POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Telekomunikacji Multimedialnej

Praca dyplomowa inżynierska

ROZPOZNAWANIE PISMA ODRĘCZNEGO Z WYKORZYSTANIEM SIECI NEURONOWYCH

Jakub Siejak

Promotor
dr inż. Adrian Dziembowski

POZNAŃ 2022

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Streszczenie

W niniejszej pracy inżynierskiej omówiona i zrealizowana została metoda rozpoznawania znaków alfanumerycznych przy pomocy dwóch rozwiązań: metody deterministycznej (użytej jako punkt odniesienia) oraz metody wyuczania sieci neuronowych, która była głównym celem badań. Przedstawione zostały aspekty teoretyczne jak i wyniki przeprowadzonych badań eksperymentalnych.

Abstract

In this thesis, a method of recognizing alphanumeric characters using two solutions was discussed and implemented: a deterministic method (used as a reference point) and a method of learning neural networks, which was the main goal of the research. Theoretical aspects as well as the results of the experimental research were presented.

Rozdział 1

Wstęp

1.1 Wprowadzenie

Praca ta porównuje różne sposoby rozwiązania problemu wykrywania znaków pisanych na obrazach. Porównaniu podlegają metody deterministyczne oparte o algorytmy deterministyczne oraz metody wykorzystujące sieci neuronowe, czyli wykorzystanie uczenia maszynowego i głębokiego uczenia. tematyka ta jest dość szeroko opisana jednak, brakuje jej wykorzystania w wielu gałęziach przemysłu oraz życia codziennego. Powodem wyboru tej tematyki była próba znalezienia optymalnego rozwiązania i osiągnięcia jak najlepszych wyników dla możliwie szerokiego zbioru znaków. Zamieszczone w pracy badania mogą zostać wykorzystane do dalszych badań z wykorzystaniem najnowszych architektur sieci neuronowych oraz dołączenia kolejnych zbiorów znaków rozpoznawanych.

1.2 Cel pracy

Celem pracy była implementacja programu rozpoznającego tekst na obrazie oraz zapisującego wyniki w formie pliku tekstowego.

1.3 Zakres pracy

Badania miały odpowiedzieć na pytanie jaka jest maksymalna wydajność każdej z metod po optymalizacji i przedstawienia tej wartości w formie procentowej skuteczności. Optymalizacja metod polegała na dobraniu odpowiedniego aparatu matematycznego wykazującego się najlepszymi wynikami oraz dobieraniu odpowiednich wartości liczbowych do parametrów algorytmów.

W drugim rozdziale przedstawiony został przegląd literatury opisujący rodzaje sztucznych neuronów oraz sieci, z których są zbudowane, a także metody i algorytmy

uczenia. Dopełnieniem tego rozdziału są informacje na temat najważniejszych pojęć związanych z uczeniem sieci neuronowych.

Trzeci rozdział odnosi się do wiedzy na temat obecnie stosowanych algorytmów widzenia komputerowego. W szczególności do zastosowanych nich aparatów matematycznych.

Rozdział czwarty opisuje cały proces badawczy od wyboru sposobu rozwiązania problemu poprzez dobór parametrów i końcową optymalizację. Jest to najważniejszy z rozdziałów podsumowujący część praktyczną wykonanej pracy wraz z wnioskami płynącymi z każdego badania i wyboru.

Bazą całego projektu jest aplikacja w postaci prostego edytora zdjęć, któremu został poświęcony rozdział piąty. Opisane są w nim funkcje programu oraz wygląd panelu głównego.

