściej opisuje sztuczny neuron jest model neuronu McCullocha-Pittsa[9] (rysunek 1). Na rysunku tym można zobaczyć jeden neuron, który posiada N wejść o N wagach W_N. Po przemnożeniu wejść przez ich odpowiadające wagi trafiają one do sumatora, gdzie są sumowane, a następnie przekazywane do bloku z funkcją aktywacji, która decyduje co ma zwrócić neuron na wyjściu y.

Rysunek 1 Neuron McCullocha-Pittsa



Źródło: <https://www.oreilly.com/library/view/artificial-intelligence-by/9781788990547/97eeab76-9e0e-4f41-87dc-03a65c3efec3.xhtml>

Sieci neuronowe mogą się składać z następujących warstw: wejściowej, ukrytej, wyjściowej. Warstwa wejściowa przyjmuje dane wejściowe np. informacje z czujników, po czym po zsumowaniu i użyciu funkcji aktywacji przekazuje je warstwie ukrytej, która decyduje i przelicza dane otrzymane od poprzedniej warstwy i przekazuje je albo kolejnej warstwie ukrytej albo jeżeli jest to ostatnia warstwa ukryta warstwie wyjściowej. Warstwa wyjściowa po otrzymaniu danych wydaje decyzje na ich podstawie i zwraca tą decyzję jako ostateczny wynik np. czy dane z czujników są poprawne czy też nie.

1.5. Rekurencyjne sieci neuronowe

Rekurencyjne sieci neuronowe (ang. Recurrent Neural Networks, RNNs) są to sieci, które wykorzystują dane w sposób sekwencyjny, zależny od siebie (w przypadku klasycznych sieci neuronowych każde informacje niezależne jak też dane wchodzące są niezależne od wychodzących). Rekurencyjna sieć neuronowa różni się od tradycyjnej możliwością propagacji wstępnej sygnałów do wejść poprzednich neuronów. Pozwala to na przewidzenie wyniku na podstawie danych poprzedzających. RNN dzięki pamięci przechowują dane o poprzednich wynikach sieci. Rekurencyjne sieci potrafią zgłębiać szeregi czasowe np. dane pogodowe, dzięki czemu są w stanie przewidzieć czy będzie słonecznie czy też będzie padało. Długość szeregów czasowych nie jest istotna dla RNN dzięki czemu szeregi czasowe mogą mieć różną długość. Przykładowym zastawaniem rekurencyjnych sieci neuronowych jest przewidywanie zjawisk występujących okresowo np. pogoda lub hossa i bessy na giełdzie. Problemem rekurencyjnych sieci neuronowych jest bardzo ograniczona pamięć krótkotrwała

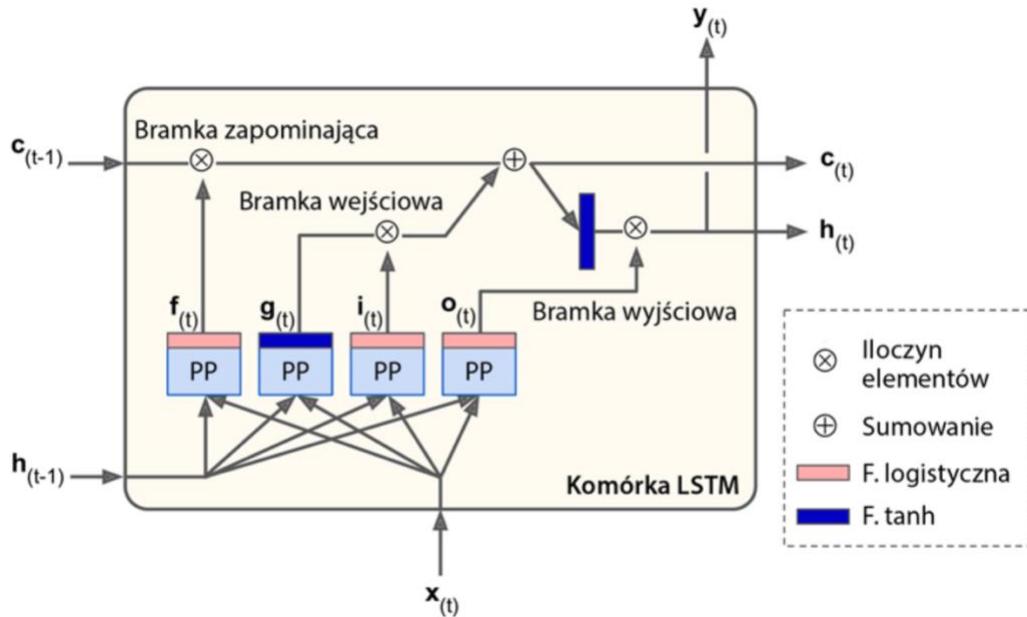
wskutek, której sieć może zapomnieć tego co się nauczyła na początku, a którą można polepszyć korzystając z komórek LTSM lub GRU [10].

1.6. Komórki LTSM

Komórka LSTM (ang. Long Short-Term Memory) opracowana przez S. Hochreitera i J. Schmidhubera w 1997 roku [11]. Komórka LSTM od zwykłej komórki różni się tym, że ma swoją pamięć, jest wydajniejsza, pozwala na szybszą naukę modelu i wykrycie zależności długookresowych. Ideą tych komórek fakt, iż sieć z nich złożona jest uczona co odrzucać, a ma co zostawiać ze stanów długotrwałych. Schemat budowy komórek LTSM przedstawia rys. 2. Możemy na nim dostrzec, że komórka LTSM ma dwa wektory stanowe: $c_{(t)}$ (długotrwały) i $h_{(t)}$ krótkotrwały. Stan długotrwały z poprzedniego kroku czasowego $c_{(t-1)}$ jest podawany na bramkę zapominającą AND, gdzie część wspomnień jest zapominana, a następnie jest kierowany na bramkę OR gdzie przychodzą nowe wspomnienia przefiltrowane przez bramkę wejściową OR, które są podawane dalej do kolejnej komórki jako stan długotrwały z obecnego kroku czasowego $c_{(t)}$, a jego kopia po zastosowaniu funkcji tangensa hiperbolicznego jest sumowana w bramce wyjściowej OR z sygnałem $o_{(t)}$ (filtrowanie), a następnie podawany jako wyjście komórki $y_{(t)}$ oraz jako stan krótkotrwały z obecnego kroku czasowego $h_{(t)}$. Nowe wspomnienia biorą się z pełni połączonych warstw (na rys.2 oznaczone skrótem PP). Pełni połączone warstwy otrzymują na wejściu wektory: krótkotrwały stanu poprzedniego $h_{(t-1)}$ oraz wejściowy $x_{(t)}$. PP składają się z warstwy głównej (wylicza ona wartości wektora $g(t)$ na podstawie wektorów $x(t)$ i $h(t-1)$, posiada ona funkcję aktywacji tangensa hiperbolicznego. W odróżnieniu od standardowej komórki wynik $g(t)$ nie jest przekazany do wektorów $y(t)$ i $h(t)$, natomiast zamiast tego istotna część $g(t)$ trafia do stanu długotrwałego zaś reszta odrzucona) oraz kontrolerów bramek wykorzystujących jako funkcję aktywacji funkcję logistyczną, której wartości min i max wynoszą odpowiednio 0 i 1. Wykorzystując te wartości wektor $f(t)$ steruje z wykorzystaniem iloczynu elementowego bramką zapominającą (0 bramka zamknięta, 1 otwarta) usuwając przy tym niepotrzebne fragmenty pamięci długotrwałej, wektor $i(t)$ steruje bramką wejściową dodając fragmenty wektora $g(t)$ do wyżej wymienionej pamięci, zaś wektor $o(t)$ bramką wyjściową co ma być wysłane z wektora stanu długotrwałego do wektorów $y(t)$ i $h(t)$. Wzory na wyliczenie $f_{(t)}$, $g_{(t)}$, $i_{(t)}$, $o_{(t)}$, $c_{(t)}$, $y_{(t)}$ przedstawia rys. 3, gdzie cytując za autorem Aurélienem Géronem z książki Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow [13] ":"

- W_{xi} , W_{xf} , W_{xo} , W_{xg} to macierze wag każdej z czterech warstw dla ich połączenia z wektorem wejściowym $x_{(t)}$.
- W_{hi} , W_{hf} , W_{ho} , Whg są macierzami wag każdej z czterech warstw dla ich połączenia z poprzednim stanem krótkotrwałym $h_{(t-1)}$.
- b_i , b_f , b_o , b_g to czlonki obciążenia każdej z czterech warstw" [13]

Rys. 2 Komórka LSTM



Źródło: Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow.

Rysunek 3 Obliczenia dla komórki LSTM [1]

$$\begin{aligned} \mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}^T \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \mathbf{h}_{(t-1)} + \mathbf{b}_i\right) \\ \mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}^T \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \mathbf{h}_{(t-1)} + \mathbf{b}_f\right) \\ \mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}^T \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \mathbf{h}_{(t-1)} + \mathbf{b}_o\right) \\ \mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}^T \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \mathbf{h}_{(t-1)} + \mathbf{b}_g\right) \\ \mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\ \mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)}) \end{aligned}$$

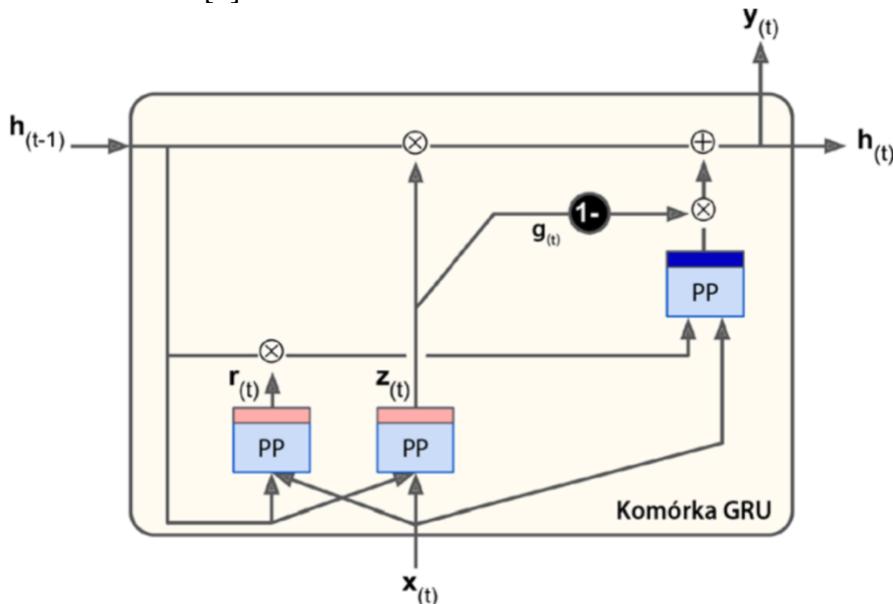
Źródło: Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow.

[1] Symbol δ oznacza sigmę, zaś symbol \otimes oznacza iloczyn elementarny

1.7. Komórki GRU

Komórka GRU została opisana w 2014 roku przez Kyunghuna Cho, Bartę van Merriënboera, Caglara Gulcehra, Dzmitrego Bahdanau, Fethiego Bougaresa, Holgera Schwenka, Yoshue Bengio w dokumencie „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation” [12]. Komórka GRU jest prostszą odmianą komórki LSTM, a przy tym równie wydajną jak LSTM. W odróżnieniu od LSTM ma jeden wektor stanowy ($h_{(t)}$), który jest połączeniem wektora stanowego długotrwałego ($c_{(t)}$) i krótkotrwałego ($h_{(t)}$), a funkcję bramek zapominającej i wejściowej pełni kontroler $z_{(t)}$ widoczny na schemacie komórki GRU, który przedstawia rys.4., używa on logistycznej funkcji aktywacji, gdy na wyjściu kontrolera $z_{(t)}$ jest 1 otwiera on bramkę wejściową (podana zostaje wartość 1), zaś sygnał w sygnale bramki zapominającej zostaje od wartości 1 zostaje odjęta jedynka do daje 0, co przy iloczynie elementarnym zamyka tą bramek. Jako iż podawany jest na wyście pełen wektor stanowy to komórka GRU nie posiada bramki wyjściowej OR. Jego funkcję przejął kontroler $r_{(t)}$, który określa co ze stanu przedającego trafi do $g_{(t)}$ (warstwy głównej). Wzory na wyliczenie $z_{(t)}$, $g_{(t)}$, $r_{(t)}$, $h_{(t)}$ przedstawia rys. 5. Opis zmiennych taki sam jak przy komórkach LSTM.

Rys.4 Komórka GRU [2]



Źródło: Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow.

Rys. 5 Obliczenia dla komórki GRU

$$\begin{aligned}\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}^T \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \mathbf{h}_{(t-1)} + \mathbf{b}_z\right) \\ \mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}^T \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \mathbf{h}_{(t-1)} + \mathbf{b}_r\right) \\ \mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}^T \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right) \\ \mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}\end{aligned}$$

Źródło: Aurélien Géron, *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*.

[1] Symbol δ oznacza sigmę, zaś symbol \otimes oznacza iloczyn elementarny

1.8. Sieci gęste

Sieć gęsta (ang. DNN- Dense Neural Network[14]) jest to sieć składająca się z samych warstw gęstych (Dense layer, inna nazwa to fully connected layer). Warstwa gęsta składa się z neuronów, gdzie informacje z warstwy poprzedniej są przekazywane do wejść wszystkich neuronów jednocześnie i których wyjścia są podawane na wejścia wszystkich neuronów warstwy kolejnej, przy czym każda dana na wejście danego neuronu ma swoją własną wagę. Sieci gęste służą ostatecznej klasyfikacji obiektów wejściowych np. czy wynik jest poprawny czy też nie.

ROZDZIAŁ 2. Środowisko pracy

2.1. Omówienie środowiska pracy

W celu stworzenia, uruchomienia i nauczenia sieci neuronowych potrzebnych do tej pracy użyto języka programowania Python w wersji 3.7 wraz z notatnikami Jupyter, sieci neuronowe były tworzone i uczone z użyciem biblioteki Tensorflow z modułem Keras. Jako system operacyjny użyto Windowsa 11 Pro. Celem usprawnienia uczenia została wykorzystana karta graficzna firmy Nvidia z wykorzystaniem narzędzia Nvidia CUDA Toolkit 11.8 wraz z biblioteką Nvidia cuDNN. Wykorzystuje ona rdzenie CUDA oraz Tensor [15](rdzenie Tensor posiadają karty graficzne Nvidii z rodziny RTX) w celu dokonywania obliczeń dokonywanych przy uczeniu sieci neuronowej [16].

2.2. Wymagania sprzętowe i programowe

Wymagania sprzętowe:

- CPU: Procesor o architekturze x86 ze wsparciem AVX
- 16 GB RAM
- Karta graficzna firmy Nvidia minimum 6 GB vramu (w wypadku braku GPU obliczenia będą trwały wolniej)

Wymagania programowe:

- Notatniki Jupyter, Python 3
- w przypadku kart graficznych Nvidia sterownik graficzny
- Nvidia cuDNN
- Nvidia CUDA Toolkit 11.8
- System operacyjny:
 - Windows 10/11
 - MacOS Ventura 13.0.1
 - Linux, testowane na Ubuntu 20.1

Zalecana konfiguracja

- CPU: 8 rdzeni z HT (8 rdzeni 16 wątków) np. Intel Core i7 11800H lub Intel Xeon e5-2667v2 lub mocniejszy
- 64 GB RAM lub więcej
- GPU: karta graficzna firmy Nvidia z rodziny RTX, w przypadku laptopa zalecany model GPU to Nvidia RTX 3060 6GB GDDR6, w przypadku komputera stacjonarnego zalecany model GPU to Nvidia RTX 3060 12GB GDDR6 lub mocniejszy o takiej samej lub większej ilości vramu
- 20GB wolnej przestrzeni dyskowej (zalecany dysk SSD)

2.3. Python

Python jest językiem programowania wysokiego poziomu stworzonym w 1991 przez Guido van Rossum, jest to język wieloparadygmatowy, z dynamicznym typowaniem zmiennych i automatycznym zarządzaniem pamięcią. Python posiada także mechanizm dziedziczenia w tym dziedziczenie wielokrotne [17]. Posiada on wsparcie wielu bibliotek w tym bibliotek do uczenia maszynowego w tym między innymi Tensorflow. Python jest językiem wieloplatformowym, dzięki czemu jest możliwe używanie go na różnych systemach operacyjnych i różnych architekturach komputerowych.

2.4. Omówienie bibliotek użytych do uczenia maszynowego

Tensorflow jest biblioteką stworzoną przez firmę Google [18] służącą do tworzenia i nauki sieci neuronowych. Kod biblioteki został napisany w języku C++, zaś jako API został użyty język Python. Pozwala on ma tworzenie sieci neuronowych o różnych architekturach.

Keras jest to biblioteka działająca jako interfejs opracowany przez François Cholleta służący ułatwieniu tworzenie modeli sieci neuronowych z wykorzystanie backendu w postaci Tensorflow [19]. Rozwijaniem i wsparciem dla ten biblioteki zajmuje się firma Google.

2.5. Jupyter notebook

Jupyter notebook jest to webowo zorientowane środowisko do tworzenia notatników, w których można uruchamiać kod, pisać tekst, dodawać multimedia, wykresy [20]. Został on budowany na podstawie bibliotek otwarto źródłowych takich jak: IPython, jQuery, Bootstrap, ZeroMQ. Notatniki składają się z komórek wejścia wyjścia, które można uruchamiać w dowolnej kolejności. Do działania Jupytera jest niezbędna przeglądarka lub program, który obsługuje notatniki Jupyter np. Visual Studio Code.

Notatniki Jupitera są plikami tekstowymi w formacie JSON zapisanymi pod rozszerzeniem .ipynb.

2.6. Nvidia Cuda Toolkit

NVIDIA® CUDA® Toolkit [21] jest to zestaw narzędzi od firmy NVIDIA, który zapewnia środowisko programistyczne do tworzenia i uruchamiania aplikacji i programów wymagających dużej wydajności z wykorzystaniem akceleracji GPU. CUDA Toolkit umożliwia opracowywaniem optymalizację i wdrażanie swoje aplikacje w systemach wbudowanych z akceleracją GPU, w stacjach roboczych, korporacyjnych centrach danych, platformach opartych na chmurze i superkomputerach HPC. Zestaw narzędzi zawiera biblioteki akcelerowane przez GPU, narzędzia do debugowania i optymalizacji, kompilator C/C++ oraz bibliotekę wykonawczą do wdrażania aplikacji. Wykorzystując wbudowane możliwości obliczeń rozproszonych w konfiguracjach z wieloma procesorami graficznymi firmy Nvidia, naukowcy i badacze mogą tworzyć aplikacje skalowalne, od stacji roboczych z pojedynczymi GPU, aż po instalacje w chmurze z tysiącami procesorów graficznych. Cuda Toolkit jest też niezbędny do uruchomienia biblioteki cuDNN dzięki czemu jest możliwe uczenie sieci neuronowych z wykorzystaniem procesorów graficznych firmy Nvidia.

2.7. Nvidia cuDNN

NVIDIA CUDA® Deep Neural Network (cuDNN[22] to wspomagana przez GPU biblioteka zawierająca podstawy dla głębokich sieci neuronowych. cuDNN zapewnia wysoko dostrojoną implementacje standardowych procedur, takich jak konwolucja do przodu i do tyłu, łączenie, normalizacja i warstwy aktywacji. cuDNN jest stosowana w uczeniu maszynowym w celu uzyskania wysokiej wydajności dzięki akceleracji GPU. Redukuje to ona czas uczenia i szybkość działania sieci neuronowych oraz przyspiesza szeroko stosowane platformy głębokiego uczenia się, w tym Keras, PyTorch, TensorFlow.

ROZDZIAŁ 3. Dane użyte do uczenia sieci

3.1. Omówienie danych

Dane zastosowane w pracy są to wyniki pomiarów meteorologicznych z okresu 4 lat zebrane na bagnach w Kopytkowie wykonywane w interwale 5 minutowy zawierające następujące informacje, rok, ms, dz, godz, mi, fco2_HQ, fco2_MQ, fco2_raw, glwody, T50cm, T2m, Kd, Ku, Ld, Lu, ppp, vdir, Tgrunt, vwc, PARd, PARu [23]. Informacje na temat ilości niepustych danych dla każdej z tych informacji przedstawia rys 3.1. Informacje na temat danych przedstawiają rys. 3.2.1., rys. 3.2.2. oraz rys. 3.2.3.

Rys 3.1. Informacje o danych uzyskane z użyciem funkcji pandas info()

```
dataToModel.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 525888 entries, 0 to 105407
Data columns (total 22 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   rok         525888 non-null   int64  
 1   ms          525888 non-null   int64  
 2   dz          525888 non-null   int64  
 3   go          525888 non-null   int64  
 4   mi          525888 non-null   int64  
 5   fco2_HQ     122607 non-null   float64 
 6   fco2_MQ     298211 non-null   float64 
 7   fco2_raw    475336 non-null   float64 
 8   glwody     525881 non-null   float64 
 9   T50cm       525888 non-null   float64 
 10  T2m         525888 non-null   float64 
 11  Kd          525888 non-null   float64 
 12  Ku          525888 non-null   float64 
 13  Ld          525888 non-null   float64 
 14  Lu          525888 non-null   float64 
 15  ppp         525888 non-null   float64 
 16  v           525888 non-null   float64 
 17  vdir        525888 non-null   float64 
 18  Tgrunt      525888 non-null   float64 
 19  vwc         525888 non-null   float64 
 20  PARd        525888 non-null   float64 
 21  PARu        525888 non-null   float64 
dtypes: float64(17), int64(5)
memory usage: 92.3 MB
```

Rys 3.2.1. Informacje o danych uzyskane z użyciem funkcji pandas describe() część 1

dataToModel.describe()									
	rok	ms	dz	go	mi	fco2_HQ	fco2_MQ	fco2_raw	glwody
count	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	122607.000000	298211.000000	475336.000000	525881.00
mean	2015.000548	6.523549	15.727820	11.500000	31.500000	-1.015673	-1.243933	-0.25396	-9.699501
std	1.414021	3.448536	8.799333	6.922193	17.260279	4.666485	4.993262	17.02511	20.903174
min	2013.000000	1.000000	1.000000	0.000000	4.000000	-29.060000	-29.720000	-934.05000	-88.10000C
25%	2014.000000	4.000000	8.000000	5.750000	17.750000	-2.050000	-2.550000	-1.70000	-20.70000C
50%	2015.000000	7.000000	16.000000	11.500000	31.500000	0.300000	0.140000	0.24000	-0.600000
75%	2016.000000	10.000000	23.000000	17.250000	45.250000	1.120000	1.010000	1.48000	5.000000
max	2017.000000	12.000000	31.000000	23.000000	59.000000	19.920000	20.150000	965.22000	19.200000
8 rows x 22 columns									

Rys 3.2.2. Informacje o danych uzyskane z użyciem funkcji pandas describe() część 2

dataToModel.describe()									
glwody	T50cm	...	Ku	Ld	Lu	ppp	v	vdir	Tgrunt
525881.000000	525888.000000	...	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000
9.699501	7.315592	...	25.631573	320.909513	356.306919	1002.712240	2.414505	192.125632	8.152936
0.903174	9.101847	...	43.922208	46.352750	50.482287	8.821064	1.524603	95.279027	5.867151
88.100000	-23.300000	...	0.000000	147.800000	211.400000	958.300000	0.000000	0.000000	-0.400000
20.700000	0.600000	...	0.600000	295.700000	318.100000	997.300000	1.200000	109.400000	2.300000
0.600000	6.400000	...	2.300000	322.100000	348.200000	1002.700000	2.200000	209.900000	7.600000
6.000000	13.800000	...	33.900000	352.800000	390.200000	1008.300000	3.400000	264.400000	13.500000
9.200000	35.200000	...	585.700000	466.200000	531.900000	1040.500000	12.200000	360.000000	21.800000

Rys 3.2.3. Informacje o danych uzyskane z użyciem funkcji pandas describe() część 3

dataToModel.describe()									
Ld	Lu	ppp	v	vdir	Tgrunt	vwc	PARd	PARu	
525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000	525888.000000
320.909513	356.306919	1002.712240	2.414505	192.125632	8.152936	0.723257	249.964709	21.739186	
46.352750	50.482287	8.821064	1.524603	95.279027	5.867151	0.122404	424.766554	63.275303	
147.800000	211.400000	958.300000	0.000000	0.000000	-0.400000	0.339000	0.000000	0.000000	
295.700000	318.100000	997.300000	1.200000	109.400000	2.300000	0.661000	0.000000	0.000000	
322.100000	348.200000	1002.700000	2.200000	209.900000	7.600000	0.760000	9.000000	0.500000	
352.800000	390.200000	1008.300000	3.400000	264.400000	13.500000	0.823000	318.000000	22.900000	
466.200000	531.900000	1040.500000	12.200000	360.000000	21.800000	0.900000	2193.100000	1490.700000	

Opis danych:

6. Rok – rok
7. ms – miesiąc
8. dz – dzień
9. go – godzina
10. mi – minuta (tylko w danych co 5min)

11. fco2_HQ – wysokiej jakości dane strumienia CO2

12. fco2_raw – oryginalne dane strumienia CO2

13. glwody – głębokość lustra wody
14. T50cm – temperatura na wysokości 50 cm nad gruntem
15. T2m – temperatura na wysokości 2m
16. Kd – dochodzące całkowite promieniowanie krótkofalowe (widzialne)
17. Ku – odbite promieniowanie krótkofalowe
18. Ld – dochodzące promieniowanie długofalowe (podczerwone)
19. Lu – odbite promieniowanie długofalowe
20. ppp – ciśnienie atmosferyczne
21. v – prędkość wiatru
22. vdir – kierunek wiatru (0 – N, 90 – E, 180 – S, 270 – W)
23. Tgrunt – temperatura gruntu
24. vwc – uwilgotnienie gruntu (volumetric water content)
25. PARd – dochodzące promieniowanie fotosyntetyczne czynne
26. PARu – odbite promieniowanie fotosyntetyczne czynne

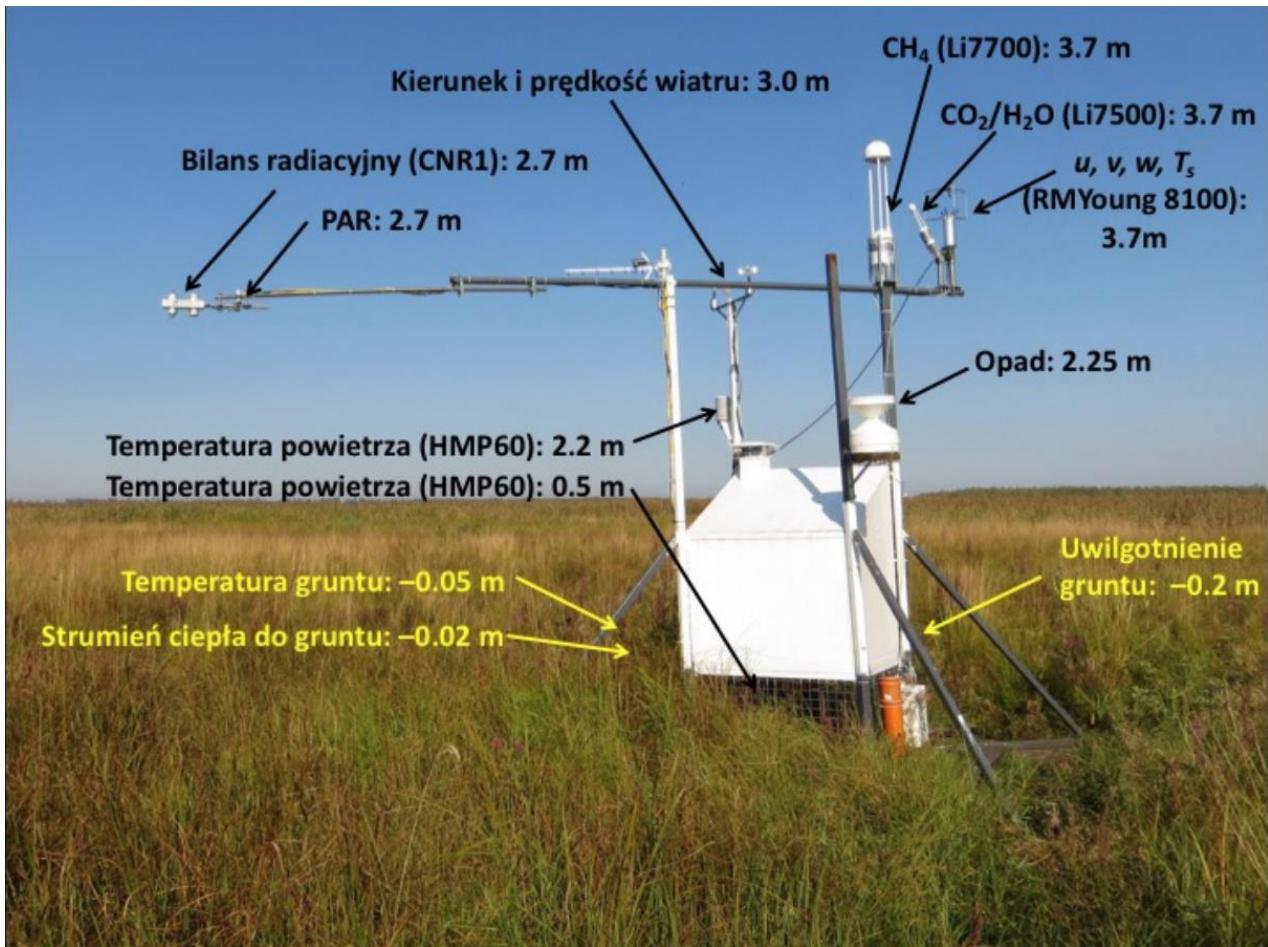
Źródło: „Wybrane Problemy Pomiarów Wymiany Gazowej Pomiędzy Powierzchnią Ziemi A Atmosferą Na Terenach Bagiennych Doświadczenia Trzyletnich Pomiarów W Biebrzańskim Parku Narodowym” prof. dr hab. Krzysztof Fortuniak, et. al.

