Politechnika
Śląska

PRACA MAGISTERSKA

Techniki głębokiego uczenia metryki

Igor BUDZYŃSKI
Nr albumu: 282602

Kierunek: Informatyka
Specjalność: Interaktywna Grafika Komputerowa

PROWADZĄCY PRACĘ
Dr inż. Damian Pęszor
KATEDRA Katedra Grafiki, Wizji Komputerowej i Systemów Cyfrowych
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2023

Tytuł pracy

Techniki głębokiego uczenia metryki

Streszczenie

Niniejsza praca ma na celu analizę wpływu różnych funkcji straty oraz liczby epok treningu na skuteczność modelu sieci neuronowej wykorzystującej głębokie uczenie metryki w zadaniu klasyfikacji obrazów. Badania przeprowadzono na zbiorze danych MNIST, a ocena dokładności modelu została wykonana przy wykorzystaniu algorytmu k najbliższych sąsiadów.

W badaniach wykorzystano kilka różnych funkcji straty, w tym strata analizy składników sąsiedztwa, strata marginesu, strata pełnomocnika kotwicy, strata kontrastowa oraz stratę trójkową. Dla każdej funkcji straty przeprowadzono trening modelu przy liczbie epok: 1, 5, 10, 20 i 50. Celem było zrozumienie, jak funkcje straty oraz liczba epok treningu wpływają na jakość wyników klasyfikacji.

Analiza wyników pokazuje, że wybór odpowiedniej funkcji straty ma znaczący wpływ na osiąganą dokładność modelu. Okazało się, że nie istnieje jedna uniwersalnie najlepsza funkcja straty dla wszystkich przypadków, ale ich skuteczność zależy od specyfiki problemu. Co więcej, wpływ liczby epok treningu na rezultaty jest zróżnicowany - dłuższy trening nie zawsze przekłada się na lepsze wyniki. Zaobserwowano przypadki przeuczenia modelu, które prowadziło do obniżenia dokładności przy zbyt długim treningu.

Wnioski z badań potwierdzają skuteczność głębokiego uczenia metryki w zadaniach klasyfikacji obrazów. Wybór odpowiednich funkcji strat oraz dostosowanie liczby epok treningu są kluczowe dla uzyskania optymalnej dokładności. Wyniki te mają ważne implikacje dla projektowania i optymalizacji modeli sieci neuronowych w zastosowaniach klasyfikacyjnych.

Słowa kluczowe

głębokie uczenie metryki, funkcje straty, liczba epok, klasyfikacja obrazów, sieci neuronowe

Thesis title

Deep metric learning techniques

Abstract

This paper aims to analyze the effect of different loss functions and the number of training epochs on the performance of a neural network model using deep learning metrics in an image classification task. The study was conducted on the MNIST dataset, and the evaluation of the model's accuracy was performed using the k nearest neighbor algorithm.

Several different loss functions were used in the study, including neighborhood component analysis loss, margin loss, proxy anchor loss, contrastive loss and triple loss. For each loss function, the model was trained at the number of epochs: 1, 5, 10, 20 and 50. The goal was to understand how the loss functions and the number of training epochs affect the quality of the classification results.

Analysis of the results shows that the choice of an appropriate loss function has a significant impact on the achieved accuracy of the model. It turned out that there is no single universally best loss function for all cases, but their effectiveness depends on the specifics of the problem. Moreover, the effect of the number of training epochs on the results varies - longer training does not always translate into better results. Cases of model overlearning were observed, leading to lower accuracy with too much training.

The conclusions of the study confirm the effectiveness of deep learning metrics in image classification tasks. Selecting appropriate loss functions and adjusting the number of training epochs are key to achieving optimal accuracy. These results have important implications for the design and optimization of neural network models in classification applications.

Key words

Deep metric learning, loss functions, number of epochs, image classification, neural networks

Spis treści

Rozdział 1

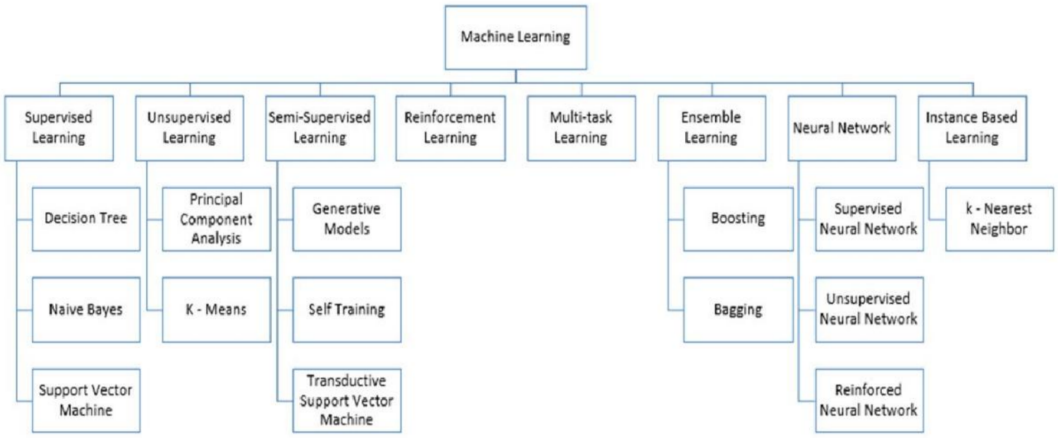
Wstęp

1.1 Uczenie maszynowe

Uczenie maszynowe, stanowiące poddziedzinę szeroko pojętej dziedziny sztucznej inteligencji, a w ramach niego głębokie uczenie, reprezentuje obszar badań z zakresu informatyki, którego istota tkwi w analizie modeli algorytmicznych oraz statystycznych. Głównym celem głębokiego uczenia jest tworzenie takich modeli, które umożliwią automatycznym systemom komputerowym realizację określonych zadań bez konieczności ręcznego formułowania jasno określonych reguł czy instrukcji, znanych jako jawne programowanie. Przykładem systemu stosującego technologię uczenia maszynowego jest wyszukiwarka internetowa, która wyuczona algorytmem jest w stanie trafnie sortować strony na podstawie wpisanych przez użytkownika słów kluczowych [43]. Wielką zaletą uczenia maszynowego jest to, że raz nauczony na zestawie danych model jest w stanie wykonywać swoje zadanie automatycznie, w takim samym zakresie jak każdy inny algorytm, którego utworzenie jawnym programowaniem mogło by okazać się niemożliwe ze względu na złożoność problemu. Istnieje wiele rodzajów uczenia maszynowego, a ich podział został przedstawiony na Rysunku 1.1 pochodzącym z pracy „Machine Learning Algorithms - A Review” Batta Mahesha.

1.2 Głębokie uczenie

Głębokie uczenie, stanowiące jedną z iteracji uczenia maszynowego, nabiera coraz większego znaczenia w sferze życia codziennego. W książce "Deep Learning"autorstwa Iana Goodfellowa, podkreśla się, że jednym z odmianowych podtypów uczenia maszynowego jest właśnie uczenie głębokie. Dzięki akumulacji danych oraz wiedzy zdobytej w poprzednich cyklach uczenia, modele sieci głębokiego uczenia rozwijają się, kreując coraz bardziej zaawansowane reprezentacje i struktury modeli. [19]. Autor pisze, że opisywana technologia jest jedyną możliwością, by sztuczna inteligencja mogła działać w skomplikowanych



Rysunek 1.1: Rodzaje uczenia maszynowego [43].

warunkach środowiska rzeczywistego. Głębokie uczenie jest szczególnym typem uczenia maszynowego, które uzyskuje bardzo dobre wyniki przy pozostaniu elastycznym dzięki reprezentowaniu świata jako zagnieżdżoną hierarchię konceptów, którego każdy koncept jest powiązany i definiowany w relacji do konceptu prostszego, a bardziej abstrakcyjne reprezentacje obliczane są w kategoriach mniej abstrakcyjnych [19].

1.3 Wykorzystanie głębokiego uczenia w XXI wieku

Współcześnie, wykorzystanie głębokiego uczenia stanowi fundamentalny filar postępu technologicznego. W swojej książce John D. Kelleher „Deep Learning" porusza temat występowania i popularności głębokiego uczenia [34]. Jak pisze autor, ta technologia jest używana przez znane na rynku firmy, takie jak Google [1], Microsoft [14] i Baidu [28], przy wyszukiwarce obrazów i automatycznych tłumaczach. Innym przykładem przytaczanym przez Kelleher'a jest serwis Facebook [38], który za pomocą tej technologii analizuje tekst z rozmów prowadzonych online. Styczność z urządzeniami, które wykorzystują systemy głębokiego uczenia, stała się codziennością dla użytkowników systemów takich jak Android czy IOS. Przykładami funkcji dostępnych w powyższych systemach, opartych na uczeniu głębokim są:

- rozpoznawanie twarzy
- rozpoznawanie głosu
- rozpoznawanie tekstu ze zdjęcia
- skaner linii papilarnych

John D. Kelleher zastosowań głębokiego uczenia szuka również w medycynie. Powyższe systemy znalazły zastosowanie m.in. w diagnostyce obrazowej, poprzez ich wykorzystanie

w przetwarzaniu zdjęć rentgenowskich oraz obrazów tomografii komputerowej, a także we wstępnej diagnostyce różnicowej pacjentów.

Kolejnym przykładem wykorzystania głębokiego uczenia, przytaczanym przez autora "Deep Learning", jest zastosowanie systemów lokalizacji i mapowania w samojezdnych pojazdach, czy też systemów monitorujących stan kierownicy w tradycyjnych samochodach [34].

1.4 Uczenie metryki

Metryka odnosi się do funkcji odległości definiującej przestrzeń metryczną, w której odległość między reprezentacjami obiektów jest interpretowana jako miara ich podobieństwa lub różnicy. Jest to matematyczna konstrukcja, która umożliwia numeryczne wyrażenie relacji między obiektami, co pozwala na opracowanie algorytmów uczenia, które dostosowują się do złożonych zależności pomiędzy danymi [33]. Uczenie systemu metryki jest podejściem, w którym przy wykorzystaniu metryki odległości dokonuje się modyfikacji lub dopasowania tak, aby lepiej odzwierciedlała ona podobieństwa i różnice między obiektami. Bardzo ciekawym sposobem wykorzystania tej technologii mogłoby być identyfikowanie ludzi na podstawie ich charakterystycznych cech zewnętrznych takich jak chód, gestykulacja rękami, czy sposób poruszania ustami.

1.5 Cel pracy

Celem pracy jest stworzenie 12 wyuczonych modeli, wykorzystujących metody głębokiego uczenia metryki stosując różne funkcje strat, a także różne opcje głębokiego uczenia oraz porównanie otrzymanych wyników.

1.6 Zakres pracy

W zakresie pracy znajduje się:

- zbadanie wpływu poniższych funkcji straty na dokładność modelu rozwiązującego problem klasyfikacji przy zastosowaniu głębokiego uczenia metryki:
  - Strata kątowa (ang. *Angular Loss*)
  - Strata kołowa (ang. *Circle Loss*)
  - Strata kontrastowa (ang. *Contrastive Loss*)
  - Strata miękki maks z dużym marginesem (ang. *Large-Margin Softmax Loss*)
  - Podniesiona strata strukturalna (ang. *Lifted Structured Loss*)

  - Strata marginesu (ang. *Margin Loss*)
  - Strata analizy składników sąsiedztwa (ang. *Neighbourhood Components Analysis Loss*)
  - Znormalizowana strata miękki maks (ang. *Normalized Softmax Loss*)
  - Strata N par (ang. *N Pairs Loss*)
  - Strata pełnomocnika kotwicy (ang. *Proxy Anchor Loss*)
  - Strata marginesu trójkowego (ang. *Triplet Margin Loss*)

- zbadanie wpływu liczby epok na dokładność modelu rozwiązującego problem klasyfikacji przy zastosowaniu głębokiego uczenia metryki,
- przeprowadzenie analizy wyników otrzymanych z zaimplementowanych rozwiązań.

Poza zakresem pracy znajduje się:

- zbadanie wpływu innych hiperparametrów związanych z treningiem sieci neuronowej takich jak współczynnik uczenia czy dobór optymalizatora,
- zbadanie wpływu parametrów związanych z poszczególnymi funkcjami straty,
- zestawienie wyników badań z innymi istniejącymi rozwiązaniami.

1.7 Charakterystyka rozdziałów

Praca ma 5 rozdziałów, w których zawarte zostały poniższe informacje:

1. **Wstęp -** w tym rozdziale znajduje się wprowadzenie do tematyki i dziedziny projektu. Rozdział również opisuje przydatność badanej technologii. Zawarta jest tu definicja celu i zakresu pracy oraz charakterystyka dalszych rozdziałów.
2. **Analiza tematu -** rozdział szczegółowo wprowadza do dziedziny problemu. Tutaj zawarta jest analiza dostępnych technologii, a także rozwiązań dostępnych w literaturze naukowej i na rynku. Zawarty jest tutaj ostateczny zarys projektowanych modeli głębokiego uczenia metryki.
3. **Przedmiot pracy -** w tym rozdziale dokładnie opisany jest przedmiot badania, który stanowi główny temat pracy. Przedstawiono w nim szczegółowe informacje na temat obszaru badawczego, definiuje kluczowe pojęcia związane z tematem i tłumaczy dlaczego wybrany przedmiot jest istotny z perspektywy naukowej lub praktycznej. Opisuje główne teorie, hipotezy oraz model badawczy, który zostanie użyty w dalszej części pracy.

4. **Badania -** w rozdziale przedstawione zostały metody badawcze, które zostały zastosowane w celu zebrania danych potrzebnych do analizy. Opisane są procedury, które były przestrzegane podczas prowadzenia badań. Przedstawiono tu również wyniki badań, ich analizę, interpretacje, a także porównanie z wynikami innych badań lub teoriami prezentowanymi w literaturze.

5. **Podsumowanie -** w ostatnim rozdziale znajduje się podsumowanie całości pracy, odnoszące się do postawionych na początku celów i hipotez. Przedstawiane są wnioski wynikające z przeprowadzonych badań oraz analizy. Są tu zasugerowane możliwe kierunki dalszych badań na podstawie wyników uzyskanych w pracy.

Rozdział 2

Analiza tematu

2.1 Historia

2.1.1 Sieci neuronowe

Początki rozwoju sieci neuronowych sięgają aż do 1943 roku, kiedy Warren McCulloch i Walter Pitts stworzyli pierwszy model sieci neuronowej [42]. Był on bardzo prosty i opierał się na prostych elementach binarnych. Swoje prace opisali w artykule „Logiczny rachunek idei zawartych w aktywności nerwowej (ang. *A logical calculus of the ideas immanent in nervous activity*)" [45]. Wynikiem modelu były proste logiczne funkcje, których charakterystyka była podobna do tej z układu nerwowego.

Przez następne lata prowadzono różne badania nad sieciami neuronowymi. Tempo rozwoju było zmienne i można je podzielić na kilka etapów [42]. Później nastąpił trwający ponad 10 lat okres zwątpienia w technologię i możliwości sieci neuronowych. Opracowane w tamtym okresie algorytmy i rozwiązania jak na przykład wymyślona przez Paula Werbosa w 1974 metoda uczenia z propagacją wsteczną [68] musiały nieraz czekać wiele lat, by zostać docenionymi. Jednak w latach 80. wiele wydarzeń przywróciło zainteresowanie sieciami neuronowymi. Jednym z nich był wydany przez Johna J. Hopfielda artykuł „Sieci neuronowe i systemy fizyczne z pojawiającymi się zbiorowymi zdolnościami obliczeniowymi (ang. *Neural networks and physical systems with emergent collective computational abilities*" [27], w którym pisał o sztucznej sieci neuronowej, która mogła posłużyć jako pamięć adresowana treścią. Swoją pracą przekonał dużą ilość specjalistów z dziedziny matematyki do przestudiowania zagadnienia sieci neuronowych [27].

W XXI wieku technologia sieci neuronowych dalej się rozwija i jest ona wykorzystywana w wielu badaniach naukowych jak choćby sztuczny model ludzkiego mózgu stworzona przez IBM, który jest w stanie symulować pracę 256 milionów synaps w czasie rzeczywistym [42].

2.1.2 Uczenie maszynowe i głębokie uczenie

Koncepcja uczenia maszynowego kształtuje się od początku lat 50. XX w. a w jej rozwoju można wskazać kilka kluczowych postaci oraz wydarzeń. W 1950 r. w artykule "Uczenie maszynowe i inteligencja"(ang. *"Computing Machinery and Intelligence"* [62] A. M. Turing sformułował koncepcję testu (Test Turinga) sprawdzającego umiejętności maszyny do naśladowania człowieka, poprzez nawiązanie z nim komunikacji podczas rozmowy kwalifikacyjnej. Termin "sztuczna inteligencja"został po raz pierwszy użyty dopiero w roku 1956 przez Johna McCarthey'ego w trakcie konferencji w Dartmouth. W 1958 r.Frank Rosenblatt w artykule "Perceptron: automat dostrzegający i rozpoznający (ang. *The perceptron, a perceiving and recognizing automaton Project Para*)"przedstawił percepton, model sztucznego neuronu, który uznawany jest za prekursora sztucznych sieci neuronowych [53]. W roku 1959 Arthur Samuel stworzył pierwszy program do gry w warcaby, który mógł poprawiać swoje wyniki na podstawie przebytych rozgrywek [54]. Od tego momentu uczenie maszynowe dynamicznie się rozwijało. W latach 90. wzrost popularności gier komputerowych spowodował jeszcze szybszy rozwój technologii uczenia maszynowego. Obecnie uczenie maszynowe jest wykorzystywane nie tylko w grach, ale i wielu innych sektorach gospodarki oraz badaniach [11].

Rozwój głębokiego uczenia jest ściśle powiązany z rozwojem uczenia maszynowego, gdyż jest jednym z jego podtypów. Głębokie uczenie poprawiło jakość rozwiązań służących do wykonywania zadań, takich jak rozpoznawanie mowy, tłumaczenia i wykrywanie obiektów. [64] Okres lat 10. i 20. XXI wieku cechuje się dużym zainteresowaniem tematem sztucznej inteligencji, co przekłada się również na dużą ilość prowadzonych badań i publikacji na temat uczenia maszynowego i głębokiego. Jednymi z przykładów są zaprojektowana w 2015 roku rekurencyjna sieć neuronowa, służąca do odszumiania sygnałów głosowych [50], której autorami byli Keiichi Osako, Rita Singh i Bhiksha Raj oraz zaprojektowany w tym samym roku przez Amana Gupta, Haohana Wanga i Madhavi Ganapathiraju model służący do odkrywania wzorców grupowania genów [20]. Innym przykładem jest sztuczna inteligencja generująca obrazy stworzona w 2015 przez Leon'a A. Gatys, Alexandra S. Ecker'a i Matthias'a Bethge'a [16].

2.2 Koncepcja uczenia maszynowego

Przy założeniu, że głównym wyznacznikiem inteligencji człowieka jest jego zdolność do uczenia się, uczenie maszynowe jest sposobem, by sprawić, że komputer również będzie inteligentny. Uczenie maszynowe to pojęcie stworzone w celu opisania naukowego badania algorytmów i modeli matematycznych, które komputery wykorzystują do wykonywania określonych zadań bez konieczności tradycyjnego programowania[12]. Przedmiotem badań w zakresie uczenia maszynowego jest odtworzenie procesów uczenia w oparciu o

ludzkie wzorców nauki i ich symulacja przez komputer. W praktyce polega to na analizie wzorców i wytycznych, dzięki którym maszyny mogą samodzielnie się uczyć i doskonalić, opierając się na doświadczeniach zdobytych z biegiem czasu. Głównym celem uczenia maszynowego jest implementacja różnorodnych algorytmów komputerowych, które pozwalają maszynom uzyskać dostęp do dostarczonych danych, przetwarzać je w procesie nauki i stopniowo zwiększać swoje umiejętności, wykonując przydzielone zadania. Dzięki temu maszyny są w stanie podejmować decyzje oraz dokonywać przewidywań w oparciu o dostępne informacje.W porównaniu z człowiekiem komputer uczy się szybciej, a efekty jego nauki szybciej rozprzestrzeniają się wśród użytkowników. [65].

2.3 Sztuczne sieci neuronowe

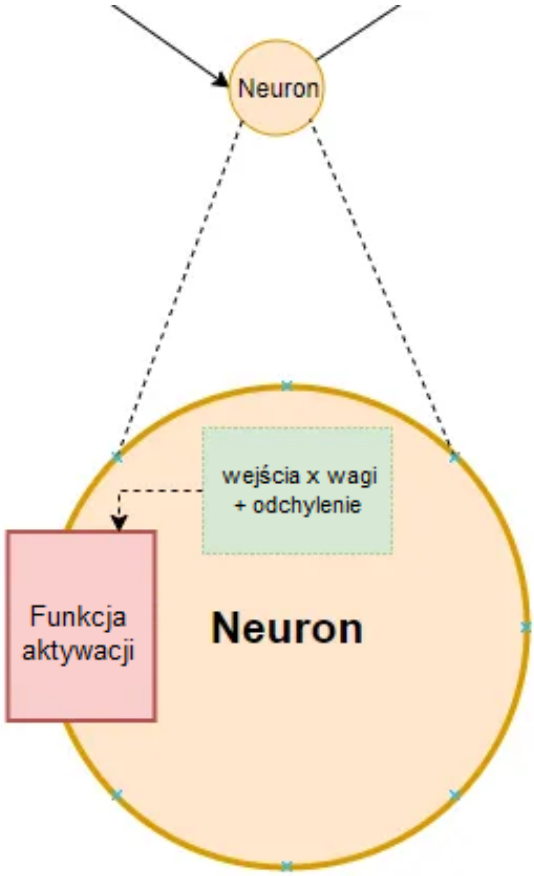
Sztuczne sieci neuronowe w sposobie działania naśladują swoje biologiczne odpowiedniki, które są odpowiedzialne za funkcjonowanie mózgu. Jeśli chodzi o możliwości obliczeniowe, ludzki mózg nie dorówna komputerom, ale jest wiele innych aspektów, które pomimo wielu lat badań nie przybliżyły komputerów do możliwości mózgu. Ludzie potrafią rozpoznawać osoby nawet w niesprzyjających warunkach takich jak złe oświetlenie lub zasłonięte części twarzy. Nie mają problemu z rozpoznawaniem głosu innego człowieka, którego dopiero co poznali w głośnym pomieszczeniu. Poza tym najważniejszą rzeczą jest umiejętność mózgu do nauki. Ludzie zdobywają nowe umiejętności, a przez całe życie posiadają tylko i wyłącznie jeden mózg, który w czasie życia stale się uczy[37].

Sztuczne sieci neuronowe są sposobem, w jaki komputer próbuje dorównać biologicznemu pierwowzorowi. Sieci te są systemami obliczeniowymi o budowie równoległej, które składają się z dużej ilości mniejszych procesów połączonych ze sobą. Sieci neuronowe częściowo wykorzystują struktury analogiczne do tych obecnych w układzie nerwowym człowieka. Każdy osobny węzeł przedstawia się jako sztuczny neuron zilustrowany na Rysunku 2.1. Sztuczny neuron jest komputerowym modelem inspirowanym działaniem naturalnych neuronów.

Funkcją neuronów, tak naturalnych, jak i sztucznych, jest przekazywanie informacji, dlatego sieci neuronowe są wykorzystywane głównie w dziedzinach bazujących na przetwarzaniu i uzyskiwaniu danych. Sztuczne sieci neuronowe mogą służyć do naśladowania prawdziwych odpowiedników tak, by umożliwić badanie zachowań zwierząt, jak również do rozpoznawania wzorców, ich przewidywania czy porównywania.

W sieci neuronowej analogicznie do modelu biologicznego, pomiędzy poszczególnymi neuronami wstępują połączenia. Te sztuczne „synapsy", czyli wejścia do węzła, są mnożone przez wagi. Wagi danego wejścia są przeliczane za pomocą funkcji matematycznych i determinują aktywację węzła. Kolejna funkcja oblicza wagę wyjścia sztucznego neuronu.

Kluczowym elementem struktury sztucznych sieci neuronowych są warstwy. Warstwa to zbiór neuronów działających wspólnie, a każda warstwa może mieć swoje unikalne

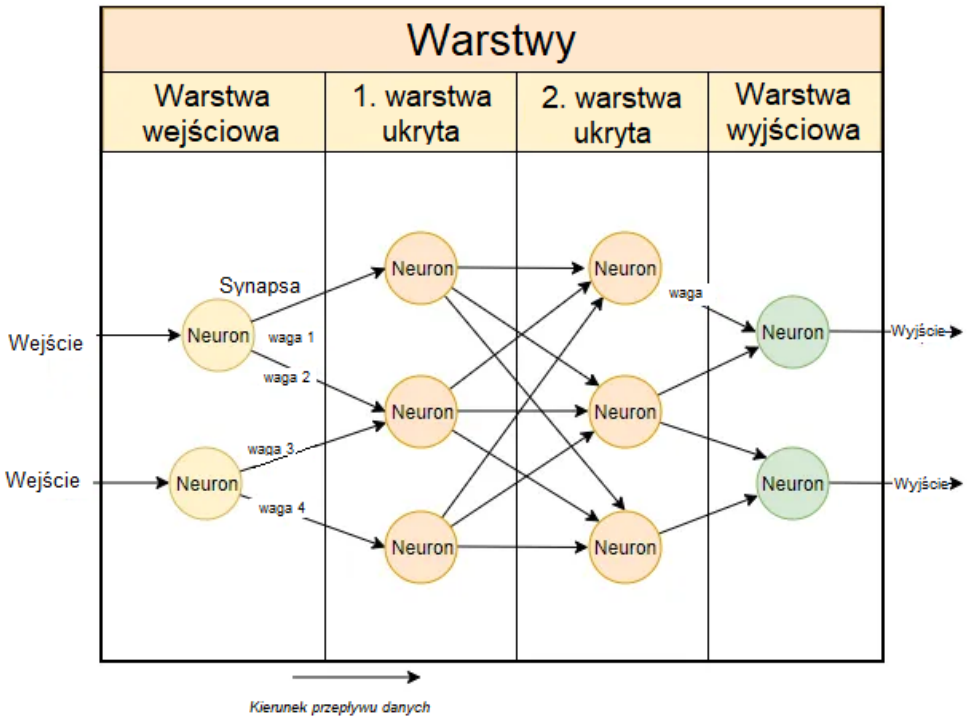


Rysunek 2.1: Sztuczny neuron - schemat działania

funkcje. W sieciach neuronowych można wyróżnić kilka rodzajów warstw, z których każda spełnia określoną rolę w przetwarzaniu danych. Składają się na nie warstwa wejściowa, wyjściowa i warstwy ukryte. Warstwa wejściowa przyjmuje dane i przekazuje je dalej do warstw ukrytych. Warstwy ukryte przetwarzają informacje, wydobywając z danych abstrakcyjne reprezentacje. Wreszcie, warstwa wyjściowa generuje końcowy wynik lub predykcję. Organizacja i liczba warstw w sieci neuronowej stanowi kluczowy element w osiąganiu efektywności i skuteczności modelu. Schemat sztucznej sieci neuronowej przedstawia Rysunek 2.2 [21].

2.4 Operacje konwolucji

Konwolucja jest operacją matematyczną, polegającą na ekstrakcji informacji z obrazów. Pomimo tego, że splot jest prostą operacją, to nadal jest bardzo użyteczna w praktyce. Łatwo można przeanalizować i zrozumieć jej działanie, a jej implementacja może zostać sprawnie przeprowadzona. Operacja konwolucji jest niezależna i liniowa. Niezależność oznacza, że zawsze wykonuje się te same operacje w każdym punkcie obrazu. Liniowość operacji odnosi się do faktu, że każdy piksel podmieniany jest przez liniowe połączenie jego sąsiadów [30].