Rozdział 2

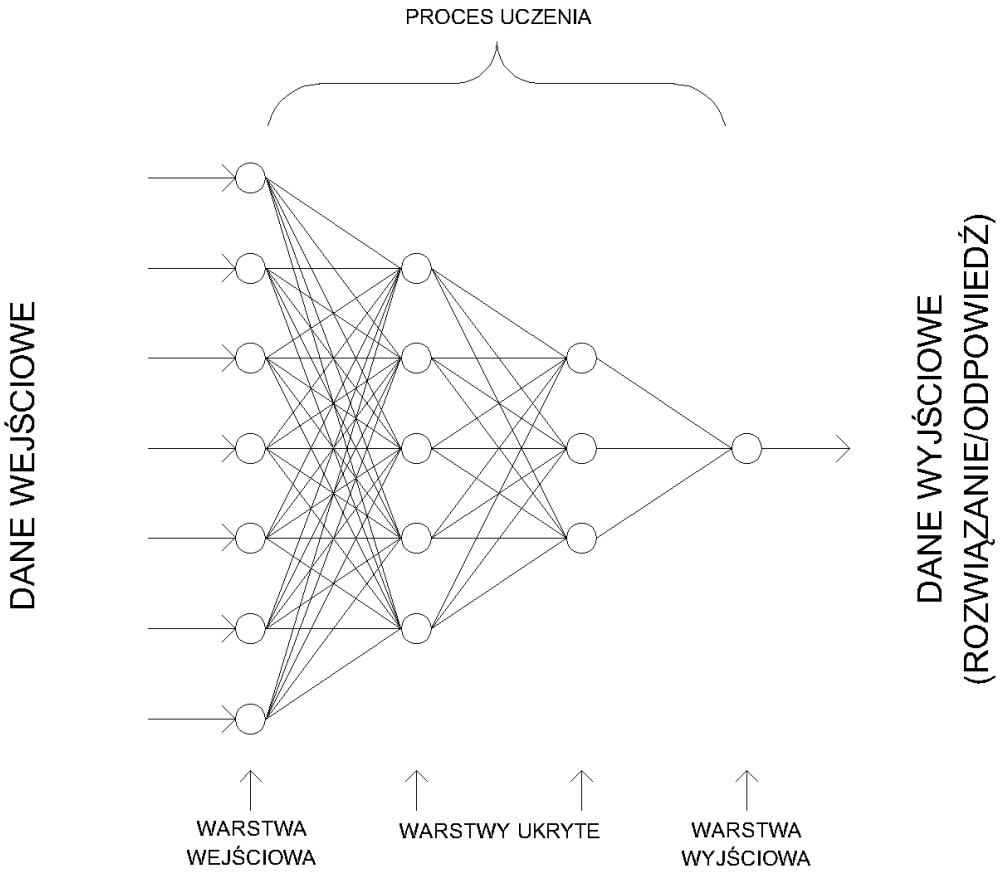
Sieci neuronowe

Sieci neuronowe zwane też sztucznymi sieciami neuronowymi mimo swojej długiej obecności w badaniach i nauce stosunkowo niedawno zyskały swoją popularność. Obecnie potencjał sztucznych sieci neuronowych jest implementowany do pracy we wszystkich gałęziach gospodarki oraz aspektach życia codziennego. Systemy i aplikacje zbudowane z wykorzystaniem modeli sieci neuronowych możemy znaleźć wszędzie wokół nas, są to między innymi:

- Rozpoznawanie tekstu,
- Prognozowanie wartości np. pogody, inflacji, giełdy,
- Rozpoznawania twarzy (blokada ekranu telefonu) i innych obiektów na obrazach (medycyna) i obrazie ruchomym (również na żywo),
- Optymalizacja produkcji,
- Modelowanie optymalnych rozwiązań np. karoserii samochodów,
- Algorytmy reklamowe uczące się na podstawie przeglądanych przez nas treści, a także portale ukazujące treści, które mogą nam się spodobać.

Powodem tak dużego zainteresowania sieciami neuronowymi i stosowania ich do rozwiązywania wszelkich problemów, zarówno tych, które pojawiły się niedawno jak i tych, które dotychczas były uznawane za nierozwiązywalne przez człowieka lub algorytmy deterministyczne jest brak potrzeby głębokiego zrozumienia rozwiązywanego problemu przez osobę tworzącą model sieci neuronowej. To co powinien wiedzieć i posiadać użytkownik to zestaw danych wejściowych i ewentualnie odpowiadający mu zestaw danych wyjściowych oraz mieć podstawowe pojęcie o rozwiązywanym problemie w celu doboru optymalnej architektury sieci neuronowej. Na rysunku 2.1 możemy zobaczyć ogólny schemat sieci neuronowej.