3.2. Pozyskanie danych

Dane pochodzą z badań prowadzonych przez prof. dr hab. Krzysztofa Fortuniaka na bagnach w Kopytkowie, przez okres 4 lat. Dane były pozyskiwane w następujący sposób cytując pracę „Wybrane Problemy Pomiarów Wymiany Gazowej Pomiędzy Powierzchnią Ziemi A Atmosferą Na Terenach Bagiennych Doświadczenia Trzyletnich Pomiarów W Biebrzańskim Parku Narodowym”, autorstwa prof. dr hab. Krzysztofa Fortuniaka, et. al.: „*Stanowisko pomiarowe w Kopytkowie [...] zbudowane jest na bazie metalowej klatki o wymiarach 1x1 m i wysokości 1,2 m, umieszczonej na 4 wspornikach o wysokości 3 m. Podstawa klatki znajduje się 0,6 m nad poziomem gruntu, a jej górna krawędź w najwyższym punkcie (klatka ma spadzisty dach) na poziomie 1,8 m. Na górze, pomiędzy dwoma wspornikami, po przekątnej, na kierunku północ-południe, zamontowana jest pozioma rura, do której przymocowane są czujniki szybkozmienne i czujniki promieniowania. Część szybkozmienna systemu składa się z anemometru akustycznego RMYoung 81000, pozwalającego na pomiary trzech składowych ruchu powietrza i temperatury, oraz dwóch analizatorów gazowych: Li-7500, rejestrującego stężenie pary wodnej i dwutlenku węgla oraz Li-7700, rejestrującego stężenie metanu[...]. Oba analizatory gazowe pracują w trybie otwartej ścieżki pomiarowej. Analizator Li-7700 umieszczony jest bezpośrednio nad północnym wspornikiem klatki, a analizator Li-7500 oraz anemometr akustyczny przytwierdzone są do poziomej rury odpowiednio w odległości 25 cm i 50 cm od Li-7700 w kierunku północnym.*

Środek ścieżki pomiarowej systemu EC znajduje się na wysokości ok. 3,7 m nad gruntem. Taka lokalizacja czujników minimalizuje wpływ klatki na pomiary turbulencyjne. Pomiarami szybkozmiennymi zarządza rejestrator CR5000 (Campbell Sci.) połączony do komputera przemysłowego TANK-101B (IEI Integration Corp.). Wyniki rejestrowanych z częstotliwością 10 Hz pomiarów składowych ruchu powietrza, temperatury sonicznej, stężenia pary wodnej, dwutlenku węgla i metanu zapisywane są na dysku twardym komputera w plikach 15-minutowych. Pozostałe czujniki stanowią część wolnozmienną systemu pomiarowego. Podłączone są one do tego samego rejestratora CR5000, wartości mierzonych parametrów rejestrowane są co 5 minut i dopisywane do pliku wynikowego na komputerze. Umieszczony na wisierniku w odległości 3,15 m od krawędzi klatki na wysokości 2,7 m radiometr różnicowy CNR1 pozwala na niezależne pomiary krótkofalowego i długofalowego promieniowania dochodzącego od atmosfery do powierzchni Ziemi oraz skierowanego od powierzchni Ziemi do atmosfery. Obok, na tym samym wisierniku, znajduje się para czujników promieniowania fotosyntetycznego aktywnego (PAR), z których jeden jest skierowany w górę, drugi w dół. Sparowanie tych czujników pozwala na wyznaczenie albedo również w tym zakresie promieniowania. Pomiaru składników bilansu cieplnego dopełnia płytka strumienia ciepła do gruntu umieszczona na głębokości 20 cm w odległości ok. 1,5 m od klatki. Informacji o stanie gruntu dostarcza dodatkowo sonda temperatury i sonda uwilgotnienia gruntu. Dwie sondy temperatury i wilgotności powietrza zamontowano na wysokościach 2,2 m i 0,5 m, co umożliwia wyznaczenie gradientów tych parametrów oraz dostarcza informacji zarówno o ich wartościach w wolnym powietrzu (górnna sonda) jak i w na poziomie szaty roślinnej (dolna sonda). Pozostałe mierzone parametry to prędkość i kierunek wiatru, suma opadów oraz ciśnienie atmosferyczne. W celu kontroli działania stacji, komputer, na którym są gromadzone dane jest podłączony do Internetu poprzez modem telefonii komórkowej.” Opisaną powyżej aparaturę pomiarową przedstawia rys. 3.3.

Rys 3.3 Aparatura pomiarowa w Kopytkowie



Źródło: „Wybrane Problemy Pomiarów Wymiany Gazowej Pomiędzy Powierzchnią Ziemi A Atmosferą Na Terenach Bagiennych Doświadczenia Trzyletnich Pomiarów W Biebrzańskim Parku Narodowym” prof. dr hab. Krzysztof Fortuniak, et. al.

3.3. Przygotowanie danych

Pliki tekstowe .txt z danymi meteorologicznymi z lat 2013-2017 są wczytywane do jednego dataframe pandas z użyciem skryptu w Pythonie zawartego w pliku readfiles_thesis.ipynb. Następnie należy się pozbyć wartości NaN (pustych wartości) na wszystkich kolumnach oprócz fco_HQ oraz usunąć wszystkie dane które nie posiadają w danym wierszu wartości fco2_raw. Następnie w oparciu o wartości kolumny fco2_HQ należy utworzyć etykiety do danych w postaci 0 i 1. Etykieta 0 jest przypisywana, gdy wartość jest w kolumnie fco2_HQ jest pusta (wynosi NaN), zaś etykieta równa 1 jest przyznawana, gdy wartość jest niepusta. Dzięki tej informacji jest wiadome czy dany wiersz posiada wiarygodne informacje wysokiej jakości (wtedy dostaje etykietę 1), a gdy informacje nie są wysokiej jakości otrzymuje etykietę 0. Następnie informacje te są zapisywane w kolumnie fco2_HQ nadpisując pierwotne dane w tej kolumnie. Potem tak przygotowany dataframe jest gotowy do przetworzenia na potrzeby uczenia i walidacji modeli jest zapisywany do pliku dataToModel_5min.csv.

3.4. Przetworzenie danych

Następnie plik zostaje wczytany przez notatnik windowsMaker.ipynb i na wstępie odrzuca on dane: rok, ms, dz, go, mi, fco2_MQ. Typ danych jest zmieniany dla float64. Następnie dane są skalowane do rozmiaru od 0 do 1. Z jego użyciem na podstawie wyżej wymienionych danych utworzone zostają okna o rozmiarze N (przy czym N jest w zakresie 2-32 z krokiem 2), zawierające N wierszy z wynikami oraz każdemu oknu zostaje przypisana etykieta 0 lub 1 w oparciu o wartość w kolumnie fco2_raw (0 lub 1) w ostatnim wierszy danego okna. Okna są zapisywane w jako tablice numpy w plikach o nazwach dataFromSensors_X.npy gdzie litera X oznacza rozmiar okna czasowego. Etykiety są zapisywane w jako tablice numpy w plikach o nazwach labels_X.npy, gdzie litera X oznacza rozmiar okna czasowego. Zbiorów danych wraz z etykietami zostaje podzielony na dwa zbiorów treningowy i walidacyjny

3.5. Zbiór treningowy

Służy on do nauczenia sieci neuronowej rozpoznawania wzorców i zależności występujących w nim dla danych etykiet przy danym oknie. Wielkość zbioru treningowego wynosi 80% wielkości zbioru danych. W przypadku wzięcia całego dataframe z punktu 3.3 i 3.4 jego wielkość wynosi w zależności od długości wybranego okna czasowego: 380244-380268 (przy czym jest to ilość etykiet i okien czasowych o podanym rozmiarze X).

3.6. Zbiór walidacyjny

Zbiór walidacyjny stosowany jest celem weryfikacji czy model sieci się nauczył, czy też jest niedouczony albo przeuczony. Podawane sieci są dane okna bez etykiet, a następnie są porównywane z etykietami jaki powinien dostać, przy czym model się na zbiorze walidacyjnym nie uczy tylko wykorzystuje zgromadzoną wiedzę celem nadania etykiet oknom czasowym (etykietami są: 0 i 1, gdzie zero oznacza dane wątpliwej jakości a 1 dane wysokiej jakości). Wielkość zbioru walidacyjnego wynosi 20% wielkości zbioru danych. W przypadku wzięcia całego dataframe z punktu 3.3 i 3.4 jego wielkość zbioru walidacyjnego wynosi, w zależności od długości wybranego okna czasowego: 95061-95067 (przy czym jest to ilość etykiet i okien czasowych o rozmiarze X).

ROZDZIAŁ 4. Sieć neuronowa służąca do klasyfikacji z użyciem komórek GRU

4.1. Zastosowanie sieci

Omawiana przeze mnie sieć neuronowa służy do klasyfikacji binarnej wyników pomiarów emisji CO₂ na terenach bagiennych na wyniki wysokiej jakości i niskiej jakości. Wyniki wysokiej jakości powinny otrzymać etykietę 1, zaś wyniki o niskiej jakości powinny otrzymywać etykietę 0. Sieć uczy się na danych treningowych, które są podzielone na okna czasowe o określonym rozmiarze, zaś etykieta (0 lub 1) przypisana do pomiarów w danym oknie zależy tylko od ostatniego pomiaru i jest nadawana całemu oknu. Dane walidacyjne mają podobną postać tj. są one podzielone na okna czasowe o określonym rozmiarze, zaś etykieta (0 lub 1) danego pomiaru również zależy tylko od ostatniego pomiaru. Po nauczeniu sieć zwraca wyniki ułamkowe z zakresu 0-1, stąd przyjęte zostało, że wyniki [0-0.5] oznaczają wyniki pomiarów niskiej jakości zaś wyniki (0.5-1] zostały zakwalifikowane jako wyniki wysokiej jakości.

4.2. Architektura sieci

Architektura sieci mówi nam z czego składa się dana sieć, ile jest warstw sieci i jak ułożone są neurony w poszczególnych jej warstwach. Architektura testowanej sieci składała się 5 warstw, w tym dwóch warstw gęstych. Pierwsze 3 warstwy są warstwami wykorzystującymi komórki sieci neuronowej Keras typu GRU, przy czym każda warstwa po warstwie pierwszej posiada mniej neuronów od poprzedniej. Dwie pierwsze warstwy GRU mają parametr return_sequences jako True. Pierwsza warstwa typu GRU jako input_shape ma ustalone odpowiednio train_X.shape[1] i train_X.shape[2]. Dwie warstwy gęste występują po warstwach GRU o mają one rozmiar odpowiednio większy niż ostatnia warstwa typu GRU oraz rozmiar 1. Posiadają one obie sigmoidalną funkcję aktywacji. Służą one ostatecznej klasyfikacji wyników z warstw poprzednich. Przy komplikacji modelu sieci keras została zastosowana funkcja błędu binary_crossentropy, a jako metryka została zastosowana ['accuracy']. Przykładową sieć neuronową wraz informacjami na temat wymiarów danych treningowych (train_X i train_y) i walidacyjnych (test_X i test_Y) oraz z informacjami jak jest zbudowana sieć zawiera rysunek nr 4.2.

Rysunek 4.2 Informacje o modelu sieci neuronowej i jej budowie

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

4.3. Porównanie modeli sieci

W trakcie poszukiwań najlepszego modelu sieci sprawdziłem wiele modeli sieci, ale w pracy skupię się na opisie trzech modeli, które są najbardziej reprezentatywne. Sprawdzane były też modele sieci z mniejszą lub większą ilością warstw, ale po sprawdzeniu wyników to najlepsze były osiągane dla sieci 5 warstwowej (3 warstwy typu GRU i dwie Dense) i na takich się skupię. Jedyną różnicą między modelami jest liczba neuronów poszczególnych warstwach sieci. We wszystkich modelach zostały zastosowane (poza wymienionymi w podpunkcie 4.2) te same parametry tj. okna czasowe, optymalizatory i pozostałe hiperparametry sieci mianowicie:

- Funkcja aktywacji w warstwie nr 1 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 2 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 3 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 1 typu Dense: Sigmoid
- Funkcja aktywacji w warstwie nr 2 typu Dense: Sigmoid
- Optymalizator: SGD (z zastosowanym domyślnym współczynnikiem uczenia)
- Okno czasowe: 16
- Batch: 128

4.3.1. Parametry modelu nr 1

Model sieci nr 1 posiada następujące ilości neuronów w poszczególnych warstwach, co widać na rysunku 4.3.1:

- Liczba neuronów w warstwie typu GRU nr 1: 64
- Liczba neuronów w warstwie typu GRU nr 2: 32
- Liczba neuronów w warstwie typu GRU nr 3: 16
- Liczba neuronów w warstwie Gęstej nr 1(warstwa 4 sieci): 25
- Liczba neuronów w warstwie Gęstej nr 2(warstwa 5 sieci): 1

Liczba epok wykonywanych podczas uczenia (fitowania) modelu wynosi 140, metryka względem, której oceniamy jest model to accuracy (dokładność).

Rysunek 4.3.1 Model sieci nr 1 i informacje o nim

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(64, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=32, return_sequences=True),
    keras.layers.GRU(units=16),
    keras.layers.Dense(units=25, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 64)       15552
gru_1 (GRU)          (None, 16, 32)       9408
gru_2 (GRU)          (None, 16)           2400
dense (Dense)        (None, 25)           425
dense_1 (Dense)      (None, 1)            26
=====

Total params: 27,811
Trainable params: 27,811
Non-trainable params: 0

```

4.3.2. Wyniki dla modelu nr 1

Wykres uczenia dla sieci z użyciem tego modelu przedstawia rysunek 4.3.2.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) i uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem tej sieci, używającej modelu sieci nr 1 przedstawia tabela 4.3.2.2. Czas uczenia modelu wynosi 6688.6203 sekund (w przybliżeniu 112 minut, czyli 1 godzinę i 52 minuty).

Rysunek. 4.3.2.1. Wykres uczenia z użyciem modelu sieci nr 1

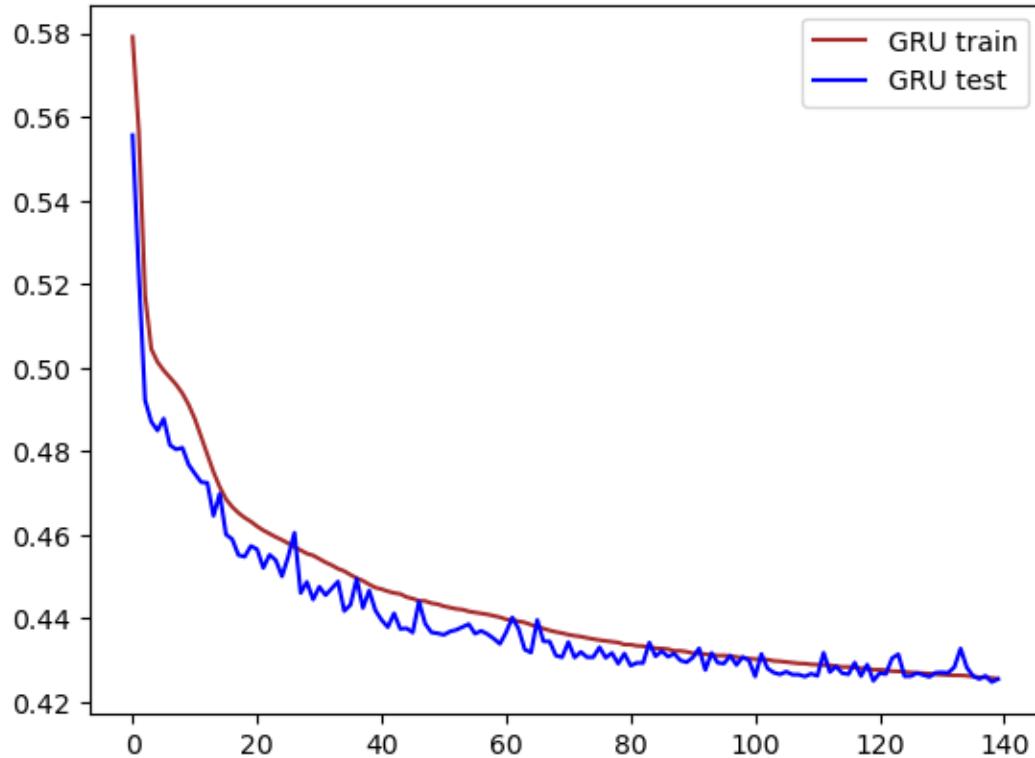


Tabela. 4.3.2.2. Wyniki klasyfikacji z użyciem modelu sieci nr 1

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5965
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8305
Dokładność klasyfikacji dla wszystkich wyników	0.7820

4.3.3. Parametry modelu nr 2

Model sieci nr 1 posiada następujące ilości neuronów w poszczególnych warstwach, co widać na rysunku 4.3.3.:

- Liczba neuronów w warstwie typu GRU nr 1: 128
- Liczba neuronów w warstwie typu GRU nr 2: 64
- Liczba neuronów w warstwie typu GRU nr 3: 32
- Liczba neuronów w warstwie Gęstej nr 1(warstwa 4 sieci): 50
- Liczba neuronów w warstwie Gęstej nr 2(warstwa 5 sieci): 1

Liczba epok wykonywanych podczas uczenia modelu wynosi 140, metryka wzgldem, ktorej oceniamy jest model to accuracy (dokladnosć).

Rysunek 4.3.3. Model sieci nr 2 i informacje o nim

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

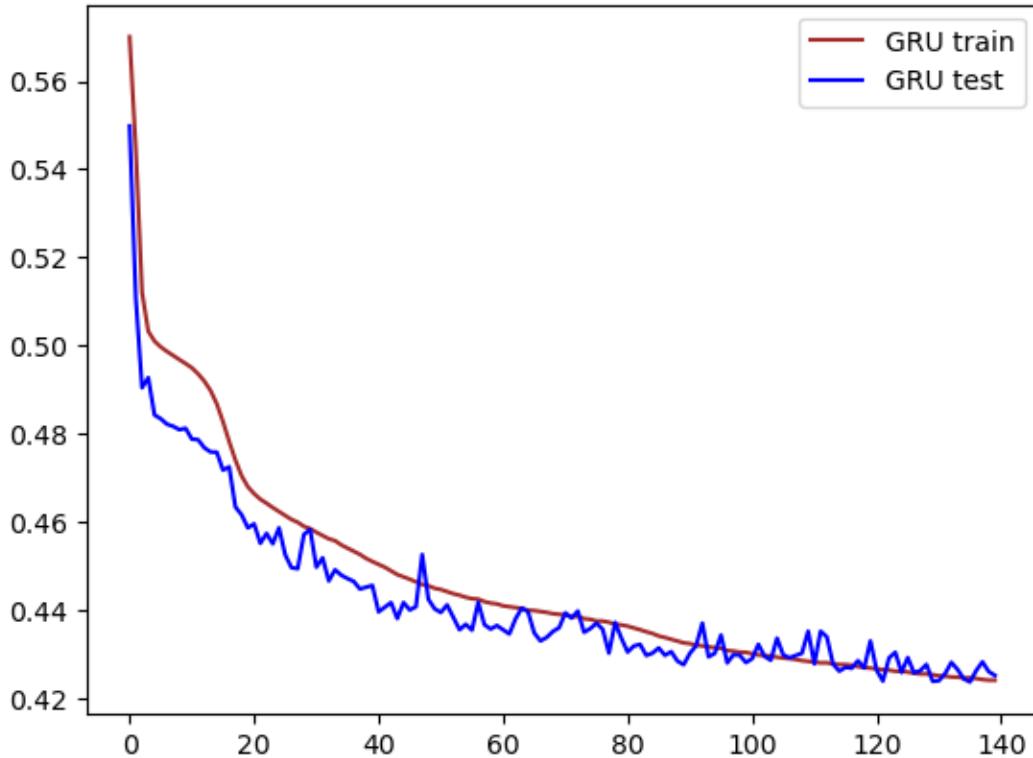
Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

4.3.4. Wyniki dla modelu nr 2

Wykres uczenia dla sieci z użyciem tego modelu przedstawia rysunek 4.3.4.1 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) i uczy się wzgldem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem tej sieci, używajacej modelu sieci nr 2 przedstawia rysunek 4.3.4.2. Modelu uczył się przez 7851.5561 sekund (około 131 minut, czyli 2 godziny 11 minut).

Rysunek. 4.3.4.1 Wykres uczenia z użyciem modelu sieci nr 2



Rysunek. 4.3.4.2 Wyniki klasyfikacji z użyciem modelu sieci nr 2

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

4.3.5. Parametry modelu nr 3

Model sieci nr 3 posiada następujące ilości neuronów w poszczególnych warstwach, co widać na rysunku 4.3.5.:

- Liczba neuronów w warstwie typu GRU nr 1: 256
- Liczba neuronów w warstwie typu GRU nr 2: 128
- Liczba neuronów w warstwie typu GRU nr 3: 64
- Liczba neuronów w warstwie Gęstej nr 1(warstwa 4 sieci): 100
- Liczba neuronów w warstwie Gęstej nr 2(warstwa 5 sieci): 1

Liczba epok wykonywanych podczas uczenia modelu wynosi 140, metryka wzgldem, ktorej oceniamy jest model to accuracy (dokladnosć).

Rysunek 4.3.5. Model sieci nr 3 i informacje o nim

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(256, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=128, return_sequences=True),
    keras.layers.GRU(units=64),
    keras.layers.Dense(units=100, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

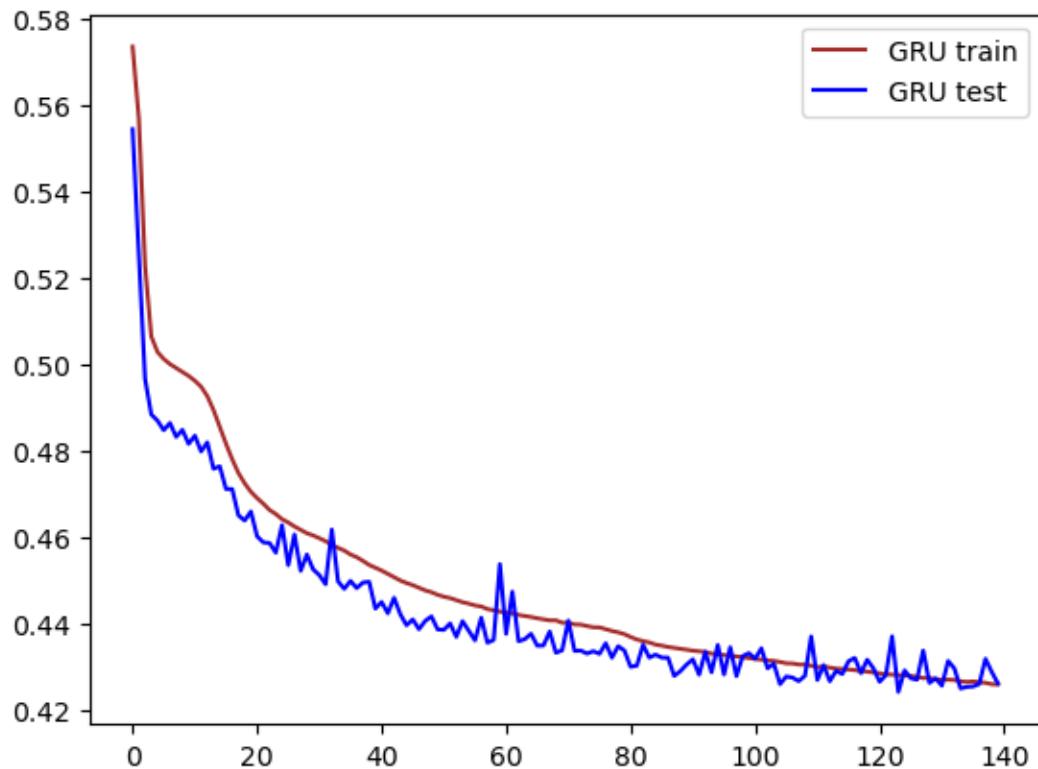
model_gru.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                   (None, 16, 256)          209664
gru_1 (GRU)                  (None, 16, 128)          148224
gru_2 (GRU)                  (None, 64)               37248
dense (Dense)                (None, 100)              6500
dense_1 (Dense)              (None, 1)                101
_________________________________________________________________
Total params: 401,737
Trainable params: 401,737
Non-trainable params: 0
```

4.3.6. Wyniki dla modelu nr 3

Wykres uczenia dla sieci z użyciem tego modelu przedstawia rysunek 4.3.6.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) i uczy się wzgldem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem tej sieci, używajacej modelu sieci nr 3 przedstawia rysunek 4.3.6.2. Czas uczenia modelu wynosi 7378.0303 sekund (w przybliżeniu 123 minuty, czyli 2 godziny i 3 minuty).

Rysunek. 4.3.6.1 Wykres uczenia z użyciem modelu sieci nr 3



Rysunek. 4.3.6.2 Wyniki klasyfikacji z użyciem modelu sieci nr 3

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6094
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8208
Dokładność klasyfikacji dla wszystkich wyników	0.7821

4.3.7. Podsumowanie modeli i ich wyników

Najlepsze wyniki daje model sieci nr 2. Model się na nich uczy bardzo wydajnie pozostałe modele są od niego gorsze, ponieważ uczą się słabiej przy tej samej ilości epok (140). W przypadku modelu nr 3 jest to okupione kosztem większej mocy obliczeniowej i czasu uczenia, niż w wypadku modelu nr 2, a nie otrzymujemy wzrostu dokładności. Model nr 1 mimo, że ma niewiele gorsze wyniki pod względem dokładności to czas uczenia w jest krótszy niż modelu nr 2. Model nr 2 jest idealnym kompromisem pomiędzy dokładnością, a czasem uczenia modelu.

4.4. Porównanie optymalizatorów sieci

Optymalizator w sieci neuronowej odpowiada za takie nauczenie sieci i znalezienie jej wag by znaleźć jak największy spadek funkcji błędu/straty przy jak największej nauce, poprzez aktualizację tych wag sieci. W przypadku zastosowania batcha są one aktualizowane, po każdym batchu, a nie po każdym z przypadków. W wypadku wybrania złego optymalizatora sieć może się uczyć albo za wolno, albo wcale się nie uczyć i aktualizować jedynie wagi w odniesieniu do danych treningowych. W porównaniach optymalizatorów zostały zastosowane te same modele i hiperparametry sieci zaś jedynym zmienianym hiperparametrem był optymalizator. Sieć ta składa się z pięciu warstw (trzech pierwszych z komórkami typu GRU oraz dwóch typu gęstego). Posiada ona 128 neuronów w pierwszej warstwie GRU, 64 neurony w drugiej warstwie GRU, 32 neurony w trzeciej warstwie GRU oraz odpowiednio 50 i jeden neuron w warstwach gęstych. Dwie pierwsze warstwy komórek GRU parametr `return_sequences` (zwracaj sekwencje) ustawiony na wartość `True` (prawda), mówi on czy warstwa ma zwracać wszystkie sekwencje czy nie. Funkcje aktywacji komórek typu GRU to tangens hiperboliczny, zaś komórek typu gęstego sigmoidalna funkcja aktywacji. Metryką zastosowaną do oceny efektu uczenia było `accuracy` (dokładność). Sieć przyjmuje dane o oknie czasowym z wielkością równą 16, zaś wielkość batcha wynosi 128 (z wyjątkiem podpunktu 4.4.1.1. oraz 4.4.1.2. gdzie wielkość batcha wynosi 16384). Ilość epok wynosi 140. Wszystkie optymalizatory (z wyjątkiem podpunktu 4.4.1.1. i 4.4.1.2., gdzie zastosowano współczynnik uczenia o wielkości 0.0007) były używane z domyslną wielkością współczynnika uczenia (0.001).

4.4.1. Optymalizator nr 1 – Optymalizator Adam

Optymalizatorem zastosowanym w tej sieci jest optymalizator Adam [24]. Informacje o budowie sieci przedstawia rys 4.4.1.

Rysunek 4.4.1. Informacje o budowie sieci używającej optymalizatora ADAM

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

4.4.2. Wyniki dla optymalizatora nr 1 - Optymalizator Adam

Wykres uczenia dla sieci z użyciem optymalizatora Adam przedstawia rysunek 4.4.2.1 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony), ale znaczco przeucza się względem danych walidacyjnych (wykres niebieski) co oznacza, że uczy się na pamięć. Wyniki klasyfikacji z użyciem tej sieci, używającej optymalizatora Adam przedstawia tabela 4.4.2.2. Czas nauki modelu wynosi 7430.1652 sekund (w przybliżeniu 123 minuty, czyli 2 godziny i 3 minuty).