Rysunek 2.2: Schemat sztucznej sieci neuronowej

2.4.1 Przykład działania

Przed rozpoczęciem operacji konwolucji należy wykonać 3 bardzo ważne kroki:

1. Dobranie filtra konwolucyjnego, który jest zdefiniowany przez użytkownika i jest prezentowany w macierzy, która posiada tyle wymiarów ile przetwarzane dane, a także nieparzystą liczba elementów w kolejnych wymiarach macierzy. Dobrany filtr będzie stosowany dla każdego piksela obrazu. W przypadku przetwarzania obrazu filtr konwolucyjny będzie dwuwymiarowy.

2. Wybór przesunięcia filtra konwolucyjnego, który będzie przesuwany po danych wejściowych, by wykonać operacje dla każdego piksela. Najczęściej spotykane jest przesunięcie o jeden, ale nie znaczy to, że nie istnieją inne warianty.

3. Wybór sposobu reprezentacji przestrzeni bez pikseli, który będzie stosowany przy granicznych danych. Istnieje kilka sposobów, na jakie można zapisać wartości wychodzące poza obszar danych wejściowych. Zazwyczaj przyjmuje się, że poza określonym obszarem komórki macierzy przyjmują wartość 0, ale czasem spotyka się metody wypełnienia ostatnią wartością przygraniczną lub zapętlania wartości macierzy w cykliczny sposób [30].

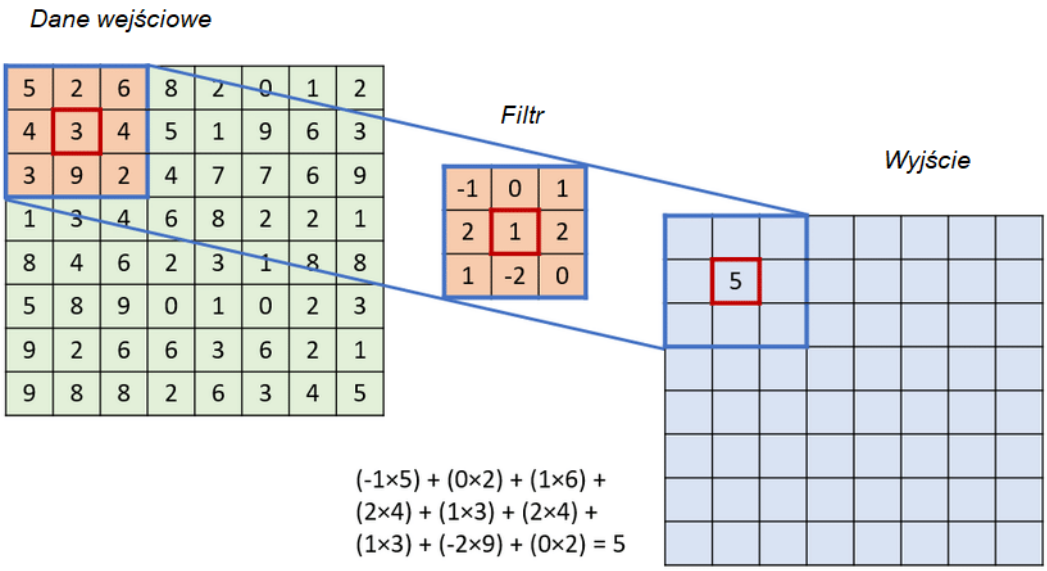
4. Dobór jaka operacja matematyczna będzie scalać wynik mnożenia wartości macieży wejściowej z filtrem konwolucyjnym. Wynik tej operacji matematycznej będzie zapisywany do macieży wyjściowej.

Proces operacji konwolucji zobrazowany na Rysunku 2.3 można opisać na przykładzie obrazu w poniższy sposób:

1. Wykonywanie operacji przemnożenia wartości piksela i wartości pikseli z nim sąsiadującymi przez wartości odpowiednich pozycji filtra konwolucyjnego, a wykonanie końcowej operacji matematycznej wybranej przed początkiem procesu.

2. Przypisanie obliczonej wartości pikselowi wynikowemu dla odpowiedniej lokalizacji filtra.

3. Powtarzanie powyższych kroków dla każdego z elementów danych wejściowych.



Rysunek 2.3: Przykład działania operacji konwolucji

Działanie operacji konwolucji na prostym przykładzie macierzy jednowymiarowej:
Przykładowa macierz wejściowa M może wyglądać następująco:

$$M = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] \tag{2.1}$$

Przykładowy filtr F dla macierzy jednowymiarowej może wyglądać w poniższy sposób:

$$F = [3, 8, 5] \tag{2.2}$$

Jako końcową operację matematyczną w ramach przykładu wybrano średnią arytmetyczną, więc na koniec wynik mnożenia wartości macierzy wejściowej z filtrem będzie trzeba podzielić przez 3

Przeprowadzenie operacji na pierwszej wartości przyjmując najczęściej stosowane założenie, że wartości nieokreślone w macierzy danych wejściowych mają wartość 0:

Jeśli rozpoczyna się operację od pierwszej wartości macierzy wejściowej należy ustawić filtr tak by jego środek znajdował się na pierwszym elemencie co w tym wypadku sprawia, że pierwszy element filtra nakłada się z nieistniejącym elementem macierzy wejściowej. Z tego powodu pierwszy element filtra mnoży się z przez 0 co wynika z powyższego założenia.

$$\begin{aligned} &M[0] \cdot F[1] + M[1] \cdot F[2] + M[2] \cdot F[3] \\ &0 \cdot 3 + 1 \cdot 8 + 2 \cdot 5 = 18 \\ &18/3 = 6 \end{aligned} \tag{2.3}$$

Nowa wartość w macierzy wynikowej dla pierwszego elementu macierzy wejściowej będzie wynosić 6. Później następuje przesunięcie do następnego elementu macierzy wejściowej i proces się powtarza.

2.4.2 Zastosowanie operacji konwolucji

Operacje konwolucji znajdują zastosowanie w analizie oraz przetwarzaniu obrazów, szczególnie w wykrywaniu krawędzi, operacjach rozmycia i wyostrzania, identyfikacji tekstur oraz wielu innych zadaniach z zakresu przetwarzania obrazów. Operacje konwolucji są także stosowane w przetwarzaniu sygnałów takich jak na przykład audio. Filtrów konwolucyjnych używa się do filtrowania sygnałów, analizy częstotliwości oraz redukcji szumów. Operacje konwolucji stosuje się też w dziedzinie biomedycyny przy przetwarzaniu i analizie sygnałów elektroencefalogramowych (EEG), elektrokardiogramowych (EKG). Operacje konwolucji są też stosowane w uczeniu maszynowym, gdyż są podstawowym procesem w konwolucyjnych sieciach neuronowych, których używa się przy problemach rozpoznawania wzorców i klasyfikacji obiektów [3, 30, 71].

2.4.3 Konwolucyjne sieci neuronowe

Konwolucyjne sieci neuronowe są jednym z najbardziej popularnych typów głębokich sieci neuronowych. Te sieci mają wiele warstw a wśród nich: konwolucyjne, nieliniowe oraz zagęszczenia. W warstwach konwolucyjnych wykorzystuje się filtry, które są aplikowane na fragmentach danych wejściowych, wykonując operacje konwolucyjne polegające na mnożeniu wartości pikseli przez odpowiednie wagi, a następnie sumowaniu i uśrednianiu uzyskanych wyników. Konwolucyjne sieci neuronowe znalazły szerokie zastosowanie w problemach związanych z wykrywaniem lokalnych wzorców i cech w danych takich jak na przykład: tekstury czy kolory [3, 70].

2.5 Sieć syjamska

Sieć syjamska (ang. *Siamese network*) jest wykorzystywana do zadania weryfikacji sygnatur biometrycznych. Jest oparta na podejściu uczenia dyskryminatywnego w ramach modeli opartych na funkcji energii, której celem jest opisanie stopnia dopasowania lub niezgodności między dwoma obiektami. Działa na zasadzie porównywania dwóch identycznych obrazów, które są podawane na wejście sieci. W wyniku nauki na tych obrazach otrzymuje się wartość binarną. Jeśli wartość wynosi "0", oznacza to, że obrazy należą do tej samej klasy, a jeśli wynosi "1", oznacza to, że obrazy należą do różnych klas [32]. Sieć syjamska jako podejście do uczenia metryki otrzymuje pary obrazów, zawierających pozytywne i negatywne próbki, aby trenować model sieci. Wykorzystuje funkcję straty (np. strata kontrastowa), aby obliczyć odległość między tymi parami obrazów. Sieć syjamska charakteryzuje się udziałem wag wspólnych dla obu gałęzi sieci, co pozwala na uzyskanie istotnego wzorca między obrazami w głębokim uczeniu metryki. Sieć syjamska może być również łączona z siecią konwolucyjną (ang. *Convolutional Neural Network*) [72].

2.6 Perceptron - przykład sieci neuronowej

Proste modele sieci neuronowych były ważnymi kamieniami milowymi w badaniach nad technologią uczenia maszynowego. Jednym z przykładowych prostych modeli sieci neuronowych jest perceptron. Rozwój sztucznych sieci neuronowych został zainicjowany przez koncepcję perceptronu, sformułowaną przez psychologa Franka Rosenblatta w 1957 roku. Koncepcja ta, opisana jako "Perceptron: automat dostrzegający i rozpoznający (ang. *The perceptron, a perceiving and recognizing automaton Project Para*)"[53], nawiązywała do funkcji mózgu i receptorów systemów biologicznych, starając się naśladować ich działanie[31].

Podstawowym elementem perceptronu jest sztuczny neuron, który przyjmuje ciąg danych wejściowych x_i, gdzie $i \in 1, 2, ..., N$, i oblicza ich ważoną sumę. Wagi w w perceptronie przyjmują ograniczone wartości rzeczywiste. Proces dostosowywania wag dąży do minimalizacji błędu przewidywania wyniku y, który jest wyjściem perceptronu. Wagi na neuronach są zmieniane w każdej iteracji nauki sztucznej sieci neuronowej w celu poprawy dokładności przewidywania [31].

Mechanizm perceptronu wykazuje analogię do procesów zachodzących w układzie odbierającym bodźce wzrokowe i koncentrującym się na tablicy czujników, jak to przedstawiono w 2.4. Jednostka S, pełni funkcję aktywacji. W kontekście sieci neuronowych, funkcja aktywacji to matematyczne przekształcenie, które decyduje, czy dane wejściowe (sumy ważonych wartości) spowodują aktywację neuronu czy też nie. Funkcje aktywacji wprowadzają nieliniowość do działania sieci, umożliwiając modelowaniu złożonych relacji i wzorców w danych. Wynik działania tego mechanizmu, czyli aktywacja jednostki

S, determinuje emisję binarnego sygnału wyjściowego. Zbiór losowo wybranych czujników jest połączony z kolejnym poziomem struktury sieci, określanym jako jednostki A. Każda z jednostek A pełni rolę podstawowej jednostki konstrukcyjnej, analogicznie do opisu wcześniejszego, przy czym wagi ($+1$ lub -1) przypisane do wejść każdej jednostki A są dobierane w sposób losowy.

Działanie perceptronu można opisać w sposób przedstawiony w Równaniu 2.4.

$$y = \begin{cases} 0, & \text{jeśli } \sum_{i=1}^{N}(x_i \cdot w_i) \leq \theta \\ 1, & \text{jeśli } \sum_{i=1}^{N}(x_i \cdot w_i) > \theta \end{cases} \tag{2.4}$$

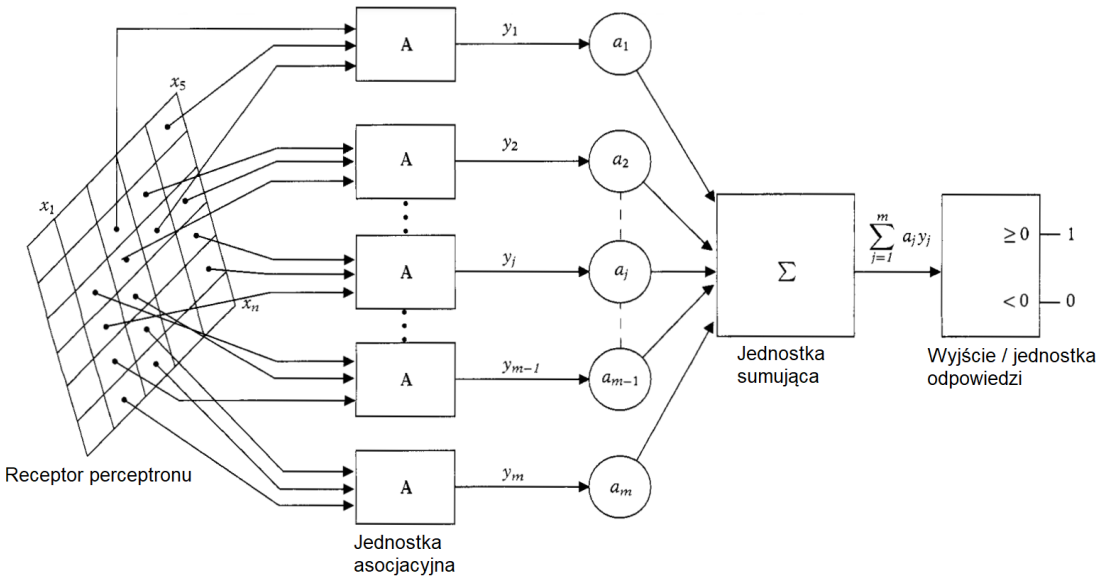
Gdzie:

- N to liczba wejść perceptronu,
- θ to próg aktywacji,
- y to sygnał wyjściowy, który przyjmuje wartość 0 lub 1 w zależności od wyniku sumy ważonej i progu aktywacji.
- x_i to wartość sygnału na i-tym wejściu,
- w_i to waga przypisana do i-tego wejścia,
- $\sum_{i=1}^{N} x_i \cdot w_i$ to suma ważona sygnałów wejściowych,

W przypadku gdy suma ważona jest mniejsza lub równa 0, perceptron nie jest aktywowany ($y = 0$) - sygnał wyjściowy wynosi 0, co jest reprezentowane przez "brak aktywacji", a w przypadku gdy suma ważona jest większa od 0, perceptron jest aktywowany ($y = 1$) - sygnał wyjściowy wynosi 1, co jest reprezentowane przez "aktywację"[31].

Sygnałem wyjściowym perceptronu jest wynik obliczeń, który jest przekształcony poprzez funkcję aktywacji i reprezentuje stan neuronu dla danego zestawu wejść. Ten sygnał wyjściowy jest następnie przekazywany do dalszych neuronów w sieci lub stanowi ostateczny wynik w zależności od liczby warstw modelu [31].

Rosenblatt rozważał różne konfiguracje perceptronów, w tym perceptrony z pojedynczą warstwą ukrytą i perceptrony wielowarstwowe. Do szkolenia perceptronów Rosenblatt zaproponował różne warianty procedury "korekcji błędów", która w praktyce polega na dostosowywaniu wag w odpowiedzi na błędy w klasyfikacji. Ostateczny werdykt na temat możliwości perceptronu zapadł, gdy Minsky i Papert (1969) [46] udowodnili, że perceptron jednowarstwowy nie jest w stanie zaimplementować funkcji logicznej XOR, co wykazało ograniczenia perceptronu w klasyfikacji wzorców. Pomimo tych ograniczeń, perceptrony stały się fundamentem dla dalszych badań nad sieciami neuronowymi [31].



Rysunek 2.4: Struktura perceptronu [31].

2.7 Charakterystyka głębokiego uczenia

Za sprawą sieci neuronowych, które są podtypem uczenia maszynowego, powstało uczenie głębokie. Ta podklasa uczenia maszynowego jest opracowywana przez naukowców w dużej mierze od 2006 roku i od momentu, w którym powstała, była wykorzystywana z dużą skutecznością w wielu dziedzinach [4]. Rysunek 2.5 przedstawia głębokie uczenie z osadzeniem go w klasyfikacji sztucznej inteligencji.

Rysunek 2.5: Głębokie uczenie i jego klasyfikacja w dziedzinie sztucznej inteligencji [60].

Charakterystycznymi cechami głębokiego uczenia są hierarchiczne podejście do uczenia oraz głęboka architektura (wykorzystanie dużej ilości warstw ukrytych). Podczas uczenia głębokiego dla poszczególnych sztucznych neuronów modelu szacowane są wagi, w taki sposób, by umożliwić przepływ danych i wykonanie określonego zadania.

Sieć neuronowa używana w głębokim uczeniu posiada wiele warstw ukrytych, co pozwala przeprowadzać wieloetapowe, nieliniowe przetwarzanie danych o hierarchicznej strukturze [55, 24]. Te informacje są wykorzystywane podczas nauki do wyodrębniania cech i klasyfikacji wzorców z danych wejściowych. W głęboki uczeniu, podczas którego przeprowadza się ekstrakcję cech z danych wejściowych, obecna jest hierarchiczna budowa sieci neuronowej, pozwalająca na reprezentowanie nieprzygotowanych informacji [7]. Dlatego głębokie uczenie obejmuje hierarchię pojęć i cech w taki sposób by pojęcia wysokiego poziomu były definiowalne przez pojęcia niskiego poziomu odwrotnie. Głębokie uczenie jest bardzo uniwersalnym podejściem, przez co stosuje się je w rozwiązywaniu problemów z prawie wszystkich dziedzin.

Uczenie głębokie, podobnie jak uczenie maszynowe w ogólności, można podzielić na podstawie charakteru dostępnych danych uczących, stopnia nadzoru i sposobu, w jaki algorytmy są trenowane, na poniższe podtypy [4]:

- głębokie uczenie nadzorowane
- głębokie uczenie częściowo nadzorowane
- głębokie uczenie nienadzorowane
- głębokie uczenie ze wzmacnianiem

2.7.1 Uczenie nadzorowane

Uczenie nadzorowane to technika szkolenia, w której zestaw danych uczących używanych do treningu algorytmu obejmuje zestawione rozwiązania problemu, nazywane etykietami. W tym podejściu głębokiego uczenia, wejścia są połączone z odpowiadającymi im dopasowanymi wyjściami w środowisku. W momencie pojawienia się konkretnych danych na określonym wejściu, zachodzi formalna operacja przepisania [4].

2.7.2 Uczenie częściowo nadzorowane

Uczenie częściowo nadzorowane jest procesem nauki, który ma miejsce w oparciu o zbiory danych, które są częściowo oznaczone. W pewnych przypadkach techniki uczenia ze wzmocnieniem oraz Generatywne Sieci Przeciwstawne są stosowane jako metody w ramach uczenia pół-nadzorowanego [4].

2.7.3 Uczenie nienadzorowane

Systemy uczenia nienadzorowanego to takie, które są zdolne do działania bez obecności etykiet danych. W tym przypadku model uczy się wewnętrznej reprezentacji lub istotnych cech w celu odkrycia nieznanych relacji, czy też struktury w danych wejściowych. Często klasyfikacja, redukcja wymiarów oraz techniki generatywne są uważane za podejścia nienadzorowanego uczenia [6].

2.7.4 Głębokie uczenie ze wzmacnianiem

Głębokie uczenie ze wzmacnianiem jest paradygmatem uczenia maszynowego, który łączy elementy głębokiego uczenia, w tym tworzenie hierarchicznych reprezentacji cech, z technikami wzmacniania, czyli uczenia poprzez interakcję z otoczeniem i systematyczną optymalizację zachowań. W tym podejściu agent uczenia maszynowego, będący decyzyjnym podmiotem, działa w pewnym środowisku, podejmuje akcje i otrzymuje nagrody lub kary w zależności od podejmowanych działań. Celem głębokiego uczenia ze wzmacnianiem jest nauczenie agenta efektywnego zachowania w celu maksymalizacji nagród w danej interakcji ze środowiskiem. [4].

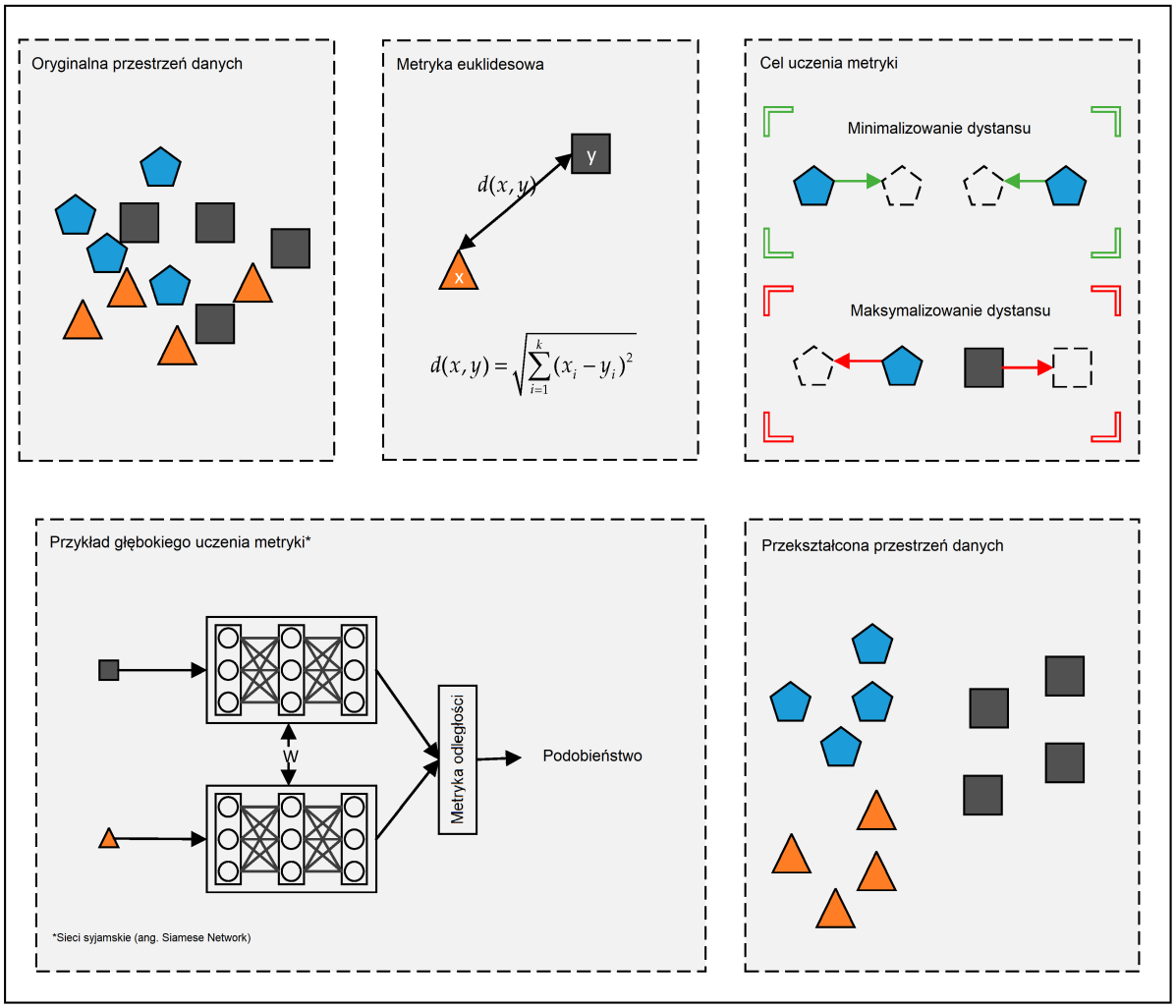
2.8 Głębokie uczenie metryki

Głębokie uczenie metryki to podejście w dziedzinie uczenia maszynowego, które ma na celu nauczenie się odpowiednich reprezentacji danych by ocenić podobieństwa lub odległości między przykładami. Metryki te są wykorzystywane do porównywania i klasyfikacji danych na podstawie ich podobieństwa. W głębokim uczeniu metryki stosuje się głębokie sieci neuronowe do nauki reprezentacji danych, które są wydajne w wyodrębnianiu istotnych cech z danych wejściowych. Te reprezentacje są następnie używane do obliczania odległości lub podobieństwa między różnymi obiektami. W procesie uczenia modelu metrycznego wykorzystuje się funkcje kosztu, które minimalizują różnice między podobnymi przykładami i maksymalizują różnice między niepodobnymi obiektami [32].

Każdy zbiór danych ma swoje specyficzne problemy związane z klasyfikacją i grupowaniem (klastrowaniem). Metryki odległości, które nie wykazują dobrej zdolności przetwarzania danych, niezależnie od problemu, mogą nie przynosić udanych rezultatów w klasyfikacji danych. Dlatego też konieczna jest dobra miara odległości, aby osiągnąć udane rezultaty dla danych wejściowych. W celu rozwiązania tego problemu, przeprowadzono wiele prac, wykorzystując podejścia uczenia metryk [32].

Uczenie metryki dostarcza nowej miary odległości poprzez analizę danych. Podejście oparte na uczeniu metryk, które przeprowadza proces uczenia na danych, posiada większą zdolność do rozróżniania próbek danych. Głównym celem uczenia metryk jest utworzenie nowej przestrzeni danych w której obiekty należące do tych samych klas będą znajdować

się blisko siebie. Podczas nauki jest to osiągane poprzez zmniejszanie odległości między próbkami tej samej klasy oraz zwiększenia odległości między próbkami różnych klas zgodnie ze schematem zaprezentowanym na Rysunku 2.6[32].



Rysunek 2.6: Głębokie uczenie metryki [32].

Głębokie uczenie metryki integruje koncepty głębokiego uczenia i uczenia metryki, co pozwala na wykorzystanie zaawansowanych modeli głębokich sieci neuronowych do ekstrakcji cech i jednoczesne nauczenie odpowiedniej metryki odległości. W głębokim uczeniu metryki, model jest trenowany tak, aby zminimalizować odległość między obiektami tej samej klasy i maksymalizować odległość między obiektami różnych klas. Kluczowym elementem głębokiego uczenia metryki jest zdefiniowanie funkcji straty, która skutecznie kieruje proces uczenia w celu osiągnięcia tego celu. Ostatecznym celem głębokiego uczenia metryki jest nauczenie modelu, który może przekształcić dane wejściowe w taki sposób, że odległości w nowej przestrzeni cech mogą być wykorzystane do różnych zastosowań, takich jak klasyfikacja, grupowanie, czy wyszukiwanie najbliższego sąsiada.

2.8.1 Obliczanie metryki odległości

Niech $X = [x_1, x_2, \ldots, x_N] \epsilon \mathbb{R}^{d \times N}$ oznaczają zbiór próbek treningowych, gdzie $x_i \epsilon \mathbb{R}^d$ jest i-tą próbką treningu, N jest całkowitą liczbą danych treningowych a d to liczba cech dla każdej próbki. Każda próbka x_i jest reprezentowana jako wektor o d wymiarach. Odległość pomiędzy przykładami x_i i x_j może obliczana poniższym wzorem [32]:

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)} \tag{2.5}$$

Gdzie M oznacza macierz ważącą lub macierz metryki, która wpływa na obliczanie odległości między dwiema próbkami x_i i x_j w przestrzeni cech.