Podczas całego procesu budowy modelu i wyuczania sieci neuronowej najwięcej czasu należy poświęcić na dobór oraz dostosowanie danych początkowych. Należy je



Rysunek 2.1: Schemat ogólny sieci neuronowej

nie tylko zdobyć, przekształcić do jednakowej formy, ale także sprawdzić czy posiada się ich wystarczającą liczbę oraz czy nie ma danych brakujących lub uszkodzonych. Przy małej ilości danych możemy powielić posiadane próbki tak by otrzymać optymalną ilość. Analogicznie można postąpić z próbkami, które znacząco przeważają nad innymi wtedy należy usunąć odpowiednie jednostki danych ze zbioru wejściowego. Dlatego też podczas doboru danych wejściowych możemy wykorzystać różne metody ich selekcji:

- **Metoda siłowa** (Brute Force) – wykorzystywanie wszystkich posiadanych danych,
- **Usuwanie danych silnie skorelowanych** – usuwanie jednego elementu z pary, dla której wartość korelacji jest większa niż wartość zaimplementowana jako progowa,
- **Usuwanie danych o niskiej wariancji** – niska wariancja świadczy o tym, że wartość informacji przenoszona przez próbkę danych jest znikoma,
- **Przycinanie** (Pruning) – usuwanie danych, które występują w nadmiernej ilości w zbiorze wejściowym, najczęściej wykorzystywane podczas tworzenia

sieci klasyfikujących dane na kategorie,

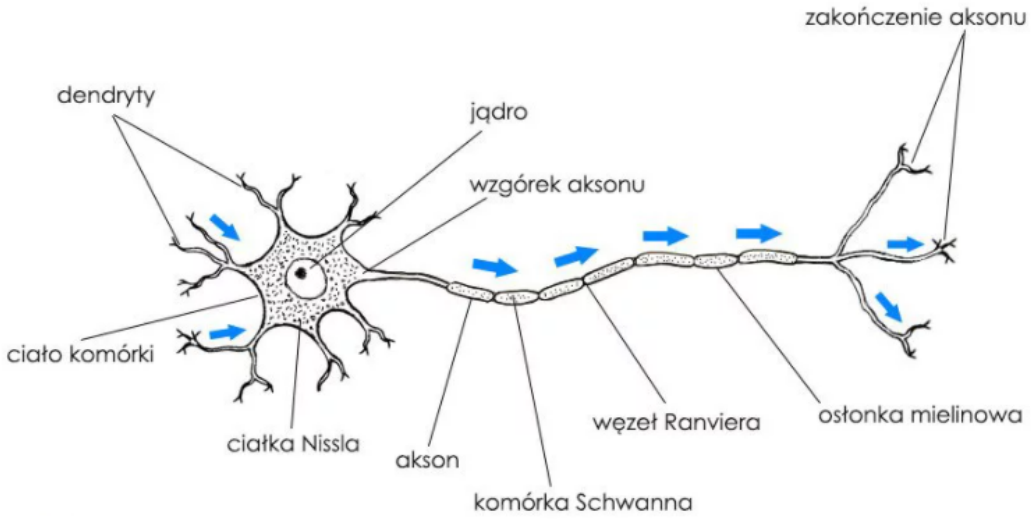
- **Powielanie** – wykorzystywane przy małej ilości danych wejściowych, zduplikowanie danych może poszerzyć zestaw danych.

2.1 Sztuczny neuron

Podstawową jednostką sztucznej sieci neuronowej jest sztuczny neuron będący odpowiednikiem neuronów znajdujących się w biologicznym mózgu. W celu symulacji uczenia podobnego do tego, które zachodzi w ludzkim mózgu należy zbudować sieć odpowiednio połączonych ze sobą neuronów uszeregowanych w warstwy. Sztuczny neuron jest podobny do biologicznego zarówno pod względem budowy jak i działania.