Rysunek. 4.4.2.1. Wykres uczenia z użyciem optymalizatora Adam

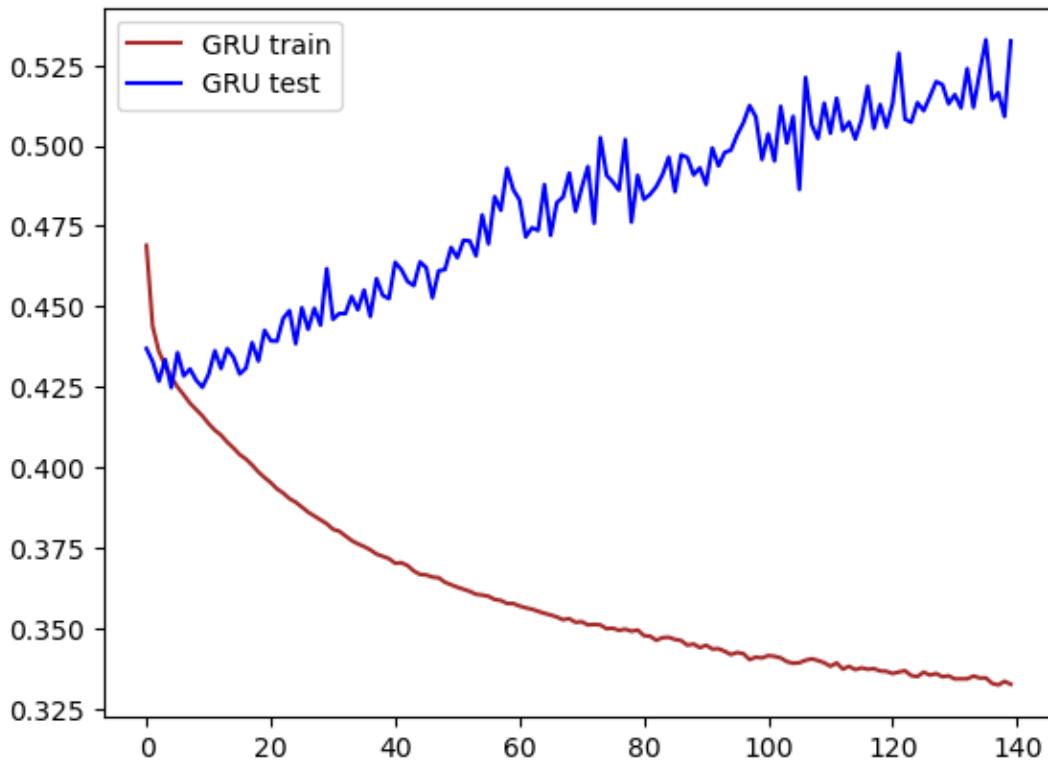


Tabela. 4.4.2.2. Wyniki klasyfikacji z użyciem optymalizatora Adam

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6802
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8656
Dokładność klasyfikacji dla wszystkich wyników	0.8237

4.4.1.1. Optymalizator nr 1.1 - Optymalizator Adam z użyciem zmienionego modelu

Optymalizatorem zastosowanym w tej sieci jest optymalizator Adam [24] z współczynnikiem uczenia (learning_rate) o wielkości 0.0007. Dodatkowo został zwiększyły batch z 128 do rozmiaru 16384. Informacje o budowie sieci przedstawia rys 4.4.1.1., zaś informacje o uczeniu modelu (batchu i liczbie epok) przedstawia tab. 4.4.1.2.

Rysunek. 4.4.1.1.1. Informacje o sieci

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
opt=keras.optimizers.Adam(learning_rate=0.0007)
model_gru.compile(loss='binary_crossentropy', optimizer=opt,metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Rysunek. 4.4.1.1.2. Funkcja fit() zastosowana na tym modelu

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=16384)
```

Python

4.4.1.2. Wyniki dla optymalizatora nr 1.1 - Optymalizator Adam z użyciem zmienionego modelu

Wykres uczenia dla sieci z użyciem optymalizatora Adam przedstawia rysunek 4.4.1.2.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony przedstawiający wartości błędu loss na przełomie epok) i w porównaniu do podpunktu 4.4.1. uczy się również względem danych walidacyjnych (wykres niebieski, przedstawiający wartości błędu walidacyjnego val_loss na przełomie epok). Wyniki klasyfikacji z użyciem tej sieci, używającej optymalizatora Adam przedstawia rysunek 4.4.1.2.2. Czas nauki modelu wynosi 305.7613 sekund (w przybliżeniu 5 minut 10 sekund).

Rysunek. 4.4.1.2.1. Wykres uczenia z użyciem optymalizatora Adam

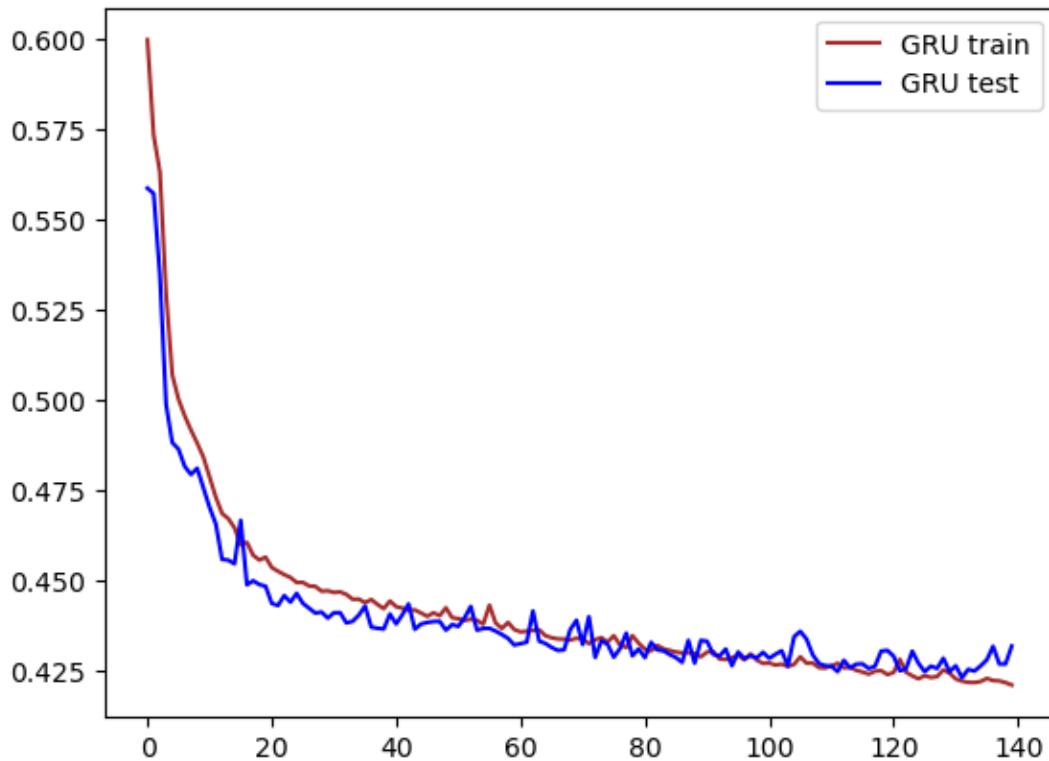


Tabela. 4.4.1.2.2. Wyniki klasyfikacji z użyciem optymalizatora Adam

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6210
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8184
Dokładność klasyfikacji dla wszystkich wyników	0.7841

4.4.2. Optymalizator nr 2

Optymalizatorem zastosowanym w tej sieci jest optymalizator SGD[25]. Informacje o budowie sieci przedstawia rys 4.4.2.

Rysunek 4.4.2. Informacje o sieci używającej optymalizatora SGD

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
-----  

Layer (type)          Output Shape         Param #
-----  

gru (GRU)            (None, 16, 128)      55680  

gru_1 (GRU)          (None, 16, 64)       37248  

gru_2 (GRU)          (None, 32)           9408  

dense (Dense)        (None, 50)           1650  

dense_1 (Dense)      (None, 1)            51  

-----  

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

4.4.3. Wyniki dla optymalizatora nr 2

Wykres uczenia sieci z użyciem optymalizatora SGD przedstawia rysunek 4.4.3.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem tej sieci, używającej optymalizatora SGD przedstawia tabela 4.4.3.2. Modelu uczył się przez 7851.5561 sekund (około 131 minut, czyli 2 godziny 11 minut).

Rysunek. 4.4.3.1. Wykres uczenia z użyciem optymalizatora SGD

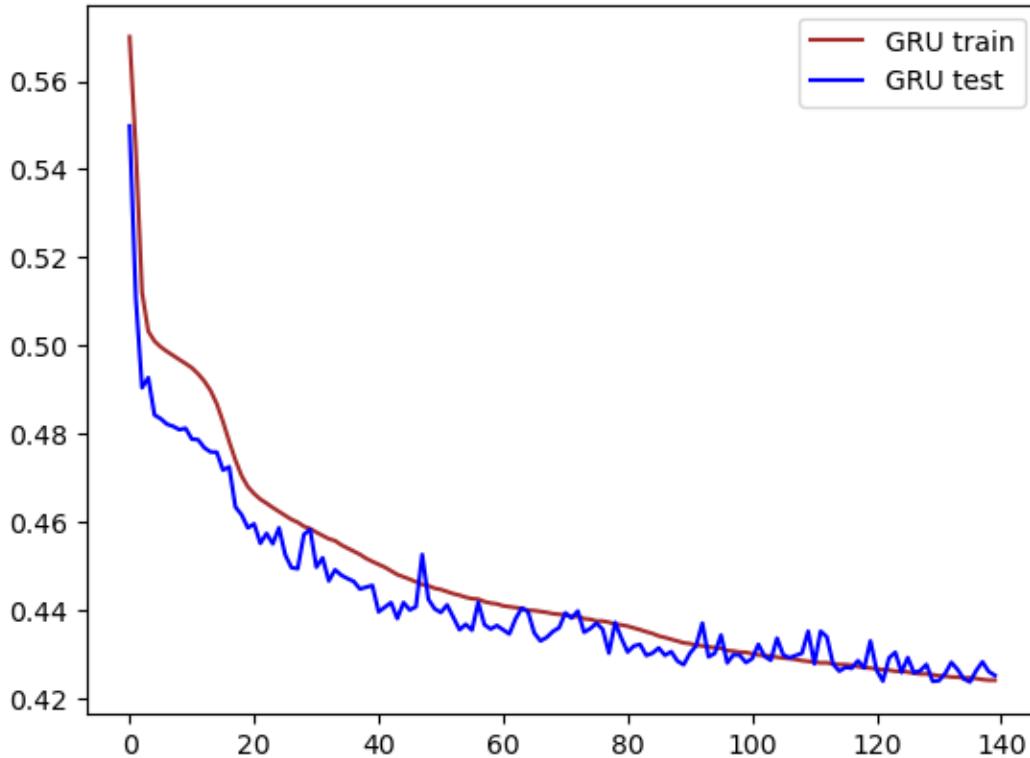


Tabela. 4.4.3.2. Wyniki klasyfikacji z użyciem optymalizatora SGD

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

4.4.4. Optymalizator nr 3

Optymalizatorem zastosowanym w tej sieci jest optymalizator Adagrad[26]. Informacje o budowie sieci przedstawia rys 4.4.4.

Rysunek 4.4.4. Informacje o sieci używającej optymalizatora Adagrad

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='adagrad', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

4.4.5. Wyniki dla optymalizatora nr 3

Wykres uczenia sieci z użyciem optymalizator Adagrad przedstawia rysunek 4.4.5.1 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Jednakże przy tej ilości epok i zastosowanym domyślnym współczynnikiem uczenia wyniki są dużo gorsze niż dla optymalizatora SGD. Należało by zastosować większy współczynnik uczenia używając tego optymalizatora przy dostrajaniu modelu lub zwiększyć ilość epok (co miało wpływ na czas uczenia się sieci neuronowej). Wyniki klasyfikacji z użyciem tej sieci, używającej optymalizatora Adagrad przedstawia tabela 4.4.5.2. Model uczył się przez 7375.1198 sekund (około 123 minuty, czyli 2 godziny 3 minuty).

Rysunek. 4.4.5.1 Wykres uczenia z użyciem optymalizatora Adagrad

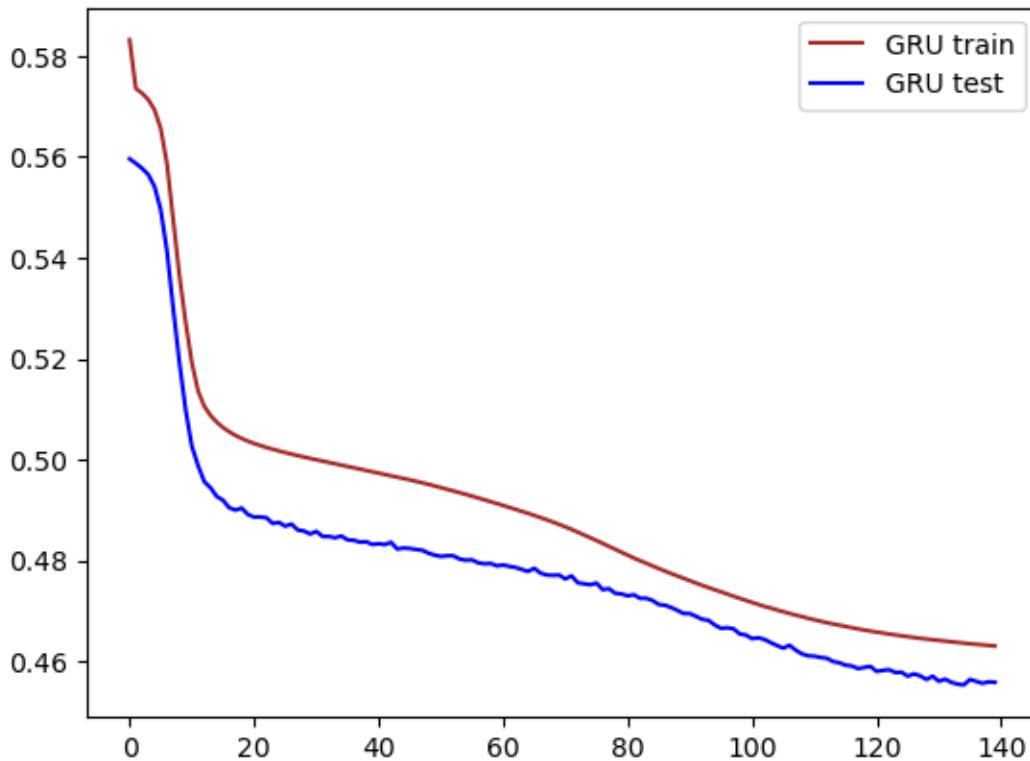


Tabela. 4.4.5.2 Wyniki klasyfikacji z użyciem optymalizatora Adagrad

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5510
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.7932
Dokładność klasyfikacji dla wszystkich wyników	0.7572

4.4.6. Podsumowanie optymalizatorów i ich wyników

Najlepsze wyniki daje optymalizator SGD, ponieważ model się uczy bardzo skutecznie w porównaniu do modelu, który używa optymalizatora ADAM z domyślną wartością współczynnika uczenia (0.001), gdzie model się nie uczy, jak powinien, co pokazuje wykres uczenia 4.4.2.1. Jednak, że przy użyciu większego batcha lub/i innego współczynnika uczenia optymalizator ADAM daje również dobre rezultaty i uczy się poprawnie co pokazuje wykres uczenia 4.4.2.2.1. Użycie optymalizatora Adagrad powoduje, że model się uczy dużo wolniej niż z użyciem optymalizatora SGD, przy tej samej ilości epok wyniki klasyfikacji dla optymalizatora SGD są dużo lepsze co widać po skuteczności i dokładności klasyfikacji wyników. Sieć uczona z użyciem optymalizatora Adagrad, aby nauczyła się na tym samym lub wyższym poziomie co z użyciem optymalizatora SGD wymaga

zastosowania większej ilości epok (co zwiększa czas uczenia kilkukrotnie) albo zwiększenia współczynnika uczenia z domyślnej wartości 0.001 [24] do wartości np. 0.01 bez większego wpływu na czas uczenia (w sekundach) modelu sieci.

4.5. Porównanie pozostałych hiperparametrów sieci

Pozostałe hiperparametry sieci neuronowej, które zostały porównane to:

- funkcja aktywacji dla warstw GRU (hiperparametr nr 1),
- metryka (hiperparametr nr 2),
- shuffle (hiperparametry nr 3),
- funkcja błędu loss (hiperparametr nr 4),
- wielkość batcha (hiperparametr nr 5)
- ilość epok (hiperparametr nr 6).

Zastosowany w podpunktach 4.5.1.-4.5.6. model sieci jest modelem z podpunktu 4.4.4. przy czym zmieniane są w odpowiednie dla danego przypadku hiperparametry.

4.5.1. Hiperparametr nr 1 - Funkcja Aktywacji Komórek GRU

Przetestowane zostały dwie funkcje aktywacji komórek GRU: Relu i Tanh. W funkcji aktywacji w komórkach GRU odpowiada za różne podziały danych z jej użyciem. Funkcja aktywacji typu tanh, będąca domyślną funkcją aktywacji posiada przedział wartości od -1 do 1, zaś funkcja aktywacji relu posiada przedział wartości od 0 do 1. Domyślną funkcją aktywacji komórek GRU jest funkcja tanh [27].

Funkcja Aktywacji Tanh

Funkcją aktywacji komórek GRU jest tangens hiperboliczny, co przedstawia rys 4.5.1.

Rys 4.5.1. Model sieci z funkcją aktywacji komórek GRU tanh

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Wyniki dla funkcji aktywacji Tanh

Wykres uczenia sieci z użyciem funkcji tanh jako funkcji aktywacji komórek GRU przedstawia rysunek 4.5.1.2. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem tej sieci, z komórkami GRU posiadającymi funkcję aktywacji tanh przedstawia tabela 4.5.1.3. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.1.2. Wykres uczenia z użyciem funkcji aktywacji komórek GRU tanh

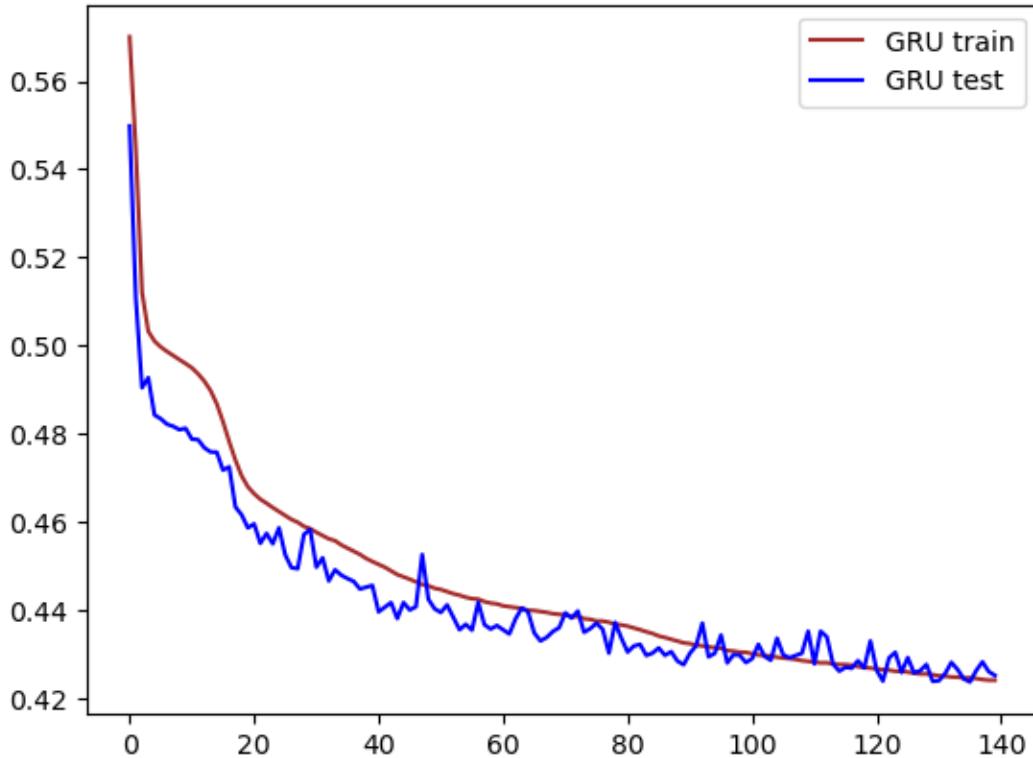


Tabela. 4.5.1.3 Wyniki klasyfikacji z użyciem funkcji aktywacji komórek GRU tanh

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

Funkcja Aktywacji Relu

Funkcją aktywacji komórek GRU jest funkcja relu, co przedstawia rys 4.5.1.4.1. oraz rys 4.5.1.4.2.

Rys 4.5.1.4.1. Model sieci z funkcją aktywacji komórek GRU relu część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
Python

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2]), activation='relu'),
    keras.layers.GRU(units=64, return_sequences=True, activation='relu'),
    keras.layers.GRU(units=32, activation='relu'),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
Python

WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_2 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

Rys 4.5.1.4.2. Model sieci z funkcją aktywacji komórek GRU relu część 2

```
model_gru.summary()
Python

Model: "sequential"

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Wyniki dla funkcji aktywacji Relu

Wykres uczenia sieci z użyciem funkcji relu jako funkcji aktywacji komórek GRU przedstawia rysunek 4.5.1.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski), ale po 100 epoce widać, że model dalej się nie uczy. Wyniki klasyfikacji z użyciem tej sieci, z komórkami GRU posiadającymi funkcję aktywacji relu przedstawia tabela 4.5.1.6. Czas uczenia modelu wynosi 154534.9910 sekund (w przybliżeniu 43 godziny).

Rysunek. 4.5.1.5. Wykres uczenia z użyciem funkcji aktywacji komórek GRU relu

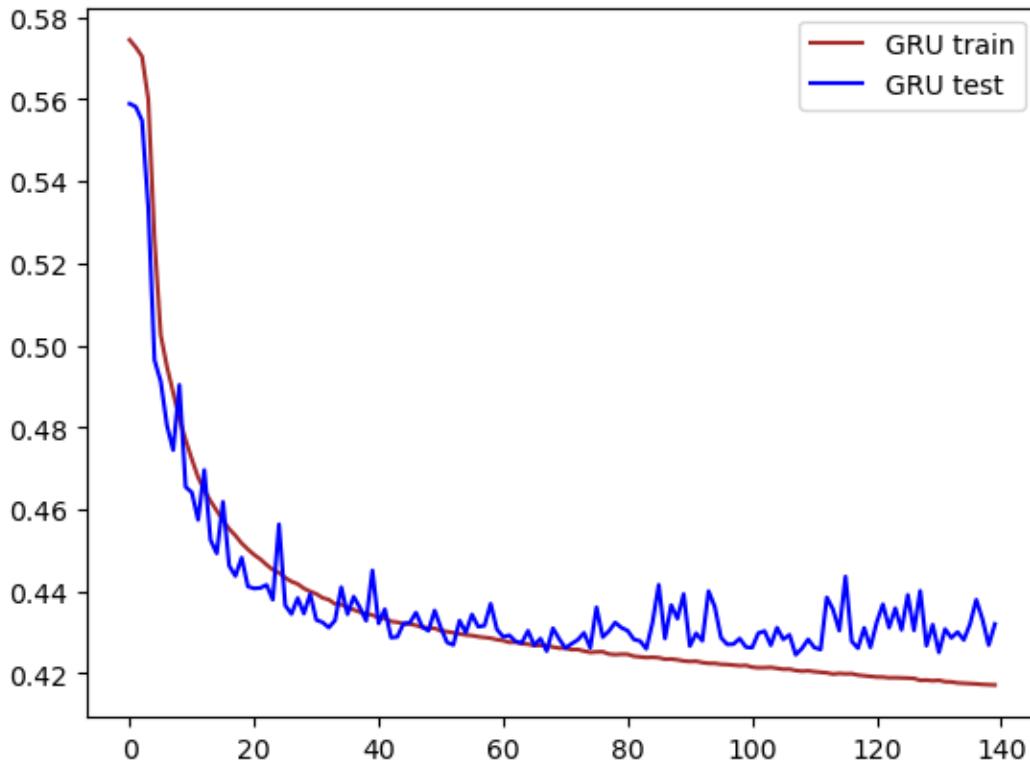


Tabela. 4.5.1.6. Wyniki klasyfikacji z użyciem funkcji aktywacji komórek GRU relu

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6064
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8334
Dokładność klasyfikacji dla wszystkich wyników	0.7862

Porównanie funkcji aktywacji

Lepsze rezultaty daje funkcja aktywacji tanh, gdyż sieć się uczy wtedy dużo szybciej(czasowo) uczy bez dużej utraty dokładności ogólnej, przy jednoczesnym wzroście dokładności wykrywania danych HQ, szczególnie, gdy wykorzystujemy karty graficzne firmy Nvidia (lepiej one współpracują z funkcją aktywacji tanh).

4.5.1. Hiperparametr nr 2 – Metryka

Metryka jest to parametr, który mówi jak zmienia się wydajność i skuteczność modelu. Nie uczestniczy on w zmianie wag, a jedynie informuje o zmianie skuteczności modelu [28]. Sprawdzone zostały metryka accuracy (dokładność) [29] oraz metryka AUC (pola powierzchni

pod krzywą) [30], gdzie domyślnie krzywą tą jest krzywa ROC (charakterystyki operacyjnej odbiornika), jest to metryka klasyfikacji oparta na informacji o prawdziwych/fałszywych pozytywach i negatywach.

Metryka accuracy

Metryką zastosowaną przy tej sieci jest metryka accuracy, co przedstawia rys 4.5.2.1.

Rys 4.5.2.1. Model sieci z metryką Accuracy

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)          9408
dense (Dense)        (None, 50)          1650
dense_1 (Dense)      (None, 1)           51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Wyniki dla metryki accuracy

Wykres uczenia sieci z użyciem metryki accuracy przedstawia rysunek 4.5.2.2. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem metryki accuracy przedstawia tabela 4.5.2.3. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.2.2. Wykres uczenia z użyciem metryki accuracy

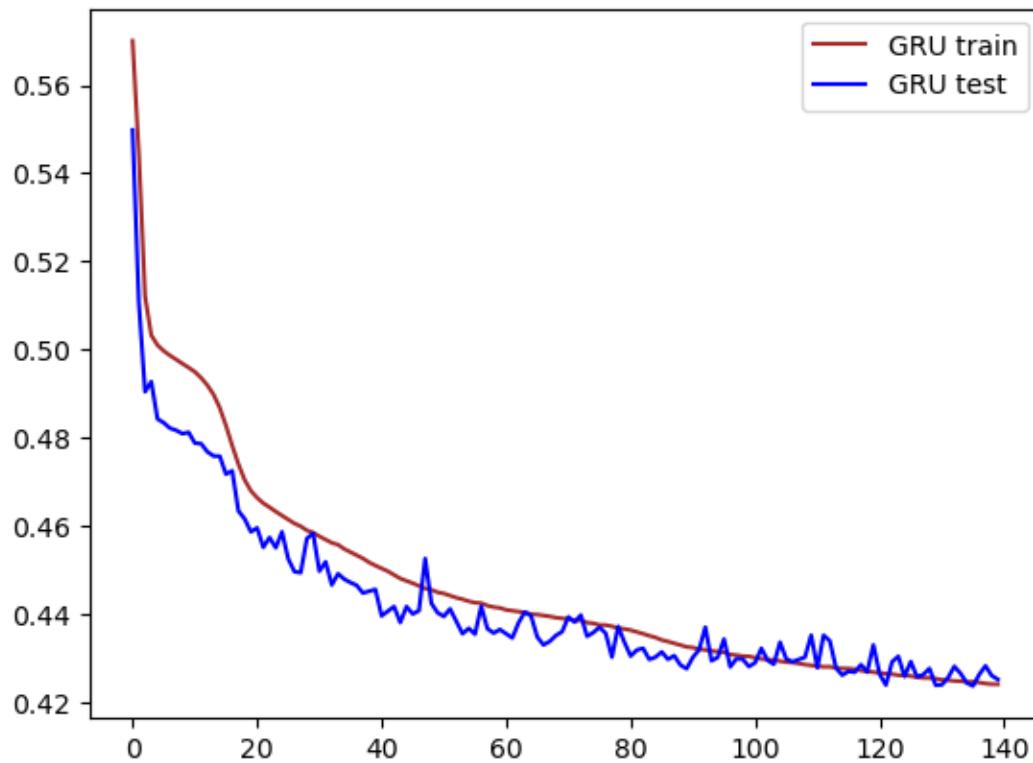


Tabela. 4.5.2.3. Wyniki klasyfikacji z użyciem metryki accuracy

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

Metryka AUC

Metryką zastosowaną przy tej sieci jest metryka AUC, co przedstawia rys 4.5.2.4.