Metryka $d_M(x_i, x_j)$ musi posiadać kilka cech:

$$d_M(x_i, x_j) > 0 \Leftrightarrow i \neq j \tag{2.6}$$
$$d_M(x_i, x_j) = 0 \Leftrightarrow x_i = x_j \tag{2.7}$$
$$d_M(x_i, x_j) = d_M(x_j, x_i) \tag{2.8}$$
$$|x_i x_j| + |x_j x_k| \geq |x_i x_k| \tag{2.9}$$

Dodatkowo macierz M musi spełniać warunki:

$$\lambda_{min} > 0 \tag{2.10}$$
$$det(M) > 0 \tag{2.11}$$

Metryka $d_M(x_i, x_j)$ musi posiadać kilka właściwości. Po pierwsze, powinna zwracać wartości nieujemne, czyli żadne odległości nie mogą być ujemne. Po drugie, powinna spełniać zasadę tożsamości nierozróżnialnych, co oznacza, że $d_M(x_i, x_j)$ będzie równe zero tylko wtedy, gdy x_i i x_j są takie same. Trzecią cechą jest symetria, czyli $d_M(x_i, x_j)$ będzie takie samo jak $d_M(x_j, x_i)$, niezależnie od kolejności x_i i x_j. Kolejną ważną właściwością jest nierówność trójkąta, która mówi, że odległość między x_i i x_j plus odległość między x_j i x_k musi być większa lub równa odległości między x_i i x_k. Ponadto, macierz M musi być symetryczna i pozytywnie półokreślona. To oznacza, że wszystkie wartości własne lub wyznaczniki M muszą być większe od zera lub równe zeru, aby macierz była pozytywnie półokreślona. Gdy rozwiniemy M:

Gdy rozwinie się M w poniższy sposób:

$$M = W^T W \tag{2.12}$$

To metrykę odległości można zapisać:

$$\begin{aligned} d_M(x_i, x_j &= \sqrt{(x_i - x_j)^T M (x_i - x_j)} \\ &= \sqrt{(x_i - x_j)^T W^T W (x_i - x_j)} \\ &= \|W x_i - W x_j\|_2 \end{aligned} \tag{2.13}$$

Dzięki właściwości liniowej transformacji macierzy W, można obserwować pewne zależności w infrastrukturze uczenia metryki. Z Równań 2.13 wynika, że odległość euklidesowa między dwoma próbkami w przekształconej przestrzeni jest równa odległości Mahalanobisa między nimi w oryginalnej przestrzeni. Ta zależność pozwala lepiej zrozumieć strukturę danych i proces uczenia metryki [32].

Posiadanie lepszej zdolności reprezentacyjnej danych pozwala na dokładniejsze przewidywanie w problemach klasyfikacji i grupowania. Głównym celem uczenia metryki jest nauka dobrej metryki odległości na podstawie dostępnych danych. Metryka odległości dostarcza nowej reprezentacji danych, która charakteryzuje się bardziej znaczącymi i skutecznymi zdolnościami dyskryminacyjnymi, uwzględniając podobieństwa między próbkami. Warto zauważyć, że liniowa transformacja jest istotnym aspektem w uczeniu metryki. Liniowe podejścia do uczenia metryki oferują większą elastyczność w definiowaniu ograniczeń w przestrzeni przekształconych danych oraz poprawiają wydajność procesu uczenia. Mają one również inne korzyści, takie jak odporność na przeuczenie. Niemniej jednak, choć metody liniowe są pomocne w nauce metryki, istnieje możliwość osiągnięcia jeszcze lepszych zdolności reprezentacyjnych danych. Aby lepiej zrozumieć dane, ważne jest, aby dostosować się do ich naturalnej struktury [32].

Należy jednak pamiętać, że liniowa transformacja ma swoje ograniczenia, szczególnie w przypadku nieliniowych struktur cech. Dlatego w uczeniu metryki stosuje się również metody nieliniowe, takie jak metody jądrowe, które przenoszą problem do przestrzeni nieliniowej. Niemniej jednak, podejścia nieliniowe mogą być podatne na przeuczenie, chociaż są praktyczne w rozwiązywaniu problemów nieliniowych. W latach 10 XXI w. coraz większe zainteresowanie wzbudza głębokie uczenie metryki, które może stanowić bardziej skondensowane rozwiązanie dla obu podejść, pozwalając jednocześnie rozwiązać pewne problemy, które występują w obu podejściach [32].

2.8.2 Różnice między uczeniem metryki a głębokim uczeniem metryki

W odróżnieniu od tradycyjnych metod uczenia maszynowego, które wymagają ręcznego konstruowania reprezentacji cech danych przed etapem klasyfikacji, uczenie głębokie dąży do nabywania wyższego poziomu abstrakcji danych poprzez włączenie tego procesu w samą strukturę modelu. W ramach uczenia głębokiego, model jest w stanie samoistnie

wyodrębniać bardziej złożone i hierarchiczne cechy z surowych danych, co pozwala na automatyczną ekstrakcję istotnych informacji jednocześnie w trakcie przetwarzania danych. Wprowadzenie biblioteki cuDNN przez NVIDIA umożliwiło wykorzystanie mocy obliczeniowej GPU w wielu strukturach (ang. *framework*) uczenia głębokiego, takich jak Caffe, TensorFlow czy PyTorch [55].

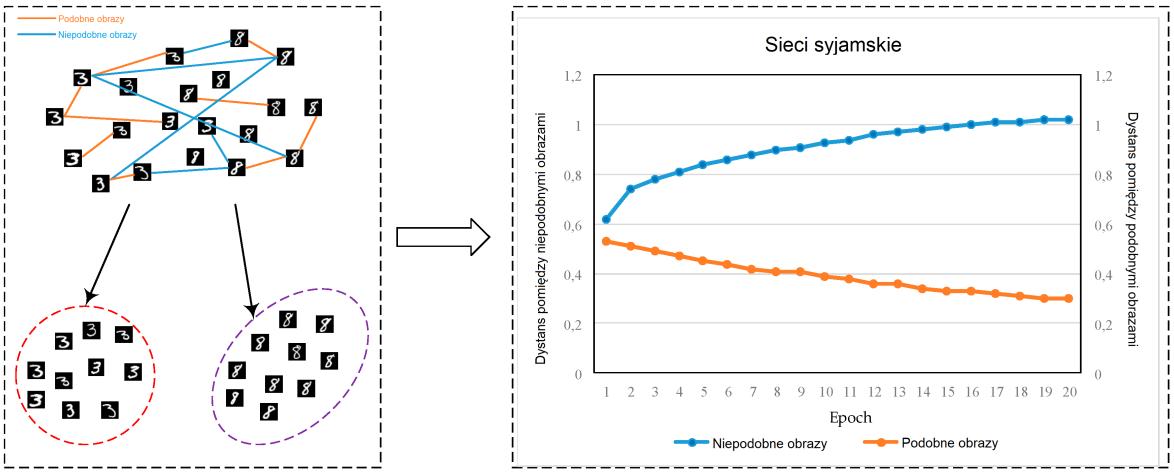
Podstawowe metryki podobieństwa, takie jak odległość euklidesowa, Matusity [44]i Bhattacharyya [8], są często stosowane w klasyfikacji danych. Jednak mają one swoje ograniczenia i nie zawsze dają satysfakcjonujące wyniki. W odpowiedzi na to, zaproponowano [17, 63, 67] podejście oparte na metryce Mahalanobisa, które pozwala na bardziej efektywną klasyfikację danych za pomocą tradycyjnego uczenia metrycznego. W tym podejściu dane są transformowane do nowej przestrzeni cech, która lepiej separuje klasy. Zwykle metody uczenia metryki skupiają się na liniowych transformacjach danych i nie uwzględniają nieliniowych zależności [73]. Dlatego nie zawsze są w stanie wydobyć pełną wiedzę z danych. Chociaż próbowano rozwiązać ten problem przez zastosowanie podejścia opartego na funkcjach jądrowych, nie zawsze przynosi to zadowalające rezultaty ze względu na pewne trudności, takie jak skalowanie [10]. W przeciwieństwie do tradycyjnych metod uczenia metryki, uczenie głębokie radzi sobie z tym problemem, wykorzystując nieliniowe funkcje aktywacji, co pozwala na efektywną ekstrakcję nieliniowych wzorców i cech w danych [32].

W dziedzinie uczenia głębokiego większość istniejących rozwiązań skupia się głównie na architekturze modelu, a nie na metryce odległości w nowej przestrzeni reprezentacji danych. Jednak w XXI wieku podejścia oparte na odległości zyskały dużą popularność w uczeniu głębokim [32]. Uczenie głębokie metryki ma na celu zwiększenie odległości między niepodobnymi próbkami, jednocześnie zmniejszając odległość między podobnymi próbkami [26]. W tym celu wykorzystuje się funkcje straty metrycznej. Poprzez dążenie do zbliżenia próbek z tych samych klas i oddalania próbek z różnych klas od siebie, sieci głębokie są w stanie efektywnie się uczyć. W swojej pracy Redukcja wymiarowości poprzez uczenie się niezmiennego mapowania (ang. *Dimensionality Reduction by Learning an Invariant Mapping*) R. Hadsel przeprowadził eksperymenty na zbiorze danych MNIST, gdzie wykorzystano funkcję straty kontrastowej [23], aby analizować odległości między podobnymi i niepodobnymi obrazami w kolejnych epokach. Wyniki eksperymentu przedstawione na Rysunku 2.7 potwierdziły, że wykorzystanie relacji odległości w sieciach syjamskich (ang. *Siamese networks*) pozwala skutecznie osiągać zamierzone cele tego podejścia [32].

2.9 Elementy głębokiego uczenia metryki

Na głębokie uczenie metryki składają się trzy ważne elementy:

- informacyjne próbki wejściowe



(a) (b)

Rysunek 2.7: Zależność odległości dla sieci syjamskiej: (a) pożądanej dyskryminacji między cyframi trzy i osiem pisanymi odręcznie, (b) zastosowanie sieci syjamskich dla cyfr trzy i osiem. [32].

- struktury modelu sieciowego
- funkcja straty metrycznej

2.9.1 Dobór próbek wejściowych

Ważnym aspektem jest selekcja odpowiednich próbek, która ma duży wpływ na skuteczność klasyfikacji i grupowania danych. Wybór próbek treningowych może być dokonywany w różny sposób, na przykład poprzez losowe wybieranie par pozytywnych czyli próbek posiadających tą samą etykietę co obiekt, który jest w danej chwili punktem odniesienia i negatywnych, czyli takich o etykiecie innej niż próbka odniesienia. Jednak bardziej zaawansowane techniki, takie jak selekcja trudnych negatywnych próbek (ang. *hard negative mining*) [57] czy selekcja informacyjnych trójek, mogą przynieść lepsze rezultaty. W głębokim uczeniu metryki można podzielić próbki w poniższy sposób:

- **Łatwe próbki:** Są to pary próbek, które należą do tej samej klasy i są podobne do siebie. Innymi słowy, odległość między nimi w przestrzeni cech jest niewielka. To próbki, które powinny być "łatwo"rozróżniane przez model, ponieważ reprezentują te same lub bardzo zbliżone cechy.
- **Trudne próbki:** Są to pary próbek, które należą do różnych klas i są sobie bardzo podobne. W przestrzeni cech odległość między nimi jest mała, mimo że należą do różnych klas.
- **Półtrudne próbki:** Półtrudne próbki są próbkami, które znajdują się pomiędzy łatwymi a trudnymi próbkami. Są to pary próbek z różnych klas, ale w przestrzeni

cech mają pewne odległości. Są trudniejsze do rozróżnienia niż łatwe próbki, ale nie tak trudne jak trudne próbki.

Wcześniejsze podejście polegające na wykorzystywaniu łatwych próbek może prowadzić do ograniczonej skuteczności modelu po osiągnięciu pewnego poziomu [57]. Dlatego istotne jest korzystanie z próbek, które posiadają wysoką zdolność dyskryminacyjną, czyli zdolność próbek do skutecznego i precyzyjnego rozróżniania między różnymi kategoriami lub klasami [15]. Dzięki temu można osiągnąć lepsze rezultaty treningu i efektywniejsze uczenie się w ramach głębokiego uczenia metryki.

W głębokim uczeniu metryki istnieje potrzeba identyfikacji trudnych negatywnych próbek, które są błędnie zaklasyfikowane jako pozytywne na podstawie danych treningowych. Metoda selekcji półtrudnych negatywnych próbek (ang. *semi-hard negative mining*) została zaproponowana w celu znalezienia negatywnych próbek znajdujących się wewnątrz marginesu. Margines to odległość, która jest utrzymana między granicami decyzyjnymi klas w przestrzeni cech, oddalonych od próbki zakotwiczenia, czyli próbki danych wybranej jako punkt odniesienia wokół której model jest uczony rozróżniania innych próbek [56]. Ta metoda oferuje bardziej płynne przejście między próbkami pozytywnymi i negatywnymi w porównaniu do selekcji trudnych negatywnych próbek. W literaturze można znaleźć również inne podejścia, takie jak próbkowanie negatywnych klas, które wykorzystują negatywne próbki z różnych klas w celu tworzenia trudnych przykładów [58]. Wybór odpowiednich negatywnych próbek ma istotne znaczenie dla skutecznego treningu modeli głębokiego uczenia metryki i może wpływać na osiągane rezultaty [32].

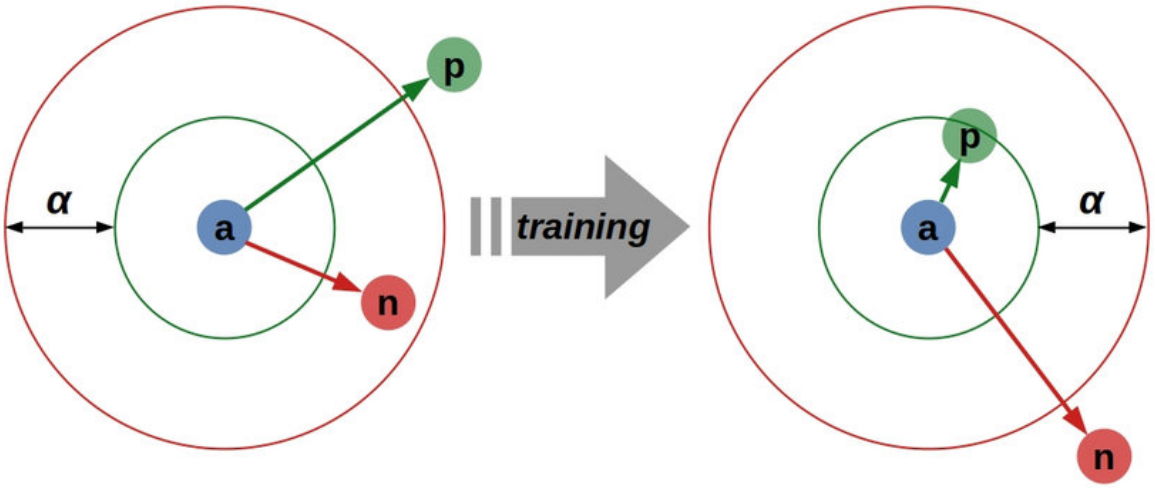
Dobór odpowiednich próbek treningowych ma kluczowe znaczenie dla efektywności uczenia się sieci. Wybór próbek informatywnych jako etapu wstępnego przetwarzania może istotnie zwiększyć skuteczność modelu sieci przed zastosowaniem głębokiego uczenia metryki. Badania w literaturze jednoznacznie wskazują na wysoki wpływ negatywnego próbkowania w głębokim uczeniu metryki. Unikanie zjawiska przeuczenia oraz redukcja złożoności czasowej poprzez selekcję wartościowych trójek (ang. *valuable triple samples*) próbek przyczyniają się do znaczących ulepszeń w wydajności sieci [32].

2.9.2 Najczęściej stosowane modele sieci w głębokim uczeniu metryki

Kierując się artykułem "Głębokie uczenie metryczne: Ankieta (ang. Deep Metric Learning: A Survey)"M. Kaya [32] w kontekście głębokiego uczenia metryki najczęściej opisywanymi przykładami są sieci syjamskie (ang. *Siamese network*) i sieci tripletowe (ang. *Triplet network*) [32, 9, 13, 25]. Poniżej pokrótce opisano wyżej wspomniane sieci.

Sieć trójkowa

Sieć tripletowa jest wykorzystywana do zadania klasyfikacji. Jej głównym celem jest zwiększenie mocy dyskryminacyjnej modelu poprzez wykorzystanie zarówno relacji wewnątrzklasowej, jak i międzyklasowej. Sieć trójkowa skupia się na trójkach próbek, składających się z próbki bazowej, nazywanej również kotwicą, (ang.*anchor*), próbki pozytywnej (ang.*positive*) i próbki negatywnej (ang.*negative*) [25]. W trakcie treningu sieć porównuje podobieństwo między parami próbek, a następnie wykorzystuje funkcję straty (na przykład stratę trójkową), która stawia sobie za cel minimalizację odległości między próbką bazową a próbką pozytywną, jednocześnie maksymalizując odległość między kotwicą a próbką negatywną. Funkcję straty z marginesem dla sieci trójkowej przedstawiono na Rysunku 2.8 [32].



Rysunek 2.8: Schemat straty trójkowej z marginesem α

2.10 Propagacja wsteczna

Algorytm propagacji wstecznej (ang. *backpropagation*) jest jednym z najważniejszych algorytmów używanych do uczenia sieci neuronowych. Jego głównym celem jest obliczenie gradientów wag sieci neuronowej w celu minimalizacji błędu predykcji. Działa na zasadzie propagacji informacji o błędzie z warstwy wyjściowej do warstw ukrytych sieci, dzięki czemu umożliwia aktualizację wag na podstawie tego błędu [40].

Algorytm ten opiera się na regule łańcuchowej z rachunku różniczkowego. W pierwszym kroku sieć jest inicjalizowana losowymi wagami. Następnie, dla każdego przykładu treningowego, obliczane są wartości wyjściowe dla każdej warstwy sieci w procesie zwanym propagacją w przód (ang. *forward propagation*). Po obliczeniu wyjść sieci, następuje krok propagacji wstecznej, w którym obliczane są gradienty wag na podstawie różnicy

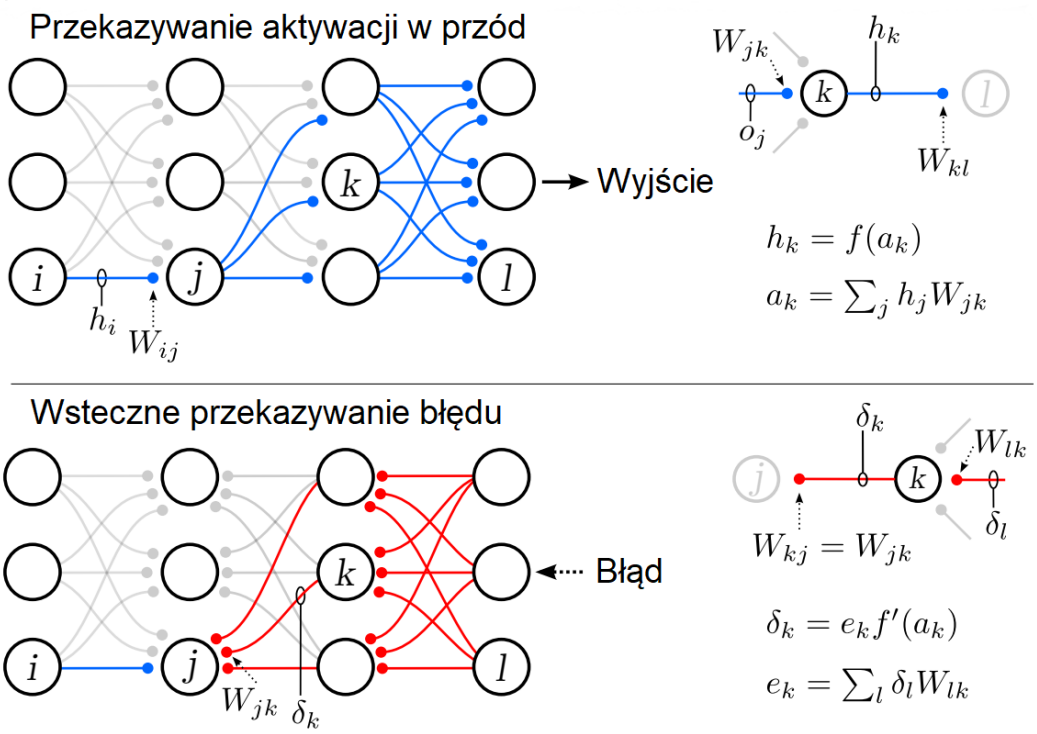
między wyjściami sieci a oczekiwanymi wartościami (czyli błędem predykcji). Ten proces jest realizowany za pomocą reguły łańcuchowej, która pozwala na obliczenie gradientu dla każdej wagi w sieci, uwzględniając wpływ tej wagi na błąd predykcji. W kolejnym kroku, na podstawie obliczonych gradientów, aktualizowane są wagi sieci przy użyciu metody optymalizacji, takiej jak na przykład. stochastyczny spadek gradientu. Aktualizacja wag odbywa się w kierunku przeciwnym do gradientu, co ma na celu minimalizację funkcji kosztu sieci [40]. Przykładowy schemat działania algorytmu propagacji wstecznej został zaprezentowany na Rysunku 2.9, gdzie:

- i, j, k, l to poszczególne neurony.
- h to wyjście z neuronu, oznaczone indeksem odpowiadającego mu neuronowi. Przykładowo h_i to wyjście neuronu i, które jest równe $f(a_i)$.
- W to waga łącząca dwa neurony. Przykładowo W_{ij} to waga łącząca neurony j i k.
- δ to sygnał błędu dla poszczególnych neuronów. δ_j to sygnał błędu dla neuronu j.
- e to błąd wejścia dla poszczególnych neuronów. e_j to błąd wejścia neuronu j.

Algorytm propagacji wstecznej jest iteracyjnie powtarzany dla wszystkich przykładów treningowych w zbiorze uczącym, aż do spełnienia warunku zakończenia, takiego jak osiągnięcie określonej liczby iteracji lub wystarczająco małego błędu predykcji. Algorytm propagacji wstecznej ma również pewne ograniczenia. Jednym z nich jest problem związany z zapętlaniem gradientów w głębokich sieciach neuronowych, co może prowadzić do problemu zanikającego gradientu lub eksplodującego gradientu. Problem zanikającego gradientu polega na tym, że gradienty funkcji kosztu stają się coraz mniejsze w miarę przechodzenia wstecz przez sieć neuronową podczas aktualizacji wag. W efekcie, wagi w początkowych warstwach sieci są aktualizowane bardzo wolno lub wcale, co znacznie utrudnia lub uniemożliwia naukę. Z drugiej strony, problem eksplodującego gradientu polega na tym, że gradienty funkcji kosztu stają się coraz większe w miarę przechodzenia wstecz przez sieć, co prowadzi do bardzo dużej aktualizacji wag, co może powodować niestabilność modelu. Istnieją sposoby rozwiązania powyższych problemów z gradientem. Techniki regularyzacji, takie jak regularyzacja L2, porzucenie (ang. *drop-out*) czy normalizacja wsadu, pomagają kontrolować wielkość wag i aktywacji w sieci neuronowej, co może pomóc w rozwiązaniu problemów z zanikającymi i eksplodującymi gradientami. Regularyzacja L2, znana również jako karanie wag, dodaje karę proporcjonalną do kwadratu wartości wag do funkcji kosztu, co zmusza model do uczenia się mniejszych wag. Porzucenie polega na losowym wyłączaniu pewnych neuronów podczas treningu, co zmusza model do uczenia się bardziej odpornych na zakłócenia reprezentacji. Normalizacja wsadu polega na normalizowaniu aktywacji na każdej warstwie na podstawie średniej i wariancji aktywacji w danym wsadzie, co może pomóc w stabilizowaniu aktywacji i przyspieszaniu procesu uczenia [40].

2.10.1 Algorytm propagacji wstecznej

Aktywność całkowita napływająca do neuronu j (wraz z obciążeniem) jest oznaczona jako a_j, a jego wyjście jako $h_j = f(a_j)$. Pomijając indeksy warstw, które informują zarówno o konkretnym neuronie, jak i o warstwie, do której należy. Funkcja błędu oblicza stopień odchylenia ostatecznych wyników sieci (y_l) od wartości docelowych (t_l). Najczęstszym wyborem jest błąd kwadratowy [40]: $E = \frac{1}{2}\Sigma_l(y_l - t_l)^2$.



Rysunek 2.9: Algorytm propagacji wstecznej (ang. *Backpropagation algorithm*) [40]

Propagacja wsteczna to metoda obliczania gradientu błędu dla każdej wagi przy aktualnych ustawieniach wszystkich wag. Najprostszym sposobem wykorzystania tego gradientu jest zmiana każdej wagi proporcjonalnie do jej negatywnego gradientu [40]. Dla warstw innych niż warstwa wyjściowa, aktualizacja wagi W_{ij} ma postać:

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = -\eta h_i \delta_j, \qquad gdzie \qquad \delta_j = e_j f'(a_j) = (\sum_k \delta_k W_{jk}) f'(a_j) \tag{2.14}$$

W powyższym równaniu:

ΔW_{ij} - Zmiana wagi między neuronami i i j.

η - Współczynnik uczenia, który kontroluje wielkość kroków aktualizacji wag.

$\frac{\partial E}{\partial W_{ij}}$ - Pochodna cząstkowa funkcji błędu E względem wagi W_{ij}.

h_i - Wyjście z neurona i.

δ_j - Sygnał błędu dla neurona j. To jest iloczyn błędu e_j i pochodnej funkcji aktywacji f' obliczonej dla wartości a_j.

e_j - Różnica między docelową wartością a wartością wyjściową neurona j.

$f'(a_j)$ - Pochodna funkcji aktywacji f obliczonej dla wartości a_j, gdzie a_j to całkowita aktywacja neurona j.

$\sum_k \delta_k W_{jk}$ - Suma iloczynów sygnałów błędu δ_k i wag W_{jk} dla wszystkich neuronów k w warstwie wyższej.

W_{jk} - Waga między neuronami j i k.

a_j - Całkowita aktywacja neurona j, czyli suma iloczynów wag i wyjść z poprzednich neuronów plus bias.

Kluczowe spostrzeżenie stojące za algorytmem propagacji wstecznej polega na tym, że terminy δ, nazywane czasem *sygnałami błędu*, mogą być obliczane rekurencyjnie za pomocą reguły łańcuchowej (zamiast mierzone poprzez wprowadzenie szumu i obserwowanie wyników). δ_j jest obliczane na podstawie wszystkich δ_k w warstwie powyżej, dlatego sygnały błędu płyną wstecz przez sieć, zaczynając od sygnałów błędu na jednostkach wyjściowych, które są pochodnymi funkcji błędu [40].

2.11 Przegląd literatury z zakresu funkcji straty dla głębokiego uczenia metryki

W tym podrozdziale porusza się kwestię analizy i doboru funkcji strat stosowanych w głębokim uczeniu metryki, które umożliwiają manipulację odległością między obiektami na podstawie ich podobieństwa. Celem jest uzyskanie jak najbardziej efektywnej reprezentacji cech dla różnych obiektów. Zostaną przedstawione różnice i sposoby wykorzystania tych funkcji. W dalszej części podrozdziału zostaną opisane poniższe funkcje strat:

- Strata marginesu trójkowego (ang. *Triplet Margin Loss*)
- Strata kątowa (ang. *Angular Loss*)
- Strata kołowa (ang. *Circle Loss*)
- Strata kontrastowa (ang. *Contrastive Loss*)
- Strata miękki maks z dużym marginesem (ang. *Large-Margin Softmax Loss*)
- Podniesiona strata strukturalna (ang. *Lifted Structured Loss*)

- Strata marginesu (ang. *Margin Loss*)
- Strata analizy składników sąsiedztwa (ang. *Neighbourhood Components Analysis Loss*)
- Znormalizowana strata miękki maks (ang. *Normalized Softmax Loss*)
- Strata N par (ang. *N Pairs Loss*)
- Strata pełnomocnika kotwicy (ang. *Proxy Anchor Loss*)

2.11.1 Strata marginesu trójkowego

Opis funkcji

[22] Strata Marginesu Trójkowego jest funkcją straty stosowaną w głębokim uczeniu, szczególnie w systemach identyfikacji i weryfikacji twarzy. Jest to specjalny przypadek uczenia z wykorzystaniem pary obrazów, gdzie wykorzystuje się trzy obrazy zamiast dwóch. Te trzy obrazy to: obraz odniesienia - kotwica (ang. *anchor*), obraz pozytywny (ang. *positive*), który jest tym samym obiektem co obraz odniesienia, ale w innym ujęciu lub kontekście, oraz obraz negatywny (ang. *negative*), który jest różnym obiektem w porównaniu z obrazem odniesienia.