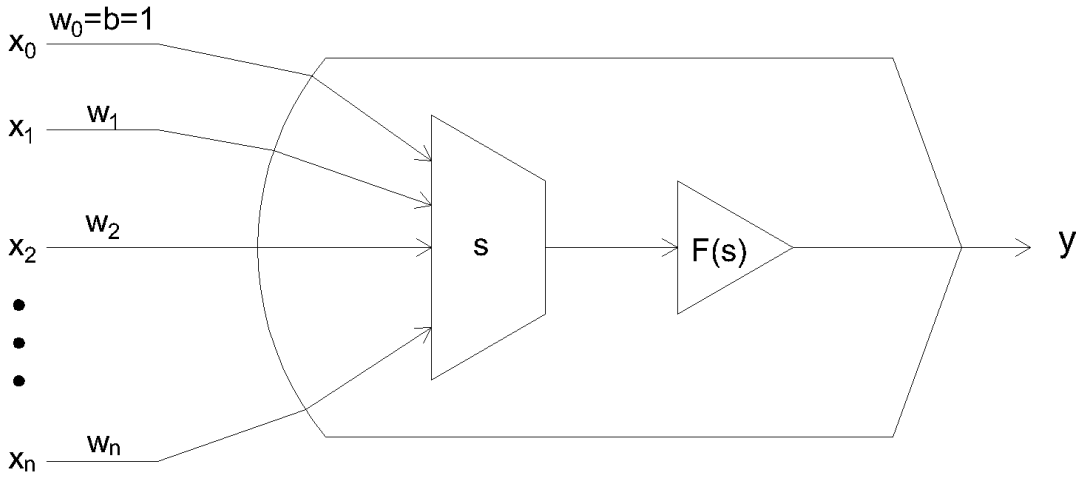
Neuron występujący w ludzkim mózgu składa się z trzech głównych elementów: ciała komórki, dendrytów i aksonu.

- **Ciało komórki** – zawiera jądro neuronu, odchodzą od niego również dendryty i akson,
- **Dendryty** – wypustki odbiorcze umożliwiające odbiór sygnałów od innych komórek nerwowych, neuron może posiadać jeden lub więcej dendrytów,
- **Akson** – wypustka wyjściowa, występująca pojedynczo w ciele neuronu, odpowiada za wysyłanie impulsów do innych komórek nerwowych.



Rysunek 2.2: Model neuronu biologicznego [1]

Porównując schematy neuronów możemy zauważyć analogie w budowie. Wejścia odpowiadają dendrytom, a wyjście aksonowi.



Rysunek 2.3: Model sztucznego neuronu, x_i – sygnały na wejściach, w_i – wagi wejść, s – agregacja danych, F(s) – oznacza funkcję aktywacji, y – wyjście

Proces jaki zachodzi w jednym neuronie możemy opisać następująco: sygnał trafiający na wejście jest przemnażany przez wagę tego wejścia, a następnie sumowany z iloczynami sygnałów i wag z innych wejść tego neuronu. Z sumy tworzona jest nieliniowa funkcja sumy, która jest przekazywana przez wyjście na inne neurony. Opis ten możemy prześledzić na rysunku 2.3. W uproszczeniu proces zachodzące w sztucznym neuronie możemy opisać za pomocą dwóch czynności: agregacji danych wejściowych oraz obliczania wartości funkcji agregacji.

2.2 Rodzaje sztucznych neuronów

Biologiczne neurony różnią się od siebie między innymi kształtem, wielkością oraz liczbą wypustek w zależności od pełnionej przez nie funkcji. Analogicznie taki podział możemy zauważyć podczas porównywania ze sobą sztucznych neuronów.

2.2.1 Neuron liniowy

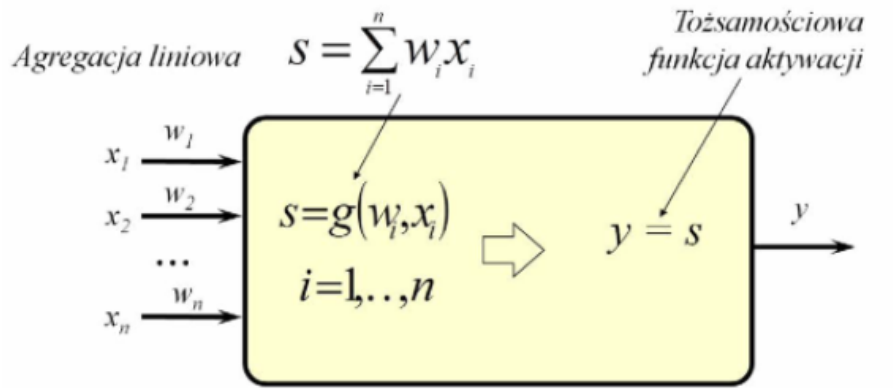
Zarówno agregacja danych jak i funkcja aktywacji są liniowe. Zastosowanie takiego neuronu da nam dobre wyniki jednak rozwiąże tylko podstawowe problemy. Przez swój brak złożoności nie nadaje się do rozwiązywania bardziej złożonych zadań. Schemat tego neuronu znajduje się na rysunku 2.4.