Rys 4.5.2.4. Model sieci z metryką AUC

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['AUC'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)                 Output Shape              Param #
=====
gru (GRU)                    (None, 16, 128)          55680
gru_1 (GRU)                  (None, 16, 64)           37248
gru_2 (GRU)                  (None, 32)              9408
dense (Dense)                (None, 50)              1650
dense_1 (Dense)              (None, 1)               51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Wyniki metryki AUC

Wykres uczenia sieci z użyciem metryki AUC przedstawia rysunek 4.5.2.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem metryki AUC przedstawia tabela 4.5.2.6. Czas uczenia modelu wynosi 7259.9978 sekund, czyli około 121 minut (w przybliżeniu 2 godziny 1 minutę).

Rysunek. 4.5.2.5. Wykres uczenia z użyciem metryki AUC

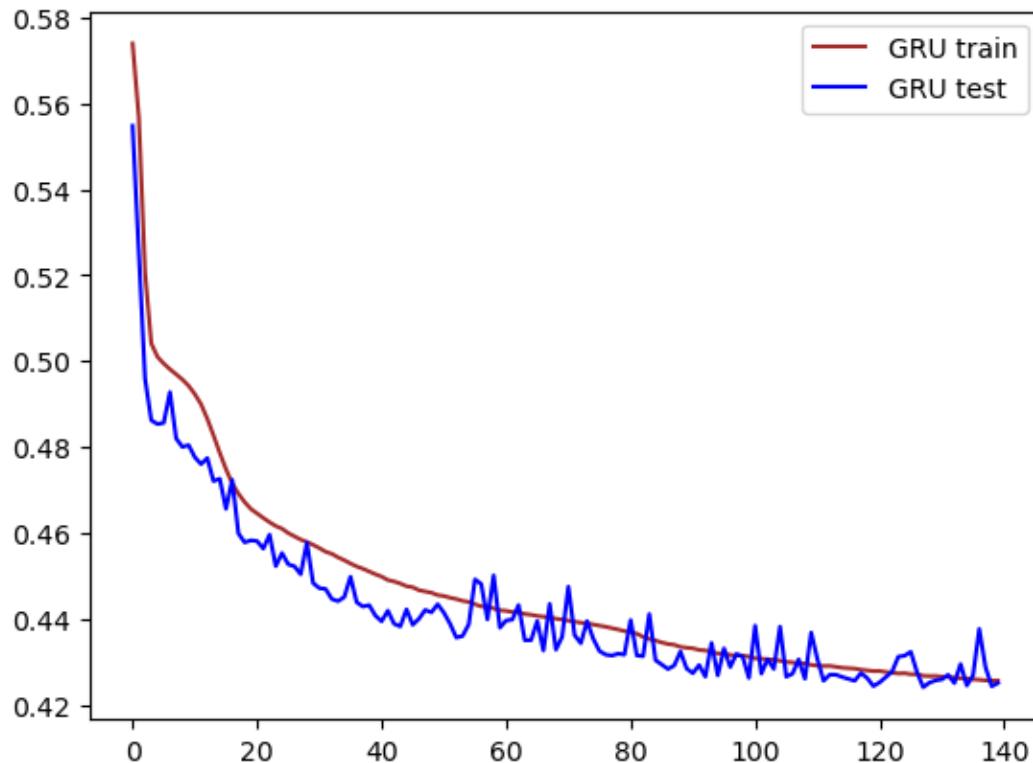


Tabela. 4.5.2.6. Wyniki klasyfikacji z użyciem metryki AUC

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6026
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8250
Dokładność klasyfikacji dla wszystkich wyników	0.7819

Porównanie metryk

Lepsze rezultaty daje użycie metryki accuracy, gdyż sieć się uczy wtedy lepiej, a przy tym niewiele wolniej niż z użyciem metryki AUC. Czas i nauczenie modeli nie odstają od siebie, gdyż metryka nie bierze udziału przy strojeniu wag modelu.

4.5.2. Hiperparametr nr 3 - Parametr shuffle

Parametr shuffle stosowany podczas dostrajania (fit) sieci odpowiada za to, czy dane uczące i treningowe są pomieszane (potasowane jak talia kart do gry w pokera) i potem podzielone na batche w przypadku, gdy parametr ten ma ustaloną wartość true (domyślna)[31] czy też są takie kolejności jakiej były i następnie, potem dzielone na batche, gdy wartość shuffle jest wartością false. W tym

podrozdziale porównuję strojenie modelu, gdy parametr shuffle jest ustawiony (domyślnie) na true oraz, gdy jest ustawiona jego wartość jako false.

Parametr shuffle z wartością true

Przy uczeniu modelu zostało zastosowany parametr shuffle ustawiony na wartość na true(domyślnie), co przedstawiają rys 4.5.3.1.1. i rys 4.5.3.1.2.

Rys 4.5.3.1.1. Model sieci z parametrem shuffle z wartością true Część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380266, 4, 15) (380266, 1) (95067, 4, 15) (95067, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 4, 128)       55680
gru_1 (GRU)          (None, 4, 64)        37248
gru_2 (GRU)          (None, 32)          9408
dense (Dense)        (None, 50)          1650
dense_1 (Dense)      (None, 1)           51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Rys 4.5.3.1.2. Model sieci z parametrem shuffle z wartością true Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=128)
```

Wyniki dla parametru shuffle równego true

Wykres uczenia sieci z użyciem parametru shuffle ustawionego na true przedstawia rysunek 4.5.3.2 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem parametru shuffle ustawionego na wartość true przedstawia tabela 4.5.3.3. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.3.2 Wykres uczenia z użyciem parametru shuffle równego true

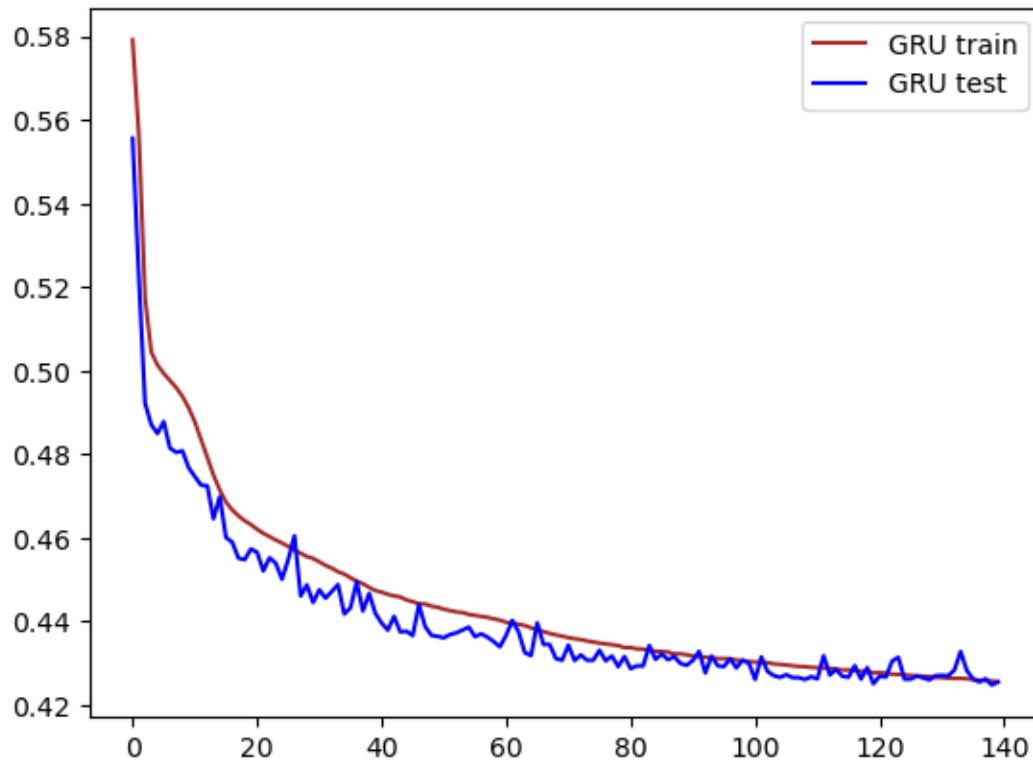


Tabela. 4.5.3.3 Wyniki klasyfikacji z użyciem parametru shuffle równego true

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5965
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8305
Dokładność klasyfikacji dla wszystkich wyników	0.7820

Parametr shuffle z wartością równą false

Przy uczeniu modelu został zastosowany parametr shuffle z wartością równą false, co przedstawia rys 4.5.3.4.1. oraz rysunek 4.5.3.4.2.

Rys 4.5.3.4.1. Model sieci z parametrem shuffle z wartością false Część 1

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
-----  

Layer (type)          Output Shape         Param #
-----  

gru (GRU)            (None, 16, 128)      55680  

gru_1 (GRU)          (None, 16, 64)       37248  

gru_2 (GRU)          (None, 32)           9408  

dense (Dense)        (None, 50)            1650  

dense_1 (Dense)      (None, 1)             51  

-----  

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Rys 4.5.3.4.2. Model sieci z parametrem shuffle z wartością false Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=128, shuffle=False)
```

Python

Wyniki dla parametru shuffle równego false

Wykres uczenia sieci z użyciem parametru Shuffle ustawionego na false przedstawia rysunek 4.5.3.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem parametru shuffle ustawionego na wartość false przedstawia tabela 4.5.3.6. Czas uczenia modelu wynosi 6957.7756 sekund, czyli około 116 minut (w przybliżeniu 1 godzinę i 56 minut).

Rysunek. 4.5.3.5. Wykres uczenia z użyciem parametru shuffle równego false

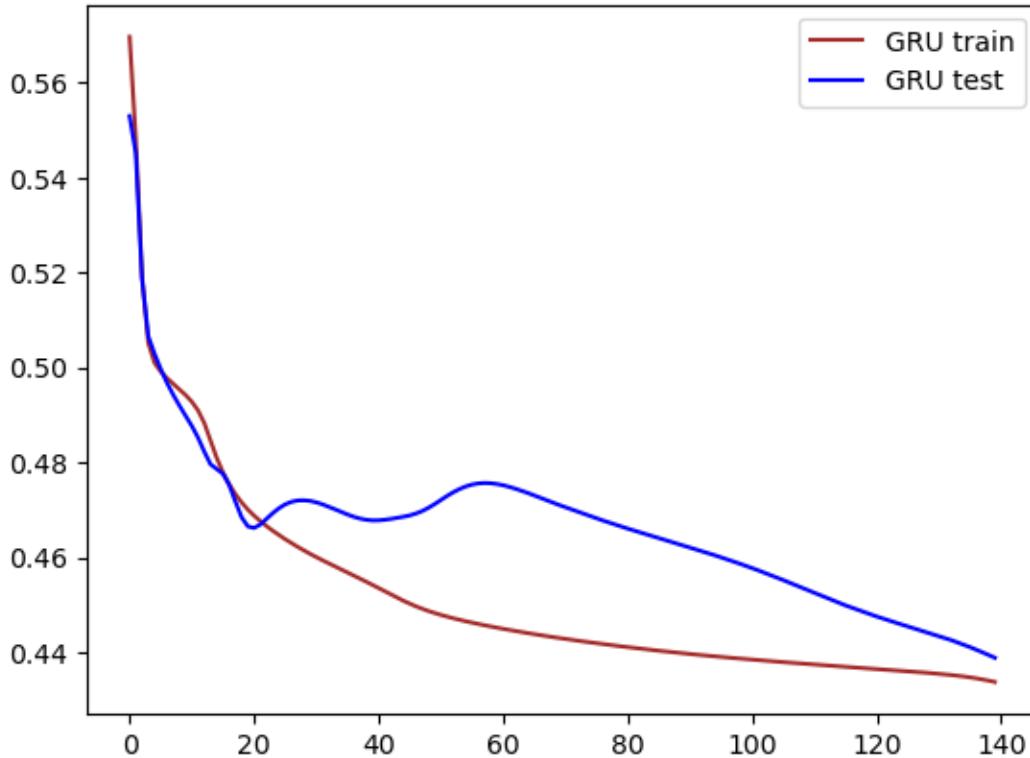


Tabela. 4.5.3.6. Wyniki klasyfikacji z użyciem parametru shuffle równego false

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5726
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8281
Dokładność klasyfikacji dla wszystkich wyników	0.7732

Porównanie parametru shuffle równego true lub false

Sieć uczy się gorzej dla parametru shuffle ustawionego na false. Ustawienie parametru shuffle na true tasuje zbiór danych (walidacyjny i uczący) dzięki czemu sieć przy uczeniu nie uczy się przypadkowych wzorców występujących w zbiorze danych, lecz skupia się na wzorcach występujących w zadanych pojedynczych oknach czasowych, co jest zachowaniem oczekiwany.

4.5.3. Hiperparametr nr 4 - Funkcja błędu

Funkcja błędu loss jest stosowana celem wyliczania wartości błędu którą model powinien zmniejszać podczas uczenia się [32]. Sprawdzona została funkcja błędu binarycrossentropy [33] (jest to

funkcja probabilistyczna funkcja błędu) oraz funkcja błędu MeanSquaredError [34] (jest to regresyjna funkcja błędu).

Funkcja błędu BinaryCrossentropy

Funkcją błędu tego modelu jest funkcja BinaryCrossentropy, co przedstawia rys 4.5.4.1.

Rys 4.5.4.1. Model sieci z funkcją błędu BinaryCrossentropy

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Wyniki dla funkcji błędu BinaryCrossentropy

Wykres uczenia sieci z użyciem funkcji błędu BinaryCrossentropy przedstawia rysunek 4.5.4.2. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem funkcji błędu BinaryCrossentropy przedstawia tabela 4.5.4.3. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.4.2. Wykres uczenia z użyciem funkcji błędu BinaryCrossentropy

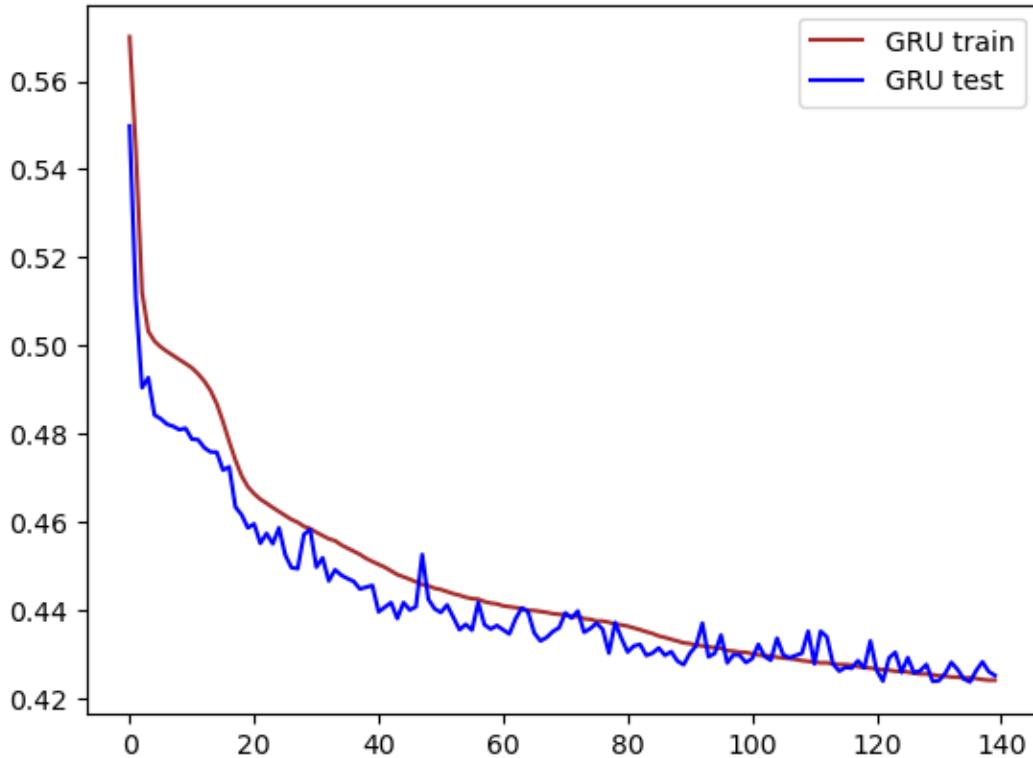


Tabela. 4.5.4.3. Wyniki klasyfikacji z użyciem funkcji błędu BinaryCrossentropy

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

Funkcja błędu MeanSquaredError

Funkcją błędu tego modelu jest funkcja MeanSquaredError, co przedstawia rys 4.5.4.4.

Rys 4.5.4.4. Model sieci z funkcją błędu MeanSquaredError

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='MeanSquaredError', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential_2"
=====
Layer (type)          Output Shape         Param #
=====
gru_6 (GRU)           (None, 16, 128)      55680
gru_7 (GRU)           (None, 16, 64)       37248
gru_8 (GRU)           (None, 32)          9408
dense_4 (Dense)       (None, 50)          1650
dense_5 (Dense)       (None, 1)           51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Wyniki dla funkcji błędu MeanSquaredError

Wykres uczenia sieci z użyciem funkcji błędu MeanSquaredError przedstawia rysunek 4.5.4.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem funkcji błędu MeanSquaredError przedstawia tabela 4.5.4.6. Czas uczenia modelu wynosi 6663.9098 sekund, czyli około 111 minut (w przybliżeniu 1 godzinę 51 minut).

Rysunek. 4.5.4.5. Wykres uczenia z użyciem funkcji błędu MeanSquaredError

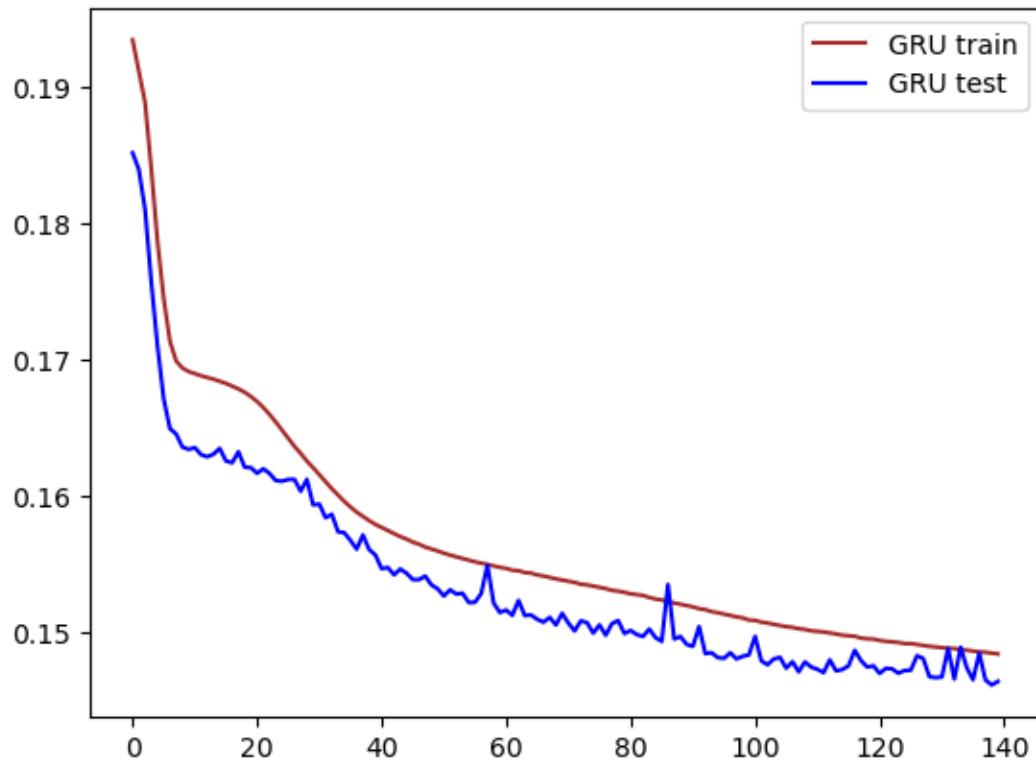


Tabela. 4.5.4.6. Wyniki klasyfikacji z użyciem funkcji błędu MeanSquaredError

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5938
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8098
Dokładność klasyfikacji dla wszystkich wyników	0.7735

Porównanie funkcji błędu

Model sieci, który używa funkcji błędu BinaryCrossentropy uczy się szybciej (przy mniejszej ilości epok uzyskuje ten sam rezultat) oraz lepiej (przy tej samej ilości epok wyniki klasyfikacji są wyższe) od modelu sieci, który używa funkcji błędu MeanSquaredError, a także ma większą skuteczność przy wykrywaniu danych wysokiej jakości oraz niskiej jakości. Pomimo tego wydajność modelu używającego funkcji MSE jest wyższa (niższy czas nauki na jedną epokę oraz całkowity czas uczenia).

4.5.4. Hiperparametr nr 5 - Wielkość batcha

W tym podrozdziale jest sprawdzany wpływ wielkości batcha nauczenia modelu [35]. Batch jest to ilość przykładów treningowych stosowanych w jednej iteracji uczenia modelu. Zastosowany został mini-batch, gdyż wielkość batcha jest większa niż 1, ale mniejsza niż wielkość całkowita zbioru danych treningowych. Przy uczeniu modelu zostały zastosowane batche odpowiednio o wielkościach 64, 128, 256, 2048 i 16368.

Batch o rozmiarze 64

Model ten korzysta z batcha 64 przy uczeniu, co przedstawiają rys 4.5.5.1.1. oraz rys 4.5.5.1.2.

Rys 4.5.5.1.1. Model sieci z korzystający z batcha o rozmiarze 64 Część 1

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Rys 4.5.5.1.2. Model sieci z korzystający z batcha o rozmiarze 64 Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=64)
```

Python

Wyniki dla batcha o wielkości 64

Wykres uczenia sieci z użyciem batcha o wielkości 64 przedstawia rysunek 4.5.5.2. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Po około setnej epoce wykres błędu i wykres błędu walidacyjnego się delikatnie rozchodzą, co wskazuje na to, że model przestaje się wtedy już uczyć. Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem batcha 64 przedstawia tabela 4.5.5.3. Czas uczenia modelu wynosi 13346.4833 sekund, czyli około 222.44 minuty (w przybliżeniu 3 godziny 42 minuty).

Rysunek. 4.5.5.2. Wykres uczenia z użyciem batcha o wielkości 64

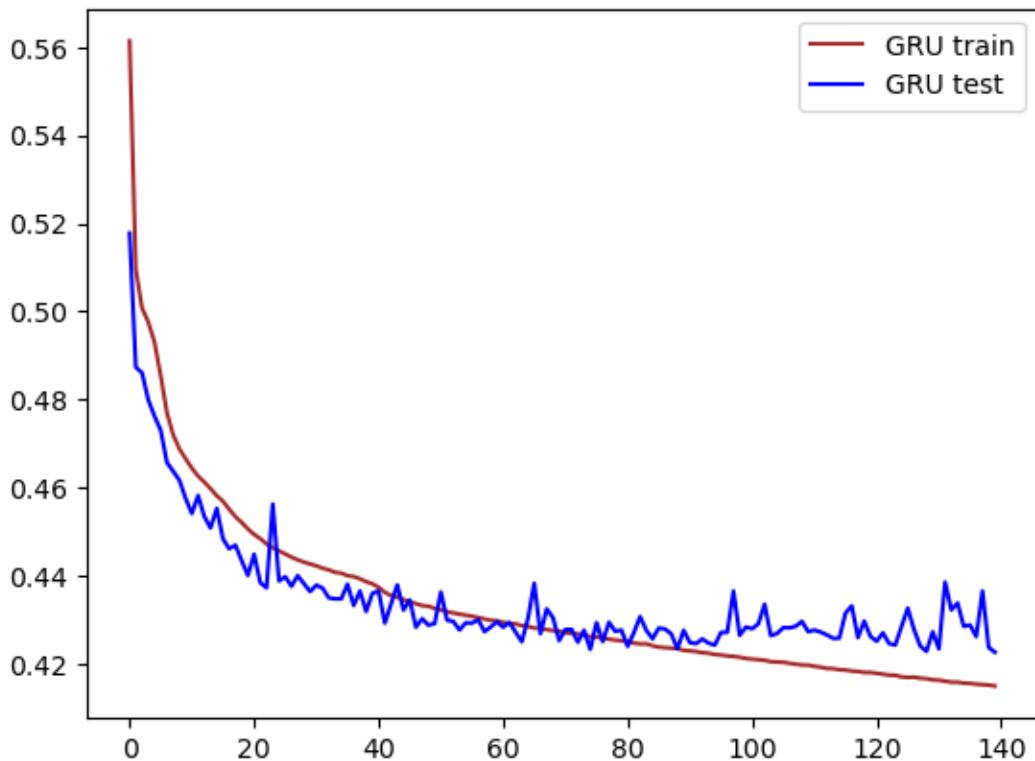


Tabela.4.5.5.3. Wyniki klasyfikacji z użyciem batcha o wielkości 64

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6174
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8276
Dokładność klasyfikacji dla wszystkich wyników	0.7872

Batch o rozmiarze 128

Model ten korzysta z batcha 128 przy uczeniu, co przedstawiają rys 4.5.5.4.1. oraz rys 4.5.5.4.2.

Rys 4.5.5.4.1. Model sieci z korzystający z batcha o rozmiarze 128 Część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
-----  

Layer (type)          Output Shape         Param #
-----  

gru (GRU)            (None, 16, 128)      55680  

gru_1 (GRU)          (None, 16, 64)       37248  

gru_2 (GRU)          (None, 32)           9408  

dense (Dense)        (None, 50)            1650  

dense_1 (Dense)      (None, 1)             51  

-----  

Total params: 104,037  

Trainable params: 104,037  

Non-trainable params: 0
```

Rys 4.5.5.4.2. Model sieci z korzystający z batcha o rozmiarze 128 Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=128)
```

Wyniki dla Batcha 128

Wykres uczenia sieci z użyciem batcha o wielkości 128 przedstawia rysunek 4.5.5.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem batcha 128 przedstawia tabela 4.5.5.6. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.5.5. Wykres uczenia z użyciem batcha o wielkości 128

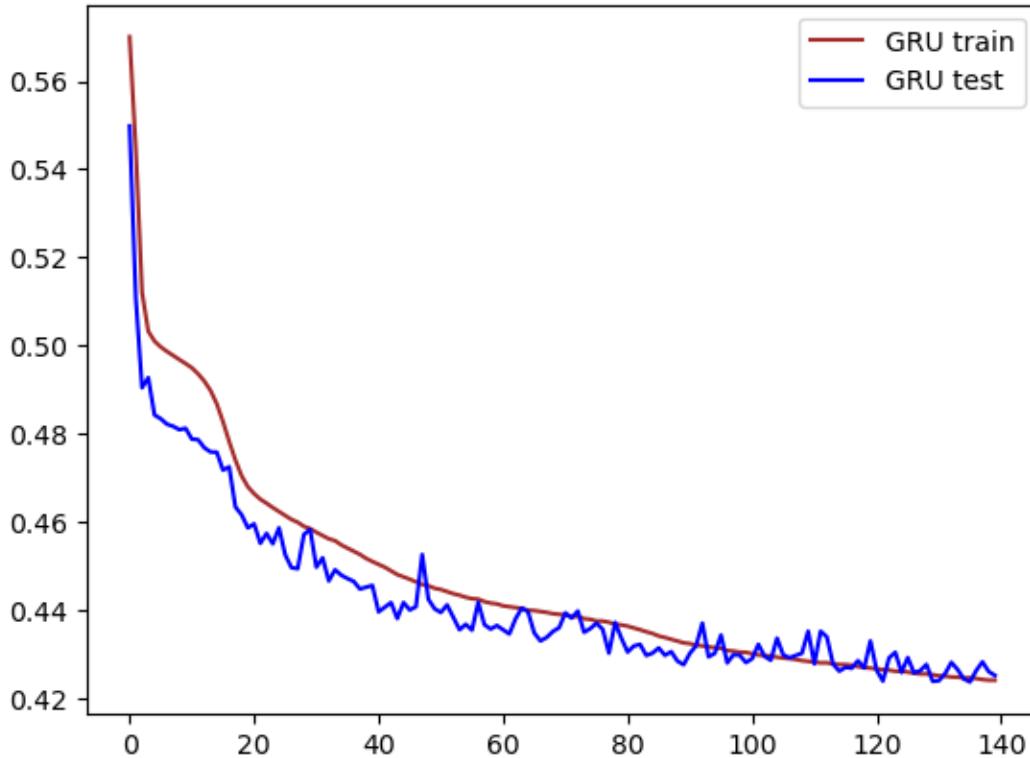


Tabela. 4.5.5.6. Wyniki klasyfikacji z użyciem batcha o wielkości 128

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

Batch o rozmiarze 256

Model ten korzysta z batcha o rozmiarze 256 przy uczeniu, co przedstawiają rys 4.5.5.7.1. oraz rys 4.5.5.7.2.

Rys 4.5.5.7.1. Model sieci z korzystający z batcha o rozmiarze 256 Część 1

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Rys 4.5.5.7.2. Model sieci z korzystający z batcha o rozmiarze 256 Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=256)
```

Python

Wyniki dla batcha o wielkości 256

Wykres uczenia sieci z użyciem batcha o wielkości 256 przedstawia rysunek 4.5.5.8. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem batcha 256 przedstawia tabela 4.5.5.9. Czas uczenia modelu 3502.7365 wynosi sekund, czyli około 58 minut.