Celem jest zminimalizowanie odległości między obrazem odniesienia a obrazem pozytywnym, a jednocześnie zwiększenie odległości między obrazem odniesienia a obrazem negatywnym. Innymi słowy, model jest trenowany tak, aby obrazy tego samego obiektu były blisko siebie w przestrzeni cech, podczas gdy obrazy różnych obiektów były od siebie oddalone.

Algorytm funkcji

Sposób, w jaki zapisujemy funkcję straty marginesu trójkowego, to:

$$L_{triplet} = [d_{ap} - d_{an} + m]_+ \tag{2.15}$$

gdzie:

- $L_{triplet}$: Jest to funkcja straty trójkowej.
- d_{ap} : To jest kwadrat odległości euklidesowej między "kotwicą a próbką pozytywną w przestrzeni cech.
- d_{an} : To jest kwadrat odległości euklidesowej między kotwicą a próbką negatywną w przestrzeni cech.

- m: Jest to margines, wartość dodatnia, którą dodajemy do różnicy między d_{ap} i d_{an}. Margines jest używany do tego, aby zmusić model do nauki cech, które są oddzielone o przynajmniej m.
- $[x]_+$: Jest to funkcja rektyfikowana jednostka liniowa - ReLU (ang. *Rectified Linear Unit*), która zwraca x, jeśli x>0, w przeciwnym razie zwraca 0. To znaczy, że $[d_{ap} - d_{an} + m]_+$ zwraca wartość $d_{ap} - d_{an} + m$, jeśli jest większa niż zero, w przeciwnym razie zwraca 0.

2.11.2 Strata kątowa

Opis funkcji

Strata kątowa to specjalny rodzaj funkcji straty, która jest używana w głębokich sieciach neuronowych. Jest to jedna z technik używanych do poprawy jakości osadzania, czyli procesu mapowania danych wejściowych na przestrzeń, w której obiekty podobne są blisko siebie, a różne obiekty są od siebie oddalone. Strata kątowa skupia się na minimalizacji kąta między wektorami osadzania dla par obiektów, które są uznawane za podobne, oraz maksymalizacji kąta między obiektami, które są uznawane za różne. Działa to poprzez wykorzystanie struktury przestrzeni osadzania, aby wpłynąć na relacje między obiektami. Strata kątowa jest wykorzystywana, na przykład, w sieciach neuronowych służących do klasyfikacji obrazów, identyfikacji twarzy lub analizy wyrażeń twarzy [66].

Algorytm funkcji

Algorytm funkcji straty opisuje, w jaki sposób obliczana jest wartość funkcji straty. Strata kątowa jest zdefiniowana jako:

$$l_{ang}(B) = \frac{1}{N} \sum_{x_a \epsilon B} \{\log[1 + \sum_{\substack{x_a \epsilon B \\ y_n \neq y_a, y_p}} \exp(f_{a,p,n})]\} \tag{2.16}$$

$$f_{a,p,n} = 4 \operatorname{tg}^2 \alpha (x_a + x_p)^T x_n - 2(1 + \operatorname{tg}^2 \alpha) x_a^T x_p \tag{2.17}$$

gdzie:

- $l_{ang}(B)$: Jest to funkcja straty kątowej dla zbioru B.
- N: Jest to liczba elementów w zbiorze B.
- x_a: Jest to punkt referencyjny, dla którego obliczamy stratę.
- B: Jest to zbiór wszystkich punktów danych.
- y_n: Jest to etykieta negatywna, różna od y_a i y_p.

- y_a: Jest to etykieta punktu referencyjnego x_a.
- y_p: Jest to etykieta punktu pozytywnego.
- $f_{a,p,n}$: Jest to funkcja obliczająca różnicę między odległościami punktów referencyjnych, pozytywnych i negatywnych.
- α: Jest to kąt pomiędzy punktami w przestrzeni osadzenia.

2.11.3 Strata kołowa

Opis funkcji

Strata kołowa jest funkcją straty opracowaną [59], aby poprawić wydajność modeli uczenia maszynowego w kontekście uczenia z nadzorem i bez nadzoru. Jest to szczególnie istotne w przypadku zadań związanych z klasyfikacją, gdzie tradycyjne funkcje straty, takie jak krzyżowa entropia, mogą nie być optymalne. Strata kołowa została zaprojektowana w celu zminimalizowania odległości między próbkami tej samej klasy, jednocześnie zwiększając odległość między próbkami różnych klas. Osiąga to poprzez wprowadzenie marginesu pomiędzy pozytywnymi a negatywnymi próbkami, co prowadzi do lepszej separacji klas.

Strata kołowa skupia się na poprawie dyskryminacji cech w przestrzeni cech, co jest kluczowe dla zadań takich jak rozpoznawanie twarzy czy klasyfikacja obrazów. W odróżnieniu od innych funkcji straty, strata kołowa nie tylko minimalizuje odległość między próbkami pozytywnymi i ich odpowiadającymi centrami klas, ale także maksymalizuje odległość między próbkami negatywnymi a centrami klas. Strata kołowa jest szczególnie przydatna w przypadku dużych zbiorów danych, gdzie próbki mogą być nierównomiernie rozłożone w przestrzeni cech.

Strata kołowa jest zdefiniowana w oparciu o relację pomiędzy odległością a kątem w przestrzeni cech. W trakcie procesu uczenia, strata kołowa dąży do zminimalizowania kąta pomiędzy wektorami cech pozytywnych próbek i ich odpowiadającymi centrami klas, podczas gdy dąży do zwiększenia kąta pomiędzy wektorami cech próbek negatywnych a centrami klas. To podejście prowadzi do efektywnej separacji klas i ich klasyfikacji [59].

Algorytm funkcji

Algorytm funkcji straty opisuje metodę wyznaczania wartości straty kołowej:

$$\mathcal{L}_{circle} = \log[1 + \sum_{j=1}^{L} \exp(\gamma \alpha_n^j (s_n^j - \Delta_n)) \sum_{i=1}^{K} \exp(\gamma \alpha_p^j (s_p^j - \Delta_p))] \tag{2.18}$$

gdzie:

- $\mathcal{L}_{circle}$ to strata kołowa.
- γ to współczynnik skali, który kontroluje szybkość, z jaką maleje strata.
- α_n^j i α_p^j to parametry marginesu dla par negatywnych i pozytywnych.
- s_n^j i s_p^j to podobieństwa między próbką kotwiczą a próbkami negatywnymi i pozytywnymi.
- Δ_n i Δ_p to predefiniowane marginesy dla par negatywnych i pozytywnych
- L to całkowita liczba próbek negatywnych.
- K to całkowita liczba próbek pozytywnych.

2.11.4 Strata kontrastowa

Opis funkcji

Strata kontrastowa jest funkcją straty używaną w uczeniu maszynowym, w szczególności w problemach, w których porównuje się parami obiekty, takie jak weryfikacja twarzy. Celem jest minimalizacja odległości między podobnymi parami i maksymalizacja odległości między różnymi parami w przestrzeni cech. W tym celu strata kontrastowa jest obliczane dla pary obiektów, co zwykle obejmuje jeden obiekt podobny i jeden obiekt różny.

W przypadku uczenia nadzorowanego, pary obiektów są etykietowane, gdzie etykieta 0 wskazuje, że obiekty są podobne, a etykieta 1 wskazuje, że obiekty są różne. Strata kontrastowa jest obliczana na podstawie różnicy między obiektami w parze i ich etykietami [23].

Algorytm funkcji

Funkcja straty kontrastowej jest zdefiniowana jako:

$$L_{contrastive} = [d_p - m_{pos}]_+ + [m_{neg} - d_n]_+ \tag{2.19}$$

gdzie:

- d_p - Jest to odległość pomiędzy pozytywnymi parami przykładów, czyli odległość pomiędzy dwoma przykładami, które są tego samego rodzaju lub klasy.
- m_{pos} - Jest to margines pozytywny, czyli minimalna odległość, którą chcemy zachować pomiędzy pozytywnymi parami przykładów.

- d_n - Jest to odległość pomiędzy negatywnymi parami przykładów, czyli odległość pomiędzy dwoma przykładami, które są różnych rodzajów lub klas.
- m_{neg} - Jest to margines negatywny, czyli maksymalna odległość, którą chcemy zachować pomiędzy negatywnymi parami przykładów.
- $[x]_+$ - Oznacza funkcję rampy, która jest równa x dla $x \geq 0$ i 0 dla $x < 0$. Stosowane jest to, aby zapewnić, że wartości w nawiasach są nieujemne.

2.11.5 Strata miękki maks z dużym marginesem

Opis funkcji

Strata miękki maks z dużym marginesem jest funkcją straty zaprojektowaną do ulepszenia zdolności klasyfikacji sieci neuronowych. Jest to ważny składnik w procesie uczenia maszynowego, który pomaga w optymalizacji modelu poprzez minimalizację różnicy między przewidywanymi wyjściami a rzeczywistymi etykietami. Standardowa funkcja miękki maks (ang. *Softmax*) jest często używana w problemach klasyfikacji, ale może prowadzić do problemów, gdy klasy są bardzo zbliżone do siebie lub gdy dane są nierównomiernie rozłożone. Strata miękki maks z dużym marginesem wprowadza margines, czyli dodatkowy parametr, który zmusza model do zwiększenia odległości między różnymi klasami, co prowadzi do lepszej separacji klas i, co za tym idzie, do poprawy zdolności generalizacji modelu. Warto zauważyć, że strata miękki maks z dużym marginesem jest szczególnie przydatny w kontekście problemów takich jak rozpoznawanie twarzy, gdzie klasy są bardzo zbliżone do siebie i wymagają bardziej dokładnej separacji [41].

Algorytm funkcji

Definicja funkcji straty miękki maks z dużym marginesem wyraża się następująco:

$$L_i = -\log\left(\frac{e^{\|W_{y_i}\|\|x_i\|\psi(0_{y_i})}}{e^{\|W_{y_i}\|\|x_i\|\psi(0_{y_i})} + \sum_{j \neq y_i} e^{\|W_j\|\|x_i\|\cos(0_j)}}\right) \tag{2.20}$$

$$\psi(0) = (-1)^k \cos(m0) - 2k, \quad 0\epsilon[\frac{k\pi}{m}, \frac{(k+1)}{\pi}m] \tag{2.21}$$

gdzie

- L_i: Strata dla i-tej próbki w zbiorze danych.
- W_{y_i}: Wiersz macierzy wag odpowiadający prawdziwej klasie próbki x_i.
- x_i: Wektor cech i-tej próbki.
- $\psi(0_{y_i})$: Funkcja modyfikująca kąt między W_{y_i} a x_i.

- 0_{y_i}: Kąt między W_{y_i} a x_i.
- 0_j: Kąt między W_j a x_i dla j-tej klasy, która nie jest prawdziwą klasą próbki x_i.
- m: Hiperparametr kontrolujący margines.
- k: Największa liczba całkowita, że 0_{y_i} należy do przedziału $[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$.

2.11.6 Podniesiona strata strukturalna

Opis funkcji

Podniesiona strata strukturalna jest funkcją straty stosowaną w uczeniu głębokim, która służy do poprawy osadzania cech przez zdefiniowanie nowego celu przewidywania struktury. Tradycyjnie, algorytmy uczenia maszynowego dążą do minimalizacji odległości między próbkami tej samej klasy i maksymalizacji odległości między próbkami różnych klas. Jednakże, taka strategia może prowadzić do nieoptymalnych rozwiązań w przypadkach, gdy różnice między klasami są subtelne [49].

Przewidywanie struktury jest związane z matrycą odległości parzystych w obrębie partii podczas treningu sieci neuronowej. Matryca odległości parzystych w obrębie partii to macierz, która zawiera odległości pomiędzy wszystkimi parami próbek w danej partii danych przetwarzanych przez sieć neuronową. Ten nowy cel opiera się na podniesionej gęstej macierzy odległości parzystych, co pozwala na uwzględnienie zarówno pozytywnych, jak i negatywnych par próbek podczas treningu, pomagając w znalezieniu optymalnego osadzenia cech, które są odpowiednie do określonych zadań, takich jak klasyfikacja, regresja czy wyszukiwanie. Podniesiona gęsta macierz odległości parzystych to specjalny rodzaj tej macierzy, który uwzględnia dodatkowy wymiar danych, dodaje dodatkowe informacje o strukturze danych. W kontekście algorytmu funkcji straty, podniesienie odnosi się do zastosowania funkcji straty, która promuje podobne lub należące do tej samej klasy pary próbek i degraduje pary próbek, które są różne lub należą do różnych klas.

W praktyce, algorytm ten jest oparty na uczeniu metryki, co pozwala na wyodrębnianie cech, które są bardziej rozróżniające i odpowiednie do określonych zadań. W ten sposób podniesiona strata strukturalna pozwala na poprawę jakości osadzania cech i ułatwia proces przetwarzania obrazów dla różnych zadań [49].

Algorytm funkcji

Funkcja podniesionej straty strukturalnej jest określona jako:

$$\tilde{J}_{i,j} = \log(\sum_{(i,k)\epsilon N} \exp\{\alpha - D_{i,k}\} + \sum_{(j,l)\epsilon N} \exp\{\alpha - D_{j,l}\}) + D_{i,j} \tag{2.22}$$

$$\tilde{J} = \frac{1}{2|P|} \max(0, \tilde{J}_{i,j})^2 \tag{2.23}$$

gdzie:

- $\tilde{J}_{i,j}$: Jest to wartość funkcji straty dla pary indeksów (i, j), reprezentujących elementy w zbiorze danych, które są porównywane. Oznacza ona pewną miarę odległości pomiędzy elementami i i j, uwzględniając również porównania z innymi elementami (k i l) z sąsiedztwa.
- N: Jest to zbiór sąsiedztwa, który składa się z indeksów k i l. Względem elementu i są to indeksy innych elementów, z którymi i jest porównywane.
- α: To parametr, który może wpływać na wagę wyrażeń w sumach eksponent.
- $D_{i,k}$ i $D_{j,l}$: Są to miary odległości pomiędzy elementami i i k, oraz j i l odpowiednio.
- $\tilde{J}$: Jest to znormalizowana wartość funkcji straty, która jest podniesiona do kwadratu i podzielona przez dwa razy liczbę elementów w zbiorze P.

2.11.7 Strata marginesu

Opis funkcji

Strata marginesu jest funkcją straty wykorzystywaną w uczeniu maszynowym, w szczególności w modelach opartych na sieciach neuronowych. Funkcja ta służy do zminimalizowania odległości między przykładami pozytywnymi, czyli próbkami posiadającymi te same etykiety a ich odpowiednikami negatywnymi, czyli próbkami posiadającymi różne etykiety w przestrzeni osadzeń. Strata marginesu jest szczególnie użyteczna w zastosowaniach, gdzie istotne jest, aby osadzenia obiektów tej samej klasy były blisko siebie, podczas gdy obiekty różnych klas były oddzielone od siebie przez pewien margines. Na przykład, w rozpoznawaniu twarzy, chcemy, żeby osadzenia różnych zdjęć tej samej osoby były blisko siebie, a osadzenia różnych osób były oddzielone. Strata marginesu zmusza model do nauczenia się osadzeń, które spełniają te warunki, poprzez nałożenie kary na pary osadzeń, które są zbyt blisko siebie lub zbyt daleko od siebie.[69].

Algorytm funkcji

Funkcja straty podniesionej struktury jest zdefiniowana jako:

$$\text{minimize} \sum_{i,j} l^{margin}(i,j) + \nu(\beta^{(0)} + \beta^{(class)}_{c(i)} + \beta^{(img)}_{i}) \tag{2.24}$$

$$l^{margin}(i,j) := (\alpha + y_{ij}(D_{ij} - \beta))_{+} \tag{2.25}$$

gdzie:

- $\sum_{i,j}$: Oznacza sumowanie po wszystkich parach (i,j).
- $l^{margin}(i,j)$: To jest funkcja straty marginesu dla pary (i,j).
- α: To jest stała, która kontroluje margines, czyli różnicę między pozytywnymi a negatywnymi parami.
- y_{ij}: To jest etykieta, która przyjmuje wartość +1 dla pozytywnych par i -1 dla negatywnych par.
- D_{ij}: To jest odległość między i-tym i j-tym elementem.
- β: To jest parametr, który jest uczony w procesie optymalizacji.
- $(\cdot)_{+}$: Oznacza funkcję ReLU, która jest równa $max(0, \cdot)$.
- ν: To jest parametr regularyzacji, który kontroluje wpływ członu regularyzacji na funkcję celu.
- $\beta^{(0)}$: To jest stały składnik regularyzacji.
- $\beta^{(class)}_{c(i)}$: To jest składnik regularyzacji związany z klasą elementu i.
- $\beta^{(img)}_{i}$: To jest składnik regularyzacji związany z obrazem i.

2.11.8 Strata analizy składników sąsiedztwa

Opis funkcji

Analiza Składników Sąsiedztwa jest to metoda ucząca się nieliniowego odwzorowania wejściowego zestawu danych do przestrzeni, w której odległości między elementami odzwierciedlają ich wzajemne podobieństwo. Kluczowym elementem tego podejścia jest funkcja straty, która jest zoptymalizowana w trakcie procesu uczenia się. Celem tego procesu jest znalezienie odwzorowania, które zwiększa prawdopodobieństwo, że najbliższy sąsiad danego punktu w przestrzeni odwzorowanej będzie z tej samej klasy co punkt odniesienia. Funkcja straty jest zdefiniowana jako różnica między prawdziwym rozkładem klas a przewidywanym rozkładem, który jest obliczany na podstawie odległości między elementami w przestrzeni odwzorowanej. Istotne jest, że funkcja ta nie zakłada żadnej

szczególnej struktury klas ani struktury granicy decyzyjnej, opierając się wyłącznie na silnej regularyzacji wprowadzonej przez ograniczenie do liniowej transformacji wejść [18].

Algorytm funkcji

Definicja funkcji strata analizy składników sąsiedztwa wyraża się następująco:

$$g(A) = \sum_i \log(\sum_{j \epsilon C_i} p_{ij}) = \sum_i \log(p_i) \tag{2.26}$$

$$p_i = \sum_{j \epsilon C_i} p_{ij} \tag{2.27}$$

$$p_{ij} = \frac{\exp(-\|Ax_i - Ax_j|^2)}{\sum_{k \neq i} \exp(-\|Ax_i - Ax_k|^2)} \tag{2.28}$$

gdzie:

- $g(A)$: Jest to funkcja straty, która jest sumą logarytmów prawdopodobieństw p_i dla wszystkich punktów danych i.
- p_i: Jest to prawdopodobieństwo, że punkt danych i zostanie wybrany jako punkt odniesienia dla innych punktów w tej samej klasie C_i. Jest to suma prawdopodobieństw p_{ij} dla wszystkich punktów j w tej samej klasie co i.
- p_{ij}: Jest to prawdopodobieństwo, że punkt danych j zostanie wybrany jako najbliższy sąsiad punktu i. Jest to wyznaczone na podstawie odległości między Ax_i i Ax_j w przestrzeni przekształconej przez macierz A. Im mniejsza odległość, tym większe prawdopodobieństwo.
- A: Jest to macierz przekształcenia, która jest uczona w celu minimalizacji funkcji kosztu $g(A)$.
- Ax_i: Jest to punkt danych i przekształcony przez macierz A.
- Ax_j: Jest to punkt danych j przekształcony przez macierz A.
- Ax_k: Jest to punkt danych k przekształcony przez macierz A.
- C_i: Jest to zbiór punktów danych należących do tej samej klasy co punkt i.

2.11.9 Znormalizowana strata miękki maks

Opis funkcji

Znormalizowana strata miękki maks jest funkcją straty, która odgrywa kluczową rolę w procesie uczenia modeli głębokich sieci neuronowych. Ta funkcja została zaprojektowana, aby zaradzić problemom związanym z tradycyjną funkcją straty miękki maks (ang.

Softmax Loss), takimi jak problemy z optymalizacją w przestrzeni o dużych wymiarach, i problem z normalizacją wektorów cech. W tradycyjnej funkcji straty miękki maks, wyjście z sieci jest transformowane do prawdopodobieństw dla różnych klas, a następnie obliczane jest prawdopodobieństwo przynależności do prawdziwej klasy. Funkcja straty jest obliczana na podstawie różnicy pomiędzy prawdziwym i przewidywanym prawdopodobieństwem. Znormalizowana strata miękki max różni się tym, że normalizuje wektory cech przed obliczeniem prawdopodobieństw. To normalizowanie wektorów cech pozwala na poprawienie zdolności generalizacji modelu, ponieważ zmniejsza wpływ różnych skal i ułatwia optymalizację. [74].

Algorytm funkcji

Definicja funkcji znormalizowanej straty miękki maks wygląda następująco:

$$L_{norm} = -\log(\frac{\exp(x^T p_y/\sigma)}{\sum_{z\epsilon Z}\exp(x^T p_z/\sigma)} \tag{2.29}$$

gdzie:

- L_{norm}: To jest znormalizowana strata miękki maks.
- x: To jest wektor cech obiektu, który jest aktualnie rozważany.
- p_y: To jest wektor wag dla prawdziwej klasy obiektu.
- p_z: To jest wektor wag dla każdej możliwej klasy z zestawu klas Z.
- Z: To jest zbiór wszystkich możliwych klas.
- σ: To jest parametr skalowania używany do normalizacji wyników.
- exp: To oznacza funkcję wykładniczą.

2.11.10 Strata N par

Opis funkcji

Strata N par jest funkcją straty wykorzystywaną w głębokim uczeniu metryki, która dąży do poprawy jakości i zbieżności procesu uczenia wykorzystywanych w stracie trójkowej. W podstawowej formie, strata trójkowa ma na celu minimalizację odległości między przykładami należącymi do tych samych klas, nazywanych pozytywnymi próbkami i maksymalizację odległości między przykładami należącymi do różnych klas, nazywanych

negatywnymi próbkami. Jednak może to prowadzić do problemów związanych z trudnościami w doborze odpowiednich przykładów negatywnych, co z kolei wpływa na proces uczenia i jakość reprezentacji [58].

Strata N par została wprowadzona w celu radzenia sobie z tymi problemami. Główną ideą jest wymuszenie równoczesnego oddzielania pozytywnych i negatywnych przykładów poprzez wprowadzenie większej liczby negatywnych przykładów dla każdego pozytywnego przykładu. W praktyce, dla każdego przykładu pozytywnego wybiera się N przykładów negatywnych, co prowadzi do bardziej zrównoważonego procesu uczenia.

Zadaniem strata N par jest minimalizowanie wartość funkcji straty poprzez maksymalizację odległości między przykładami pozytywnymi a negatywnymi. Poprzez wprowadzenie większej liczby negatywnych przykładów, model jest zmuszony do tworzenia bardziej rozdzielczych i skoncentrowanych reprezentacji cech, co z kolei przekłada się na lepszą zdolność rozróżniania między różnymi klasami w procesie klasyfikacji lub identyfikacji [58].

W efekcie, strata N par przyczynia się do poprawy jakości reprezentacji cech oraz lepszej wydajności modeli w zadaniach takich jak klasyfikacja obrazów, rozpoznawanie twarzy czy wyszukiwanie wzorców. Jej zastosowanie umożliwia bardziej skuteczne wykorzystanie danych treningowych, co ma istotny wpływ na końcową wydajność i skuteczność modelu w rzeczywistych zastosowaniach.

Algorytm funkcji

Funkcja straty N par jest zdefiniowana jako:

$$\mathcal{L}_{N-pair}(\{(x_i, x_i^+)\}_{i=1}^N; f) = \frac{1}{N}\sum_{i=1}^{N}\log(1+\sum_{j\neq 1}\exp(f_i^\top f_j^+ - f_i^\top f_i^+)) \tag{2.30}$$

gdzie:

- $\mathcal{L}_{N-pair}$: To jest funkcja straty N par.
- x_i: To jest i-ty obraz w zestawie danych.
- x_i^+: To jest pozytywny przykład dla obrazu x_i.
- N: To jest liczba par obrazów w zestawie danych.
- f: To jest funkcja, która mapuje obrazy wejściowe na wektory cech.
- f_i: To jest wektor cech obrazu x_i, uzyskany przez zastosowanie funkcji f do x_i.
- f_i^+: To jest wektor cech obrazu x_i^+, uzyskany przez zastosowanie funkcji f do x_i^+.
- $f_i^\top$: To jest transpozycja wektora cech f_i.

W powyższym równaniu, dla każdej pary obrazów x_i i x_i^+, oblicza się różnicę między iloczynem skalarnym $f_i^\top f_j^+$ i $f_i^\top f_i^+$ dla wszystkich $j \neq i$, a następnie oblicza się eksponent tej różnicy. Sumuje się te eksponenty dla wszystkich $j \neq i$, dodaje się 1, a następnie oblicza się logarytm tego wyniku. Ostatecznie, oblicza się średnią z tych logarytmów dla wszystkich i.

2.11.11 Strata pełnomocnika kotwicy

Opis funkcji

Strata pełnomocnika kotwicy jest to funkcja straty opracowana w celu ulepszenia działania modeli uczących się głębokiej reprezentacji na zbiorach danych obrazów. Funkcja ta została zaprojektowana w celu zarządzania trudnościami związanymi z uczeniem głębokich sieci neuronowych, takimi jak optymalizacja na dużą skalę i przetwarzanie dużej liczby klas [35].

Model wykorzystuje osadzenia obrazów, reprezentujące obrazy wejściowe w przestrzeni osadzeń, i generuje pełnomocników dla każdej klasy, które są osadzeniami odniesienia dla wszystkich obrazów w danej klasie. Strata pełnomocnika kotwicy wykorzystuje tych pełnomocników w celu minimalizacji odległości między obrazami z tej samej klasy i maksymalizacji odległości między obrazami z różnych klas.

W czasie uczenia, model minimalizuje różnicę pomiędzy osadzeniem obrazu a najbliższym pełnomocnikiem tej samej klasy i maksymalizuje różnicę pomiędzy osadzeniem obrazu a najbliższym pełnomocnikiem innej klasy. W rezultacie, model nauczy się reprezentacji, które skutecznie separują obrazy różnych klas i ściśle grupują obrazy tej samej klasy [35].

Algorytm funkcji

Funkcja straty pełnomocnika kotwicy jest matematycznie zdefiniowana w następujący sposób:

$$l(x) = \frac{1}{|P^+|} \sum_{p \epsilon P^+} \log(1 + \sum_{x \epsilon X_p^+} e^{-\alpha(s(x,p)-\delta)}) + \frac{1}{|P|} \sum_{p \epsilon P} \log(1 + \sum_{x \epsilon X_p^-} e^{\alpha(s(x,p)+\delta)}) \qquad (2.31)$$

gdzie:

- $l(x)$: To jest funkcja straty pełnomocnika kotwicy.
- P^+: To jest zbiór wszystkich par pozytywnych, czyli par obrazów, które są uznawane za podobne.
- P: To jest zbiór wszystkich par obrazów, zarówno pozytywnych, jak i negatywnych.

- X_p^+: To jest zbiór wszystkich pozytywnych obrazów dla danego obrazu odniesienia p.
- X_p^-: To jest zbiór wszystkich negatywnych obrazów dla danego obrazu odniesienia p.
- α: To jest hiperparametr, który kontroluje wagę wykładnika w równaniu.
- $s(x, p)$: To jest funkcja podobieństwa między obrazem x a obrazem odniesienia p.
- δ: To jest margines, który kontroluje, jak duże różnice między parą pozytywną a parą negatywną są uznawane za akceptowalne.