Poniższy wzór opisuje wartość agregacji liniowej, która dla neuronu liniowego jest również wartością wyjściową tego neuronu:

$$s_l = b + \sum_i x_i w_i, \tag{2.1}$$

$$y = s_l, \tag{2.2}$$

gdzie: s_l – wartość agregacji liniowej, y – wyjście neuronu, b – wyraz wolny (BIAS), x_i – wartość na i-tym wejściu, w_i – waga i-tego wejścia.



Rysunek 2.4: Model neuronu liniowego [2]

BIAS – wyraz wolny, do którego nie jest przypisywany żaden sygnał wejściowy ze zbioru wejściowego. Często BIAS jest oznaczany jako wejście zerowe, a jego wartość ustala się jako 1. Nie każda sieć neuronowa korzysta z wyrazu wolnego, mimo to jego wartość może ulec zmianie na drodze uczenia.

2.2.2 Neuron radialny

Działanie neuronu radialnego opiera się na radialnej agregacji danych wejściowych oraz funkcji aktywacji w postaci funkcji Gaussa.

Wartość agregacji radialnej obliczana jest za pomocą następującego wzoru:

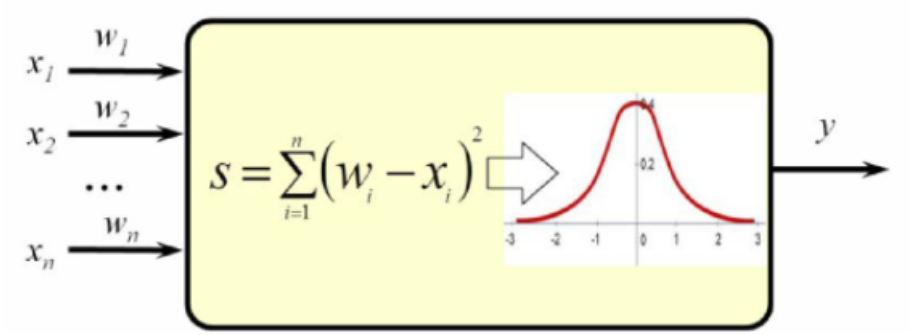
$$s_r = \sum_i (x_i - w_i)^2, \tag{2.3}$$

gdzie: s_r – wartość agregacji radialnej, x_i – wartość na i-tym wejściu, w_i – waga i-tego wejścia.

Neurony radialne znajdują zastosowanie w Radialnych sieciach neuronowych (RBFN) oraz w drugiej warstwie sieci Kohonena, regresyjnej, probabilistycznej i regresyjnej.

2.2.3 Neuron sigmoidalny

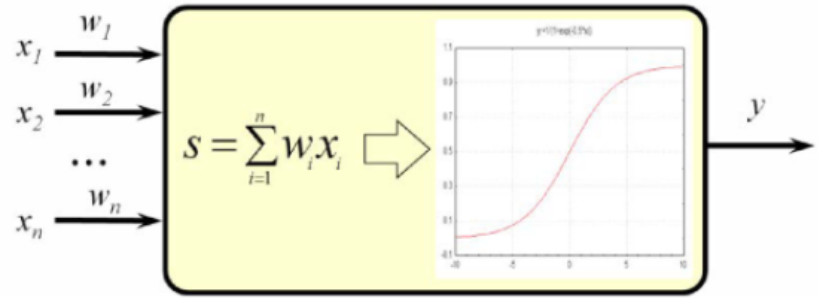
Popularny neuron nieliniowy o liniowej agregacji danych oraz sigmoidalnej funkcji aktywacji. Neuron ten często użytkowany jest wraz z elementem BIAS. Wystę-



Rysunek 2.5: Model neuronu radialnego [2]

powanie neuronów sigmoidalnych można zauważyć w sieciach neuronowych stworzonych do rozwiązywania problemów w rozmaitych dziedzinach głównie w warstwach ukrytych. Ma to swoje podłoże najprawdopodobniej w zbliżonej budowie tego neuronu do biologicznego odpowiednika występującego w ludzkim mózgu.

Agregacja danych wejściowych jest obliczana za pomocą wzoru 2.1, wynika to z wykorzystania tej samej metody agregacji co neuron liniowy.