Rysunek. 4.5.5.8. Wykres uczenia z użyciem batcha o wielkości 256

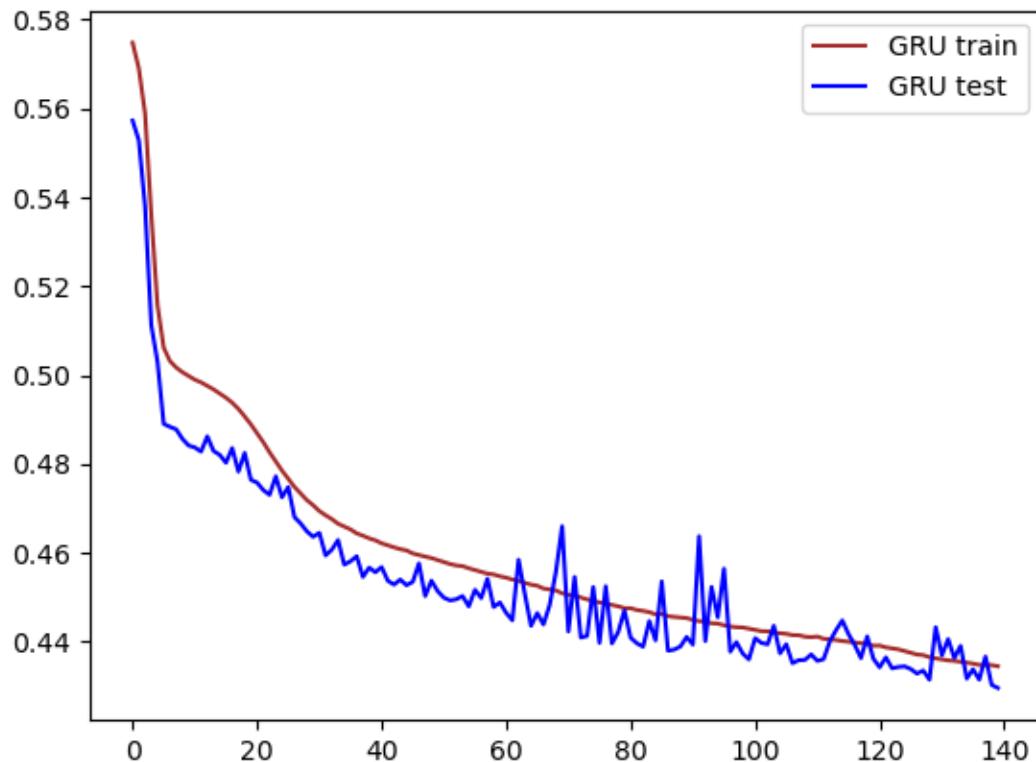


Tabela. 4.5.5.9. Wyniki klasyfikacji z użyciem batcha o wielkości 256

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6035
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8140
Dokładność klasyfikacji dla wszystkich wyników	0.7777

Batch o rozmiarze 2048

Model ten korzysta z batcha o rozmiarze 2048 przy uczeniu, co przedstawiają rys 4.5.5.10.1. oraz rys 4.5.5.10.2. W odróżnieniu od modelu dla batchy 64, 128 i 256 został zastosowany większy współczynnik uczenia: 0.9.

Rys 4.5.5.10.1. Model sieci z korzystający z batcha o rozmiarze 2048 Część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
opt=keras.optimizers.SGD(learning_rate=0.9)
model_gru.compile(loss='MSE', optimizer=opt, metrics=['accuracy'])

model_gru.summary()

Model: "sequential_1"
-----  

Layer (type)          Output Shape         Param #
-----  

gru_3 (GRU)           (None, 16, 128)      55680  

gru_4 (GRU)           (None, 16, 64)       37248  

gru_5 (GRU)           (None, 32)          9408  

dense_2 (Dense)       (None, 50)          1650  

dense_3 (Dense)       (None, 1)           51  

-----  

Total params: 104,037  

Trainable params: 104,037  

Non-trainable params: 0
```

Rys 4.5.5.10.2. Model sieci z korzystający z batcha o rozmiarze 2048 Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=2048)
```

Python

Wyniki dla batcha o wielkości 2048

Wykres uczenia sieci z użyciem batcha o wielkości 2048 przedstawia rysunek 4.5.5.11. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego z użyciem batcha 2048 przedstawia tabela 4.5.5.12. Czas uczenia modelu wynosi 555.7902 sekund, czyli około 9 minut 15 sekund.

Rysunek. 4.5.5.11. Wykres uczenia z użyciem batcha o wielkości 2048

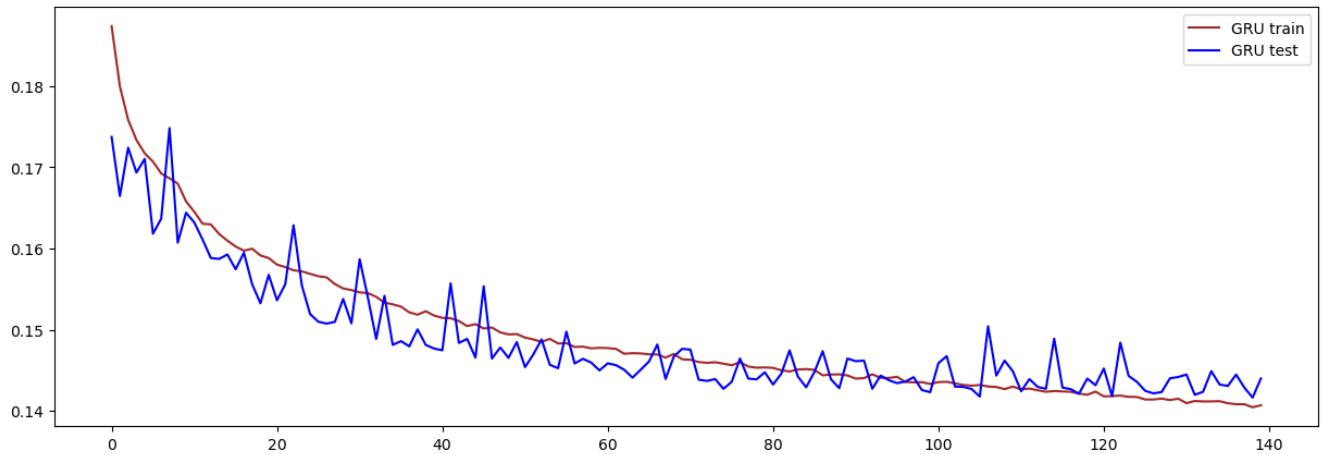


Tabela. 4.5.5.12. Wyniki klasyfikacji z użyciem batcha o wielkości 2048

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6528
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8057
Dokładność klasyfikacji dla wszystkich wyników	0.7845

Batch o rozmiarze 16368

Model ten korzysta z batcha o rozmiarze 16368 przy uczeniu, co przedstawiają rys 4.5.5.13.1. oraz rys 4.5.5.13.2. W odróżnieniu od modelu dla batchy 64, 128 i 256 został zastosowany inny optymalizator (ADAM zamiast SGD) ze zwiększoną współczynnikiem uczenia: 0.0007.

Rys 4.5.5.13.1. Model sieci z korzystający z batcha o rozmiarze 16368 Część 1

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
opt=keras.optimizers.Adam(learning_rate=0.0007)
model_gru.compile(loss='binary_crossentropy', optimizer=opt,metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====

Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====

Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

Rys 4.5.5.13.2. Model sieci z korzystający z batcha o rozmiarze 16368 Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=16384)
```

Python

Wyniki dla batcha o wielkości 16368

Wykres uczenia sieci z użyciem batcha o wielkości 16368 przedstawia rysunek 4.5.5.14. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczzonego z użyciem batcha 16368 przedstawia tabela 4.5.5.15. Czas uczenia modelu wynosi 305.7613 sekund, czyli około 5 minut 6 sekund.

Rysunek. 4.5.5.14. Wykres uczenia z użyciem batcha o wielkości 16368

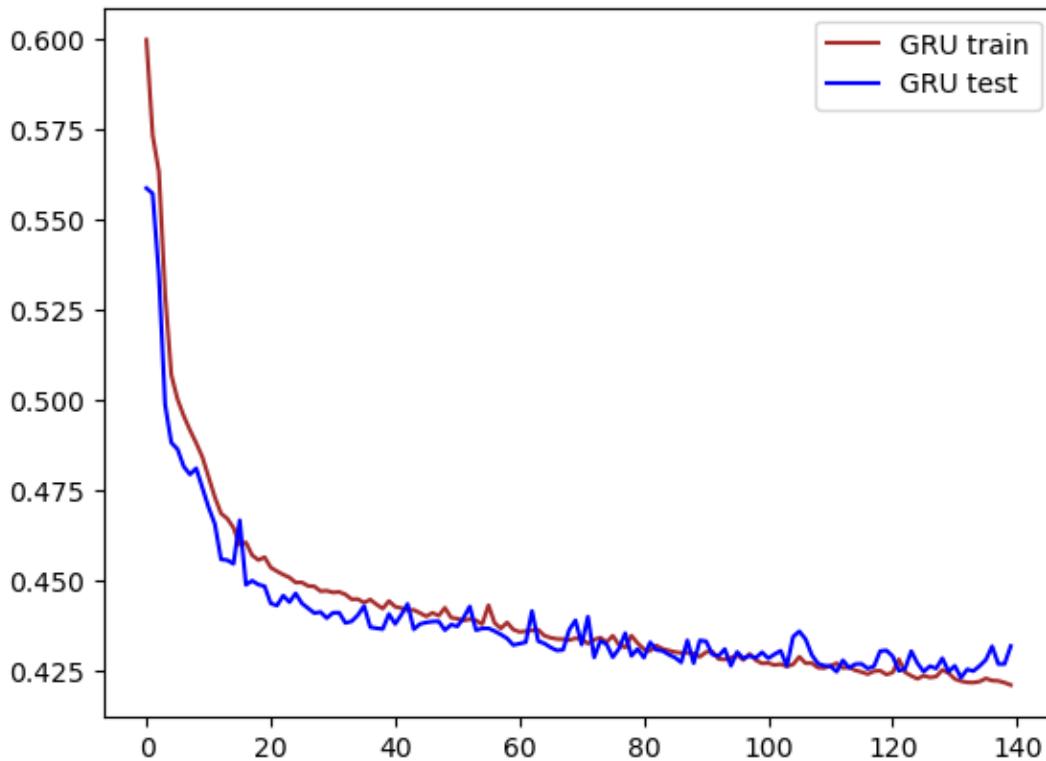


Tabela. 4.5.5.15. Wyniki klasyfikacji z użyciem batcha o wielkości 16368

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6210
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8184
Dokładność klasyfikacji dla wszystkich wyników	0.7841

Porównanie wielkości batchy

Najlepsze wyniki przy domyślnym współczynniku uczenia osiąga model, który wykorzystuje batcha o wielkości 128, zaś mimo iż wyniki walidacji dla modelu z batchem 64 są odrobinę większe niż przy batchu 128 to przy batchu 64 występuje przy ostatnich 40 epokach brak uczenia. Jednak, dla większych batchy(na przykład 2048, 16368), gdy użyjemy innego współczynnika uczenia lub innego optymalizatora (tak jak w przypadku batcha 16368) to czas wykonania pojedynczej epoki jest mniejszy niż przy pozostałych sprawdzonych wielkościach batcha i wynosi odpowiednio 93s z użyciem batcha 64, około 54s w przypadku batcha 128, około 25s w przypadku batcha 256, około 4 s z użyciem batcha 2048 oraz 2 s z użyciem batcha 16368. Zyskujemy również wtedy na dokładności modelu. Jednak, że batche tak duże (np. 16368 i większe) wymagają karty graficznej z dużą ilością VRAMu (dla batcha

16368 było to około 9.6GB, co oznacza, że karta musi mieć około 11-12 GB VRAMu) z tego powodu, że wrzucają większe rozmiarowo batche do pamięci karty graficznej, co sprawia, że model się uczy szybciej, bo nie występuje efekt wąskiego gardła (z ang. Bottleneck) przy przesłaniu danych za każdym razem do pamięci karty. Większy batch sprawia, że więcej informacji jest przesyłane na raz, dzięki czemu nie trzeba co chwilę przesyłać danych do pamięci karty graficznej (jak np. w wypadku batcha 64)

4.5.5. Hiperparametr nr 6 - Liczba epok

W podrozdziale tym jest sprawdzana różna ilość epok uczenia, odpowiednio są to 80 140 i 200. Epoka jest to pojedyncze przejście do przodu i jedno przejście do tyłu przez wszystkie przypadki treningowe [35]. Wagi sieci są aktualizowane po każdej epoce.

Liczba epok - 80

Model sieci, który uczy się na przełomie 80 epok przedstawiają rys 4.5.6.1.1 oraz rys 4.5.6.1.2.

Rys 4.5.6.1.1. Model uczony przez 80 epok Część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
-----  

Layer (type)          Output Shape         Param #
-----  

gru (GRU)            (None, 16, 128)      55680  

gru_1 (GRU)          (None, 16, 64)       37248  

gru_2 (GRU)          (None, 32)           9408  

dense (Dense)        (None, 50)           1650  

dense_1 (Dense)      (None, 1)            51  

-----  

Total params: 104,037  

Trainable params: 104,037  

Non-trainable params: 0
```

Rys 4.5.6.1.2. Model uczyony przez 80 epok Część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=80, validation_data=(test_X, test_y), batch_size=128)
```

Python

Wyniki dla 80 epok

Wykres uczenia sieci na przełomie 80 epok przedstawia rysunek 4.5.6.2. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego przez 80 epok przedstawia tabela 4.5.6.3. Czas uczenia modelu wynosi 3797.2632 sekund, czyli około 64 minuty (w przybliżeniu 1 godzinę 4 minuty).

Rysunek. 4.5.6.2. Wykres uczenia modelu sieci przez 80 epok

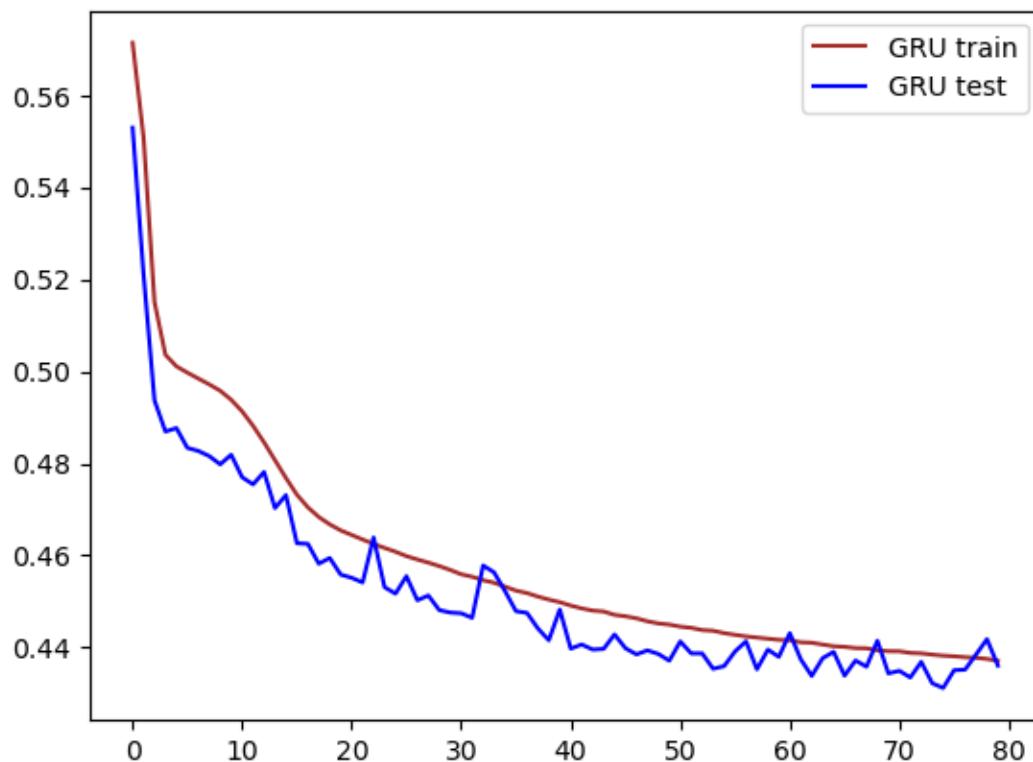


Tabela. 4.5.6.3. Wyniki klasyfikacji wyników przez model sieci uczyony przez 80 epok

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5839
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8220
Dokładność klasyfikacji dla wszystkich wyników	0.7751

Liczba epok - 140

Model sieci, który uczy się na przełomie 140 epok przedstawiają rys 4.5.6.4.1. oraz rys 4.5.6.4.2.

Rys 4.5.6.4.1. Model uczony przez 140 epok część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
```

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 16, 128)	55680
gru_1 (GRU)	(None, 16, 64)	37248
gru_2 (GRU)	(None, 32)	9408
dense (Dense)	(None, 50)	1650
dense_1 (Dense)	(None, 1)	51

```
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Rys 4.5.6.4.2. Model uczony przez 140 epok część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=140, validation_data=(test_X, test_y), batch_size=128)
```

Wyniki dla 140 epok

Wykres uczenia sieci na przełomie 140 epok przedstawia rysunek 4.5.6.5. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci uczonego przez 140 epok przedstawia tabela 4.5.6.6. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.5.6.5. Wykres uczenia modelu sieci przez 140 epok

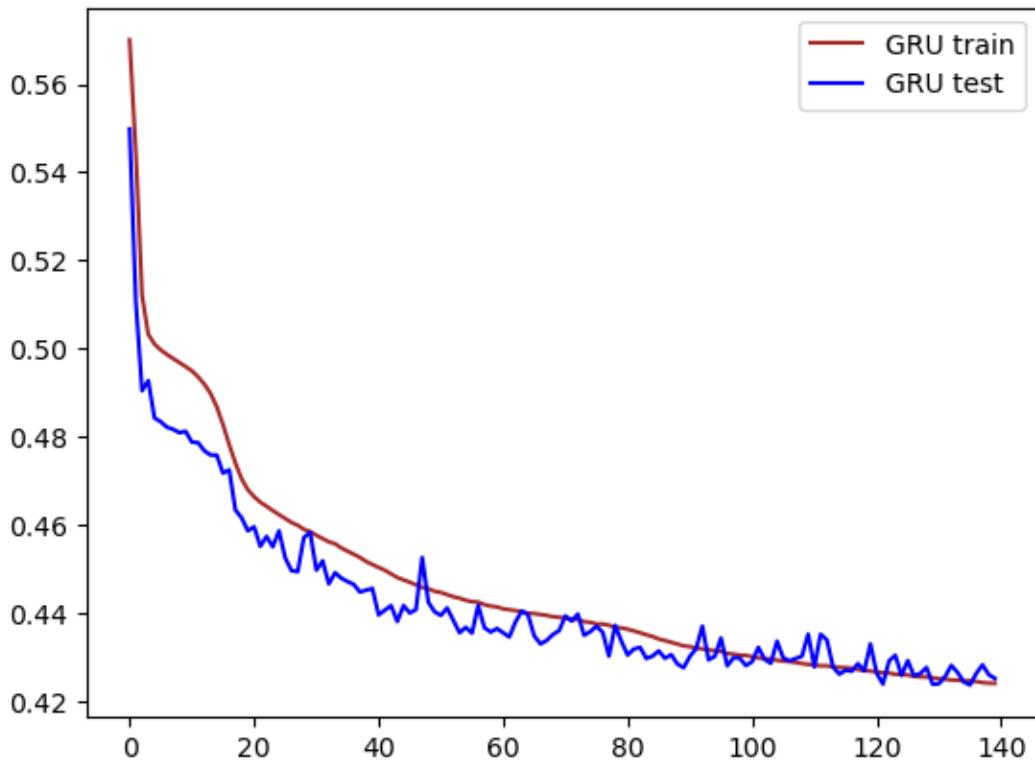


Tabela. 4.5.6.6. Wyniki klasyfikacji wyników przez model sieci uczony przez 140 epok

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

Liczba epok 200

Model sieci, który uczy się na przełomie 200 epok przedstawiają rys 4.5.6.7.1. oraz rys 4.5.6.7.2.

Rys 4.5.6.7.1. Model uczony przez 200 epok część 1

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
```

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 16, 128)	55680
gru_1 (GRU)	(None, 16, 64)	37248
gru_2 (GRU)	(None, 32)	9408
dense (Dense)	(None, 50)	1650
dense_1 (Dense)	(None, 1)	51

```
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

Rys 4.5.6.7.2. Model uczony przez 200 epok część 2

```
gru_history = model_gru.fit(train_X, train_y, epochs=200, validation_data=(test_X, test_y), batch_size=128)
```

Python

Wyniki dla 200 epok

Wykres uczenia sieci na przełomie 200 epok przedstawia rysunek 4.5.6.8. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski), ale po około 150 epoce model się już dalej nie uczy i następuje przerwanie nauki i wykres błędu dla danych walidacyjnych po 150 epoce staje się płaski. Wyniki klasyfikacji z użyciem modelu sieci uczonego przez 200 epok przedstawia tabela 4.5.6.9. Czas uczenia modelu wynosi 9352.3184 sekund, czyli około 156 minut (w przybliżeniu 2 godziny 36 minut).

Rysunek. 4.5.6.8. Wykres uczenia modelu sieci przez 200 epok

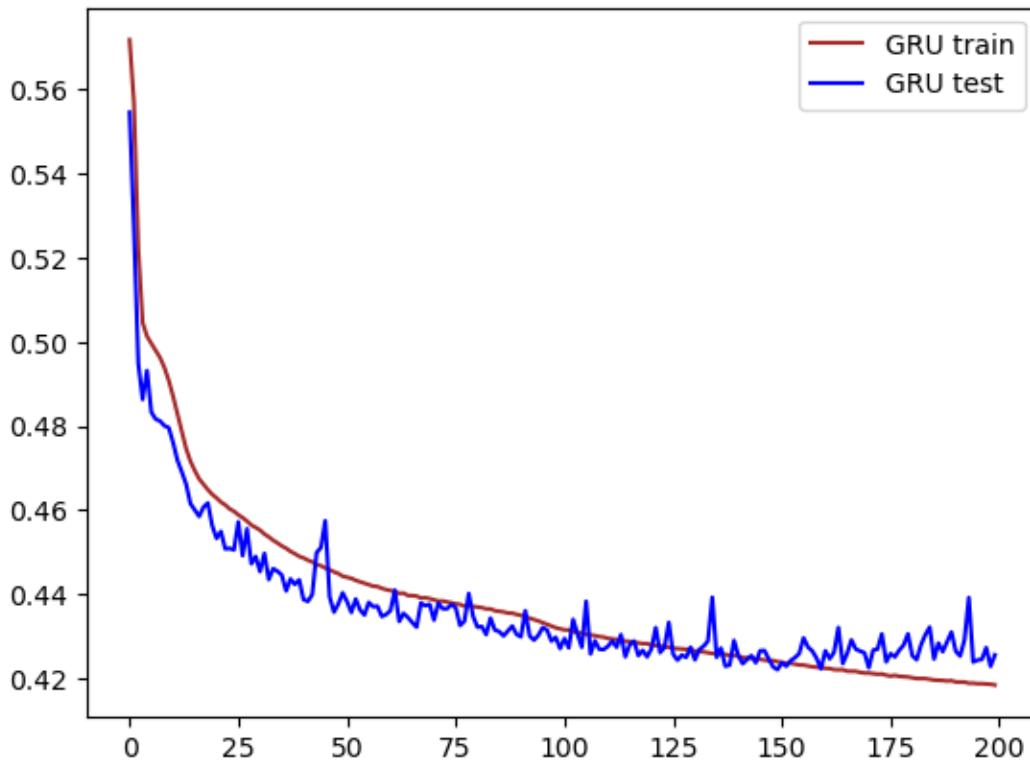


Tabela. 4.5.6.9. Wyniki klasyfikacji wyników przez model sieci uczony przez 200 epok

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6066
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8317
Dokładność klasyfikacji dla wszystkich wyników	0.7857

Porównanie ilości epok

Model, który uczy się przez 140 epok osiąga najlepsze wyniki, zaś mimo iż wyniki walidacji dla modelu uczonego przez 200 epok teoretycznie są odrobinę większe niż przy 140 epokach to po epoce około 150 występuje przy ostatnich 50 epokach brak uczenia (funkcja błędu dla danych walidacyjnych staje się płaska). W wypadku uczenia modelu przez 80 epok, sieć się uczy, ale nie jest tak dobrze nauczona jak 140 epokach przez co osiąga gorsze wyniki walidacji wyników. Sieć wykorzystująca ten konkretny model uczy się najlepiej, gdy liczba epok wynosi w przybliżeniu około 140.

4.5.6. Podsumowanie pozostałych hiperparametrów i ich wyników

Najlepsze wyniki sieć osiąga, gdy są zastosowane następujące hiperparametry: funkcja aktywacji dla komórek GRU jest funkcją Tanh, metryka jest metryką AUC/accuracy (metryka nie ma wpływu na wynik), zaś parametr shuffle jest ustawiony na true (zostawiony domyślnie), funkcja błędu jest funkcją BinaryCrossentropy, zastosowany w uczeniu batch ma rozmiar 16368 (dla domyślnego współczynnika uczenia jest to batch 128), a liczba epok wynosi 140. Ponadto sieć wykorzystująca funkcję aktywacji tanh w komórkach typu GRU ma krótszy czas procesu uczenia niż wykorzystująca funkcję aktywacji relu, co w połączeniu z dużym batchem daje bardzo małe czasy uczenia (przy batchu 16368 z optymalizatorem ADAM z LR 0.0007 i ilością epok 140 jest to około **5 minut**).

4.6. Porównanie różnych wielkości okna czasowego

Okno czasowe jest to wielkość grupy pomiarów, która otrzymuje etykietę na podstawie ostatniego pomiaru. Etykieta jest przyznawana na podstawie informacji czy dane pomiaru FCO2_RAW w ostatnim wierszu danej grupy są HQ (wysokiej jakości) czy też nie. Pozostałe dane FCO2_HQ w danym oknie czasowym nie mają wpływu na przyznaną etykietę. Przy czym informacje zawarte w kolumnie FCO2_HQ nie są przekazywane danej i nie uczestniczą w procesie uczenia i walidacji modelu (wyniki są walidowane tylko na postawie etykiet całego okna). Przetestowane zostały okna czasowe o wielkościach **4, 16 i 32**. Zastosowany model sieci jest to model sieci neuronowej z podpunktu 4.4.2., składający się z pięciu warstw (trzech pierwszych z komórkami typu GRU oraz dwóch typu gęstego). Posiada on 128 neurony typu GRU w warstwie pierwszej, 64 komórek typu GRU w warstwie drugiej, 32 komórki w trzeciej warstwie GRU oraz odpowiednio 50 i jeden neuron w warstwach gęstych. Warstwa pierwsza i druga typu GRU posiada parametr return_sequences (zwracaj sekwencje) ustawiony na wartość True (prawda). Funkcje aktywacji komórek typu GRU to tangens hiperboliczny, zaś komórek typu gęstego sigmoidalna funkcja aktywacji. Metryka metryką zastosowaną do uczenia było accuracy (dokładność). Optymalizatorem stosowanym do uczenia sieci był optymalizator SGD. Optymalizatorem zastosowanym w sieci tej jest optymalizator SGD. Wielkość batcha wynosi 128.

4.6.1. Okno czasowe nr 1

Przykładowe okno czasowe o wielkości 4 przedstawia rysunek 4.6.1.1. Etykieta przyznana temu oknu wynosi 0, gdyż ostatni pomiar nie był HQ. O tym czy pomiar jest HQ czy nie mówi pierwsza kolumna danych, pomiary są ułożone w kolejności chronologicznej od najstarszych do najnowszych. Przy czym informacje zawarte w kolumnie FCO2_HQ nie są przekazywane danej i nie uczestniczą w procesie uczenia i walidacji modelu (wyniki są walidowane tylko na postawie etykiet całego okna). Przykładowe okno czasowe, które uczestniczy w uczeniu modelu przedstawia rys.4.6.1.2. Informacje na temat budowy modelu oraz rozmiarze danych testowych i treningowych dla okna czasowego nr 1 przedstawia rys. 4.6.1.3.