Wartości $s(x, p)$, α, i δ są ustalane podczas procesu uczenia. Model stara się dostosować te parametry tak, aby zminimalizować wartość funkcji straty $l(x)$, co prowadzi do lepszego wydajności modelu.

2.12 Przegląd istniejących rozwiązań

Głębokie uczenie metryki znajduje zastosowanie w wielu dziedzinach:

- **Rozpoznawanie twarzy:** Technika ta pozwala na porównanie dwóch obrazów twarzy i określenie, czy należą one do tej samej osoby. Jest to kluczowe dla systemów bezpieczeństwa, które muszą weryfikować tożsamość osób wchodzących na chronione tereny lub uzyskujących dostęp do zabezpieczonych urządzeń.
- **Rekomendacje produktów:** Poprzez analizę cech różnych produktów, głębokie uczenie metryki pozwala na rekomendowanie użytkownikom produktów o podobnych cechach do tych, które już wybrali.
- **Wyszukiwanie obrazów:** Technika ta pozwala na porównanie obrazów i wyszukiwanie obrazów podobnych do danego obrazu, co jest kluczowe dla wyszukiwarek internetowych i platform udostępniających zdjęcia.
- **Śledzenie wizualne:** Głębokie uczenie metryki pozwala na porównanie obiektów na kolejnych klatkach obrazu, co umożliwia śledzenie ruchu obiektów. Jest to niezbędne dla pojazdów autonomicznych i systemów monitoringu.
- **Ponowna identyfikacja osób:** Technika ta pozwala na porównanie obrazów osób z różnych kamer, co umożliwia ponowną identyfikację osób, co jest kluczowe dla systemów monitoringu i bezpieczeństwa.

Potencjalne wykorzystanie tej technologii w różnych obszarach może wyglądać następująco:

- **Urządzenia mobilne:** Na przykład urządzenia mobilne firmy Apple [5] mogą wykorzystywać głębokie uczenie metryki w technologii rozpoznawania twarzy, co pozwala na weryfikację tożsamości użytkownika i odblokowanie urządzenia.
- **Platformy handlu internetowego:** Na przykład platforma handlowa Amazon [51] może wykorzystywać głębokie uczenie metryki do analizy cech produktów i rekomendowania użytkownikom produktów o podobnych cechach do tych, które już wybrali.
- **Pojazdy autonomiczne:** Na przykład pojazdy autonomiczne firmy Tesla [61] mogą wykorzystywać głębokie uczenie metryki do analizy obrazów z kamer i śledzenia ruchu obiektów, co pozwala na bezpieczną jazdę.

Należy zaznaczyć, że to, czy te firmy i urządzenia rzeczywiście wykorzystują głębokie uczenie metryki, jest tylko hipotezą, ponieważ konkretne algorytmy i technologie, które wykorzystują, są własnością i nie są ujawniane publicznie. Dlatego nie ma potwierdzenia, czy rzeczywiście wykorzystują głębokie uczenie metryki w swoich rozwiązaniach.

Rozdział 3

Przedmiot pracy

3.1 Opis użytych narzędzi

3.1.1 PyTorch

PyTorch to zaawansowana biblioteka do obliczeń numerycznych, zaprojektowana głównie do wsparcia prac badawczych oraz tworzenia aplikacji w dziedzinie uczenia maszynowego i głębokiego uczenia.

Główne cechy PyTorch obejmują:

- **Tensory:** Tensory stanowią podstawową strukturę danych w bibliotece PyTorch. Są to wielowymiarowe tablice, które umożliwiają przechowywanie i przetwarzanie danych numerycznych. Tensory w PyTorch przypominają macierze, ale mogą mieć dowolną liczbę wymiarów. Są one wykorzystywane do reprezentowania cech, danych wejściowych, wag modelu i wyników pośrednich podczas obliczeń. Tensory w PyTorch są nie tylko kontenerami na dane, ale również umożliwiają wykonywanie zaawansowanych operacji matematycznych, operacji na GPU oraz automatyczną propagację gradientów, co jest kluczowe w procesie uczenia maszynowego i trenowania modeli głębokich sieci neuronowych.

- **Autograd:** Autograd to mechanizm PyTorch umożliwiający automatyczne obliczanie gradientów funkcji. Jest to kluczowy element w procesie uczenia maszynowego, ponieważ umożliwia efektywne aktualizowanie parametrów modelu poprzez optymalizację na podstawie gradientów funkcji kosztu.

- **Dynamiczny graf obliczeniowy:** PyTorch używa dynamicznego grafu obliczeniowego, co oznacza, że graf ten jest tworzony w trakcie wykonywania programu. To umożliwia bardziej elastyczną i intuicyjną manipulację modelem oraz łatwiejsze usuwanie błędów.

- **Modułowość:** PyTorch umożliwia tworzenie modeli poprzez składanie różnych

warstw i komponentów. To ułatwia konstrukcję skomplikowanych architektur modeli.

- **Wsparcie dla GPU:** PyTorch pozwala na przyspieszenie obliczeń poprzez wykorzystanie kart graficznych (ang. *Graphics Processing Unit*) w skrócie GPU. To znacząco przyspiesza trening modeli głębokiego uczenia.

- **Dynamiczne definiowanie modeli:** Możliwość dynamicznego definiowania modeli w PyTorch sprawia, że jest on idealny do eksperymentowania z różnymi architekturami modeli oraz podejściami do uczenia maszynowego.

PyTorch jest powszechnie stosowany w środowiskach badawczych, akademickich i przemysłowych do tworzenia, trenowania i wdrażania modeli uczenia maszynowego i głębokiego uczenia, umożliwiając naukowcom i inżynierom skupienie się na tworzeniu innowacyjnych rozwiązań w dziedzinie sztucznej inteligencji.

3.1.2 PyTorch Metric Learning

PyTorch Metric Learning stanowi zaawansowaną platformę programistyczną, która umożliwia badaczom oraz praktykom prowadzenie analiz i eksploracji w obszarze głębokiego uczenia metryki. Dzięki swojej elastycznej i rozbudowanej strukturze, biblioteka ta pozwala na projektowanie, implementację oraz testowanie różnorodnych algorytmów i strategii związanych z nauką reprezentacji danych w kontekście przestrzeni metrycznych.

Jednym z kluczowych aspektów możliwych do realizacji dzięki PyTorch Metric Learning jest możliwość tworzenia modeli, które potrafią automatycznie dostosować swoje wagi i parametry w celu zoptymalizowania odległości między różnymi elementami w przestrzeni metrycznej. To pozwala na skupienie się na istotnych cechach danych i tworzenie bardziej spójnych i mierzalnych reprezentacji.

Biblioteka ta umożliwia przetestowanie i dostosowanie różnych metryk odległości, w tym zarówno klasycznych euklidesowych, jak i bardziej złożonych nieliniowych, dostosowując się do specyficznych właściwości danych i problemów analizy.

PyTorch Metric Learning ułatwia również proces ewaluacji modeli, pozwalając na ocenę ich wydajności przy użyciu różnorodnych wskaźników ewaluacyjnych, które pomagają zrozumieć, w jaki sposób model radzi sobie z określonymi zadaniami analizy podobieństwa lub różnicy między elementami.

Dzięki swojej zintegrowanej strukturze z platformą PyTorch, biblioteka ta otwiera drzwi do wykorzystania innych elementów ekosystemu PyTorch, takich jak modele, funkcje strat oraz mechanizmy optymalizacji, co w efekcie umożliwia tworzenie kompleksowych i wydajnych rozwiązań z zakresu uczenia metrycznego.

3.1.3 TensorBoard

TensorBoard, stworzony przez Google, jest narzędziem wizualizacyjnym, które stanowi nieodłączny element biblioteki TensorFlow ale jest możliwy do użycia razem z biblioteką PyTorch, służącą do monitorowania i analizy procesów treningu modeli głębokiego uczenia. Jednym z kluczowych aspektów TensorBoard jest możliwość wizualizacji grafu obliczeniowego, co pozwala na wyświetlenie architektury modelu oraz zobaczenie połączeń między jego warstwami. To umożliwia lepsze zrozumienie przepływu danych w trakcie propagacji wstecznej oraz procesu uczenia.

Kolejną ważną funkcją jest monitorowanie krzywych uczenia, gdzie możemy śledzić zmiany funkcji straty oraz metryk ewaluacyjnych w kolejnych epokach treningu. To umożliwia ocenę postępów w procesie uczenia oraz wykrycie potencjalnych problemów, które mogą wymagać interwencji. TensorBoard umożliwia także wizualizację rozkładu wag w poszczególnych warstwach modelu. Dzięki temu możemy zidentyfikować nieliniowe efekty, takie jak zjawisko zanikającego gradientu, które mogą wpływać na skuteczność modelu. Dodatkowo, TensorBoard oferuje profilowanie, co pozwala na analizę wydajności modelu i identyfikację potencjalnych źródeł opóźnień, co jest istotne w optymalizacji procesu uczenia.

3.1.4 Scikit-learn

Scikit-learn jest biblioteką o otwartym źródle programistyczną w języku Python, dedykowaną analizie danych oraz zastosowaniom w dziedzinie uczenia maszynowego. Jej głównym celem jest dostarczanie elastycznych i efektywnych narzędzi do modelowania i analizy danych, umożliwiających badaczom i praktykom eksplorację oraz rozwiązywanie problemów związanych z klasyfikacją, regresją, klastrowaniem czy redukcją wymiarowości.

Biblioteka ta oferuje różnorodne metody uczenia maszynowego, takie jak algorytm k-najbliższych sąsiadów, drzewa decyzyjne, maszyny wektorów nośnych, algorytmy klastrowania oraz wiele innych.

Scikit-learn charakteryzuje się modułową strukturą, umożliwiając użytkownikom składanie różnych komponentów w złożone procesy analizy danych. Ponadto, biblioteka oferuje narzędzia do oceny wydajności modeli, dopasowywanie hiperparametrów, a także wizualizacji wyników.

Biorąc pod uwagę jego funkcje i zastosowanie, Scikit-learn może stanowić podstawowe narzędzie dla osób pracujących w dziedzinie uczenia maszynowego, pozwalając na skuteczną analizę i modelowanie danych w celu odkrywania wzorców, tworzenia prognoz i podejmowania decyzji opartych na danych.

3.1.5 CUDA

Zjednoczona Architektura Przetwarzania na Urządzeniach (ang. *Compute Unified Device Architecture*) w skrócie CUDA to platforma obliczeniowa stworzona przez firmę NVIDIA, specjalnie zaprojektowana dla przyspieszenia obliczeń za pomocą kart graficznych (GPU) w zastosowaniach naukowych, technicznych i obliczeniach równoległych. CUDA umożliwia efektywne wykorzystanie mocy obliczeniowej kart graficznych, które są wyposażone w setki lub nawet tysiące rdzeni obliczeniowych. Platforma CUDA dostarcza zestaw narzędzi programistycznych oraz interfejsów programowania aplikacji, takich jak język programowania CUDA C/C++, który umożliwia programistom tworzenie aplikacji obliczeniowych wykorzystujących zdolności obliczeniowe GPU.

3.2 Opis użytej sieci neuronowej

Celem niniejszej pracy jest przeprowadzenie analizy jakości zastosowania technik głębokiego uczenia w powszechnie występujących problemach, które podlegają obszarowi głębokiego uczenia. Przed rozpoczęciem eksperymentów koniecznym krokiem jest implementacja sieci neuronowej opartej na technikach głębokiego uczenia, której wynikami będą numeryczne reprezentacje punktów danych w przestrzeni metrycznej, w sposób, który zachowuje podstawowe relacje podobieństwa lub niepodobieństwa między tymi punktami danych. Proces trenowania modelu obejmuje wykorzystanie jednej z baz danych, na których już wcześniej przeprowadzano trening licznych sieci neuronowych. Pozwoli to na porównanie osiągnięć modelu implementującego głębokie uczenie metryki z innymi wariantami uczenia maszynowego.

W ramach badań przewidziane jest przeprowadzenie porównawczej analizy różnych funkcji strat charakterystycznych dla głębokiego uczenia metryki. Ewaluacja zostanie przeprowadzona w oparciu o dokładność prognozowanych danych, średnią wartość funkcji straty oraz czas trwania procesu treningu modelu. Celem eksperymentów jest ocena, w jaki sposób modyfikacje funkcji straty oraz liczby epok wpłyną na wymienione wyżej wskaźniki efektywności.

Jako, że jednym z często poruszanych problemów w głębokim uczeniu jest klasyfikacja obrazów [52, 36, 2, 29, 39], to po przeprowadzonej analizie zdecydowano się na skonstruowanie modelu konwolucyjnej sieci neuronowej, gdyż charakteryzuje się ona zdolnością do wydobywania abstrakcyjnych cech z obrazów. Dodatkowo dodanie do modelu podstawowego mechanizmu sieci trójkowej jakim jest wykorzystanie trójek danych: kotwicy, próbki pozytywnej oraz próbki negatywnej, pomiędzy którymi model w czasie uczenia się będzie zmieniał odległości. To podejście wykazuje skuteczność w zadaniach klasyfikacji za pomocą głębokiego uczenia metryki.

3.2.1 Struktura modelu

Projekt modelu wykonano z wykorzystaniem biblioteki PyTorch Metric Learning [48]. Ta biblioteka zapewniła dostęp do już zaimplementowanych funkcji strat, klas odległości, które obliczają parami odległości między osadzeniami (ang. *embeddings*) wejściowymi na przykład **LpDistance**, klas służących do wyboru próbek treningowych (ang. *miner*) jak na przykład **TripletMarginMiner** dobierający kotwicę, próbki pozytywne i próbki negatywne, a także klasy, których zadaniem jest zredukowanie straty pochodzącej z każdego tripletu danych wejściowych do jednej wartości (ang. *reducer*) np. **MeanReducer**.

Opis warstw

- **Warstwa konwolucyjna**: Pierwsza warstwa konwolucyjna, przyjmuje obrazy o pojedynczym kanale i wykorzystuje filtr o rozmiarze 3x3 i generuje wyjście o 32 kanałach. Filtry te przesuwane są po obrazie z przesunięciem o 1, wykrywając lokalne cechy. Wynikiem jest zestaw 32 map cech, które reprezentują przetworzone dane wejściowe.

- **Funkcja aktywacji** ReLU (ang. *Rectified Linear Unit*): Po warstwie konwolucyjnej stosowana jest funkcja aktywacji rektyfikowana jednostka liniowa w skrócie ReLU, która wprowadza nieliniowość, eliminując wartości ujemne i wzmacniając pozytywne aktywacje. Wiele problemów związanych z analizą obrazów i danych przestrzennych jest nieliniowo separowalnych. Oznacza to, że nie można ich skutecznie rozwiązać za pomocą jedynie operacji liniowych. Funkcje aktywacji, takie jak ReLU wprowadzają nieliniowość, co pozwala na tworzenie bardziej skomplikowanych granic decyzyjnych.

- **Druga warstwa konwolucyjna**: Druga warstwa konwolucyjna przetwarza wyjście z poprzedniej warstwy wykorzystując filtr o rozmiarze 3x3 przesuwający się o 1, generując zestaw 64 map cech. Każda warstwa konwolucyjna wyodrębnia określone cechy z obrazu wejściowego. Druga warstwa konwolucyjna może analizować cechy wyodrębnione przez pierwszą warstwę, co pozwala na uczenie bardziej skomplikowanych i abstrakcyjnych cech, które są kombinacją cech niższego poziomu. Każda kolejna warstwa konwolucyjna działa na coraz bardziej abstrakcyjnych poziomach cech. To pozwala modelowi na automatyczne uczenie się cech o rosnącej złożoności, co jest kluczowe w przypadku zadań związanych z obrazami.

- **Warstwa łączenia**: Po drugiej warstwie konwolucyjnej stosowana jest operacja maksymalnego łączenia (ang. *max pooling*) z jądrem o rozmiarze 2x2, co zmniejsza rozmiar przetwarzanych map cech i jednocześnie wydobywa istotne cechy z obszarów lokalnych. Jest kilka powodów, dla których zdecydowano się użyć warstwy łączenia. Po pierwsze warstwa łączenia zmniejsza rozmiar mapy cech, co pomaga w ograniczeniu liczby parametrów i obliczeń w sieci, co z kolei może poprawić wydajność i

zmniejszyć ryzyko nadmiernego dopasowania. Po drugie warstwa łączenia sprawia, że model staje się bardziej odporny na małe przesunięcia w obrazach. Dzięki temu rozpoznawanie wzorców nie jest zbyt czułe na dokładne położenie cechy na obrazie.

- **Warstwa porzucenia**:Następna umieszczona jest warstwa porzucenia (ang. *dropout*), która losowo wyłącza niektóre neurony zerując kanały w celu zapobieżenia przeuczeniu modelu. To pomaga w generalizacji i zwiększa zdolność modelu do radzenia sobie z nowymi danymi.

- **Warstwa liniowa**: Następnie, dane są spłaszczane do wektora i przekazywane do warstwy liniowej z 9216 wejściami i 128 wyjściami. Wagi tej warstwy są dostosowywane w procesie uczenia.

- **Normalizacja partii**: Po warstwie liniowej, zastosowana jest normalizacja partii (ang. *batch*) w celu standaryzacji rozkładu aktywacji, co pomaga w stabilizacji procesu uczenia i przyspiesza zbieżność.

- **Warstwa wyjściowa - Normalizacja L2**: Ostatecznie, przed zwróceniem wyników, wyjście z warstwy liniowej jest normalizowane w oparciu o normę L2 w celu utworzenia osadzeń o jednostkowej długości, dzięki czemu uzyskuje się rozwiązania w jednostkowej hipersferze, co sprawi, że wszystkie wyniki zostaną zamknięte w cyklicznej przestrzeni.

3.2.2 Funkcja straty

Podczas procesu uczenia maszynowego, sieć neuronowa jest dostosowywana w celu minimalizacji funkcji straty. Im niższa wartość funkcji straty, tym lepszy jest model w dopasowaniu się do danych treningowych i w generowaniu dokładniejszych wyników. Dobór odpowiedniej funkcji straty jest kluczowym elementem, który może wpłynąć na jakość modelu i został wybrany jako jeden z zmiennych parametrów, których wpływ zostanie przetestowany w badaniach.

Podczas badań zostaną zastosowane poniższe funkcje straty:

- Strata kątowa (ang. *Angular Loss*)
- Strata kołowa (ang. *Circle Loss*)
- Strata kontrastowa (ang. *Contrastive Loss*)
- Strata miękki maks z dużym marginesem (ang. *Large-Margin Softmax Loss*)
- Podniesiona strata strukturalna (ang. *Lifted Structured Loss*)
- Strata marginesu (ang. *Margin Loss*)

- Strata analizy składników sąsiedztwa (ang. *Neighbourhood Components Analysis Loss*)
- Znormalizowana strata miękki maks (ang. *Normalized Softmax Loss*)
- Strata N par (ang. *N Pairs Loss*)
- Strata pełnomocnika kotwicy (ang. *Proxy Anchor Loss*)
- Strata marginesu trójkowego (ang. *Triplet Margin Loss*)

3.2.3 Parametry treningu

W tym fragmencie omówione zostaną istotne parametry oraz techniki używane podczas etapu treningu modelu. Dokładne określenie tych parametrów jest niezwykle istotne ze względu na ewentualne odtworzenie eksperymentu przez innych badaczy, a także dla pełnego zrozumienia metodyki przeprowadzanych badań. Wartości tych parametrów mają wpływ na efektywność i tempo uczenia modelu, a także na jego końcową jakość. Szczegółowe informacje na temat wykorzystanych w badaniach stałych i zmiennych hiperparametrów modelu znajdują się poniżej.

Optymalizator

Do optymalizacji modelu wykorzystano optymalizator Adam - Adaptacyjna estymacja momentu (ang. *Adaptive Moment Estimation*). Głównym celem tego algorytmu jest dostosowywanie współczynnika uczenia (ang. *learning rate*) dla każdego parametru modelu w oparciu o wcześniejsze historie gradientów.

Rozmiar partii

Rozmiar partii (ang. *batch size*) określa liczbę przykładów treningowych, które są przetwarzane jednocześnie przez model podczas każdej iteracji w procesie uczenia. Innymi słowy, rozmiar partii definiuje, ile danych wejściowych jest przekazywanych do modelu jednocześnie, zanim nastąpi aktualizacja wag na podstawie obliczonych gradientów. Modele był trenowany na partii o rozmiarze 64.

Współczynnik uczenia

Współczynnik uczenia (ang. *learning rate*) jest jednym z kluczowych hiperparametrów w procesie uczenia modeli opartych na optymalizacji gradientowej, takich jak sieci neuronowe. Określa on krok, o którym algorytm optymalizacji przesuwa się w przestrzeni parametrów modelu podczas aktualizacji wag w każdej iteracji. Modele był trenowany z użyciem współczynnika uczenia o wartości $0,001$.

Liczba epok

Liczba epok (ang. *epoch number*) oznacza liczbę pełnych przejść przez zbiór danych treningowych. Każda epoka reprezentuje jedno pełne przejście. Jest to bardzo ważny hiperparametr który wpływa na to, jak długo model jest trenowany co może mieć kluczowy wpływ na jakość wytrenowanego modelu. W przeprowadzonych eksperymentach liczba epok jest parametrem zmiennym i wynosi: 1, 5, 10 ,20 ,50.

Warunki sprzętowe

Model był trenowany na karcie graficznej NVIDIA GeForce 1060. Podana karta graficzna ma poniższe specyfikacje techniczne:

Architektura: Pascal

Liczba rdzeni CUDA: 1280

Zegar bazowy: Zazwyczaj około 1506 MHz

Zegar w trybie Boost: Zazwyczaj około 1708 MHz

Pamięć VRAM: Zazwyczaj 6 GB GDDR5

Szerokość szyny pamięci: 192-bit

Szybkość pamięci: Zazwyczaj 8 Gbps

Wydajność pamięci: Zazwyczaj 192 GB/s

Interfejs pamięci: PCI Express 3.0 x16

Wsparcie dla DirectX: 12

Wsparcie dla OpenGL: 4.5

Wsparcie dla Vulkan: Tak

Wsparcie dla CUDA: Tak

Wsparcie dla NVIDIA G-SYNC: Tak

Wsparcie dla NVIDIA GameStream: Tak

Wsparcie dla VR Ready: Tak

Liczba złącz: Zazwyczaj 3x DisplayPort, 1x HDMI, 1x DVI

Zalecany zasilacz: Zazwyczaj 400 W

Wymagane złącza zasilania: Zazwyczaj 1x 6-pin

Rozmiar karty: Zazwyczaj dwuslotowy

Rozdział 4

Badania

4.1 Metodyka badań

4.1.1 Przeprowadzone badania

W celu zebrania danych badawczych przeprowadzono kilka kroków poza wytrenowaniem modelu. Po pierwsze zebrano wszystkie wyjścia sieci neuronowej i zapisano je w liście. Te działania były potrzebne by przygotować klasyfikator KNN na podstawie zbioru danych treningowych, które w tym wypadku były pozycjami na hipersferze zwróconymi przez uczący się model sieci neuronowej.

Powyższe kroki musiały zostać przeprowadzone by KNN mógł znajdować najbliższych sąsiadów w nowo utworzonej metryce podczas ewaluacji. W trakcie testowania modelu zapisywane są wyjścia z sieci i odpowiadające im etykiety danych. Na koniec obliczana jest dokładność modelu za pomocą zestawienia przewidzianych przez KNN etykiet z zapisanych w czasie ewaluacji wyjść z ich faktycznymi etykietami.

Dodatkowo w trakcie trenowania modelu w każdej iteracji zapisywana jest wartość straty, na podstawie której tworzony jest wykres przedstawiający zmianę średniej wartości straty w czasie.

4.1.2 Rygor badawczy

W przeprowadzonych badaniach został wprowadzony rygor, którego celem było zapewnienie powtarzalności i dokładności wyników. Aby osiągnąć zamierzony cel zostały wprowadzone poniższe działania:

- Dane treningowe i testowe są takie same dla każdego uczonego modelu sieci neuronowej oraz dla każdego eksperymentu. Dane treningowe pomimo, że są tasowane, to za każdym uruchomieniem przemieszają się w tej samej kolejności dzięki ustawianemu na początku ziarnu (ang. *seed*). Ziarno jest kodem, który wpływa na to w jaki

sposób PyTorch generuje wartości losowe, co przekłada się między innymi na tasowanie wartości treningowych. Dzięki niemu za każdym uruchomieniem programu do uczenia modelu uzyska się dane treningowe ustawione w tej samej kolejności co umożliwia odtworzenie wyników.

- Funkcja straty jest jednym z hiperparametrów, którego wpływ zbadano w eksperymentach, ale potrzebne było wprowadzenie rygoru odnośnie parametrów tej funkcji, ze względu na zbyt dużą ilość parametrów, które mogły wpłynąć na ostateczny wynik. Niektóre funkcje straty posiadają swoje zmienne jak na przykład wielkość marginesu czy klasy redukcji. Ze względu na dużą liczbę potencjalnych parametrów zdecydowano się zawsze używać domyślnych wartości. Wpływ parametrów poszczególnych funkcji straty pozostawia możliwość przeprowadzenia badań o tej tematyce w przyszłości.

4.1.3 Opis stanowiska badawczego

W celu trenowania i testowania modeli sieci neuronowej została utworzona aplikacja, która implementuje opisany wyżej model, przeprowadza pętlę uczenia oraz ewaluację, a także umożliwia zmianę hiperparametrów. W celu ewaluacji modelu stosuje się algorytm KNN - k najbliższych sąsiadów (ang. *K Nearest Neighbors*). Hiperparametry modelu zostały dobrane drogą eksperymentalną.

Test, którego wynik zadecydował o doborze stałych parametrów wyglądał w poniższy sposób:

1. Ustawiono parametry, które będą ulegać zmianie w czasie docelowych badań. Wybrana liczba epok była równa 10 a jako funkcję straty wybrano stratę marginesu trójkowego z domyślnymi wartościami parametrów.

2. Przeprowadzono serię testów, w których zmieniano wartość parametru *batchsize* oznaczającego rozmiar partii kolejno na: 32, 64, 128, 512, 1024 i do następnej części testu dobrano wartość 64, gdyż na niej model osiągnął największą dokładność równą $99,03$.

3. Następnie przeprowadzono kolejne eksperymenty, w których zmieniano wartość parametru *learningrate* oznaczającego współczynnik uczenia kolejno na: $0,01$, $0,001$, $0,0001$ i do następnej części testu dobrano wartość $0,001$, gdyż na niej model osiągnął największą dokładność równą $99,11$.

4. ostatnim zmienianym parametrem było k czyli wartość oznaczającą ile sąsiadów ma wziąć pod uwagę algorytm KNN podczas klasyfikacji. Podczas testu sprawdzono wartości: 1, 3 i 5. Do docelowych badań wybrano wartość 3, gdyż na niej model osiągnął największą dokładność równą $99,13$.

4.2 Zbiory danych

4.2.1 Opis danych

MNIST

Jako zbiór treningowy i testowy użyto bazy danych MNIST [47] (ang. *Modified National Institute of Standards and Technology database*). To popularny zbiór danych używany w dziedzinie uczenia maszynowego do testowania i oceny algorytmów rozpoznawania obrazów.

Oryginalna baza MNIST zawiera zestaw 60 000 obrazów treningowych oraz 10 000 obrazów testowych, które zostały zebrane z próbek pisanych ręcznie przez pracowników amerykańskiego Narodowego Instytutu Standaryzacji i Technologii. Obrazy te zostały następnie przetworzone i przeskalowane, aby były o rozmiarze 28x28 pikseli i w odcieniach szarości.