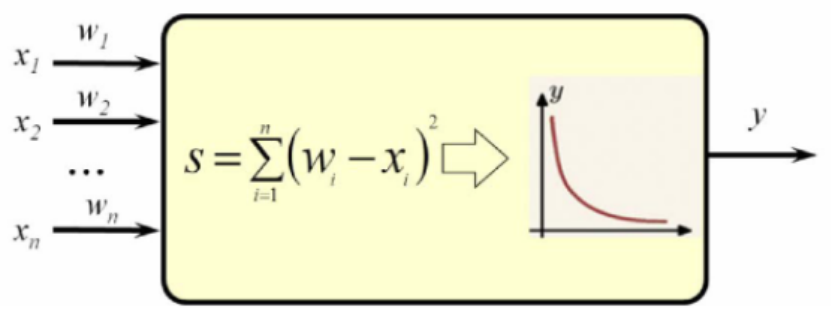
Rysunek 2.6: Model neuronu sigmoidalnego [2]

Rzadziej stosowaną odmianą neuronu sigmoidalnego jest neuron tangensoidalny. Różnica pomiędzy nimi kryje się w wartościach funkcji aktywacji, mimo że wykresy obu wyglądają prawie identycznie to wykres sigmoidalny przyjmuje jedynie wartości dodatnie dlatego też neuron ten nazywany jest także "unipolarnym". Neuron tangensoidalny jest natomiast "bipolarny"gdyż zbiór wartości funkcji należy do przedziału $< -1; 1 >$.

2.2.4 Neuron Kohonena

Charakterystyka neuronu Kohonena jest zbliżona do neuronu radialnego. Różnica polega na zastosowaniu odwrotności sygnału sumarycznego pobudzenia jako funkcji aktywacji. Zatem wzór na agregację danych wejściowych jest zgodny ze

wzorem 2.1. Oznacza to, że funkcja aktywacji działa odwrotnie proporcjonalnie do wartości uzyskanej po obliczeniu agregacji.

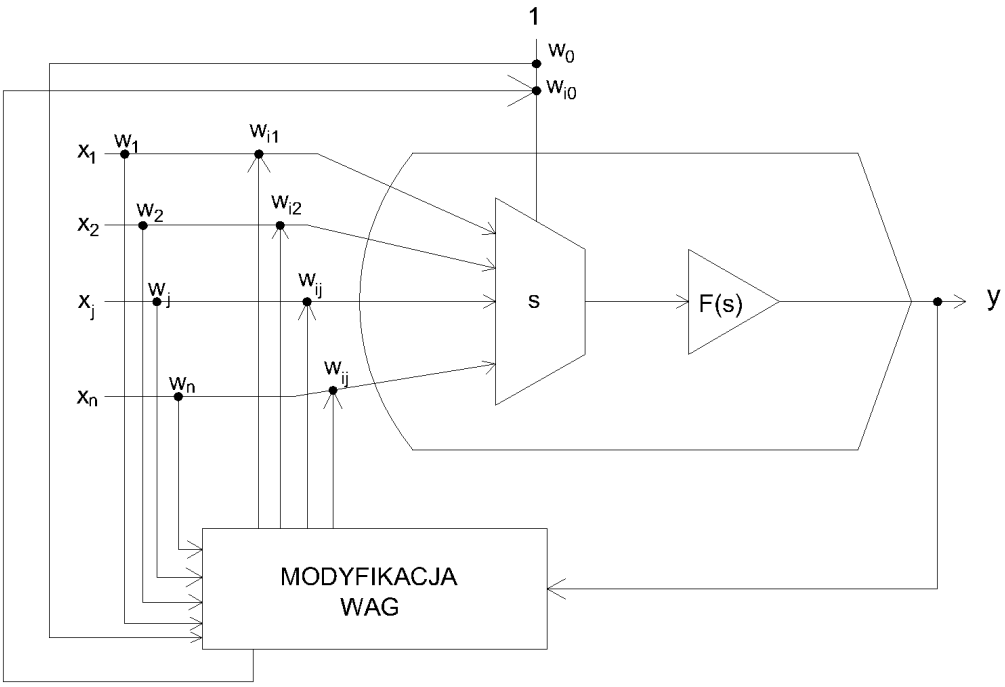


Rysunek 2.7: Model neuronu Kohonena [2]

Neurony te używane są głównie w sieciach Kohonena.