Rysunek 4.6.1.1. Przykładowe okno czasowe o wielkości 4 przed usunięciem informacji FCO2_HQ

[[[0. 0.49157834 0.86486486 0.40034662 0.40213523 0.02145491 0.00307325 0.50816583 0.32602568 0.54202192 0.3442623 0.93694444 0.04504505 0.80035651 0.02466828 0.00174415]
[0. 0.49215751 0.86486486 0.40034662 0.40213523 0.02656721 0.00375619 0.50722362 0.32633887 0.54202192 0.39344262 0.93833333 0.04504505 0.80035651 0.03004879 0.00207956]
[0. 0.49206274 0.86486486 0.40034662 0.40213523 0.01969494 0.00273177 0.50942211 0.32602568 0.5408039 0.44262295 0.93833333 0.04504505 0.80035651 0.02261996 0.00160998]
[0. 0.49188372 0.86486486 0.40034662 0.40391459 0.02732149 0.00392693 0.50753769 0.32633887 0.53958587 0.41803279 0.92527778 0.04954955 0.80035651 0.03064156 0.00221373]]]

Rysunek 4.6.1.2. Przykładowe okno czasowe o wielkości 4

```
[[[0.49157834 0.86486486 0.40034662 0.40213523 0.02145491 0.00307325 0.50816583 0.32602568 0.54202192 0.3442623 0.93694444 0.04504505 0.80035651 0.02466828 0.00174415]
[0.49215751 0.86486486 0.40034662 0.40213523 0.02656721 0.00375619 0.50722362 0.32633887 0.54202192 0.39344262 0.93833333 0.04504505 0.80035651 0.03004879 0.00207956]
[0.49206274 0.86486486 0.40034662 0.40213523 0.01969494 0.00273177 0.50942211 0.32602568 0.5408039 0.44262295 0.93833333 0.04504505 0.80035651 0.02266199 0.00160998]
[0.49188372 0.86486486 0.40034662 0.40391459 0.02732149 0.00392693 0.50753769 0.32633887 0.53958587 0.41803279 0.92527778 0.04954955 0.80035651 0.03064156 0.00221373]]]
```

Rysunek 4.6.1.3. Informacje o modelu sieci dla okna czasowego o wielkości 4

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380266, 4, 15) (380266, 1) (95067, 4, 15) (95067, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 4, 128)           55680
gru_1 (GRU)                  (None, 4, 64)            37248
gru_2 (GRU)                  (None, 32)              9408
dense (Dense)                (None, 50)              1650
dense_1 (Dense)              (None, 1)               51
=================================================================
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0
```

4.6.2. Wyniki dla okna czasowego nr 1

Wykres uczenia sieci wykorzystując okno czasowe o wielkości 4 przedstawia rysunek 4.6.2.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci wykorzystującego okno czasowe o rozmiarze 4 przedstawia tabela 4.6.2.2. Czas uczenia modelu wynosi 7170.9076 sekund, czyli około 120 minut (w przybliżeniu 2 godziny).

Rysunek. 4.6.2.1. Wykres uczenia z użyciem okna czasowego o rozmiarze 4

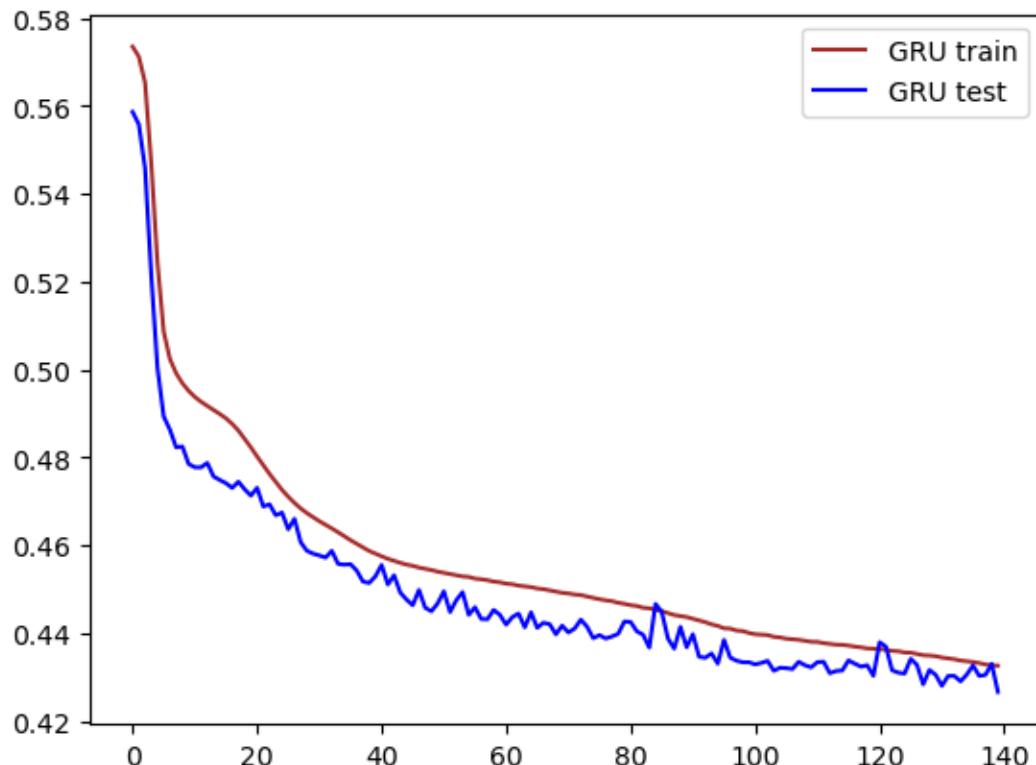


Tabela. 4.6.2.2. Wyniki klasyfikacji z użyciem okna czasowego o rozmiarze 4

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6061
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8155
Dokładność klasyfikacji dla wszystkich wyników	0.7790

4.6.3. Okno czasowe nr 2

Przykładowe okno czasowe o wielkości 16 przedstawia rysunek 4.6.3.1. Etykieta przyznana temu oknu wynosi 1, gdyż ostatni pomiar był pomiarem wysokiej jakości HQ. O tym czy pomiar jest wysokiej jakości czy nie mówi pierwsza kolumna danych, pomiary są ułożone w kolejności chronologicznej od najstarszych do najnowszych Przy czym informacje zawarte w kolumnie FCO2_HQ nie są przekazywane danej i nie uczestniczą w procesie uczenia i walidacji modelu (wyniki są walidowane tylko na postawie etykiet całego okna). Przykładowe okno czasowe, które uczestniczy w uczeniu modelu przedstawia rys.4.6.3.2. Informacje na temat budowy modelu oraz rozmiarze danych testowych i treningowych dla okna czasowego nr 2 przedstawia rys. 4.6.3.3.

Rysunek 4.6.3.1. Przykładowe okno czasowe o wielkości 16 przed usunięciem informacji FCO2_HQ

[[[0.	0.49204694	0.86579683	0.3864818	0.39323843	0.	0.	0.50942211	0.32226746	0.57003654	0.3442623	0.92472222	0.04504505	0.80035651	0.00041038	0.]
[0.	0.492326	0.86579683	0.3864818	0.39323843	0.	0.	0.50879397	0.32226746	0.57003654	0.36065574	0.92916667	0.04504505	0.80035651	0.00036478	0.]
[0.	0.49236286	0.86579683	0.3864818	0.39323843	0.	0.	0.50722362	0.32226746	0.57003654	0.36065574	0.92805556	0.04504505	0.80035651	0.00022799	0.]
[0.	0.49419514	0.8667288	0.3864818	0.39323843	0.	0.	0.50628141	0.32226746	0.57003654	0.36885246	0.92444444	0.04504505	0.80035651	0.00013679	0.]
[0.	0.49303153	0.8667288	0.3864818	0.39323843	0.	0.	0.50565327	0.32226746	0.56881851	0.3442623	0.92444444	0.04504505	0.80035651	0.0000912	0.]
[0.	0.49213645	0.8667288	0.3864818	0.39323843	0.	0.	0.50502513	0.32226746	0.56638246	0.39344262	0.92166667	0.04504505	0.80035651	0.0000456	0.]
[0.	0.49095705	0.8667288	0.3882149	0.39501779	0.	0.	0.50471106	0.32258065	0.56516443	0.40983607	0.93194444	0.04504505	0.80035651	0.	0.]
[0.	0.4910255	0.8667288	0.3882149	0.39501779	0.	0.	0.50471106	0.32258065	0.56516443	0.35245902	0.93611111	0.04504505	0.80035651	0.	0.]
[0.	0.49203641	0.8667288	0.3882149	0.39501779	0.	0.	0.50471106	0.32258065	0.56516443	0.28688525	0.94305556	0.04504505	0.80035651	0.	0.]
[0.	0.49152569	0.8667288	0.3882149	0.39501779	0.	0.	0.50471106	0.32258065	0.56638246	0.35245902	0.94416667	0.04504505	0.80035651	0.	0.]
[0.	0.49196797	0.8667288	0.3882149	0.39501779	0.	0.	0.50376884	0.32258065	0.56881851	0.32786885	0.95611111	0.04504505	0.80035651	0.	0.]
[0.	0.49152043	0.8667288	0.3882149	0.39501779	0.	0.	0.50345477	0.32258065	0.57369062	0.37704918	0.95833333	0.04504505	0.80035651	0.	0.]
[0.	0.49542193	0.8667288	0.3882149	0.39501779	0.	0.	0.50345477	0.32258065	0.5773447	0.3852459	0.9425	0.04504505	0.80035651	0.	0.]
[0.	0.49180475	0.8667288	0.38994801	0.39501779	0.	0.	0.50376884	0.32289383	0.57856273	0.36885246	0.94416667	0.04504505	0.80035651	0.	0.]
[0.	0.49179948	0.8667288	0.38994801	0.39501779	0.	0.	0.50345477	0.32258065	0.57978076	0.40983607	0.95027778	0.04504505	0.80035651	0.	0.]
[1.	0.49195217	0.86579683	0.38994801	0.39679715	0.	0.	0.5031407	0.32289383	0.57978076	0.35245902	0.95083333	0.04504505	0.80035651	0.	0.]]]

Rysunek 4.6.1.2. Przykładowe okno czasowe o wielkości 16

[[[0.49157834	0.86486486	0.40034662	0.40213523	0.02145491	0.00307325	0.50816583	0.32602568	0.54202192	0.3442623	0.93694444	0.04504505	0.80035651	0.02466828	0.00174415]]
[0.49215751	0.86486486	0.40034662	0.40213523	0.02656721	0.00375619	0.50722362	0.32633887	0.54202192	0.39344262	0.93833333	0.04504505	0.80035651	0.03004879	0.00207956]]
[0.49206274	0.86486486	0.40034662	0.40213523	0.01969494	0.00273177	0.50942211	0.32602568	0.5408039	0.44262295	0.93833333	0.04504505	0.80035651	0.02266199	0.00160998]]
[0.49188372	0.86486486	0.40034662	0.40391459	0.02732149	0.00392693	0.50753769	0.32633887	0.53958587	0.41803279	0.92527778	0.04954955	0.80035651	0.03064156	0.00221373]]
[0.49244183	0.86486486	0.40034662	0.40213523	0.02388535	0.00341472	0.50659548	0.32602568	0.53714982	0.40163934	0.93027778	0.04954955	0.80035651	0.02708495	0.00194539]]
[0.49212066	0.86486486	0.39688042	0.40213523	0.02623198	0.00375619	0.50439698	0.32633887	0.53471376	0.40983607	0.92444444	0.04954955	0.80035651	0.02941042	0.00214664]]
[0.49122031	0.86486486	0.39514731	0.40213523	0.02505867	0.00358545	0.5053392	0.3257125	0.53471376	0.36885246	0.93694444	0.04954955	0.80035651	0.02813369	0.00201248]]
[0.49105709	0.86486486	0.39514731	0.40035587	0.02614817	0.00358545	0.5053392	0.3257125	0.53593179	0.45901639	0.91777778	0.04954955	0.80035651	0.02922803	0.00214664]]
[0.49202062	0.86486486	0.39514731	0.40035587	0.02681864	0.00358545	0.50282663	0.3257125	0.53836784	0.40983607	0.93027778	0.04954955	0.80035651	0.02995759	0.00214664]]
[0.49046212	0.86486486	0.39514731	0.40035587	0.0263996	0.00375619	0.50062814	0.3257125	0.54323995	0.45901639	0.93166667	0.04954955	0.80035651	0.02931923	0.00201248]]
[0.49146251	0.86486486	0.39514731	0.40035587	0.02313108	0.00324398	0.50219849	0.3257125	0.54689403	0.3852459	0.91916667	0.04954955	0.80035651	0.02567142	0.00187831]]
[0.4917205	0.86486486	0.39514731	0.40213523	0.02003017	0.00273177	0.50125628	0.3257125	0.54933009	0.36885246	0.9275	0.04954955	0.80035651	0.0223428	0.0015429]]
[0.49204168	0.86486486	0.39514731	0.40213523	0.01533691	0.00204883	0.50219849	0.32539931	0.55176614	0.39344262	0.93222222	0.04504505	0.80035651	0.01732707	0.0011404]
[0.49211013	0.86486486	0.39514731	0.40213523	0.0132417	0.00170736	0.50282663	0.32539931	0.55054811	0.40983607	0.92916667	0.04504505	0.80035651	0.0148648	0.00093916]]
[0.49158361	0.86486486	0.39514731	0.40213523	0.01173315	0.00136589	0.5031407	0.32539931	0.54933009	0.3442623	0.94083333	0.04504505	0.80035651	0.01326889	0.00080499]]
[0.49188899	0.86486486	0.39514731	0.40213523	0.00879987	0.00085368	0.5053392	0.32508613	0.54933009	0.40163934	0.93027778	0.04504505	0.80035651	0.01012266	0.00053666]]]]

Rysunek 4.6.3.3. Informacje o modelu sieci dla okna czasowego o wielkości 16

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380256, 16, 15) (380256, 1) (95065, 16, 15) (95065, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=50, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
gru (GRU)            (None, 16, 128)      55680
gru_1 (GRU)          (None, 16, 64)       37248
gru_2 (GRU)          (None, 32)           9408
dense (Dense)        (None, 50)           1650
dense_1 (Dense)      (None, 1)            51
=====
Total params: 104,037
Trainable params: 104,037
Non-trainable params: 0

```

4.6.4. Wyniki dla okna czasowego nr 2

Wykres uczenia sieci wykorzystując okno czasowe o wielkości 16 przedstawia rysunek 4.6.4.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci wykorzystującego okno czasowe o rozmiarze 16 przedstawia tabela 4.6.4.2. Czas uczenia modelu wynosi 7851.5561 sekund, czyli około 131 minut (w przybliżeniu 2 godziny 11 minut).

Rysunek. 4.6.4.1. Wykres uczenia z użyciem okna czasowego o rozmiarze 16

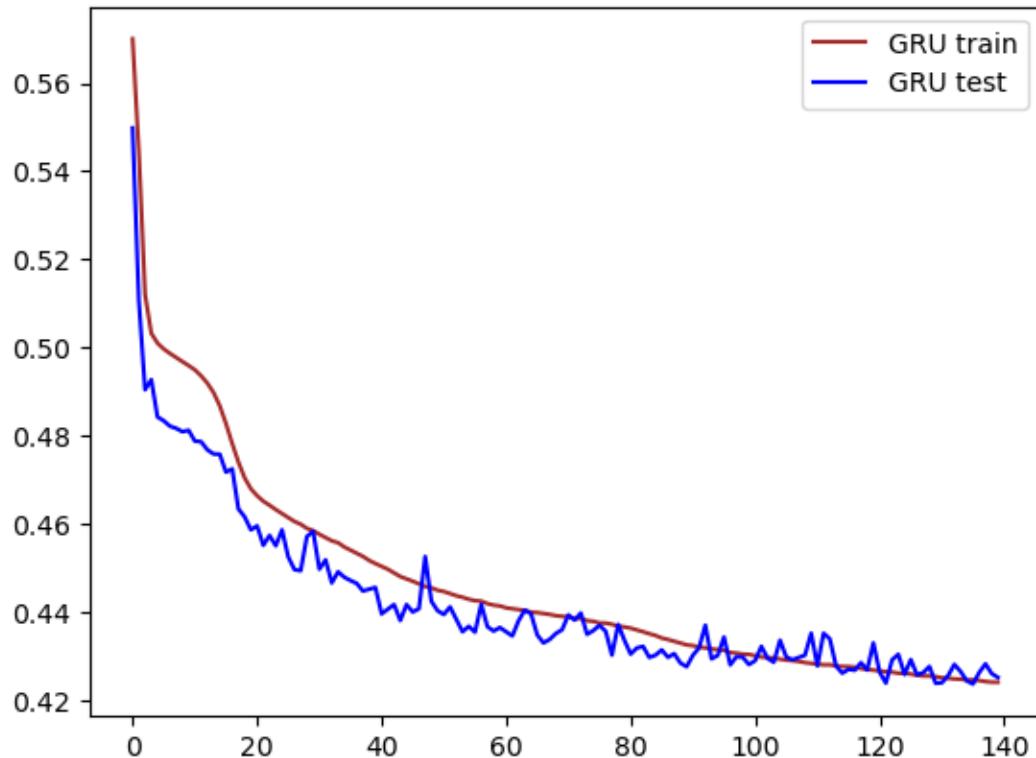


Tabela. 4.6.4.2. Wyniki klasyfikacji z użyciem okna czasowego o rozmiarze 16

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6116
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8230
Dokładność klasyfikacji dla wszystkich wyników	0.7836

4.6.5. Okno czasowe nr 3

Przykładowe okno czasowe o wielkości 32 przedstawia rysunek 4.6.5.1. Etykieta przyznana temu oknu wynosi 0, gdyż ostatni pomiar nie był HQ. O tym czy pomiar jest HQ czy nie mówi pierwsza kolumna danych, pomiary są ułożone w kolejności chronologicznej od najstarszych do najnowszych. Przy czym informacje zawarte w kolumnie FCO2_HQ nie są przekazywane danej i nie uczestniczą w procesie uczenia i validacji modelu (wyniki są walidowane tylko na postawie etykiet całego okna). Przykładowe okno czasowe, które uczestniczy w uczeniu modelu przedstawia rys.4.6.5.2. Informacje na temat budowy modelu oraz rozmiarze danych testowych i treningowych dla okna czasowego nr 3 przedstawia rys. 4.6.5.3.

Rysunek 4.6.5.1. Przykładowe okno czasowe o wielkości 32 przed usunięciem informacji FCO2_HQ

[[[0.	0.49157834 0.86486486 0.40034662 0.40213523 0.02145491 0.00307325 0.50816583 0.32602568 0.54202192 0.3442623 0.93694444 0.04504505 0.80035651 0.02466828 0.00174415]
[0.	0.49215751 0.86486486 0.40034662 0.40213523 0.02656721 0.00375619 0.50722362 0.32633887 0.54202192 0.39344262 0.93833333 0.04504505 0.80035651 0.03004879 0.00207956]
[0.	0.49206274 0.86486486 0.40034662 0.40213523 0.01969494 0.00273177 0.50942197 0.32602568 0.54080393 0.4426295 0.93833333 0.04504505 0.80035651 0.02266199 0.00160998]
[0.	0.49188372 0.86486486 0.40034662 0.40391459 0.02732149 0.00392693 0.50753769 0.32633887 0.53958587 0.41803279 0.92527778 0.04954955 0.80035651 0.03064156 0.00221373]
[0.	0.49244183 0.86486486 0.40034662 0.40213523 0.02388535 0.00341472 0.50659548 0.32602568 0.53714982 0.40163934 0.93027778 0.04954955 0.80035651 0.02708495 0.00194539]
[0.	0.49212066 0.86486486 0.40034662 0.40213523 0.02623198 0.00375619 0.50439698 0.32633887 0.53471376 0.40983607 0.92444444 0.04954955 0.80035651 0.02941042 0.00214664]
[0.	0.49122031 0.86486486 0.39514731 0.40213523 0.02505867 0.00358545 0.5053392 0.3257125 0.53471376 0.36885246 0.93694444 0.04954955 0.80035651 0.02813369 0.00202148]
[0.	0.49105709 0.86486486 0.39514731 0.40035587 0.02614817 0.00358545 0.5053392 0.3257125 0.535593179 0.45901639 0.91777778 0.04954955 0.80035651 0.02922803 0.00214664]
[0.	0.49202062 0.86486486 0.39514731 0.40035587 0.02681864 0.00358545 0.50282663 0.3257125 0.53836784 0.40983607 0.93027778 0.04954955 0.80035651 0.02995759 0.00214664]
[0.	0.49046212 0.86486486 0.39514731 0.40035587 0.0263996 0.00375619 0.50062814 0.3257125 0.54323995 0.45901639 0.93166667 0.04954955 0.80035651 0.02931923 0.00202148]
[0.	0.49146251 0.86486486 0.39514731 0.40035587 0.02313108 0.00324398 0.50219849 0.3257125 0.54689403 0.3852459 0.91916667 0.04954955 0.80035651 0.02567142 0.00187831]
[0.	0.4917205 0.86486486 0.39514731 0.40213523 0.02003017 0.00273177 0.50125628 0.3257125 0.54933009 0.36885246 0.9275 0.04954955 0.80035651 0.0223428 0.0015429]
[0.	0.49204168 0.86486486 0.39514731 0.40213523 0.01533691 0.00204883 0.50219849 0.32539931 0.55176614 0.39344262 0.93222222 0.04504505 0.80035651 0.01732707 0.0011404]
[0.	0.49211013 0.86486486 0.39514731 0.40213523 0.0132417 0.00170736 0.50282663 0.32539931 0.55054811 0.40983607 0.92916667 0.04504505 0.80035651 0.0148648 0.00093916]
[0.	0.49158361 0.86486486 0.39514731 0.40213523 0.0173135 0.00136589 0.5031407 0.32539931 0.54933009 0.342623 0.94083333 0.04504505 0.80035651 0.01326889 0.00080499]
[0.	0.49188899 0.86486486 0.39514731 0.40213523 0.00879987 0.00085368 0.5053392 0.32508613 0.54933009 0.40163934 0.93027778 0.04504505 0.80035651 0.01012266 0.00053666]
[0.	0.49182054 0.86486486 0.39514731 0.40213523 0.00846463 0.00086294 0.50565632 0.32508613 0.54811206 0.3852459 0.94638889 0.04504505 0.80035651 0.00952989 0.0046958]
[0.	0.49128876 0.86486486 0.39514731 0.40213523 0.00829702 0.00068294 0.50596734 0.32508613 0.54445798 0.3852459 0.95638889 0.04504505 0.80035651 0.0093931 0.00053666]
[0.	0.49174156 0.86486486 0.39514731 0.40213523 0.0101408 0.00102442 0.50125628 0.32477294 0.54323995 0.39344262 0.94555556 0.04504505 0.80035651 0.01144499 0.00067083]
[0.	0.49089913 0.86486486 0.39514731 0.40213523 0.00846463 0.00086294 0.50219849 0.32477294 0.536065574 0.94694444 0.04504505 0.80035651 0.00943869 0.00046958]
[0.	0.49235232 0.86486486 0.39341421 0.40213523 0.00720751 0.00034147 0.50345477 0.32445976 0.54689403 0.36065574 0.94583333 0.04504505 0.80035651 0.00820756 0.00033541]
[0.	0.49145198 0.86486486 0.39341421 0.40035587 0.00620181 0.00034147 0.50376884 0.32445976 0.55054811 0.37704918 0.93972222 0.04504505 0.80035651 0.00711322 0.00020125]
[0.	0.49192058 0.86486486 0.39168111 0.40035587 0.00486088 0.00017074 0.50471106 0.32445976 0.55420219 0.3442623 0.9311111 0.04504505 0.80035651 0.0055629 0.00013417]
[0.	0.49196797 0.86486486 0.39168111 0.39857651 0.00385518 0.50565327 0.32414657 0.5590743 0.42622951 0.93583333 0.04504505 0.80035651 0.00460535 0.00006708]
[0.	0.49114466 0.86486486 0.39168111 0.39857651 0.00293329 0.50690955 0.32414657 0.5590743 0.29508197 0.93027778 0.04504505 0.80035651 0.0037846 0.00020125]
[0.	0.49273668 0.86486486 0.39168111 0.39857651 0.00217901 0.50690955 0.32383339 0.56029233 0.39344262 0.95388889 0.04504505 0.80035651 0.00310063 0.00020125]
[0.	0.49355279 0.86486486 0.39168111 0.39857651 0.00159236 0.50753769 0.32383339 0.56029233 0.37704918 0.9375 0.04504505 0.80035651 0.00246227 0.00013417]
[0.	0.49240498 0.86486486 0.39168111 0.39857651 0.00083808 0.50847499 0.32383339 0.56151035 0.31147541 0.93166667 0.04504505 0.80035651 0.00168711 0.00006708]
[0.	0.49179422 0.86486486 0.39894801 0.39679715 0.00067047 0.50816583 0.3235202 0.5590743 0.32477294 0.54445798 0.36065574 0.94694444 0.04504505 0.80035651 0.00136793 0.00006708]
[0.	0.49168891 0.86486486 0.38994801 0.39679715 0.00025142 0.50816583 0.3235202 0.5590743 0.35245902 0.935 0.04504505 0.80035651 0.00086635 0.00006708]
[0.	0.49123611 0.86486486 0.38994801 0.39679715 0. 0. 0.50847499 0.3235202 0.55785627 0.39344262 0.91333333 0.04504505 0.80035651 0.00045598 0.]
[0.	0.48884571 0.86486486 0.38994801 0.39679715 0. 0. 0.50879397 0.32320702 0.5590743 0.34342623 0.9211111 0.04504505 0.80035651 0.0000912 0.]]]

Rysunek 4.6.5.2. Przykładowe okno czasowe o wielkości 32

[[[0.49157834 0.86486486 0.40034662 0.40213523 0.02145491 0.00307325 0.50816583 0.32602568 0.54202192 0.3442623 0.93694444 0.04504505 0.80035651 0.02466828 0.00174415]
[0.49215751 0.86486486 0.40034662 0.40213523 0.02656721 0.00375619 0.50722362 0.32633887 0.54202192 0.39344262 0.93833333 0.04504505 0.80035651 0.03004879 0.00207956]
[0.49206274 0.86486486 0.40034662 0.40213523 0.01969494 0.00273177 0.50942197 0.32602568 0.54080393 0.4426295 0.93833333 0.04504505 0.80035651 0.02266199 0.00160998]
[0.49188372 0.86486486 0.40034662 0.40391459 0.02732149 0.00392693 0.50753769 0.32633887 0.53958587 0.41803279 0.92527778 0.04954955 0.80035651 0.03064156 0.00221373]
[0.49244183 0.86486486 0.40034662 0.40213523 0.02388535 0.00341472 0.50659548 0.32602568 0.53714982 0.40163934 0.93027778 0.04954955 0.80035651 0.02708495 0.00194539]
[0.49212066 0.86486486 0.40034662 0.40213523 0.02623198 0.00375619 0.50439698 0.32633887 0.53471376 0.40983607 0.92444444 0.04954955 0.80035651 0.02941042 0.00214664]
[0.49122031 0.86486486 0.39514731 0.40213523 0.02505867 0.00358545 0.5053392 0.3257125 0.53471376 0.36885246 0.93694444 0.04954955 0.80035651 0.02813369 0.00202148]
[0.49105709 0.86486486 0.39514731 0.40035587 0.02614817 0.00358545 0.5053392 0.3257125 0.535593179 0.45901639 0.91777778 0.04954955 0.80035651 0.02922803 0.00214664]
[0.49202062 0.86486486 0.39514731 0.40035587 0.02681864 0.00358545 0.50282663 0.3257125 0.53836784 0.40983607 0.93027778 0.04954955 0.80035651 0.02995759 0.00214664]
[0.49046212 0.86486486 0.39514731 0.40035587 0.0263996 0.00375619 0.50062814 0.3257125 0.54323995 0.45901639 0.9311111 0.04504505 0.80035651 0.0055629 0.00013417]
[0.49196797 0.86486486 0.39514731 0.40035587 0.02313108 0.00324398 0.50219849 0.3257125 0.54689403 0.3852459 0.91916667 0.04954955 0.80035651 0.02567142 0.00187831]
[0.49114466 0.86486486 0.39514731 0.40213523 0.02003017 0.00273177 0.50125628 0.3257125 0.54933009 0.36885246 0.9275 0.04954955 0.80035651 0.0223428 0.0015429]
[0.49204168 0.86486486 0.39514731 0.40213523 0.01533691 0.00204883 0.50219849 0.32539931 0.55176614 0.39344262 0.93222222 0.04504505 0.80035651 0.01732707 0.0011404]
[0.49211013 0.86486486 0.39514731 0.40213523 0.0132417 0.00170736 0.50282663 0.32539931 0.55054811 0.40983607 0.92916667 0.04504505 0.80035651 0.0148648 0.00093916]
[0.49158361 0.86486486 0.39514731 0.40213523 0.0173135 0.00136589 0.5031407 0.32539931 0.54933009 0.342623 0.94083333 0.04504505 0.80035651 0.01326889 0.00080499]
[0.49188899 0.86486486 0.39514731 0.40213523 0.00879987 0.00085368 0.5053392 0.32508613 0.54933009 0.40163934 0.93027778 0.04504505 0.80035651 0.01012266 0.00053666]
[0.49182054 0.86486486 0.39514731 0.40213523 0.00846463 0.00086294 0.50565632 0.32508613 0.54811206 0.3852459 0.94638889 0.04504505 0.80035651 0.00952989 0.0046958]
[0.49128876 0.86486486 0.39514731 0.40213523 0.00829702 0.00068294 0.50596734 0.32508613 0.54445798 0.3852459 0.95638889 0.04504505 0.80035651 0.0093931 0.00053666]
[0.49174156 0.86486486 0.39514731 0.40213523 0.0101408 0.00102442 0.50125628 0.32477294 0.54323995 0.39344262 0.94555556 0.04504505 0.80035651 0.01144499 0.00067083]
[0.49089913 0.86486486 0.39514731 0.40213523 0.00846463 0.00068294 0.50219849 0.32477294 0.54445798 0.36065574 0.94694444 0.04504505 0.80035651 0.00943869 0.00046958]
[0.49235232 0.86486486 0.39341421 0.40213523 0.00720751 0.00034147 0.50345477 0.32445976 0.54689403 0.36065574 0.94583333 0.04504505 0.80035651 0.00820756 0.00033541]
[0.49145198 0.86486486 0.39341421 0.40035587 0.00620181 0.00034147 0.50376884 0.32445976 0.55054811 0.37704918 0.93972222 0.04504505 0.80035651 0.00711322 0.00020125]
[0.49196797 0.86486486 0.39168111 0.40035587 0.00486088 0.00017074 0.50471106 0.32445976 0.55420219 0.3442623 0.9311111 0.04504505 0.80035651 0.0055629 0.00013417]
[0.49114466 0.86486486 0.39168111 0.39857651 0.00293329 0.50690955 0.32414657 0.5590743 0.29508197 0.93027778 0.04504505 0.80035651 0.0037846 0.00020125]
[0.49273668 0.86486486 0.39168111 0.39857651 0.00217901 0.50690955 0.32383339 0.56029233 0.39344262 0.95388889 0.04504505 0.80035651 0.00310063 0.00020125]
[0.49355279 0.86486486 0.39168111 0.39857651 0.00159236 0.50753769 0.32383339 0.56029233 0.37704918 0.9375 0.04504505 0.80035651 0.0246227 0.00013417]
[0.49240498 0.86486486 0.39168111 0.39857651 0.00083808 0.50847499 0.32383339 0.56151035 0.31147541 0.