Każdy obraz w bazie MNIST przedstawia jedną cyfrę od 0 do 9, a etykieta przypisana do każdego obrazu wskazuje, która cyfra jest przedstawiona. Baza ta była wykorzystywana jako zestaw testowy i benchmark w wielu badaniach naukowych oraz jako narzędzie dydaktyczne do nauki i demonstracji algorytmów klasyfikacji, sieci neuronowych i innych technik związanych z analizą obrazów.

Baza MNIST była jednym z pierwszych i najbardziej rozpoznawalnych zbiorów danych w dziedzinie uczenia maszynowego. Pomogła w stworzeniu podwalin dla nowych algorytmów i modeli, a także w popularyzacji dziedziny rozpoznawania obrazów i głębokiego uczenia. Chociaż MNIST jest stosunkowo prostym zestawem danych, wciąż jest używany jako punkt wyjścia do nauki i eksperymentowania z nowymi technikami uczenia maszynowego.

Ta baza danych została wybrana ze względu na jej popularność, co pozwoli zbadać jak dokładna jest sieć neuronowa oparta na głębokim uczeniu metryki w porównaniu do innych typów uczenia maszynowego i algorytmów służących do klasyfikacji obrazów. Dużą zaletą doboru popularnej bazy danych jest łatwa reprodukowalność wyników.

4.3 Wyniki badania wpływu liczby epok

4.3.1 Strata kątowa

W tej części badań do nauki modelu użyto kątowej funkcji straty [48]. Poniżej zaprezentowano zmianę wartości straty podczas treningu widoczną na Rysunku 4.1 oraz dokładność wyuczonego modelu w zależności od liczby epok, która została zaprezentowana na Rysunku 4.2.

Zmiana wartości straty podczas treningu

Rysunek 4.1: Wykres średniej wartość straty dla straty kątowej w zależności od liczby epok)

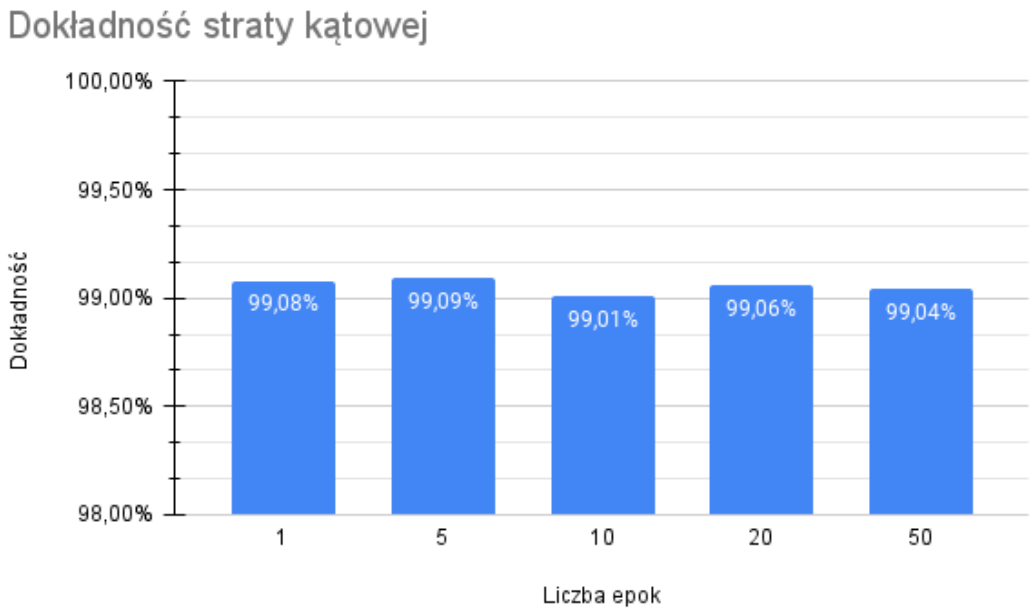
Z analizy wykresu na Rysunku 4.1 wynika, że funkcja straty uzyskiwała wartości średnie z zakresu pomiędzy ponad $1,89$ a prawie $1,95$. Nie zaobserwowano spodziewanej tendencji malejącej, co sugeruje, że średnia wartość straty w modelu podczas procesu treningu nie wykazuje zależności od liczby epok ze względu na niewielką i nierówną zmianę dokładności przy rosnącej liczbie epok..

Dokładność wyuczonego modelu

Z przeprowadzonej analizy wykresu widocznego na Rysunku 4.2 można wywnioskować, że dokładność uzyskiwana przez trenowany model charakteryzuje się wysokimi rezultatami, mieszczącymi się w zakresie od $99,01\%$ do $99,09\%$. Nie stwierdzono istotnego wpływu liczby epok na osiąganą dokładność modelu ze względu na niewielką i nierówną zmianę dokładności przy rosnącej liczbie epok.

4.3.2 Strata kołowa

W ramach prowadzonych badań nad nauką modelu, zastosowano funkcję straty kołowej, jaką opisano w źródle [48]. Przedstawione poniżej jest zmienność wartości tej funkcji w trakcie procesu treningowego, zobrazowana na wykresie zaprezentowanym na Rysunku 4.3. Również zaobserwowano zmiany w osiąganej dokładności w zależności od liczby epok, co zilustrowano w Rysunku 4.4.



Rysunek 4.2: Wykres dokładności wyuczonego modelu dla straty kątowej w zależności od liczby epok)

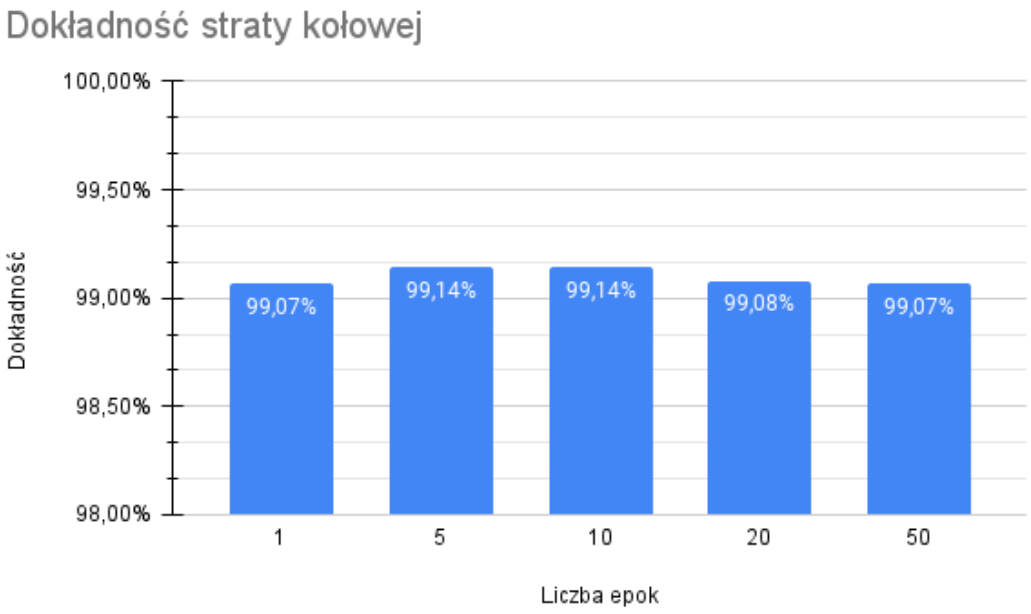
Zmiana wartości straty podczas treningu

Rysunek 4.3: Wykres średniej wartość straty dla straty kołowej w zależności od liczby epok)

Z wyników przedstawionych na wykresie 4.3 można wywnioskować, że funkcja straty przyjmowała średnie wartości mieszczące się w zakresie od ponad $12,5$ do prawie $13,2$. Obserwowana jest subtelna tendencja malejąca, szczególnie po zrealizowaniu 5 epok, jed-

nakże to w przypadku treningu trwającego 5 i 10 epok osiągnięto najwyższe wartości tej funkcji straty.

Dokładność wyuczonego modelu



Rysunek 4.4: Wykres dokładności wyuczonego modelu dla straty kołowej w zależności od liczby epok)

Wyniki przedstawione na wykresie z Rysunku 4.4 pozwala stwierdzić, że dokładność osiągana przez trenowany model wykazuje wysokie rezultaty, mieszczące się w przedziale od $99,07\%$ do $99,14\%$. Nie zanotowano istotnego wpływu liczby epok na osiągane wyniki dokładności modelu. Niemniej jednak, zaobserwowano nietypową relację między średnią wartością funkcji straty a dokładnością. Wartości dokładności były najlepsze wtedy, gdy model osiągał najwyższe średnie wartości straty w określonych liczbach epok. Jako, że najlepsze wyniki zostały osiągnięte dla liczby epok 5 i 10, a charakterystyka wykresu z Rysunku 4.4 od tego momentu jest malejąca, to można mówić o przeuczeniu modelu.

4.3.3 Strata kontrastowa

Badania objęły zastosowanie straty kontrastowej omówionej w dokumentacji [48]. Prezentowany poniżej wykres w Rysunku 4.5 ukazuje ewolucję wartości tej funkcji w trakcie procesu treningu. Dodatkowo, ilustracja w Rysunku 4.6 przedstawia zależność pomiędzy liczbą epok a osiągniętą dokładnością wytrenowanego modelu.

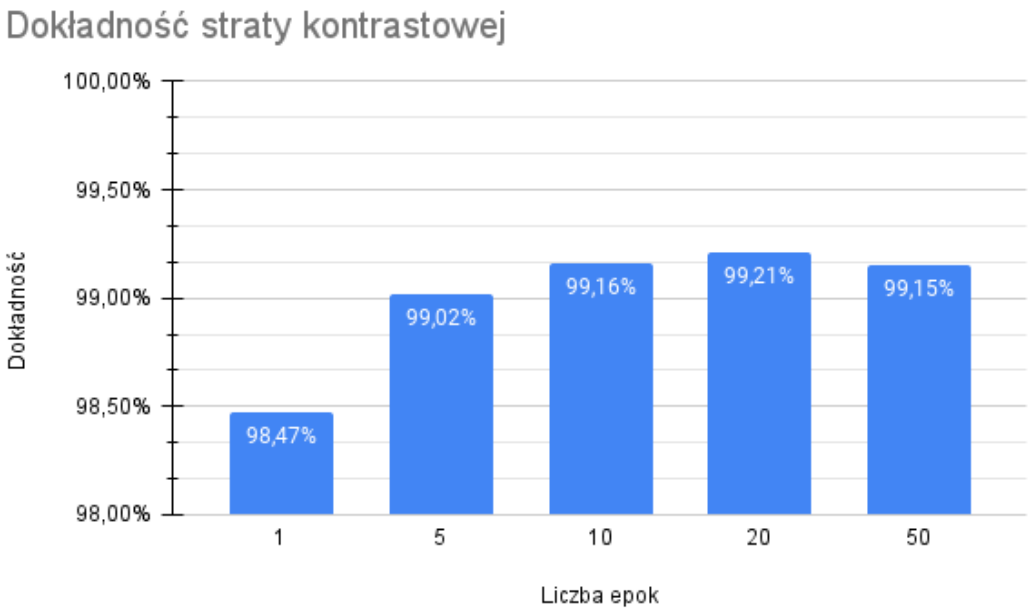
Rysunek 4.5: Wykres średniej wartość straty dla straty kontrastowej w zależności od liczby epok

Zmiana wartości straty podczas treningu

Analizując wykres widoczny na Rysunku 4.5, można wysnuć wniosek, że średnie wartości funkcji straty mieściły się w zakresie od ponad $0,78$ do prawie $0,88$. Widoczna jest tendencja malejąca, która przypomina charakterystykę wykresu malejącej funkcji logarytmicznej, które świadczy o dobrym dopasowaniu modelu. To oznacza, że model jest coraz bliższy optymalnego dopasowania do danych.

Dokładność wyuczonego modelu

Analiza przeprowadzona na podstawie wykresu z Rysunku 4.6 wskazuje, że dokładność osiągana przez trenowany model wykazuje wysokie rezultaty, mieszczące się w przedziale od $98,47\%$ do $99,21\%$. Zauważono wpływ liczby epok na osiągane wyniki dokładności modelu. Poprawa dokładności modelu wykazuje charakter przypominający rosnącą funkcję logarytmiczną, z wyjątkiem ostatniego modelu trenowanego przy 50 epokach, gdzie zaobserwowano niewielki spadek dokładności względem poprzednika uczącego się przy liczbie epok równej 20. Wynika to z coraz lepszego dopasowania modelu do danych, a w przypadku ostatniego testu dla 50 epok można stwierdzić przeuczenie sieci neuronowej co obrazuje się przez spadek dokładności.



Rysunek 4.6: Wykres dokładności wyuczonego modelu dla straty kontrastowej w zależności od liczby epok)

4.3.4 Strata miękki maks z dużym marginesem

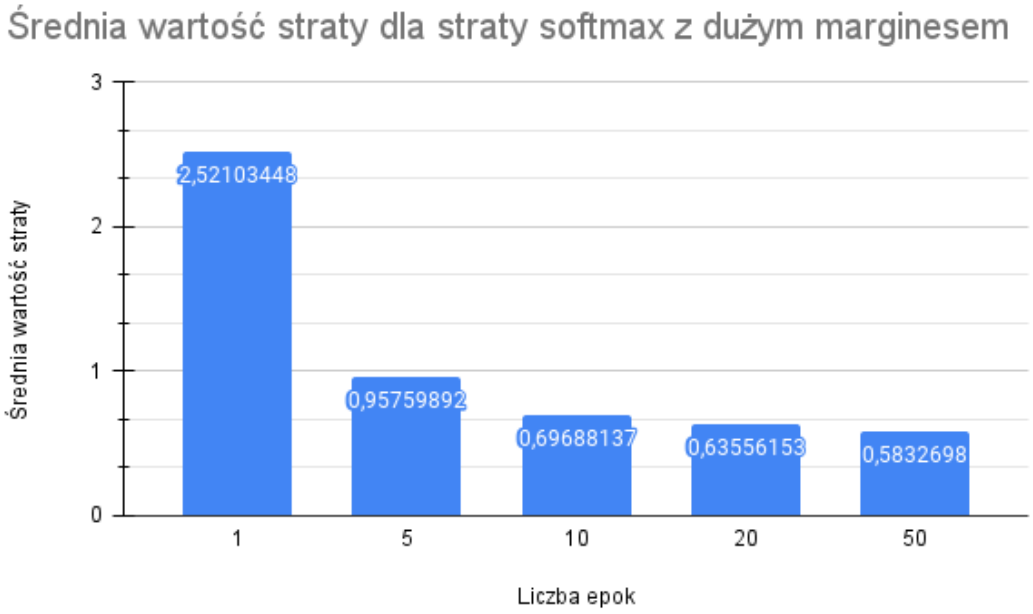
Przy realizacji badań skorzystano ze straty miękki maks z dużym marginesem [48]. Na Rysunku 4.7 zobrazowano dynamiczne wahania wartości straty podczas procesu treningowego. Ilustracja w Rysunku 4.8 prezentuje natomiast, jak liczba epok wpływa na jakość osiąganą przez wyuczoną reprezentację modelu.

Zmiana wartości straty podczas treningu

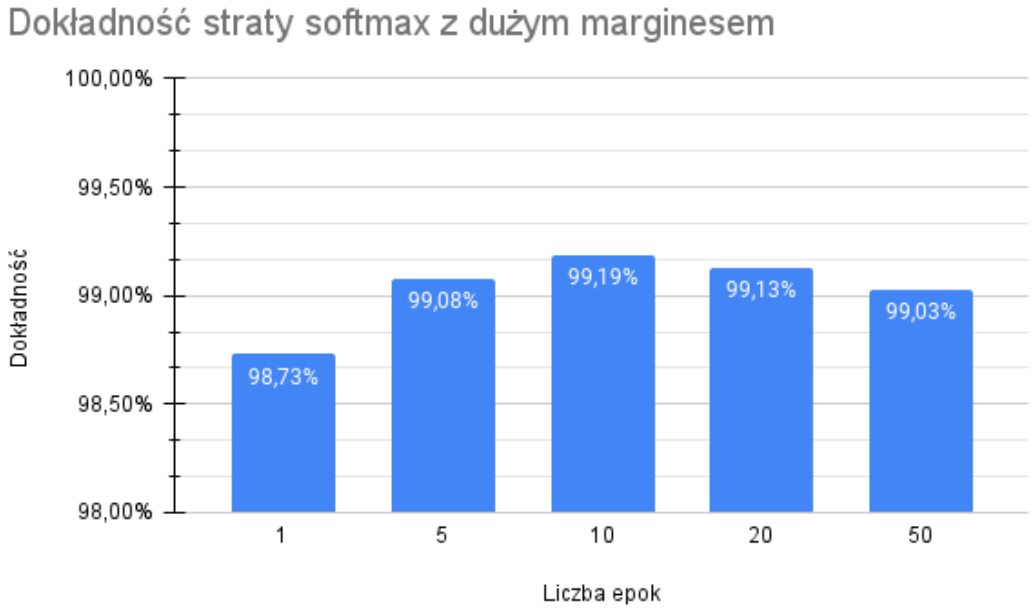
Z analizy wykresu oznaczonego jako 4.7 wynika, że funkcja straty osiągała średnie wartości mieszczące się w zakresie od ponad $0,58$ do prawie $2,522$. Zauważalna jest istotna tendencja malejąca, przypominająca charakterystykę wykresu funkcji logarytmicznej. Wielkość skoku można zaobserwować, biorąc pod uwagę skrajne wartości straty uzyskane dla 1 i 50 epok.

Dokładność wyuczonego modelu

Analizując wykres oznaczony jako 4.8, można stwierdzić, że dokładność osiągana przez trenowany model prezentuje znakomite wyniki, mieszczące się w przedziale od $98,73\%$ do $99,19\%$. Dostrzeżono wpływ liczby epok na osiąganą dokładność modelu. Wydaje się, że poprawa dokładności modelu nie jest wprost związana z wartościami średniej straty. Obserwuje się, że najlepsza dokładność została osiągnięta dla 10 epok, podczas gdy najniższą średnią wartość straty zanotowano dla 50 epok treningu.



Rysunek 4.7: Wykres średniej wartość straty dla straty miękki maks z dużym marginesem w zależności od liczby epok)



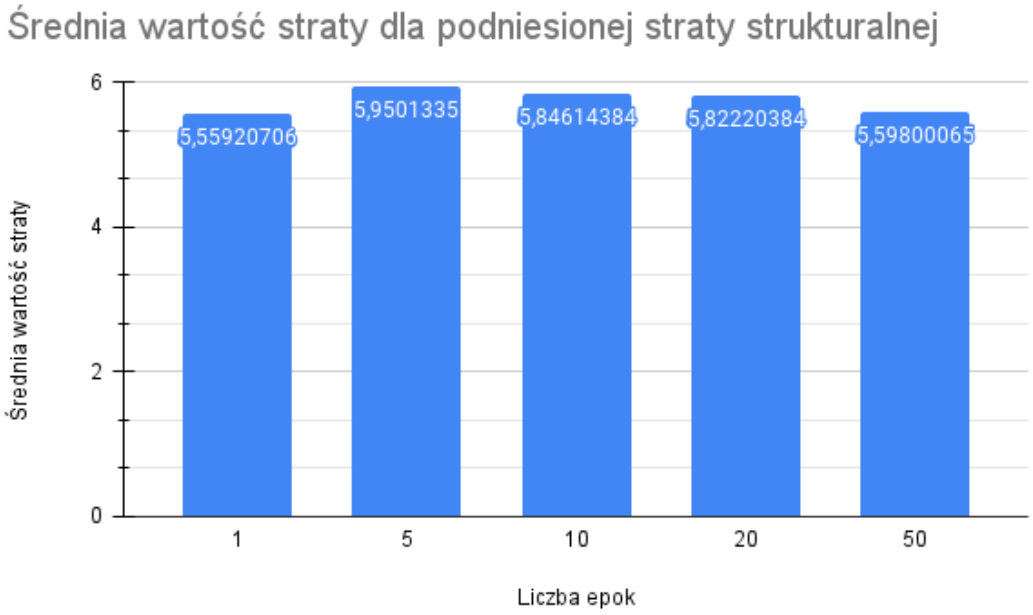
Rysunek 4.8: Wykres dokładności wyuczonego modelu dla straty miękki maks z dużym marginesem w zależności od liczby epok)

4.3.5 Podniesiona strata strukturalna

W ramach przeprowadzonych badań naukowych wykorzystano mechanizm podniesionej straty strukturalnej [48]. Na wykresie w Rysunku 4.9 przedstawiono fluktuacje wartości straty w trakcie procesu treningowego. W Rysunku 4.10 ukazano relację między liczbą

epok a zdolnością modelu do osiągania poprawnych wyników.

Zmiana wartości straty podczas treningu



Rysunek 4.9: Wykres średniej wartość straty dla podniesionej straty strukturalnej w zależności od liczby epok)

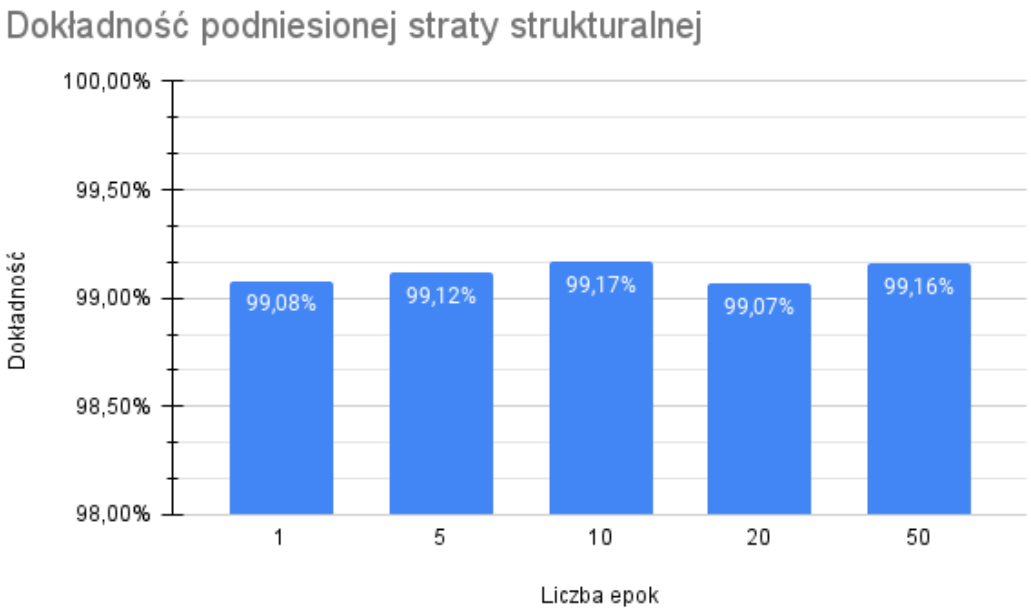
Analiza wykresu oznaczonego jako 4.9 wskazuje, że funkcja straty przyjmowała średnie wartości z zakresu od ponad $5,559$ do prawie $5,951$. Nie stwierdzono obecności tendencji malejącej. Na tej podstawie można wywnioskować, że średnia wartość straty w modelu podczas procesu treningu nie wykazuje zależności od liczby epok, ponieważ brak jest zauważalnych zmian względem tego parametru.

Dokładność wyuczonego modelu

Z analizy wykresu widocznego na Rysunku 4.10 można wywnioskować, że dokładność osiągana przez model podczas treningu charakteryzuje się wysokimi rezultatami, mieszczącymi się w przedziale od $99,07\%$ do $99,17\%$. Nie odnotowano istotnego wpływu liczby epok na osiągane wyniki dokładności modelu.

4.3.6 Strata marginesu

W toku prowadzonych badań zastosowano podejście oparte na wykorzystaniu straty marginesu [48]. Wariacje wartości straty w trakcie procesu treningu zilustrowano na Rysunku 4.11. Dodatkowo, relacja pomiędzy liczbą epok, a efektywnością modelu została zaprezentowana na Rysunku 4.12.



Rysunek 4.10: Wykres dokładności wyuczonego modelu dla straty strukturalnej w zależności od liczby epok)

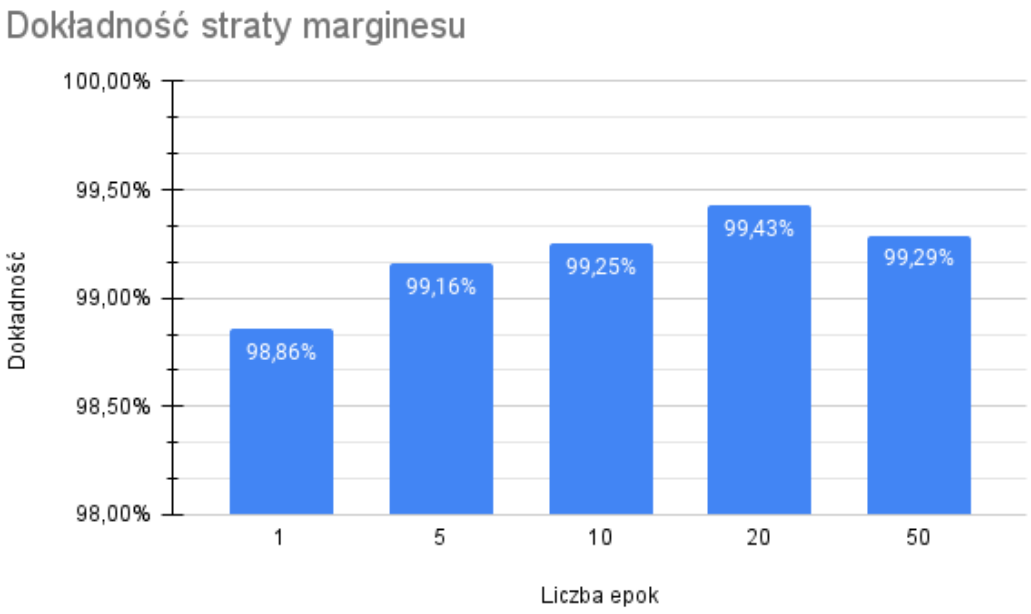
Zmiana wartości straty podczas treningu

Rysunek 4.11: Wykres średniej wartość straty dla straty marginesu w zależności od liczby epok)

Analiza przedstawiona na wykresie z Rysunku 4.11 wskazuje, że funkcja straty osiągała średnie wartości mieszczące się w zakresie od ponad 0,177 do prawie 0,33. Obserwuje się wyraźną tendencję malejącą, przypominającą charakterystykę wykresu funkcji logaryt-

micznej, co jednoznacznie wskazuje na pozytywny wpływ liczby epok na średnią wartość funkcji straty.

Dokładność wyuczonego modelu



Rysunek 4.12: Wykres dokładności wyuczonego modelu dla straty marginesu w zależności od liczby epok)

Analiza przeprowadzona na podstawie wykresu widocznego na Rysunku 4.12 wskazuje, że dokładność osiągana przez trenowany model wykazuje znakomite rezultaty, mieszczące się w zakresie od $98,86\%$ do $99,43\%$. Dostrzeżono wpływ liczby epok na uzyskiwane wyniki dokładności modelu. Obserwowana poprawa dokładności modelu ma charakter przypominający rosnącą funkcję logarytmiczną, z wyjątkiem ostatniego modelu trenowanego przy 50 epokach, gdzie zaobserwowano spadek dokładności, co nie pokrywa się z ciągle malejącą wartością średnią funkcji straty. W przypadku treningu dla 50 epok zjawiska można mówić o zbyt długim czasie nauki co spowodowało przeuczenie modelu.

4.3.7 Strata analizy składników sąsiedztwa

Badania obejmowały wykorzystanie strategii opartej na funkcji straty analizy składników sąsiedztwa [48]. Na wykresie w Rysunku 4.13 zaprezentowano oscylacje wartości straty w trakcie treningu. Ponadto, wpływ liczby epok na efektywność osiąganą przez model ukazano na Rysunku 4.14.