Rysunek 4.6.5.3. Informacje o modelu sieci dla okna czasowego o wielkości 32

```

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(380244, 32, 15) (380244, 1) (95061, 32, 15) (95061, 1)

model_gru = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    keras.layers.GRU(units=64, return_sequences=True),
    keras.layers.GRU(units=32),
    keras.layers.Dense(units=40, activation='sigmoid'),
    keras.layers.Dense(units=1, activation='sigmoid')])
model_gru.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

model_gru.summary()

Model: "sequential"
=====
Layer (type)                 Output Shape              Param #
=====
gru (GRU)                    (None, 32, 128)          55680
gru_1 (GRU)                  (None, 32, 64)           37248
gru_2 (GRU)                  (None, 32)              9408
dense (Dense)                (None, 40)              1320
dense_1 (Dense)              (None, 1)               41
=====
Total params: 103,697
Trainable params: 103,697
Non-trainable params: 0
=====
```

4.6.6. Wyniki dla okna czasowego nr 3

Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 4.6.6.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem modelu sieci wykorzystującego okno czasowe o rozmiarze 16 przedstawia tabela 4.6.6.2. Czas uczenia modelu wynosi 8681.4993 sekund, czyli około 145 minut (w przybliżeniu 2 godziny 25 minut).

Rysunek. 4.6.6.1. Wykres uczenia z użyciem okna czasowego o rozmiarze 32

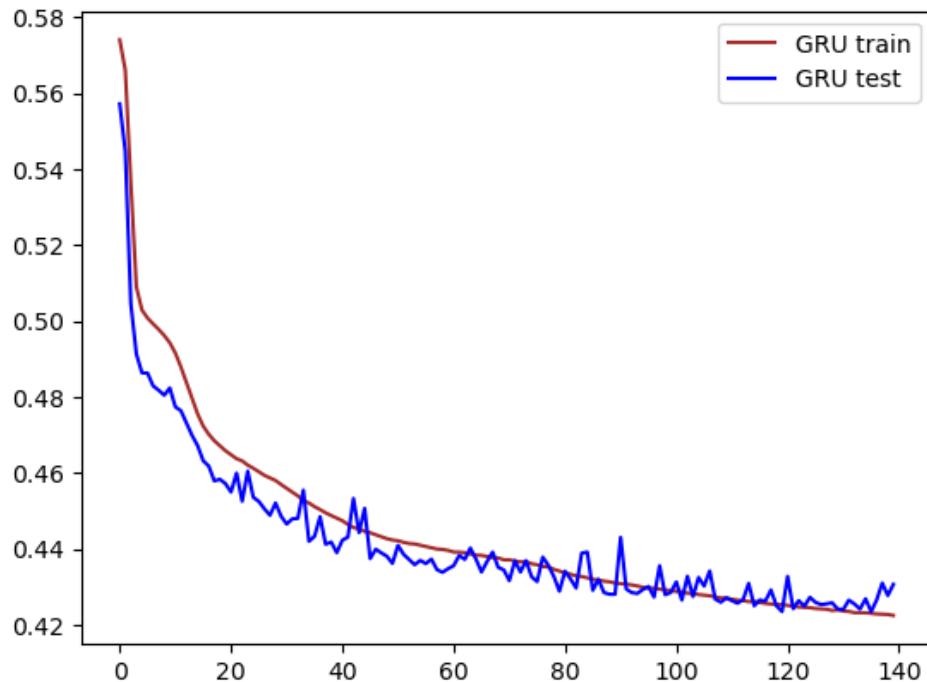


Tabela. 4.6.6.2. Wyniki klasyfikacji z użyciem okna czasowego o rozmiarze 32

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6090
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8235
Dokładność klasyfikacji dla wszystkich wyników	0.7831

4.6.7. Podsumowanie okien czasowych i ich wyników

Najlepsze wyniki daje użycie okna czasowego o wielkości 16, jednakże okno o rozmiarze 32 daje zbliżone wyniki, ale szybkość uczenia sieci z jego wykorzystaniem jest niemal dwa razy większe, co sprawia, że jest ono sensownym kompromisem pomiędzy dokładnym nauczeniem modelu, a czasem jego uczenia.

4.7. Pozostałe przetestowane modele sieci

Poza wymienionymi w pracy modelami sieci przetestowane zostały inne modele, ale nie uwzględniałem ich w treści pisemnej pracy, gdyż były one wariantami omówionych w pracy modeli, a opisywanie każdego modelu mijało się z celem, gdyż chciałem porównać te modele, które przedstawiały różne warianty hiperparametrów sieci. Informacje o wszystkich przetestowanych modelach sieci znajdują się w pliku ModelsComparision.xlsx. w zakładce modele_wszystkie, zaś

w zakładce modele_posortowane znajdują się wszystkie modele sieci posortowane w kolejności ich dokładności, tego czy się uczą itd. W zakładce modele_sieci znajdują się tabela z modelami sieci, które zostały omówione w części pisemnej niniejszej pracy inżynierskiej, zaś w zakładce posortowane_modele_sieci_praca_inżynierska znajdują się posortowane modele sieci opisane w pracy posortowane w kolejności według w kolejności ich dokładności, czasu wykonywania, tego czy się uczą itd.

4.8.Zastosowanie nazewnictwo plików

Cele ułatwienia szukania modeli sieci oraz archiwizacji notatniki Jupiter załączone do pracy są nazwane według następującej konwencji:

Thesis_OPTIMALIZATOR_NNLGRU1_NNLGRU2_NNLGRU3_NNLDENSE1_NNLDENSE2_windows_WS_batch_BS_LR_LRS.ipynb

Gdzie:

- Thesis – Praca inżynierska
- OPTIMALIZATOR – użyty optymalizator
- NNLGRU1- Sieć neuronowa (z ang. neural network), liczba komórek w warstwie typu GRU nr 1
- NNLGRU2- Sieć neuronowa (z ang. neural network), liczba komórek w warstwie typu GRU nr 2
- NNLGRU3- Sieć neuronowa (z ang. neural network), liczba komórek w warstwie typu GRU nr 3
- NNLDENSE1- Sieć neuronowa (z ang. neural network), liczba komórek w warstwie gęstej nr 1
- NNLDENSE2- Sieć neuronowa (z ang. neural network), liczba komórek w warstwie gęstej nr 2
- windows – informacja, że następna liczba to rozmiar okna
- WS – rozmiar okna (ang. Windows size)
- Batch - informacja, że następna liczba to rozmiar batcha
- BS - rozmiar batcha (ang. batch size)
- LR - informacja, że następna liczba to wielkość współczynnika uczenia
- LRS – wielkość współczynnika uczenia (ang. learning rate size), przy czym jest on zapisywany w formacie liczbaprzekropka_liczbapokropce np. learning rate 0.001 jest zapisany jako 0_001
- .ipynb – rozszerzenie pliku notatnika python

Przykład

Thesis_SGD_140_64_32_50_1_windows_16_batch_2048_LR_0_9.ipynb

4.9. Wybór najlepszego modelu sieci z najlepszymi jej hiperparametrami i rozmiarem okna

Podczas testowania modeli wyszło, że najlepsze rezultaty pod względem czasu i wyników daje użycie modelu o następujących parametrach:

- Funkcja aktywacji w warstwie nr 1 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 2 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 3 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 1 typu Dense: Sigmoid
- Funkcja aktywacji w warstwie nr 2 typu Dense: Sigmoid
- Liczba neuronów w warstwie typu GRU nr 1: 256
- Liczba neuronów w warstwie typu GRU nr 2: 128
- Liczba neuronów w warstwie typu GRU nr 3: 64
- Liczba neuronów w warstwie Gęstej nr 1(warstwa 4 sieci): 100
- Liczba neuronów w warstwie Gęstej nr 2(warstwa 5 sieci): 1
- Optymalizator: Adam (współczynnik uczenia 0.0004)
- Okno czasowe: 32
- Batch: 16384
- Metryka: accuracy
- Shuffle: true
- Funkcja błędu: Binary crossentropy
- Liczba epok: 140

Osiąga on najlepsze rezultaty w najszybszym czasie i dzięki zastosowaniu funkcji aktywacji komórek GRU [27] jako Tanh współpracuje on bardzo dobrze z kartami graficznymi firmy NVIDIA wykorzystującymi rdzenie CUDA [36] oraz Tensor. Duży rozmiar batcha sprawia, że model się szybko się uczy pod względem czasowym. Jego wadą jest fakt, iż wymaga systemu z dużą ilością pamięci RAM oraz wydajnej karty graficznej z dużą ilością VRAMU. Wykres uczenia sieci wykorzystując najlepszy model rysunek 4.9.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z użyciem najlepszego modelu sieci przedstawia tabela 4.9.2. oraz rysunek 4.9.3 Czas nauki modelu wynosi 305.7613 sekund (w przybliżeniu 5 minut 10 sekund).

Rysunek. 4.9.1. Wykres uczenia z użyciem najlepszego modelu

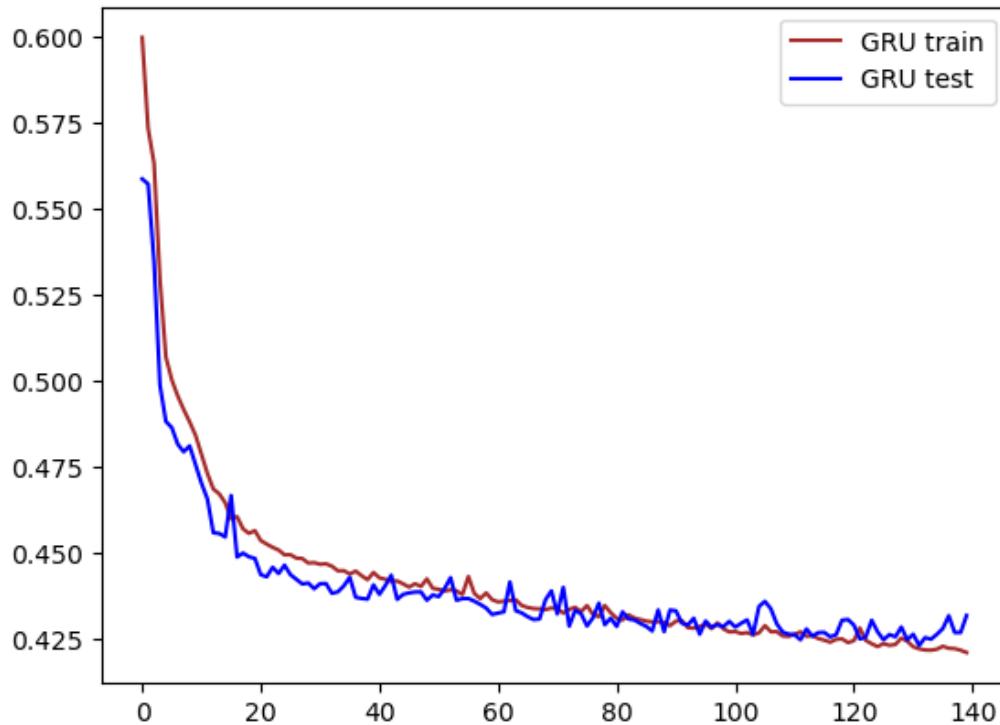
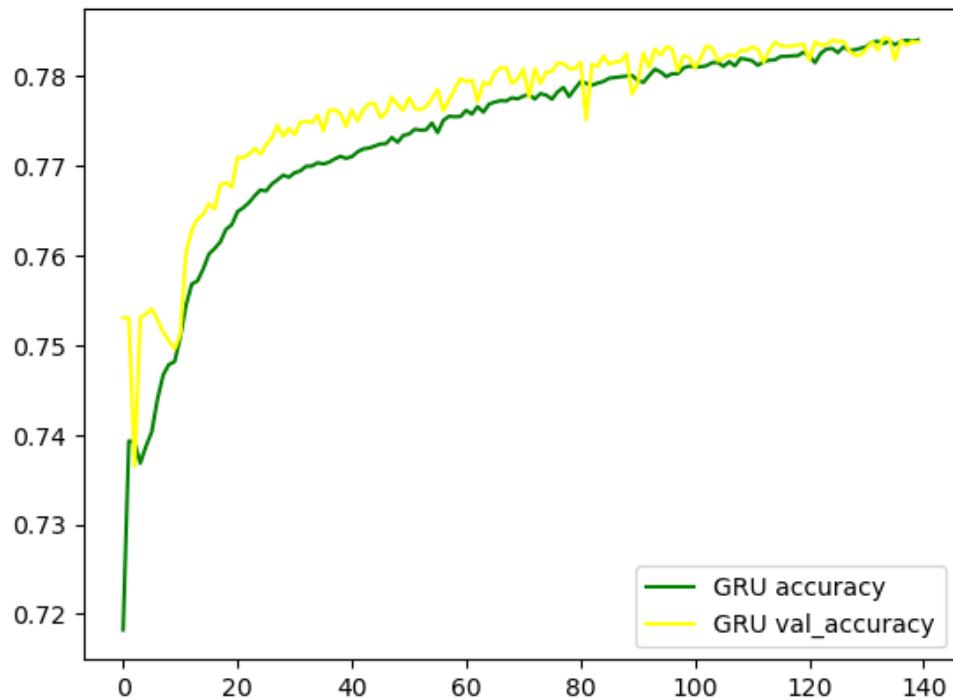


Tabela. 4.9.2. Wyniki klasyfikacji z użyciem najlepszego modelu

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6210
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8185
Dokładność klasyfikacji dla wszystkich wyników	0.7841

Rysunek. 4.9.3. Wykres dokładności z użyciem najlepszego modelu



4.10. Podsumowanie klasyfikacji binarnej wyników pomiarów emisji co2 na terenach bagiennych z użyciem komórek GRU

Podczas moich badań i poszukiwań najlepszego modelu sieci neuronowej celem klasyfikacji pomiarów meteorologicznych na terenach bagiennych udało mi się dowieść, że komórki GRU nadają się do tego bardzo dobrze, dla niektórych modeli sieci osiągają one wysokie wyniki dokładności ogólnej dla tych danych, nawet 78.81%, przy czym dane są obarczone pewnymi niedoskonałościami przy nadawaniu im etykiet (wysokiej jakości lub niskiej) oraz miejscowymi brakami etykiet na co nie mam wpływu. Klasyfikacja binarna z użyciem komórek GRU jest skuteczna, w niektórych modelach sieci wykrywanie wyników o wysokiej jakości oraz o niskiej jakości jest na dużym poziomie odpowiednio w przybliżeniu 61.4% i 83.2%. Sieć wykorzystująca komórki GRU, dzięki odpowiedniemu dobiorowi hiperparametrów oraz sprzętu jest w stanie robić to w krótkim czasie (rzędu kilku minut przy tak dużym zbiorze danych) bez utraty na jakości dokładności klasyfikacji.

ROZDZIAŁ 5. Wpływ GPU na wydajność uczenia modelu.

5.1. Procedura testowa

Testowanie modelu w punkcie 4 polegało na sprawdzaniu nauczenia sieci oraz jej dokładności w zależności od modelu sieci, jej hiperparametrów oraz wielkości okna. Ponadto testy obejmowały również sprawdzenie wpływu karty graficznej na szybkość obliczeń oraz ich dokładność na czym skupię się w tym punkcie. Wybór wielu hiperparametrów skutkował zwiększeniem czasu uczenia modelu przy takich zbliżonej dokładności lub/i ilości epok. Przykładem takiego hiperparametru jest funkcja aktywacji komórek GRU. Gdy jest wybrana inna niż tanh (na przykład stosowana przeze mnie funkcja aktywacji relu) czas uczenia modelu jest dużo dłuższy niż z wykorzystaniem funkcji tanh. Funkcja aktywacji komórek GRU będąca funkcją tanh, sprawia, że, gdy tensorflow korzysta z karty graficznej czas uczenia jest jeszcze krótszy. Celem zobrazowania tego zjawiska wybrałem najbardziej wymagającą sieć pod względem zasobów, ale przy tym wybrałem taką, która daje wyniki w postaci nauczzonego modelu sieci najwyższym stopniu dokładności. Nie wybierałem modelu o bardzo wysokiej zasobozerności np. model z 10000 neuronów GRU w pierwszej warstwie, 5000 GRU w drugiej itd. by tylko pokazać, że karta pomaga i potrzebne jest dużo pamięci RAM i VRAM, zaś sam model sieci nie jest optymalny i przeucza się po 4 epokach, gdzie tych epok byłoby np. 1000. Moim celem było pokazanie wpływu karty graficznej Nvidii wykorzystującej rdzenie cuda [37] oraz rdzenie tensor na obliczenia z użyciem modelu sieci, który jest standardowym pod względem użycia zasobów komputera oraz takim który pokazuje naukę modelu w dłuższej perspektywie czasowej (na przełomie epok). Jako model sieci wybrałem model o następujących parametrach:

- Funkcja aktywacji w warstwie nr 1 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 2 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 3 typu GRU: Tanh
- Funkcja aktywacji w warstwie nr 1 typu Dense: Sigmoid
- Funkcja aktywacji w warstwie nr 2 typu Dense: Sigmoid
- Liczba neuronów w warstwie typu GRU nr 1: 128
- Liczba neuronów w warstwie typu GRU nr 2: 64
- Liczba neuronów w warstwie typu GRU nr 3: 32
- Liczba neuronów w warstwie Gęstej nr 1(warstwa 4 sieci): 50
- Liczba neuronów w warstwie Gęstej nr 2(warstwa 5 sieci): 1
- Optymalizator: Adam (współczynnik uczenia 0.0007)
- Okno czasowe: 32
- Batch: 16384
- Metryka: accuracy
- Shuffle: true
- Funkcja błędu: Binary crossentropy
- Liczba epok: 140

5.2. Konfiguracje testowe

Konfiguracje testowe:

- Konfiguracja nr 1:
 - CPU: Intel Xeon E5-2667v2 (8 rdzeni 16 wątków) [38]
 - RAM: 32GB 1886MHz DDR3
 - GPU: Nvidia GeForce RTX 3060 12GB [39]
 - Storage: 512GB SSD SATA 3.0
 - OS: Windows 10
- Konfiguracja nr 1.1:
 - CPU: Intel Xeon E5-2667v2 (8 rdzeni 16 wątków)
 - RAM: 32GB 1886MHz DDR3
 - GPU: Nvidia GeForce RTX 3060 12GB (karta nie używana przy uczeniu modelu)
 - Storage: 512GB SSD SATA 3.0
 - OS: Windows 10
- Konfiguracja nr 2 Laptop MSI KATANA GF66-UE [40]:
 - CPU: Intel Core i7 11800H (8 rdzeni 16 wątków) [41]
 - RAM: 16GB 3200MHz DDR4/64GB 3200MHz DDR4(nastąpiła zmiana ilości ramu w trakcie pisania pracy na 64GB)
 - GPU1: Nvidia GeForce RTX 3060 6GB (wersja mobilna) [42]
 - GPU2: Intel UHD Graphics Jasper Lake 32 EU (karta nie jest wykorzystywana do obliczeń przez Tensorflow)
 - Storage: 512GB SSD PCIE Nvme + 512GB SSD PCIE Nvme + 1TB SSD SATA 3.0
 - OS: Windows 11
- Konfiguracja nr 2.1 Laptop MSI KATANA GF66-UE:
 - CPU: Intel Core i7 11800H (8 rdzeni 16 wątków)
 - RAM: 16GB 3200MHz DDR4/64GB 3200MHz DDR4(nastąpiła zmiana ilości ramu w trakcie pisania pracy na 64GB)
 - GPU1: Nvidia GeForce RTX 3060 6GB (wersja laptopowa) (karta nie używana przy uczeniu modelu)
 - GPU2: Intel UHD Graphics Jasper Lake 32 EU (karta nie jest wykorzystywana do obliczeń przez Tensorflow)
 - Storage: 512GB SSD PCIE Nvme + 512GB SSD PCIE Nvme + 1TB SSD SATA 3.0
 - OS: Windows 11
- Konfiguracja nr 3:
 - CPU: Intel Pentium Silver J5005 (4 rdzenie 4 wątki) [43]
 - RAM: 16 GB 2400 MHz DDR4
 - GPU: Intel UHD Graphics 605 EU (karta nie jest wykorzystywana do obliczeń przez Tensorflow)
 - Storage: 256GB SSD SATA 3.0
 - OS: Windows 11

5.3. Użycie GPU

Konfiguracje sprzętowe 1 oraz 2 używały karty graficznej firmy Nvidia (odpowiednio RTX 3060 12GB oraz RTX 3060 6GB w wersji mobilnej), zaś w konfiguracjach sprzętowych 1.1 oraz 2.1 karta graficzna nie była używana, mimo jej fizycznej obecności w sprzęcie testowym. Wyłączenie użycia karty graficznej do uczenia modelu odbywało się z wykorzystaniem kodu przedstawionego na rys. 5.3., wtedy

do uczenia sieci wykorzystywany był wyłącznie procesor (CPU) (odpowiednio dla konfiguracji sprzętowej 1.1 Intel Xeon E5-2667v2 oraz Intel Core i7 11800H dla konfiguracji sprzętowej nr 2.1). Konfiguracja sprzętowa nr 3 nie posiada karty graficznej firmy Nvidia, stąd wykorzystywała wyłącznie CPU (w tym wypadku Intel Pentium Silver J5005).

Rysunek 5.3. Wyłączenie wykorzystania karty graficznej przez Tensorflow

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

5.4. Wyniki dla konfiguracji nr 1

Konfiguracja nr 1 używając modelu z podpunktu 5.1. osiągnęła następujący całkowity czas uczenia modelu: 519.6805 sekund. Średni czas uczenia jednej epoki wyniósł: 3 sekundy, przy czym przybliżony czas uczenia na jeden krok (ang. Step) wyniósł 146 ms/step. Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 5.3.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1 przedstawia tabela 5.3.2. Użycie pamięci VRAM karty graficznej przedstawia rys. 5.3.3.

Rysunek. 5.3.1. Wykres uczenia z wykorzystaniem konfiguracji nr 1

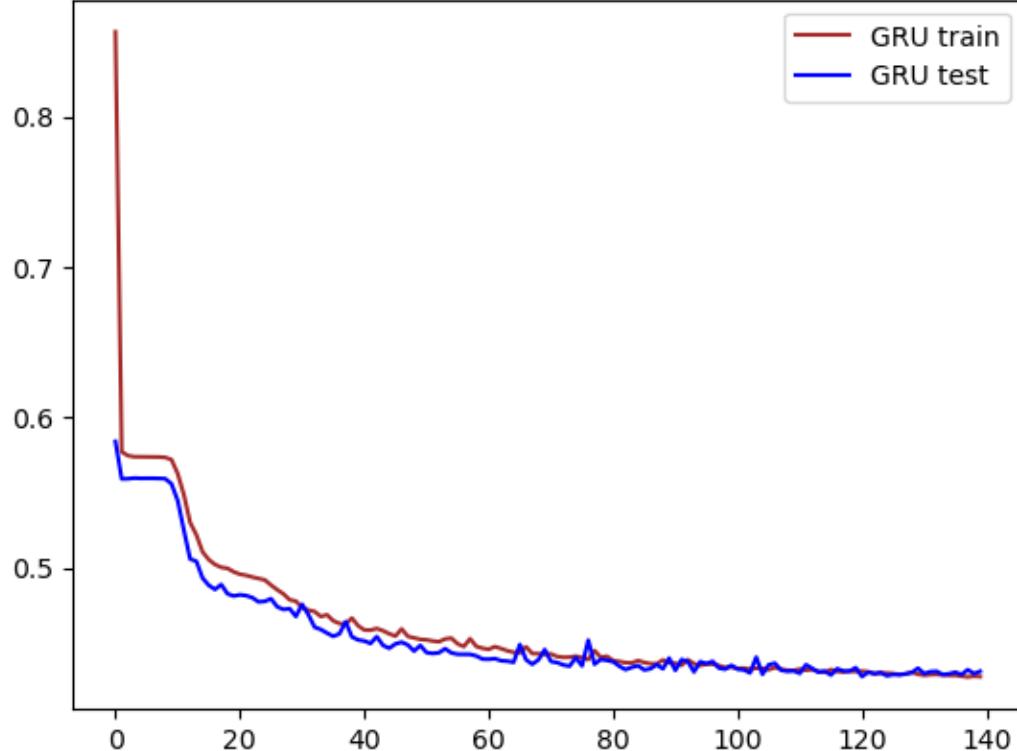


Tabela. 5.3.2. Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etyktę wysokiej jakości, HQ (1)	0.6083
Dokładność klasyfikacji dla wyników posiadających etyktę niskiej jakości, nie HQ (0)	0.8184
Dokładność klasyfikacji dla wszystkich wyników	0.7808
Czas uczenia dla 140 epok (w sekundach)	520

Rysunek. 5.3.4. Użycie pamięci VRAM dla konfiguracji nr 1

```
tf.config.experimental.get_memory_info('GPU:0')

{'current': 1096521472, 'peak': 5198026240}
```

5.5. Wyniki dla konfiguracji nr 1.1

Konfiguracja nr 1.1 wykorzystując modelu z podpunktu 5.1 osiągnęła następujący całkowity czas uczenia modelu: 15890.6453 sekund. Średni czas uczenia jednej epoki wyniósł: 108 sekund, przy czym przybliżony czas uczenia na jeden krok (ang. Step) wyniósł 5s/step. Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 5.5.1 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1.1 przedstawia tabela 5.5.2.

Rysunek. 5.5.1. Wykres uczenia z wykorzystaniem konfiguracji nr 1.1

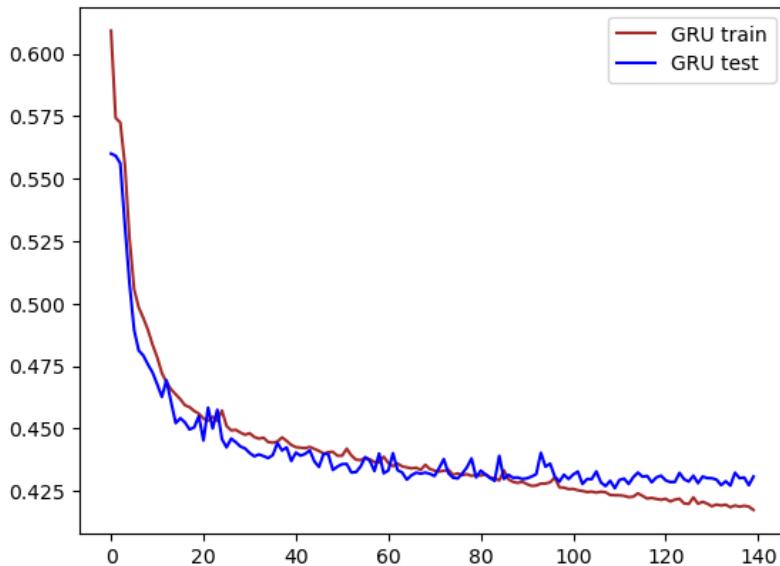


Tabela. 5.5.2. Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1.1

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6193
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8223
Dokładność klasyfikacji dla wszystkich wyników	0.7854
Czas uczenia dla 140 epok (w sekundach)	15891

5.6. Wyniki dla konfiguracji nr 2

Konfiguracja nr 2 używając modelu z podpunktu 5.1. osiągnęła następujący całkowity czas uczenia modelu: 514.4872 sekundy. Średni czas uczenia jednej epoki wyniósł: 4 sekundy, przy czym przybliżony czas uczenia na jeden krok (ang. Step) wyniósł 147 ms/step. Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 5.6.1. Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1 przedstawia tabela 5.6.2. Użycie pamięci VRAM karty graficznej przedstawia rys. 5.6.3.