Rysunek 4.13: Wykres średniej wartość straty dla straty analizy składników sąsiedztwa w zależności od liczby epok)

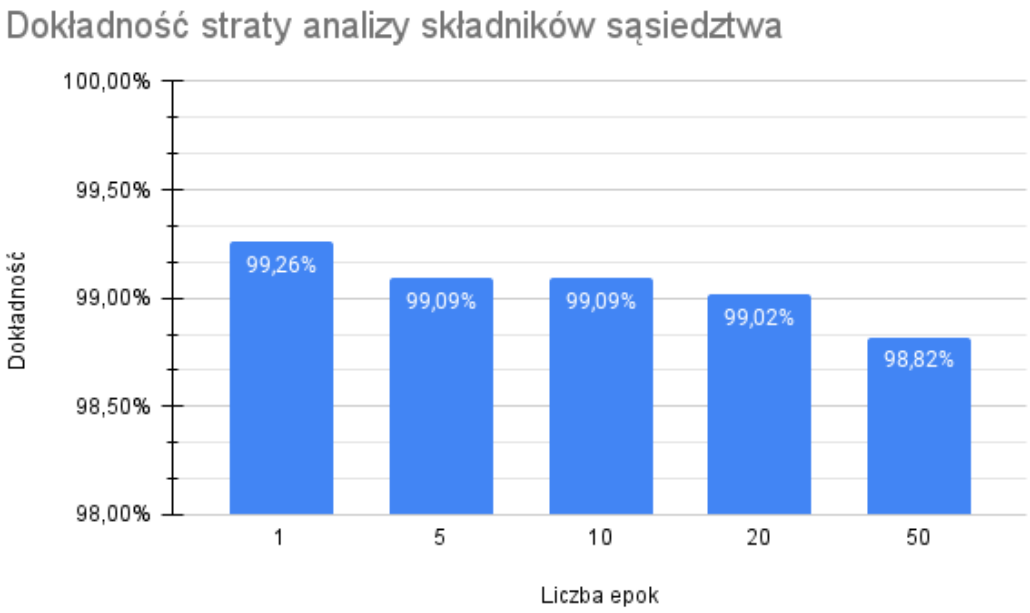
Zmiana wartości straty podczas treningu

Analiza wykresu widocznego na Rysunku 4.13 ukazuje, że średnie wartości funkcji straty kształtują się w przedziale od ponad $0,28$ do prawie $0,335$. Dostrzegalna jest wyraźna liniowa tendencja malejąca, co wskazuje na istnienie zależności między liczbą epok treningu a wartością średniej straty.

Dokładność wyuczonego modelu

Z analizy wykresu z Rysunku 4.14 wynika, że dokładność osiągana przez trenowany model wykazuje znakomite rezultaty, mieszczące się w zakresie od $98,82\%$ do $99,26\%$. Stwierdzono wpływ liczby epok na uzyskiwane wyniki dokładności modelu. Obserwowana zależność dokładności modelu od liczby epok przypomina malejącą funkcję liniową. Można zaobserwować, że wraz ze spadkiem wartości średniej straty maleje również dokładność modelu.

W tej części badań do nauki modelu znormalizowaną stratę miękki maks [48]. Poniżej zaprezentowano zmianę wartości straty podczas treningu widoczną na Rysunku 4.15 oraz dokładność wyuczonego modelu w zależności od liczby epok, która została zaprezentowana na Rysunku 4.16.



Rysunek 4.14: Wykres dokładności wyuczonego modelu dla straty analizy składników sąsiedztwa w zależności od liczby epok)

Rysunek 4.15: Wykres średniej wartość straty dla znormalizowanej straty miękki maks w zależności od liczby epok)

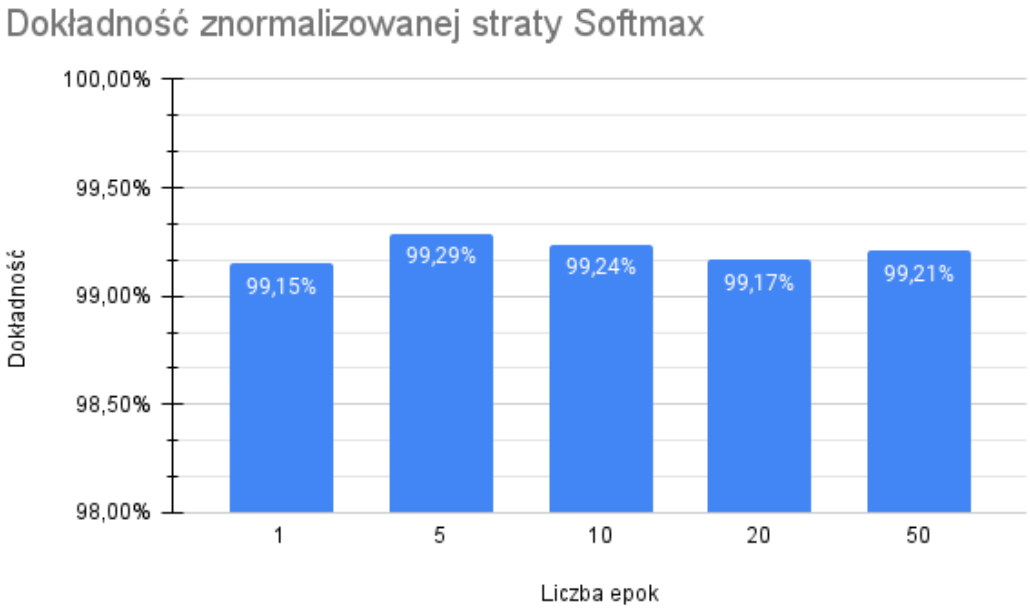
4.3.8 Znormalizowana strata miękki maks

Zmiana wartości straty podczas treningu

Analiza przeprowadzona na podstawie wykresu z Rysunku 4.15 wskazuje, że średnie wartości funkcji straty utrzymywały się w zakresie od ponad $0,019$ do prawie $0,74$. Ob-

serwowana jest wyraźna zależność między liczbą epok a średnią wartością straty. Pierwszy model, uczony przez 1 epokę, charakteryzował się największą wartością straty, natomiast kolejny model uzyskał prawie najlepsze wyniki. Po wykonaniu 10 epok, można zaobserwować ponowny wzrost średniej wartości funkcji straty, co może sugerować przeuczenie modelu.

Dokładność wyuczonego modelu



Rysunek 4.16: Wykres dokładności wyuczonego modelu dla znormalizowanej straty miękki maks w zależności od liczby epok)

Analiza przeprowadzona na podstawie wykresu oznaczonego jako 4.16 wskazuje, że dokładność osiągana przez trenowany model prezentuje znakomite wyniki, mieszczące się w zakresie od $99,15\%$ do$99,29\%$. Nie odnotowano istotnego wpływu liczby epok na uzyskiwane wyniki dokładności modelu. Dodatkowo, nie zauważono związku pomiędzy dokładnością modelu a średnią wartością funkcji straty.

W ramach prowadzonych badań nad nauką modelu, zastosowano funkcję straty N par [48]. Przedstawione poniżej jest zmienność wartości tej funkcji w trakcie procesu treningowego, zobrazowana na wykresie zaprezentowanym w Rysunku 4.17. Również zaobserwowano zmiany w osiąganej dokładności w zależności od liczby epok, co zilustrowano w Rysunku 4.18.

Rysunek 4.17: Wykres średniej wartość straty dla straty N par w zależności od liczby epok)

4.3.9 Strata N par

Zmiana wartości straty podczas treningu

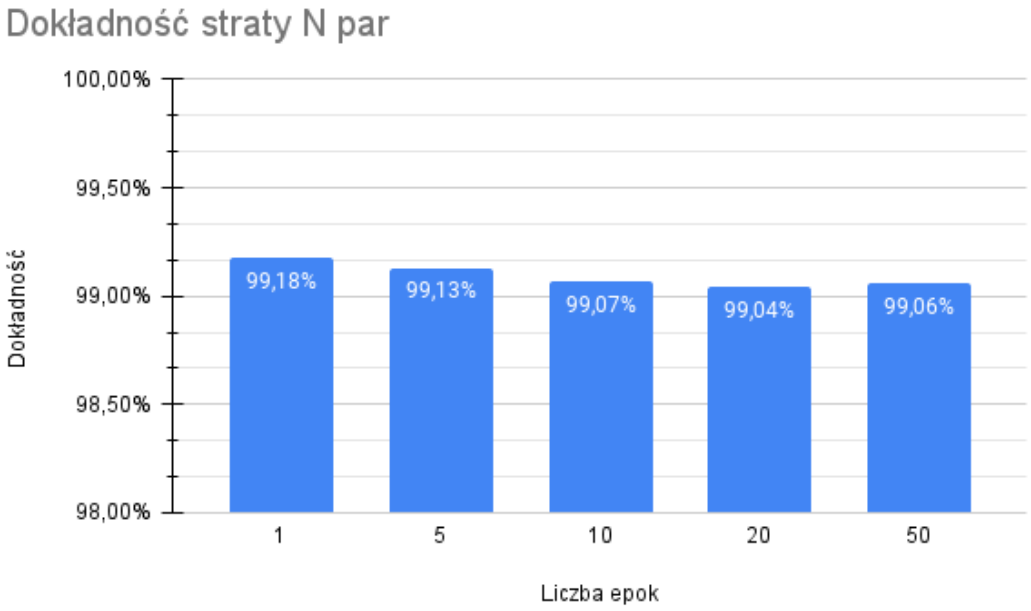
Wynik analizy wykresu z Rysunku 4.17 ukazuje, że funkcja straty przybierała średnie wartości z zakresu od ponad $0,659$ do prawie $0,676$. Nie obserwuje się spodziewanej tendencji malejącej. W rezultacie można wywnioskować, że średnia strata w trenowanym modelu nie wykazuje zależności od liczby epok, gdyż brak jest zauważalnej zmiany w tym parametrze.

Dokładność wyuczonego modelu

Z analizy przedstawionej na wykresie widocznego na Rysunku 4.18 wynika, że dokładność osiągana przez trenowany model prezentuje wysokie osiągnięcia, mieszczące się w przedziale od $99,04\%$ do $99,18\%$. Zaobserwowano delikatny wpływ liczby epok na uzyskiwane wyniki dokładności modelu. Wraz z wzrostem liczby epok dokładność modelu maleje.

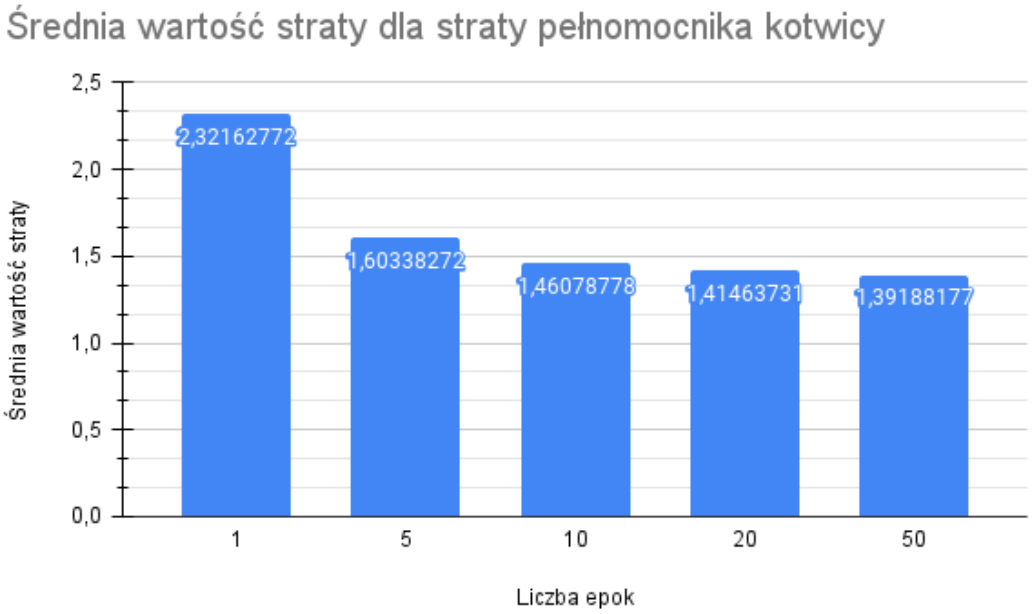
4.3.10 Strata pełnomocnika kotwicy

W toku przeprowadzanych badań zastosowano funkcję straty pełnomocnika kotwicy [48]. Zmienność wartości tej funkcji straty w trakcie treningu została przedstawiona na wykresie w Rysunku 4.19. Dodatkowo, związki pomiędzy liczbą epok a efektywnością osiąganą przez model ukazano na Rysunku 4.20.



Rysunek 4.18: Wykres dokładności wyuczonego modelu dla straty N par w zależności od liczby epok)

Zmiana wartości straty podczas treningu

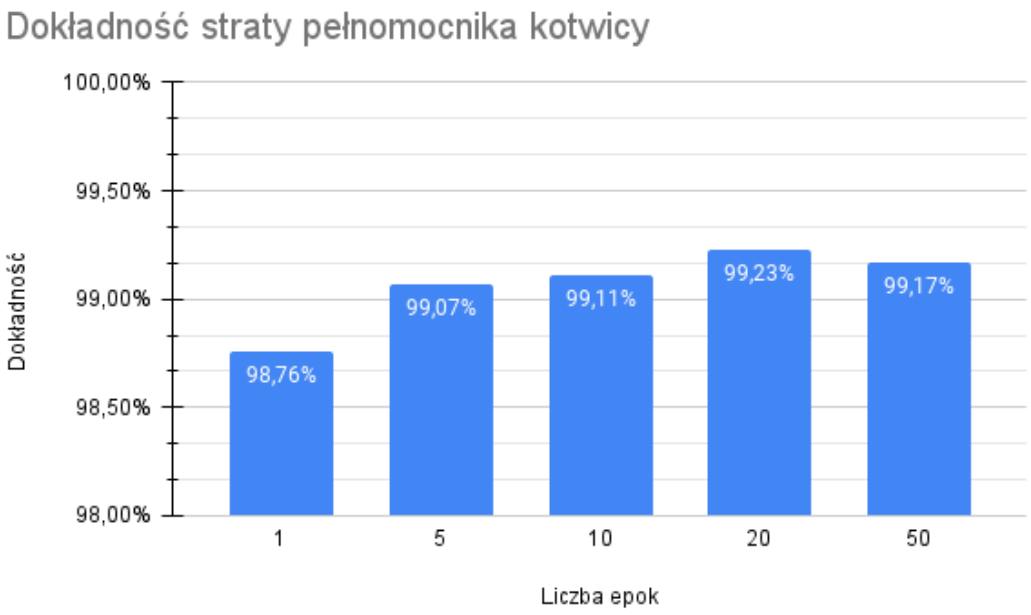


Rysunek 4.19: Wykres średniej wartość straty dla straty pełnomocnika kotwicy w zależności od liczby epok)

Z analizy wykresu zobrazowanego na Rysunku 4.19 wynika, że funkcja straty osiągała średnie wartości mieszczące się w zakresie od ponad 1,39 do prawie 2,322. Dostrzegalna jest istotna tendencja malejąca, która przypomina charakterystykę wykresu malejącej funkcji

logarytmicznej.

Dokładność wyuczonego modelu



Rysunek 4.20: Wykres dokładności wyuczonego modelu dla straty pełnomocnika kotwicy w zależności od liczby epok)

Analiza wykresu widocznego na Rysunku 4.20 wskazuje, że dokładność osiągana przez trenowany model prezentuje bardzo korzystne wyniki, mieszczące się w zakresie od 98,76% do 99,23%. Zaobserwowano wpływ liczby epok na uzyskiwane wyniki dokładności modelu. Dynamika poprawy dokładności modelu wykazuje cechy przypominające rosnącą funkcję logarytmiczną. Wyjątek stanowi ostatniego modelu trenowanego przy 50 epokach, gdzie zaistniał niewielki spadek dokładności.

Przy realizacji badań skorzystano ze straty marginesu trójkowego [48]. Na Rysunku 4.21 zobrazowano dynamiczne wahania wartości straty podczas procesu treningowego. Ilustracja w Rysunku 4.22 prezentuje natomiast, jak liczba epok wpływa na jakość osiąganą przez wyuczoną reprezentację modelu.

4.3.11 Strata marginesu trójkowego

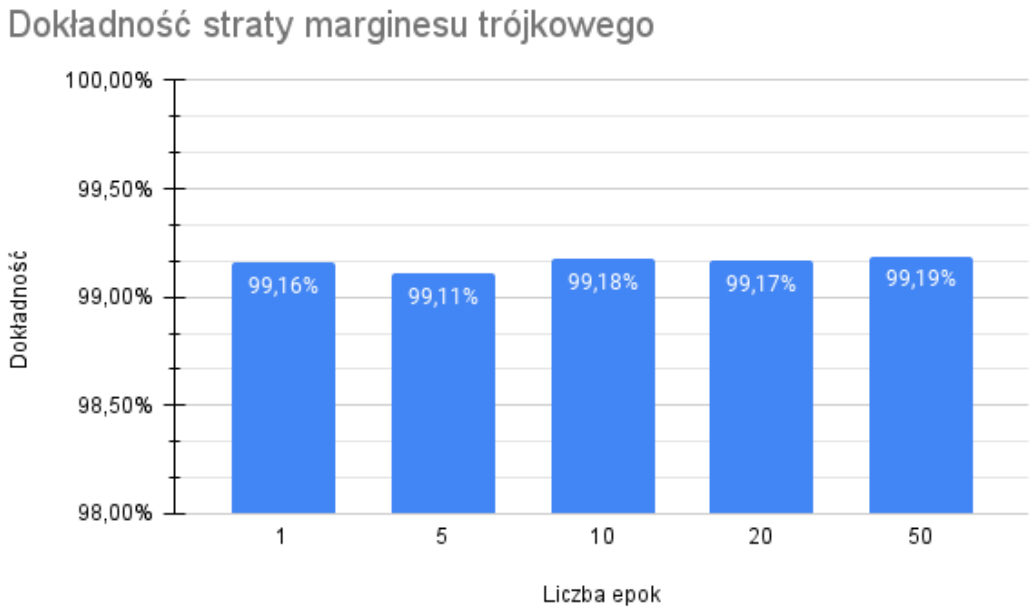
Zmiana wartości straty podczas treningu

Analiza przeprowadzona na podstawie wykresu z Rysunku 4.21 wskazuje, że funkcja straty uzyskiwała średnie wartości znajdujące się w zakresie od ponad $0,0145$ do prawie $0,016$. Nie obserwowano spodziewanej tendencji malejącej. Na tej podstawie można wysnuć wniosek, że średnia wartość straty w trenowanym modelu nie wykazuje zależności

Rysunek 4.21: Wykres średniej wartość straty dla straty marginesu trójkowego w zależności od liczby epok)

od liczby epok.

Dokładność wyuczonego modelu



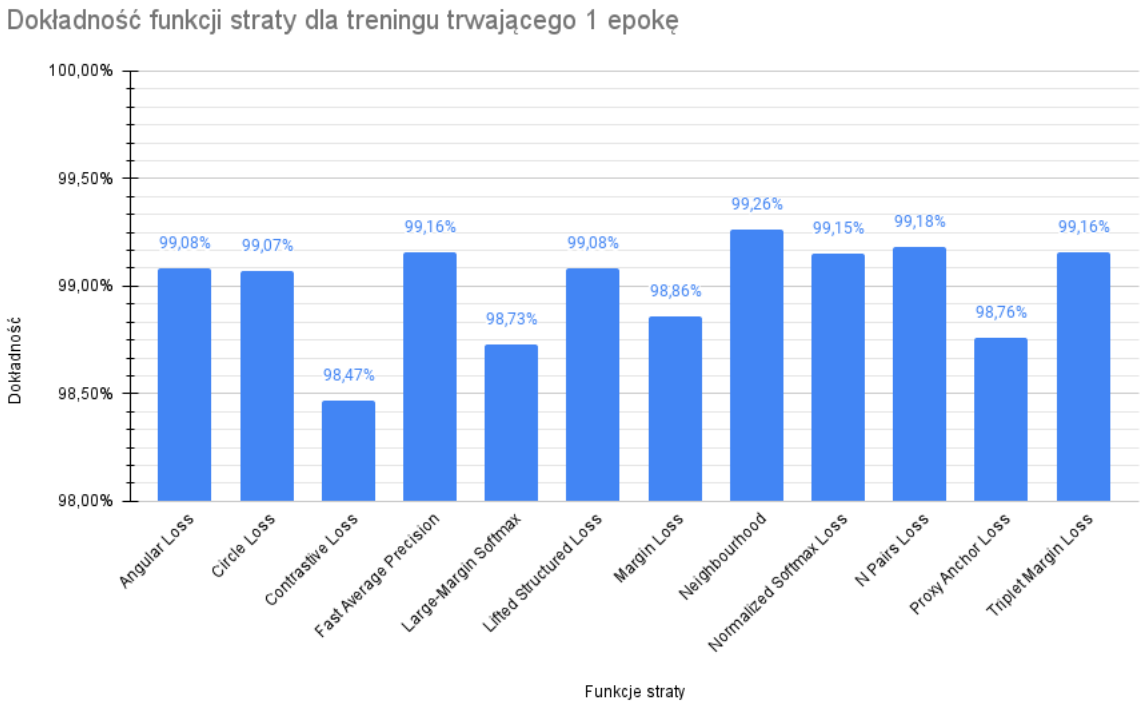
Rysunek 4.22: Wykres dokładności wyuczonego modelu dla straty marginesu trójkowego w zależności od liczby epok)

Wynik analizy wykresu widocznego na Rysunku 4.22 wskazuje, że dokładność osiągana

przez trenowany model prezentuje bardzo dobre osiągnięcia, mieści się w zakresie od 99, 11% do 99, 19%. Nie stwierdzono istotnego wpływu liczby epok na uzyskiwane wyniki dokładności modelu.

4.4 Wyniki badania wpływu funkcji straty

Dokładność modelu przy 1 epoce

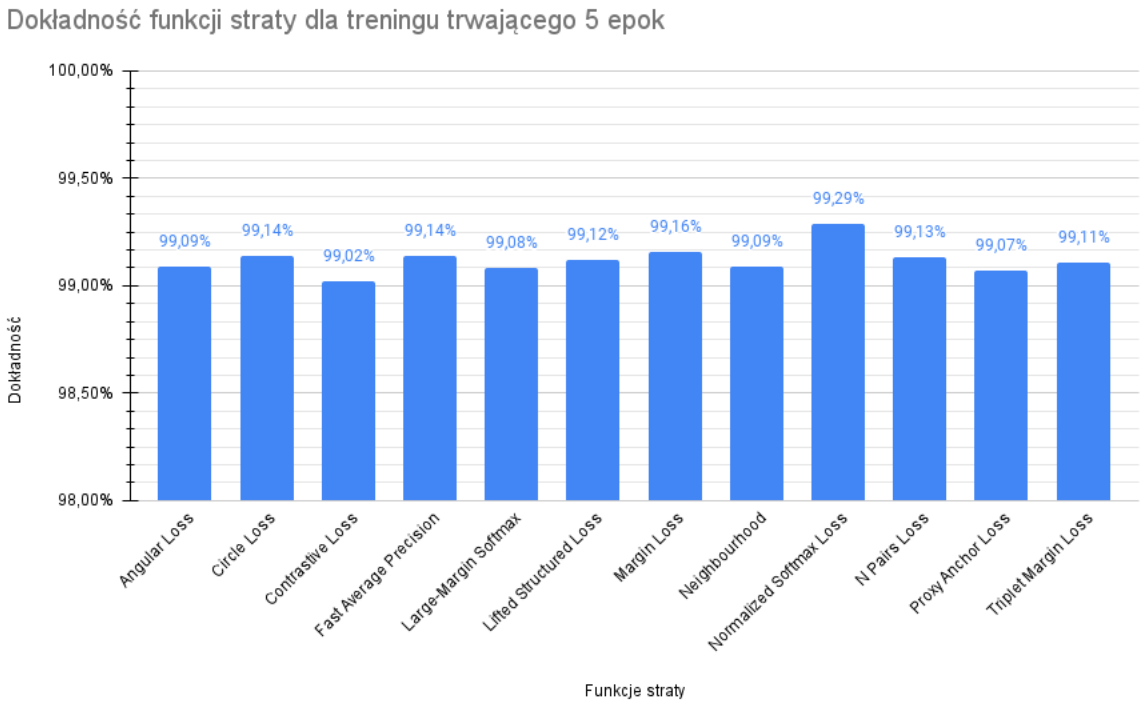


Rysunek 4.23: Wykres dokładności wyuczonego modelu dla badanych funkcji strat trenowanego przez 1 epokę

Analiza przeprowadzona na podstawie wykresu z Rysunku 4.23 wskazuje, że dokładność osiągana przez trenowane modele przy liczbie epok równej 1 charakteryzuje się wynikami z przedziału od 98, 47% do 99, 26%. W kontekście zbadanych funkcji straty, najkorzystniejsze wyniki osiągnęła funkcja straty analizy składników sąsiedztwa, podczas gdy najmniej korzystne wyniki uzyskano przy zastosowaniu funkcji straty kontrastowej, gdyż osiągnęła ona najmniejszy wynik dokładności spośród badanych funkcji straty.

Dokładność modelu przy 5 epokach

Z wyników analizy przedstawionych na wykresie widocznym na Rysunku 4.24 wynika, że dokładność osiągana przez trenowany model po 5 epokach mieściła się w zakresie od 99, 02% do 99, 29%. Spośród zbadanych funkcji straty, znormalizowana strata miękki maks



Rysunek 4.24: Wykres dokładności wyuczonego modelu dla badanych funkcji strat trenowanego przez 5 epok

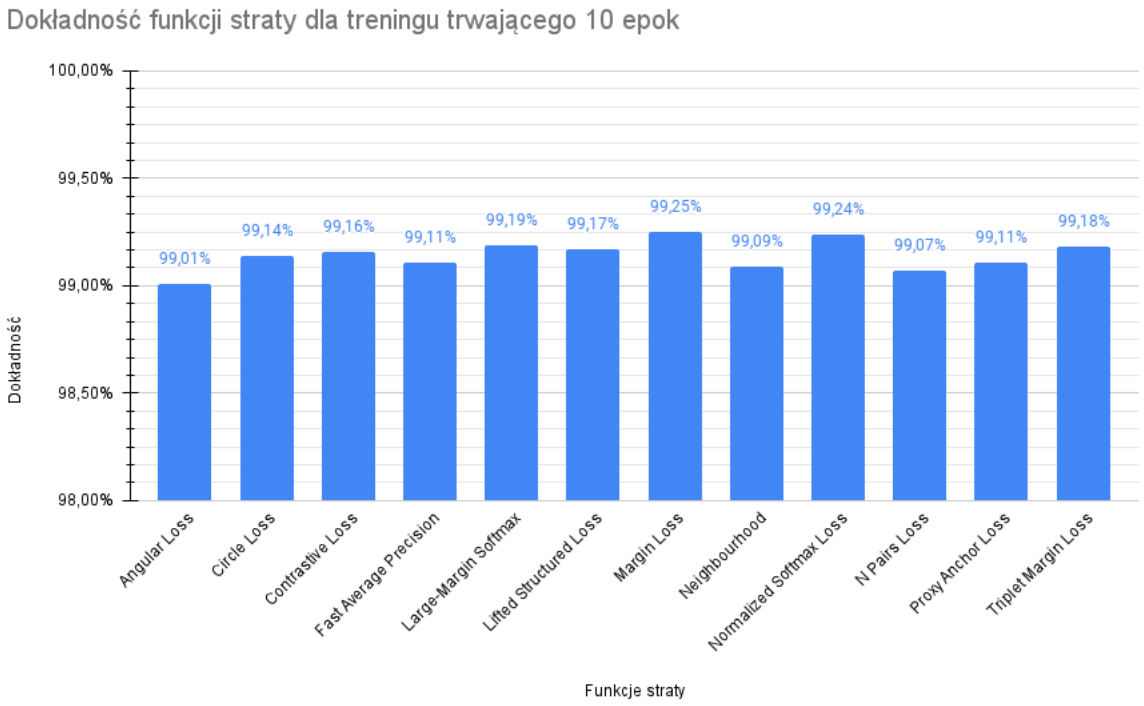
okazała się najskuteczniejsza, podczas gdy funkcja straty kontrastowej uzyskała najmniej korzystne wyniki. Zauważalne jest, że zarówno najlepszy, jak i najgorszy wynik są wyższe niż te uzyskane w przypadku treningu przy liczbie epok równej 1.