Rysunek. 5.6.1. Wykres uczenia z wykorzystaniem konfiguracji nr 2

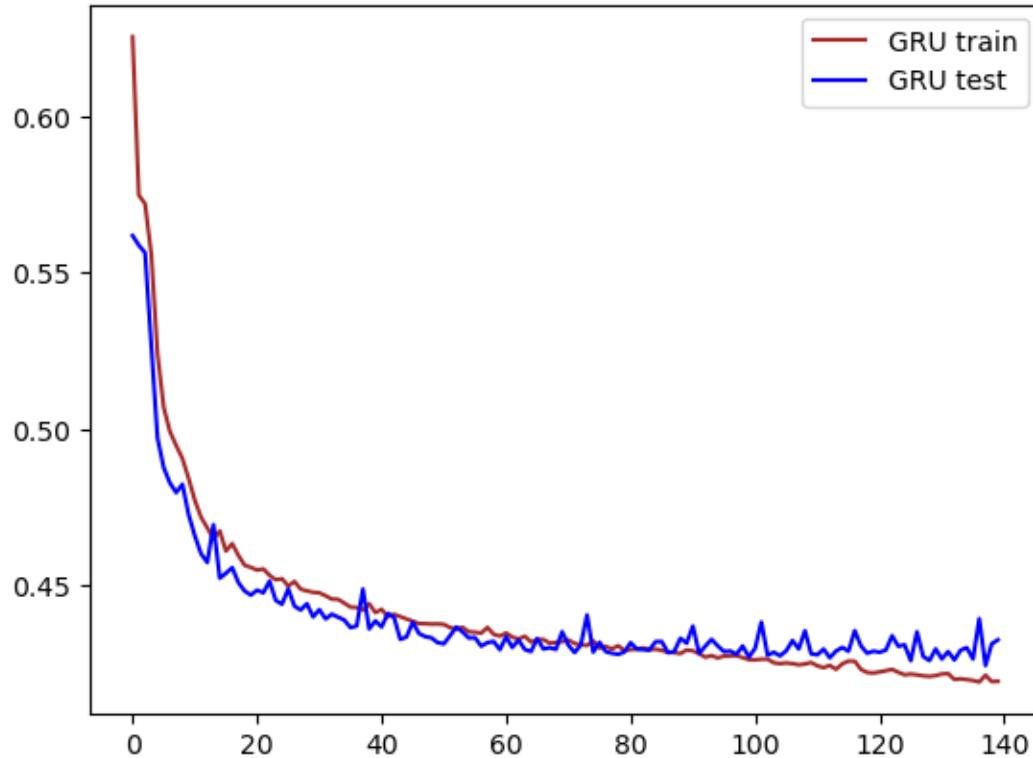


Tabela. 5.6.2. Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 2

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6241
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8201
Dokładność klasyfikacji dla wszystkich wyników	0.7857
Czas uczenia dla 140 epok (w sekundach)	514

Rysunek. 5.6.3. Użycie pamięci VRAM dla konfiguracji nr 2

```
tf.config.experimental.get_memory_info('GPU:0')

{'current': 1096521472, 'peak': 5257622272}
```

5.7. Wyniki dla konfiguracji nr 2.1

Konfiguracja nr 2.1 wykorzystując modelu z podpunktu 5.1 osiągnęła następujący całkowity czas uczenia modelu: 13881.8779 sekund. Średni czas uczenia jednej epoki wyniósł: 95 sekund, przy czym przybliżony czas uczenia na jeden krok (ang. Step) wyniósł 4 ms/step. Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 5.7.1 Widzimy na nim, że sieć uczy się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 2.1 przedstawia tabela 5.7.2.

Rysunek. 5.7.1. Wykres uczenia z wykorzystaniem konfiguracji nr 2.1

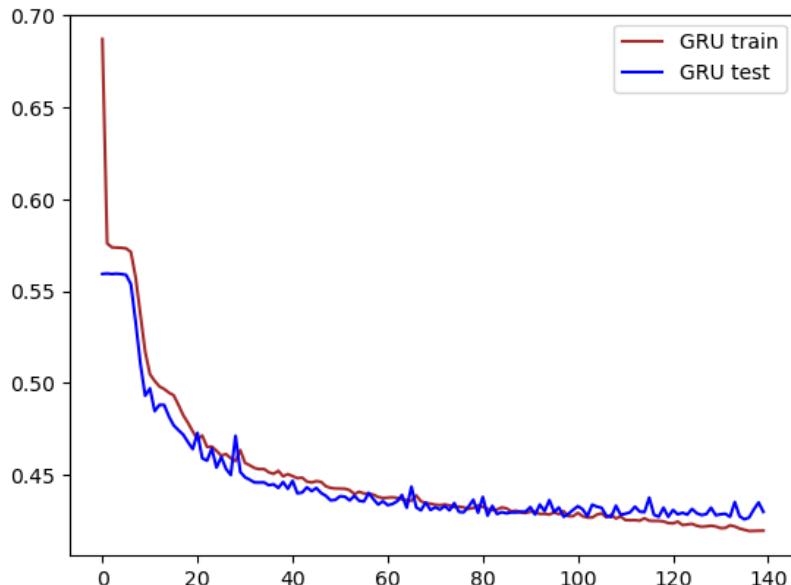


Tabela. 5.7.2. Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 2.1

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5991
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8349
Dokładność klasyfikacji dla wszystkich wyników	0.7844
Czas uczenia dla 140 epok (w sekundach)	13881

5.8. Wyniki dla konfiguracji nr 3

Konfiguracja nr 3 używając modelu z podpunktu 5.1. osiągnęła następujący całkowity czas uczenia modelu: 67348.7573 sekund. Średni czas uczenia jednej epoki wyniósł: 480 sekund, przy czym przybliżony czas uczenia na jeden krok (ang. Step) wyniósł 20 s/step. Wykres uczenia sieci wykorzystując okno czasowe o wielkości 32 przedstawia rysunek 5.8.1. Widzimy na nim, że sieć uczy

się na danych treningowych (wykres czerwony) oraz dobrze uczy się względem danych walidacyjnych (wykres niebieski). Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 1 przedstawia tabela 5.8.2.

Rysunek. 5.8.1. Wykres uczenia z wykorzystaniem konfiguracji nr 3

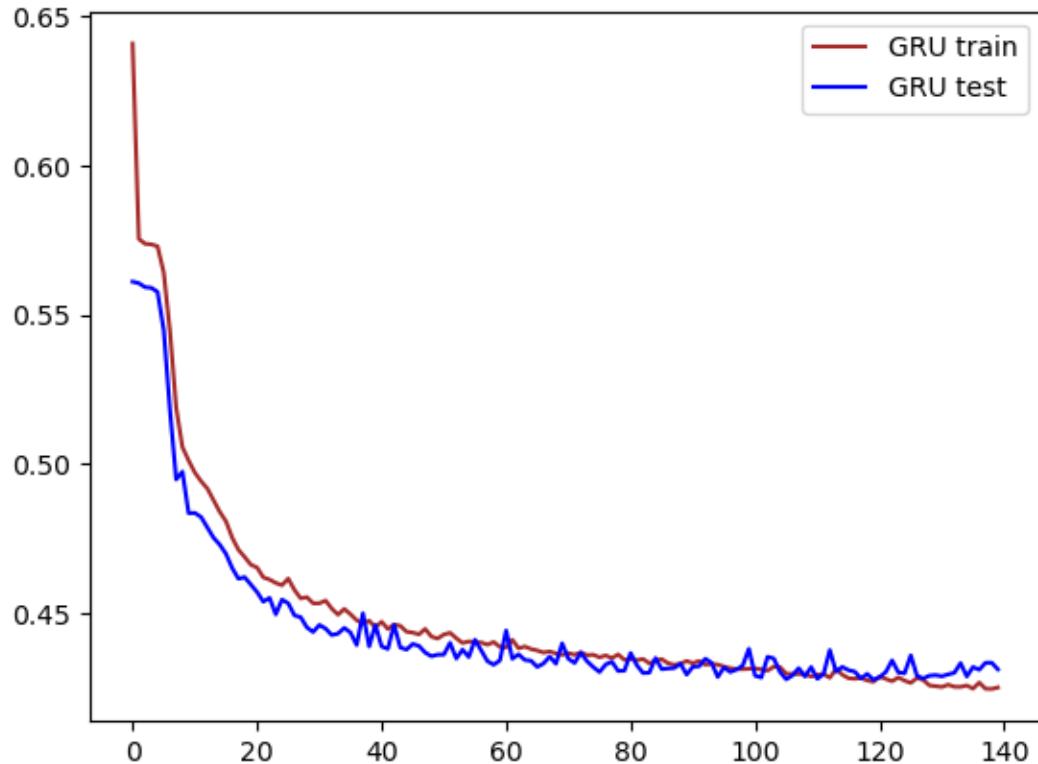


Tabela. 5.8.2. Wyniki klasyfikacji z wykorzystaniem konfiguracji nr 3

Etykieta	Wartość
Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.5987
Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8279
Dokładność klasyfikacji dla wszystkich wyników	0.7818
Czas uczenia dla 140 epok (w sekundach)	67349

5.9. Podsumowanie i wnioski

Najlepsze wyniki pod względem czasu osiągnął model wykorzystujący konfigurację sprzętową nr 2, dzięki użyciu karty graficznej RTX 3060 6GB w wersji mobilnej, niewiele gorszy rezultat (wręcz na granicy błędu pomiarowego) osiągnęła konfiguracja nr 1 z RTX 3060 12GB. Jednakże konfiguracja nr 1 ma tą przewagę, że dzięki większej ilości VRAMU można uczyć dzięki niej ten sam model z podpunktu 5.1 z batchem 32768 lub użyć modelu z podpunktu 5.1 z 2 razy większą ilością neuronów w poszczególnych warstwach (odpowiednio 256, 128, 64, 100 ,1), co by nie było możliwe na konfiguracji nr 2, gdyż zabrakłoby VRAMU karty graficznej. Wyniki wszystkich konfiguracji pod względem dokładności klasyfikacji są zbliżone, w granicach błędu pomiarowego z powodu randomowego przyznawania wag początkowych oraz tego, że parametr shuffle tasuje zbiór danych w sposób pseudolosowy. Na podstawie danych najlepsze rezultaty osiąga konfiguracja nr 2. Użycie karty graficznej firmy Nvidia sprawia, że czasy uczenia modelu nr 1 i 2 są odpowiednio około 30.5 i 27 razy mniejsze niż w wypadku modeli 1.1 i 2.1, które kart graficznych nie używają. W wypadku konfiguracji nr 1 czas uczenia modelu skraca się około z 4 godzin 25 minut do 8 minut 40 sekund. Podobnie jest w przypadku konfiguracji nr 2, gdzie czas uczenia modelu skraca się z 3 godzin 51 minut do 8 minut 36 sekund. W przypadku samego procesora wpływ ilości wątków i IPC jest również widoczny, gdyż model sieci z użyciem konfiguracji nr 3 uczył się 18 godzin 42 minut, zaś konfiguracje 1.1. oraz 2.1 osiągnęły czasy uczenia modelu na poziomie odpowiednio 4 godzin 25 minut i 3 godzin 51 minut. Konfiguracja nr 3 posiada procesor 4 rdzeniowy i 4 wątkowy, zaś konfiguracje nr 1, 1.1., 2 i 2.1. posiadają procesory 8 rdzeniowe i 16 wątkowe. Przyrost wydajności w kontekście czasu uczenia modelu jest w przypadku konfiguracji nr 1.1 4.24 razy większy niż dla konfiguracji nr 3, zaś dla konfiguracji 2.1 jest on 4.85 razy większy względem konfiguracji nr 3. Różnica czasowa uczenia modelu między konfiguracją nr 1.1 oraz 2.1 wynosi około 14% na korzyść konfiguracji nr 2.1, jest ona rezultatem tego, że i7 11700H jest procesorem wydajniejszym, o mniejszej litografii(10nm vs 22nm), o wyższym taktowaniu turbo boost(4.3GHz vs 4.0GHz), o większym IPC niż Xeon e5-2667v2. Pokazuje to, że procesor nie ma, aż tak dużego procentowego wpływu na wydajność uczenia modelu jak wydajna karta graficzna Nvidia, ale w wypadku gdy jej nie posiadamy ilość rdzeni i ich wydajność decyduje o czasie uczenia modelu. Zbiorcze wyniki wszystkich testowanych konfiguracji zawiera tabela 5.9.1.

Tabela. 5.9.1 Tabela zbiorcza dla wszystkich konfiguracji

Konfiguracja	Etykieta	Wartość
Konfiguracja testowa nr 1	Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6083
Konfiguracja testowa nr 1	Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8184
Konfiguracja testowa nr 1	Dokładność klasyfikacji dla wszystkich wyników	0.7808
Konfiguracja testowa nr 1	Czas uczenia jednej epoki	3 sekundy
Konfiguracja testowa nr 1	Całkowity czas uczenia dla 140 epok	519.6805 sekundy
Konfiguracja testowa nr 1	Zużycie VRAMU GPU	{'current': 1096521472, 'peak': 5198026240}
Konfiguracja testowa nr 1.1	Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6193
Konfiguracja testowa nr 1.1	Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8223
Konfiguracja testowa nr 1.1	Dokładność klasyfikacji dla wszystkich wyników	0.7854
Konfiguracja testowa nr 1.1	Czas uczenia jednej epoki	108 sekundy
Konfiguracja testowa nr 1.1	Całkowity czas uczenia dla 140 epok	15890.6453 sekundy
Konfiguracja testowa nr 1.1	Zużycie VRAMU GPU	Brak (GPU nie używane)
Konfiguracja testowa nr 2	Dokładność klasyfikacji dla wyników posiadających etykietę wysokiej jakości, HQ (1)	0.6241
Konfiguracja testowa nr 2	Dokładność klasyfikacji dla wyników posiadających etykietę niskiej jakości, nie HQ (0)	0.8201
Konfiguracja testowa nr 2	Dokładność klasyfikacji dla wszystkich wyników	0.7857
Konfiguracja testowa nr 2	Czas uczenia jednej epoki	4 sekundy
Konfiguracja testowa nr 2	Całkowity czas uczenia dla 140 epok	514.4872 sekundy

Konfiguracja testowa nr 2	Zużycie VRAMU GPU	{'current': 1096521472, 'peak': 5257622272}
Konfiguracja testowa nr 2.1	Dokładność klasyfikacji dla wyników posiadających etykię wysokiej jakości, HQ (1)	0.5991
Konfiguracja testowa nr 2.1	Dokładność klasyfikacji dla wyników posiadających etykię niskiej jakości, nie HQ (0)	0.8349
Konfiguracja testowa nr 2.1	Dokładność klasyfikacji dla wszystkich wyników	0.7844
Konfiguracja testowa nr 2.1	Czas uczenia jednej epoki	95 sekundy
Konfiguracja testowa nr 2.1	Całkowity czas uczenia dla 140 epok	13881.8779 sekund
Konfiguracja testowa nr 2.1	Zużycie VRAMU GPU	Brak (GPU nie używane)
Konfiguracja testowa nr 3	Dokładność klasyfikacji dla wyników posiadających etykię wysokiej jakości, HQ (1)	0.5987
Konfiguracja testowa nr 3	Dokładność klasyfikacji dla wyników posiadających etykię niskiej jakości, nie HQ (0)	0.8279
Konfiguracja testowa nr 3	Dokładność klasyfikacji dla wszystkich wyników	0.7818
Konfiguracja testowa nr 3	Czas uczenia jednej epoki	480 sekund
Konfiguracja testowa nr 3	Całkowity czas uczenia dla 140 epok	67348.7573 sekund
Konfiguracja testowa nr 3	Zużycie VRAMU GPU	Brak (GPU nie używane)

ROZDZIAŁ 6. PODSUMOWANIE

6.1. Zbiór uczący oraz walidacyjny

Zbiór uczący służył celem nauczenia sieci neuronowej rozpoznawania wzorców i zależności występujących w nim dla danych etykiet przy danym oknie. Wielkość zbioru treningowego wynosi 80% rozmiaru zbioru danych. Jego wielkość dla użytego w pracy zbioru danych wynosi, w zależności od długości wybranego okna czasowego: 380244-380268 (przy czym jest to ilość etykiet i okien czasowych o podanym rozmiarze X).

Zbiór walidacyjny zastosowany jest celem weryfikacji nauki modelu. Do sieci przekazywane są dane okna bez etykiet, a następnie są porównywane z etykietami jakie powinny być nadane danemu oknu, z zastrzeżeniem, iż model się nie uczy na zbiorze walidacyjnym, lecz wykorzystuje zgromadzoną wiedzę celem nadania etykiet oknom czasowym (etykietami są: 0 i 1, gdzie zero oznacza dane wątpliwej jakości (non-HQ) a 1 dane wysokiej jakości (HQ)). Wielkość zbioru walidacyjnego wynosi 20% wielkości zbioru danych. Jego wielkość dla użytego w pracy zbioru danych wynosi, w zależności od długości wybranego okna czasowego: 95061-95067 (przy czym jest to ilość etykiet i okien czasowych o rozmiarze X).

6.2. Osiągnięta skuteczność modelu oraz wybrany model

Najwyższe osiągnięte wyniki klasyfikacji to (0.6144 dla danych HQ (1), 0.832 dla danych non-HQ (0) oraz 0.7882 ogólnej), przy czasie uczenia modelu wynoszącym 17 minut 35 sekund przy liczbie epok wynoszącej 140. Model ten posiadał 256 komórek GRU w pierwszej warstwie, 128 komórek GRU w drugiej warstwie, 64 komórki GRU w trzeciej warstwie, 100 neuronów typu dense w warstwie czwartej oraz 1 neuron typu dense w warstwie ostatniej, funkcją aktywacji dla komórek GRU była funkcja tanh, funkcją aktywacji dla warstw typu dense była funkcja sigmoid, zastosowany batch wynosił 16384, okno 32, optymalizator Adam z learning rate 0.0004, zastosowana funkcja błędu to BinaryCrossentropy, parametr shuffle z wartością true.

6.3. Osiągnięta szybkość uczenia modelu oraz wybrany model

Dzięki użyciu modelu sieci, który dobrze wykorzystuje zasoby karty graficznej firmy Nvidia najlepsze wyniki pod względem czasu osiągnął model z podpunktu 5.1 wykorzystujący konfigurację sprzętową nr 2, dzięki użyciu karty graficznej RTX 3060 6GB w wersji mobilnej i mocnego procesora i7 11800H. Osiągnięty czas uczenia modelu wynosił 8 minut 36 sekund. Model ten posiadał 128 komórek GRU w pierwszej warstwie, 64 komórki GRU w drugiej warstwie, 32 komórki GRU w trzeciej warstwie, 50 neuronów typu dense w warstwie czwartej oraz 1 neuron typu dense w warstwie ostatniej, funkcją aktywacji dla komórek GRU była funkcja tanh, funkcją aktywacji dla warstw typu dense była funkcja sigmoid zastosowany batch wynosił 16384, okno 32, optymalizator Adam z learning rate 0.0007, zastosowana funkcja błędu to BinaryCrossentropy, parametr shuffle z wartością true, liczba epok wynosiła 140.

Ponadto po kupnie i użyciu RTX 3060 GB udało się osiągnąć najwyższe wyniki klasyfikacji (0.6144 dla danych HQ (1), 0.8321 dla danych non-HQ (0) oraz 0.7882 ogólnej), przy czasie uczenia modelu wynoszącym 17 minut 35 sekund, przy czym model ten był bardzo wymagający pod względem zasobów, gdyż używał ponad 9.5GB VRAMU. Model ten posiadał 256 komórek GRU w pierwszej warstwie, 128 komórek GRU w drugiej warstwie, 64 komórki GRU w trzeciej warstwie, 100 neuronów typu dense

w warstwie czwartej oraz 1 neuron typu dense w warstwie ostatniej, funkcją aktywacji dla komórek GRU była funkcja tanh, funkcją aktywacji dla warstw typu dense była funkcja sigmoid, zastosowany batch wynosił 16384, okno 32, optymalizator Adam z learning rate 0.0004, zastosowana funkcja błędu to BinaryCrossEntropy, liczba epok wynosiła 140.

6.4. Napotkane trudności

Przez fakt, że w pewnym momencie czas uczenia modelu był długi (około 2h) potrzebowałem go zmniejszyć zwiększając rozmiar batcha, ale napotkałem problem w postaci braku VRAMU na karcie graficznej w laptopie, gdyż obecny tam mobilny RTX 3060 posiada 6GB, a model do uczenia potrzebował więcej stąd zakupiłem do mojej prywatnej stacji roboczej kartę graficzną RTX 3060 z 12GB VRAMU. Dzięki temu możliwe było nauczenie modelu nawet z użyciem batcha 32768.

6.5. Dalszy rozwój

Celem poprawy wyników należałoby utworzyć nowy zbiór danych ewentualnie lepiej przyznać etykiety przy obecnym zbiorze danych, a także sprawdzić inne architektury sieci niż GRU np. LSTM[44], CNN [45], RNN [46]. Należałoby również wykorzystać mocniejszy procesor, zarówno CPU jak i GPU, celem zmniejszenia jeszcze czasu uczenia. W wypadku procesora graficznego o większej ilości VRAM niż 12GB np. Nvidia Quadro RTX 6000 24GB [47] możliwe by było sprawdzenie dużo większych batchów lub modeli o większej liczbie neuronów i warstw. Można by również było zgromadzić dane na przełomie większego okresu czasowego np. dekady przez co możliwe by było poprawienie lub znalezienie kolejnych wzorców, które pozwalałyby na określenie poprawności wyników pomiarów z większą dokładnością.

BIBLIOGRAFIA

- [1] Prędki, A. (Ed.). (2018). Wybrane zastosowania metod analitycznych w naukach ekonomicznych. UEK.
- [2] Quinlan, J. R. (1987). "Simplifying decision trees". *International Journal of Man-Machine Studies*. **27** (3): 221–234.
- [3] Ho, T. K. (1995, August). Random decision forests. In Proceedings of 3rd international conference on document analysis and recognition (Vol. 1, pp. 278-282). IEEE.
- [4] McCallum, A. (2019). Graphical Models, Lecture2: Bayesian Network Representation. PDF). Retrieved, 22.
- [5] McLachlan, G. J. (2005). Discriminant analysis and statistical pattern recognition. John Wiley & Sons. ISBN 978-0-471-69115-0.
- [6] Jukiewicz, M. (2014). Wykorzystanie maszyny wektorów nośnych oraz liniowej analizy dyskryminacyjnej jako klasyfikatorów cech w interfejsach mózg-komputer. Poznan University of Technology Academic Journals. Electrical Engineering, (79), 25-30.
- [7] Xue, H., Huynh, D. Q., & Reynolds, M. (2017, November). Bi-prediction: Pedestrian trajectory prediction based on bidirectional LSTM classification. In 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA) (pp. 1-8). IEEE.
- [8] Podlaski, K., Durka, M., Gwizdała, T., Miniak-Górecka, A., Fortuniak, K., & Pawlak, W. (2021, June). LSTM Processing of Experimental Time Series with Varied Quality. In International Conference on Computational Science (pp. 581-593). Springer, Cham.
- [9] Marsland, S. (2011). Machine learning: an algorithmic perspective. Chapman and Hall/CRC. ISBN 978-1-4665-8328-3.
- [10] Ławrynowicz, A. (2020). Rekurencyjne sieci neuronowe, LSTM i GRU Wydział Informatyki Politechniki Poznańskiej https://www.cs.put.poznan.pl/alaawrynowicz/PJN_5-2020.pdf online, data uzyskania dostępu 17.12.2022
- [11] Komórka LTSM, Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
- [12] Komórka GRU opis na stronie <https://arxiv.org/abs/1406.1078>, Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
- [13] Géron, A. (2020). Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Helion ISBN: 978-83-283-6003-7
- [14] Nazari, F., & Yan, W. (2021). Convolutional versus Dense Neural Networks: Comparing the Two Neural Networks Performance in Predicting Building Operational Energy Use Based on the Building Shape. arXiv preprint arXiv:2108.12929.
- [15] Dokumentacje rdzeni tensor <https://www.nvidia.com/en-us/data-center/tensor-cores/> online, data uzyskania dostępu 24.01.2023
- [16] Dokumentacja użycia biblioteki Tensorflow na kartach Nvidii <https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/index.html> online, data uzyskania dostępu 24.01.2023
- [17] Dokumentacja języka Python <https://www.python.org/doc/> online, data uzyskania dostępu 17.12.2022

- [18] Tensorflow Google <https://www.tensorflow.org/> online, data uzyskania dostępu 17.12.2022
- [19] Dokumentacja projektu Keras <https://keras.io/> online, data uzyskania dostępu 17.12.2022
- [20] Vu, Linda (June 14, 2021). ["Project Jupyter: A Computer Code that Transformed Science"](#). Berkeley Lab Computing Sciences.
- [21] Cuda Toolkit, <https://developer.nvidia.com/cuda-toolkit> online, data uzyskania dostępu 17.12.2022
- [22] CuDnn, <https://developer.nvidia.com/cudnn> online, data uzyskania dostępu 17.12.2022
- [23] Fortuniak, K. (2016). Wymiana metanu i dwutlenku węgla pomiędzy powierzchnią Ziemi a atmosferą na terenach bagiennych–przegląd opublikowanych wyników pomiarów bezpośrednich i oszacowań. Wybrane problemy pomiarów wymiany gazowej pomiędzy powierzchnią ziemi a atmosferą na terenach bagiennych. Doświadczenia trzyletnich pomiarów w Biebrzańskim Parku Narodowym;
- [24] Dokumentacja optymalizatora Adam <https://keras.io/api/optimizers/adam/> online, data uzyskania dostępu 2.01.2023
- [25] Dokumentacja optymalizatora SGD <https://keras.io/api/optimizers/sgd/> online, data uzyskania dostępu 2.01.2023
- [26] Dokumentacja optymalizatora Adagrad <https://keras.io/api/optimizers/adagrad/> online, data uzyskania dostępu 2.01.2023
- [27] Komórki GRU https://keras.io/api/layers/recurrent_layers/gru/ online, data uzyskania dostępu 10.12.2022
- [28] Dokumentacja metryk w Kerasie <https://keras.io/api/metrics/> online, data uzyskania dostępu 29.01.2023
- [29] Dokumentacja metryki accuracy https://keras.io/api/metrics/accuracy_metrics/#accuracy-class online, data uzyskania dostępu 29.01.2023
- [30] Dokumentacja metryki AUC https://keras.io/api/metrics/classification_metrics/#auc-class online, data uzyskania dostępu 29.01.2023
- [31] Dokumentacja keras.models.fit() https://keras.io/api/models/model_training_apis/#fit-method online, data uzyskania dostępu 29.01.2023
- [32] Dokumentacja funkcji błędu https://www.tensorflow.org/api_docs/python/tf/keras/losses/ online, data uzyskania dostępu 29.01.2023
- [33] Dokumentacja funkcji błędu BinaryCrossEntropy https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy online, data uzyskania dostępu 29.01.2023
- [34] Dokumentacja funkcji błędu MeanSquaredError https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError online, data uzyskania dostępu 29.01.2023
- [35] Dokumentacja API trenującego model w tym batcha i liczby epok https://keras.io/api/models/model_training_apis/ online, data uzyskania dostępu 29.01.2023

- [36] Rdzenie Cuda <https://developer.nvidia.com/cuda-zone> online, data uzyskania dostępu 8.12.2022
- [37] Sterowniki CUDA <https://developer.nvidia.com/cuda-downloads> online, data uzyskania dostępu 8.12.2022
- [38] Specyfikacja CPU Intel Xeon E5-2667v2
<https://www.intel.pl/content/www/pl/pl/products/sku/75273/intel-xeon-processor-e52667-v2-25m-cache-3-30-ghz/specifications.html> online, data uzyskania dostępu 29.01.2023
- [39] Dokumentacja GPU Nvidia RTX 3060 12GB <https://www.nvidia.com/pl-pl/geforce/graphics-cards/30-series/rtx-3060-3060ti/> online, data uzyskania dostępu 29.01.2023
- [40] Dokumentacja laptopa MSI Katana GF66-UE <https://www.msi.com/Laptop/Katana-GF66-11UX/Specification>, data uzyskania dostępu 29.01.2023
- [41] Specyfikacja CPU Intel Core i7 11800H
<https://www.intel.pl/content/www/pl/pl/products/sku/213803/intel-core-i711800h-processor-24m-cache-up-to-4-60-ghz/specifications.html> online, data uzyskania dostępu 29.01.2023
- [42] Dokumentacja GPU Nvidia RTX 3060 6GB (wersja mobilna) obecnego w laptopie MSI Katana GF66-UE <https://laptopmedia.com/video-card/nvidia-geforce-rtx-3060-laptop-85w/> online, data uzyskania dostępu 29.01.2023
- [43] Specyfikacja CPU Pentium Silver J5005
<https://www.intel.pl/content/www/pl/pl/products/sku/128984/intel-pentium-silver-j5005-processor-4m-cache-up-to-2-80-ghz/specifications.html> online, data uzyskania dostępu 29.01.2023
- [44] LSTM vs GRU <https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/> online, data uzyskania dostępu 11.12.2022
- [45] Konwulcyjne sieci neuronowe (CNN)
<https://www.tensorflow.org/tutorials/images/cnn?hl=pl> online, data uzyskania dostępu 29.01.2023
- [46] Rekurencyjne sieci neuronowe (RNN)
<https://www.tensorflow.org/guide/keras/rnn?hl=pl> online, data uzyskania dostępu 29.01.2023
- [47] Dokumentacja GPU Nvidia Quadro RTX 6000 <https://www.nvidia.com/content/dam/en-za/Solutions/design-visualization/quadro-product-literature/quadro-rtx-6000-us-nvidia-704093-r4-web.pdf> online, data uzyskania dostępu 29.01.2023