Dokładność modelu przy 10 epokach

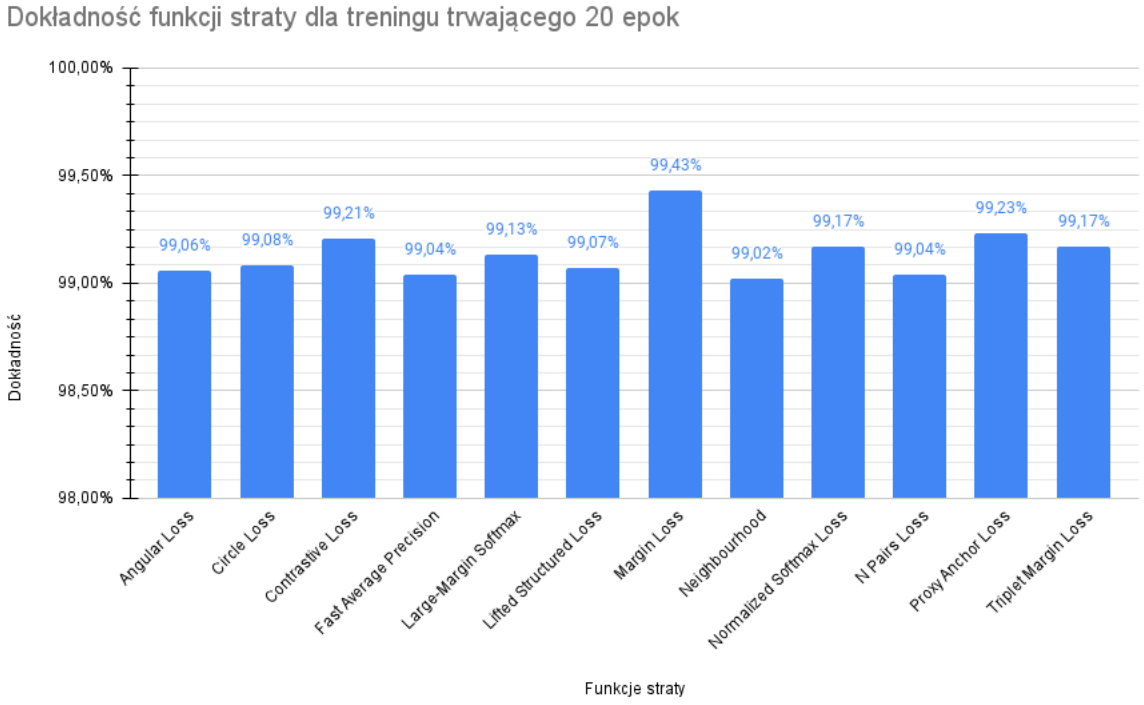
Z wyników analizy przedstawionych na wykresie z Rysunku 4.25 wynika, że dokładność osiągana przez trenowany model po 10 epokach mieściła się w zakresie od $99,01\%$ do $99,25\%$. Spośród zbadanych funkcji straty, funkcja straty marginesu okazała się najskuteczniejsza, a funkcja straty kątowej uzyskała najsłabszy wynik. Można zauważyć, że zarówno najlepszy, jak i najgorszy wynik są gorsze niż te uzyskane w przypadku treningu przy 5 epokach.

Dokładność modelu przy 20 epokach

Z wyników analizy przedstawionych na wykresie widocznym na Rysunku 4.26 wynika, że dokładność osiągana przez trenowany model po 20 epokach mieściła się w zakresie od $99,02\%$ do $99,43\%$. Spośród zbadanych funkcji straty, funkcja straty marginesu okazała się najskuteczniejsza, podczas gdy funkcja straty analizy składników sąsiedztwa uzyskała najgorszy wynik. Najsłabszy odnotowany wynik z powrotem wrócił do wartości $99,02\%$, a odnotowano najlepszy jak dotąd wynik dokładności o wartości $99,43\%$.

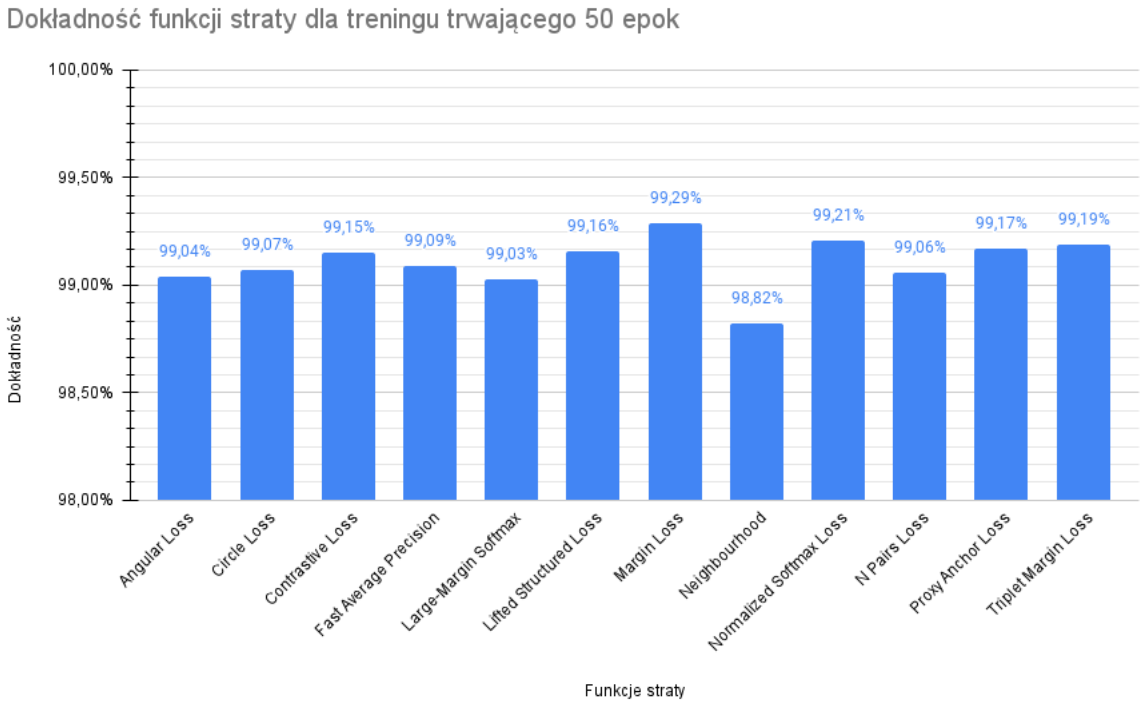


Rysunek 4.25: Wykres dokładności wyuczonego modelu dla badanych funkcji strat trenowanego przez 10 epok



Rysunek 4.26: Wykres dokładności wyuczonego modelu dla badanych funkcji strat trenowanego przez 20 epok

Dokładność modelu przy 50 epokach



Rysunek 4.27: Wykres dokładności wyuczonego modelu dla badanych funkcji strat trenowanego przez 50 epok

Z wyników analizy przedstawionych na wykresie z Rysunku 4.27 wynika, że dokładność osiągana przez model po 50 epokach mieściła się w zakresie od $98,82\%$ do $99,29\%$. Spośród zbadanych funkcji straty, funkcja straty marginesu okazała się najskuteczniejsza, podczas gdy funkcja straty analizy składników sąsiedztwa uzyskała najgorszy wynik. Na pozycji najgorszej i najlepszej funkcji się nie zmieniły, ale obydwie osiągnęły dużo słabsze dokładności niż w przypadku 20 epok.

4.5 Wnioski z wyników badań

Poniżej opisano wnioski po analizie zebranych podczas badań danych:

- Wykresy dotyczące dokładności i średniej wartości straty w zależności od różnych funkcji strat oraz liczby epok ukazują, że osiągnięta dokładność modelu w większości przypadków jest wysoka, plasując się w przedziale powyżej 98.46%. Tendencja do uzyskiwania dobrych wyników wskazuje na skuteczność zastosowanych strategii treningowych.
- Analiza wykresów dla różnych funkcji strat wykazała, że istnieje potencjalna korelacja między dokładnością a średnią wartością straty. Jednakże warto zauważyć, że nie

zawsze jest to jednoznaczne. Niektóre funkcje strat, na przykład strata marginesu i strata pełnomocnika kotwicy, prowadziły do lepszych wyników dokładności, sugerując pewną zgodność między obiema miarami. Natomiast funkcja analizy składników sąsiedztwa uzyskiwała niższe wyniki, co może sugerować, że w pewnych przypadkach odchylenie między dokładnością a stratą może być większe.

- W przypadku analizy wpływu liczby epok na dokładność modelu, obserwowano zróżnicowane efekty. W niektórych przypadkach, takich jak strategie straty marginesu i pełnomocnika kotwicy, zwiększenie liczby epok prowadziło do niewielkiego wzrostu dokładności, przy czym widoczny był też efekt przeuczenia modelu, który można było zaobserwować poprzez spadek dokładności przy zbyt długim treningu. Jednak nie zaobserwowano spójnego trendu, a zwiększenie liczby epok nie zawsze przekładało się na większą dokładność.

- Średnia wartość straty w większości przypadków wykazywała tendencję malejącą wraz ze wzrostem liczby epok, co sugeruje skuteczność procesu treningowego w redukcji straty. Niemniej jednak, istniały przypadki, w których nie można było zaobserwować spodziewanej tendencji malejącej, co wskazuje na pewne zróżnicowanie w zachowaniu modelu w zależności od doboru funkcji straty.

  Podsumowując, analiza wyników badań w kontekście doboru różnych funkcji strat oraz liczby epok dostarcza wniosków potwierdzających skuteczność niektórych strategii treningowych w osiąganiu wysokich wyników dokładności. Jednakże, istotne jest również zauważenie zależności między wyborem funkcji straty, liczbą epok a ostatecznymi osiągniętymi rezultatami.

Rozdział 5

Podsumowanie

W ramach niniejszego badania podjęto analizę wpływu zastosowanej funkcji straty oraz liczby epok na osiągane rezultaty dokładności modelu sieci neuronowej, który wykorzystuje technikę głębokiego uczenia metryki. Ocena skuteczności tego modelu przeprowadzona była przy wykorzystaniu algorytmu KNN.

Z badań można wysnuć poniższe wnioski:

- Głębokie uczenie metryki wykazuje znakomite osiągnięcia w kontekście problemów klasyfikacji obrazów. Przeprowadzone badania polegały na trenowaniu modeli sieci neuronowych przy zastosowaniu różnorodnych funkcji strat oraz wariantów liczby epok treningu. W wyniku tych eksperymentów, osiągnięto rezultaty sytuujące się w przedziale od 98.47% do 99.43% skuteczności klasyfikacyjnej.

- Należy podkreślić, że zarówno wybór funkcji straty, jak i decyzja dotycząca liczby epok treningu, mają istotny wpływ na dokładność wytrenowanego modelu. Niemniej jednak, nie dla każdej funkcji straty zwiększenie liczby epok prowadzi do poprawy rezultatów. Zaobserwowano, że istnieje potencjalne ryzyko nadmiernego dopasowania modelu, co może prowadzić do obniżenia dokładności wraz z dłuższym procesem treningu.

- W trakcie doboru hiperparametrów, kluczowe znaczenie ma właściwy dobór liczby epok w zależności od stosowanej funkcji straty. Okazało się, że pewne funkcje strat osiągają lepsze wyniki przy krótszym okresie treningu, podczas gdy inne wymagają większej liczby epok w celu precyzyjnego ukształtowania tworzonej metryki klasyfikacyjnej.

Potencjalne przyszłe kierunki badań:

- Jednym z perspektywicznych kierunków badań stanowi dokładniejsza ocena wpływu innych hiperparametrów, takich jak współczynnik uczenia, przy stałym wykorzystaniu tych samych funkcji strat. Celem jest przeprowadzenie porównawczej analizy

dokładności osiągniętych modeli w zależności od zastosowanych hiperparametrów. Przedstawione badanie pozwoli na lepsze zrozumienie, jak różne konfiguracje parametrów wpływają na jakość wyników w kontekście optymalizacji modelu.

- Kolejnym badaniem jest ocena osiągalnych rezultatów przy użyciu innych zbiorów danych treningowych. Badanie może obejmować zagadnienia klasyfikacyjne o odmiennych klasach niż odręcznie rysowane symbole pochodzące z zestawu MNIST.

- Możliwym rozwinięciem jest przeprowadzenie badań dotyczących wpływu parametrów konkretnych funkcji strat, które w obecnej pracy przyjęto jako wartości domyślne. Precyzyjna kalibracja tych parametrów może prowadzić do osiągnięcia lepszej jakości reprezentacji przestrzeni cech, co w rezultacie przekłada się na wyższą dokładność modelu.

- Alternatywnym obszarem eksploracji jest zbadanie możliwości zastosowania głębokiego uczenia metryki w kontekście innych problemów niż klasyfikacja. Rozpoznawanie twarzy oraz identyfikacja osób stanowią przykłady takich problemów.

Zakreślony cel niniejszej pracy, obejmujący analizę oddziaływania wykorzystanych funkcji straty oraz liczby epok treningu, został z powodzeniem zrealizowany. Z przeprowadzonych badań wyłoniono konkluzje, które mają istotne implikacje dla dalszych studiów badawczych. Przedstawione w analizie wnioski stanowią punkt wyjścia dla przyszłych badań, które zaowocują eksploracją nowych kierunków badawczych.

Bibliografia

[1] *About google, our culture and company news.* URL: `https://about.google/` (term. wiz. 15. 08. 2023).

[2] Carlos Affonso, André Luis Debiaso Rossi, Fábio Henrique Antunes Vieira, André Carlos Ponce de Leon Ferreira i in. „Deep learning for biological image classification". W: *Expert systems with applications* 85 (2017), s. 114–122.

[3] Saad Albawi, Tareq Abed Mohammed i Saad Al-Zawi. „Understanding of a convolutional neural network". W: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, s. 1–6.

[4] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal i Vijayan K Asari. „The history began from alexnet: A comprehensive survey on deep learning approaches". W: *arXiv preprint arXiv:1803.01164* (2018).

[5] *Apple.* URL: `https://www.apple.com/business/` (term. wiz. 27. 08. 2023).

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage i Anil Anthony Bharath. „A brief survey of deep reinforcement learning". W: *arXiv preprint arXiv:1708.05866* (2017).

[7] Yoshua Bengio, Aaron Courville i Pascal Vincent. „Representation learning: A review and new perspectives". W: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), s. 1798–1828.

[8] Anil Bhattacharyya. „On a measure of divergence between two statistical populations defined by their probability distribution". W: *Bulletin of the Calcutta Mathematical Society* 35 (1943), s. 99–110.

[9] J Bromley, I Guyon, Y Lecun i in. „Signature Verification Using a Siamese Time Delay Neural Network [C] Advances in Neural Information Processing Systems 6". W: *7th NIPS Conference*. Morgan Kaufmann Publishers Inc. 1993.

[10] Xinyuan Cai, Chunheng Wang, Baihua Xiao, Xue Chen i Ji Zhou. „Deep nonlinear metric learning with independent subspace analysis for face verification". W: *Proceedings of the 20th ACM international conference on Multimedia*. 2012, s. 749–752.

[11] Özer Çelik. „A research on machine learning methods and its applications". W: *Journal of Educational Technology and Online Learning* 1.3 (2018), s. 25–40.

[12] Siddhartha Chaudhary, Shivam Yadav, Shweta Kushwaha i Surya Ratan Pratap Shahi. „A brief review of machine learning and its applications". W: *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology* 12.SUP 1 (2020), s. 218–223.

[13] Sumit Chopra, Raia Hadsell i Yann LeCun. „Learning a similarity metric discriminatively, with application to face verification". W: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. T. 1. IEEE. 2005, s. 539–546.

[14] *Cloud, computers, Apps and Gaming*. URL: `https://www.microsoft.com/` (term. wiz. 15. 08. 2023).

[15] Yin Cui, Feng Zhou, Yuanqing Lin i Serge Belongie. „Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop". W: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 1153–1162.

[16] Leon A Gatys, Alexander S Ecker i Matthias Bethge. „A neural algorithm of artistic style". W: *arXiv preprint arXiv:1508.06576* (2015).

[17] Amir Globerson i Sam Roweis. „Metric learning by collapsing classes". W: *Advances in neural information processing systems* 18 (2005).

[18] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis i Russ R Salakhutdinov. „Neighbourhood components analysis". W: *Advances in neural information processing systems* 17 (2004).

[19] Ian Goodfellow, Yoshua Bengio i Aaron Courville. *Deep learning*. MIT press, 2016.

[20] Aman Gupta, Haohan Wang i Madhavi Ganapathiraju. „Learning structure in gene expression data using deep architectures, with an application to gene clustering". W: *2015 IEEE international conference on bioinformatics and biomedicine (BIBM)*. IEEE. 2015, s. 1328–1335.

[21] Neha Gupta i in. „Artificial neural network". W: *Network and Complex Systems* 3.1 (2013), s. 24–28.

[22] Mai Lan Ha i Volker Blanz. „Deep ranking with adaptive margin triplet loss". W: *arXiv preprint arXiv:2107.06187* (2021).

[23] Raia Hadsell, Sumit Chopra i Yann LeCun. „Dimensionality reduction by learning an invariant mapping". W: *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*. T. 2. IEEE. 2006, s. 1735–1742.

[24] Geoffrey Hinton, Yann LeCun i Yoshua Bengio. „Deep learning". W: *Nature* 521.7553 (2015), s. 436–444.

[25] Elad Hoffer i Nir Ailon. „Deep metric learning using triplet network". W: *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer. 2015, s. 84–92.

[26] Elad Hoffer i Nir Ailon. „Semi-supervised deep learning by metric embedding". W: *arXiv preprint arXiv:1611.01449* (2016).

[27] John J Hopfield. „Neural networks and physical systems with emergent collective computational abilities." W: *Proceedings of the national academy of sciences* 79.8 (1982), s. 2554–2558.

[28] *Investor overview*. URL: `https://ir.baidu.com/` (term. wiz. 15. 08. 2023).

[29] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar i Pierre-Alain Muller. „Deep learning for time series classification: a review". W: *Data mining and knowledge discovery* 33.4 (2019), s. 917–963.

[30] David Jacobs. „Correlation and convolution". W: *Class Notes for CMSC* 426 (2005), s. 401–409.

[31] Laveen N Kanal. „Perceptron". W: *Encyclopedia of Computer Science*. 2003, s. 1383–1385.

[32] Mahmut Kaya i Hasan Şakir Bilge. „Deep metric learning: A survey". W: *Symmetry* 11.9 (2019), s. 1066.

[33] Mahmut KAYA i Hasan Şakir BİLGE. „Deep Metric Learning: A Survey". W: *Symmetry* 11.9 (2019). ISSN: 2073-8994. DOI: `10.3390/sym11091066`. URL: `https://www.mdpi.com/2073-8994/11/9/1066`.

[34] John D Kelleher. *Deep learning*. MIT press, 2019.

[35] Sungyeon Kim, Dongwon Kim, Minsu Cho i Suha Kwak. „Proxy anchor loss for deep metric learning". W: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, s. 3238–3247.

[36] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber i Laura E Barnes. „Hdltex: Hierarchical deep learning for text classification". W: *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2017, s. 364–371.

[37] Anders Krogh. „What are artificial neural networks?" W: *Nature biotechnology* 26.2 (2008), s. 195–197.

[38] *Kup Gogle VR I inteligentne okulary*. URL: `https://www.meta.com/pl/` (term. wiz. 15. 08. 2023).

[39] Shutao Li, Weiwei Song, Leyuan Fang, Yushi Chen, Pedram Ghamisi i Jon Atli Benediktsson. „Deep learning for hyperspectral image classification: An overview". W: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), s. 6690–6709.

[40] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman i Geoffrey Hinton. „Backpropagation and the brain". W: *Nature Reviews Neuroscience* 21.6 (2020), s. 335–346.

[41] Weiyang Liu, Yandong Wen, Zhiding Yu i Meng Yang. „Large-margin softmax loss for convolutional neural networks". W: *arXiv preprint arXiv:1612.02295* (2016).

[42] Bohdan Macukow. „Neural networks–state of art, brief history, basic models and architecture". W: *Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference, CISIM 2016, Vilnius, Lithuania, September 14-16, 2016, Proceedings 15*. Springer. 2016, s. 3–14.

[43] Batta Mahesh. „Machine learning algorithms-a review". W: *International Journal of Science and Research (IJSR).[Internet]* 9 (2020), s. 381–386.

[44] Kameo Matusita. „Decision rules, based on the distance, for problems of fit, two samples, and estimation". W: *The Annals of Mathematical Statistics* (1955), s. 631–640.

[45] Warren S McCulloch i Walter Pitts. „A logical calculus of the ideas immanent in nervous activity". W: *The bulletin of mathematical biophysics* 5 (1943), s. 115–133.

[46] Marvin Minsky i Seymour Papert. „An introduction to computational geometry". W: *Cambridge tiass., HIT* 479.480 (1969), s. 104.

[47] *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.* URL: `http://yann.lecun.com/exdb/mnist/` (term. wiz. 27. 08. 2023).

[48] Kevin Musgrave. *Pytorch metric learning¶*. URL: `https://kevinmusgrave.github.io/pytorch-metric-learning/` (term. wiz. 15. 08. 2023).

[49] Hyun Oh Song, Yu Xiang, Stefanie Jegelka i Silvio Savarese. „Deep metric learning via lifted structured feature embedding". W: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 4004–4012.

[50] Keiichi Osako, Rita Singh i Bhiksha Raj. „Complex recurrent neural networks for denoising speech signals". W: *2015 IEEE workshop on applications of signal processing to audio and acoustics (WASPAA)*. IEEE. 2015, s. 1–5.

[51] *PL About Amazon 2022*. 2022. URL: `https://www.aboutamazon.pl/` (term. wiz. 27. 08. 2023).

[52] Mark Pritt i Gary Chern. „Satellite image classification with deep learning". W: *2017 IEEE applied imagery pattern recognition workshop (AIPR)*. IEEE. 2017, s. 1–7.

[53] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[54] A. L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers". W: *IBM Journal of Research and Development* 3.3 (1959), s. 210–229. DOI: 10.1147/rd.33.0210.

[55] Jürgen Schmidhuber. „Deep learning in neural networks: An overview". W: *Neural networks* 61 (2015), s. 85–117.

[56] Florian Schroff, Dmitry Kalenichenko i James Philbin. „Facenet: A unified embedding for face recognition and clustering". W: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, s. 815–823.

[57] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua i Francesc Moreno-Noguer. „Discriminative learning of deep convolutional feature point descriptors". W: *Proceedings of the IEEE international conference on computer vision*. 2015, s. 118–126.

[58] Kihyuk Sohn. „Improved deep metric learning with multi-class n-pair loss objective". W: *Advances in neural information processing systems* 29 (2016).

[59] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang i Yichen Wei. „Circle loss: A unified perspective of pair similarity optimization". W: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, s. 6398–6407.

[60] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang i Joel S Emer. „Efficient processing of deep neural networks: A tutorial and survey". W: *Proceedings of the IEEE* 105.12 (2017), s. 2295–2329.

[61] *Tesla*. URL: https://www.tesla.com/pl_pl/about (term. wiz. 27. 08. 2023).

[62] A. M. TURING. „I.—Computing Machinery and intelligence". W: *Mind* LIX.236 (1950), 433–460. DOI: 10.1093/mind/lix.236.433.

[63] Fei Wang i Jimeng Sun. „Survey on distance metric learning and dimensionality reduction in data mining". W: *Data mining and knowledge discovery* 29.2 (2015), s. 534–564.

[64] Haohan Wang i Bhiksha Raj. „On the origin of deep learning". W: *arXiv preprint arXiv:1702.07800* (2017).

[65] Hua Wang, Cuiqin Ma i Lijuan Zhou. „A brief review of machine learning and its application". W: *2009 international conference on information engineering and computer science*. IEEE. 2009, s. 1–4.

[66] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu i Yuanqing Lin. „Deep metric learning with angular loss". W: *Proceedings of the IEEE international conference on computer vision*. 2017, s. 2593–2601.

[67] Kilian Q Weinberger i Lawrence K Saul. „Distance metric learning for large margin nearest neighbor classification." W: *Journal of machine learning research* 10.2 (2009).

[68] Paul Werbos. „Beyond regression: New tools for prediction and analysis in the behavioral sciences". W: *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA* (1974).

[69] Chao-Yuan Wu, R Manmatha, Alexander J Smola i Philipp Krahenbuhl. „Sampling matters in deep embedding learning". W: *Proceedings of the IEEE international conference on computer vision*. 2017, s. 2840–2848.

[70] Jianxin Wu. „Introduction to convolutional neural networks". W: *National Key Lab for Novel Software Technology. Nanjing University. China* 5.23 (2017), s. 495.

[71] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do i Kaori Togashi. „Convolutional neural networks: an overview and application in radiology". W: *Insights into imaging* 9 (2018), s. 611–629.

[72] Dong Yi, Zhen Lei, Shengcai Liao i Stan Z Li. „Deep metric learning for person re-identification". W: *2014 22nd international conference on pattern recognition*. IEEE. 2014, s. 34–39.

[73] Jun Yu, Xiaokang Yang, Fei Gao i Dacheng Tao. „Deep multimodal distance metric learning using click constraints for image ranking". W: *IEEE transactions on cybernetics* 47.12 (2016), s. 4014–4024.

[74] Andrew Zhai i Hao-Yu Wu. „Classification is a strong baseline for deep metric learning". W: *arXiv preprint arXiv:1811.12649* (2018).

Dodatki

Spis terminów i skrótów

CUDA	zjednoczona architektura przetwarzania na urządzeniach (ang. *Compute Unified Device Architecture*)
GPU	jednostka przetwarzania grafiki (ang. *Graphics Processing Unit*)
KNN	k najbliższych sąsiadów (ang. *k nearest neighbors*)
MNIST	zmodyfikowana baza danych Narodowego Instytutu Standardów i Technologii (ang. *Modified National Institute of Standards and Technology database*)
Adam	adaptacyjna estymacja momentu (ang. *Adaptive Moment Estimation*)
ReLU	rektyfikowana jednostka liniowa (ang.*Rectified Linear Unit*)
kotwica	próbka stosowana jako obiekt odniesienia (ang. *anchor*)
próbka pozytywna	próbka posiadająca tą samą etykietę co próbka odniesienia (ang. *positive*)
próbka negatywna	próbka posiadająca inną etykietę co próbka odniesienia (ang. *negative*)
liczba epok	liczba pełnych przebiegów przez zestaw treningowy (ang. *epochs*)
wielkość partii	liczba przykładów danych, które są przetwarzane jednocześnie przez model podczas każdej iteracji procesu uczenia (ang. *batch size*)
funkcja straty	metryka, która mierzy, jak bardzo prognozy modelu różnią się od rzeczywistych wartości danych treningowych (ang. *loss function*)
CNN	konwolucyjna sieć neuronowa (ang. *convolution neural network*)

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- zbiory danych użyte w eksperymentach,
- film pokazujący działanie opracowanego oprogramowania

Spis rysunków