2.2.5 Neuron Hebba

Typ neuronu powstały dzięki badaniom Donalda Hebba. W trakcie swoich obserwacji zauważył, że komórki nerwowe ulegają wzmocnieniu jeśli zostaną powiązane ze sobą będą aktywowane jednocześnie



Rysunek 2.8: Model neuronu Hebba

Neurony Hebba mogą być uczone metodą z nauczycielem lub bez. Wzór na zmianę wagi sygnału wyjściowego:

$$\Delta w_{ij} = \eta y_j y_i, \tag{2.4}$$

gdzie: $\Delta\ w_{ij}$ – zmiana wagi, η – stała uczenia o wartości w przedziale $< 0, 1 >$

Dla uczenia z nauczycielem wzór wygląda następująco:

$$\Delta w_{ij} = \eta y_j d_i, \tag{2.5}$$

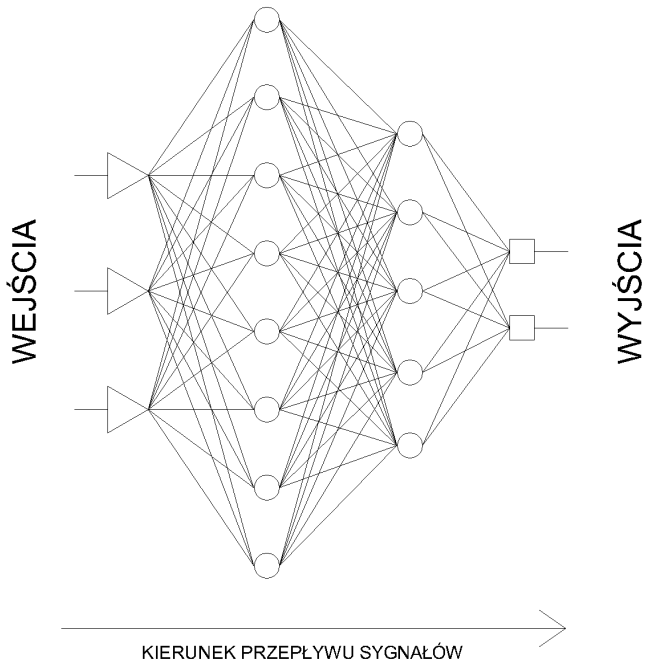
gdzie: d_i – wartość zadana

Sygnał wyjściowy y_i został zastąpiony d_i wartością zadaną.

2.3 Rodzaje sztucznych sieci neuronowych

2.3.1 Jednokierunkowa sieć neuronowa

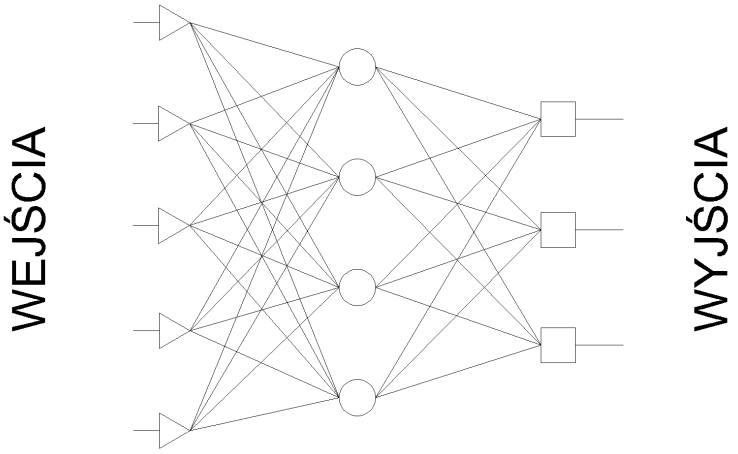
Podstawowy rodzaj sieci gdzie przepływ danych następuje od warstwy wejściowej do wyjściowej w jednym kierunku włącznie z warstwami ukrytymi, często używa się na nie określenia sieci typu "feedforward".



Rysunek 2.9: Model sieci jednokierunkowej

MLP

Wielowarstwowy perceptron (Multilayer Perceptron) jest jedną z popularniejszych struktur sieci neuronowych. Typowy model tej sieci składa się z warstwy wejściowej i wyjściowej oraz do dwóch warstw ukrytych. Warstwa wyjściowa może być zbudowana z neuronów sigmoidalnych lub liniowych. Natomiast warstwy ukryte oparte są tylko o neurony sigmoidalne.



Rysunek 2.10: Model perceptronu wielowarstwowego

Najczęstszą metodą uczenia sieci MLP jest wsteczna propagacja błędu.

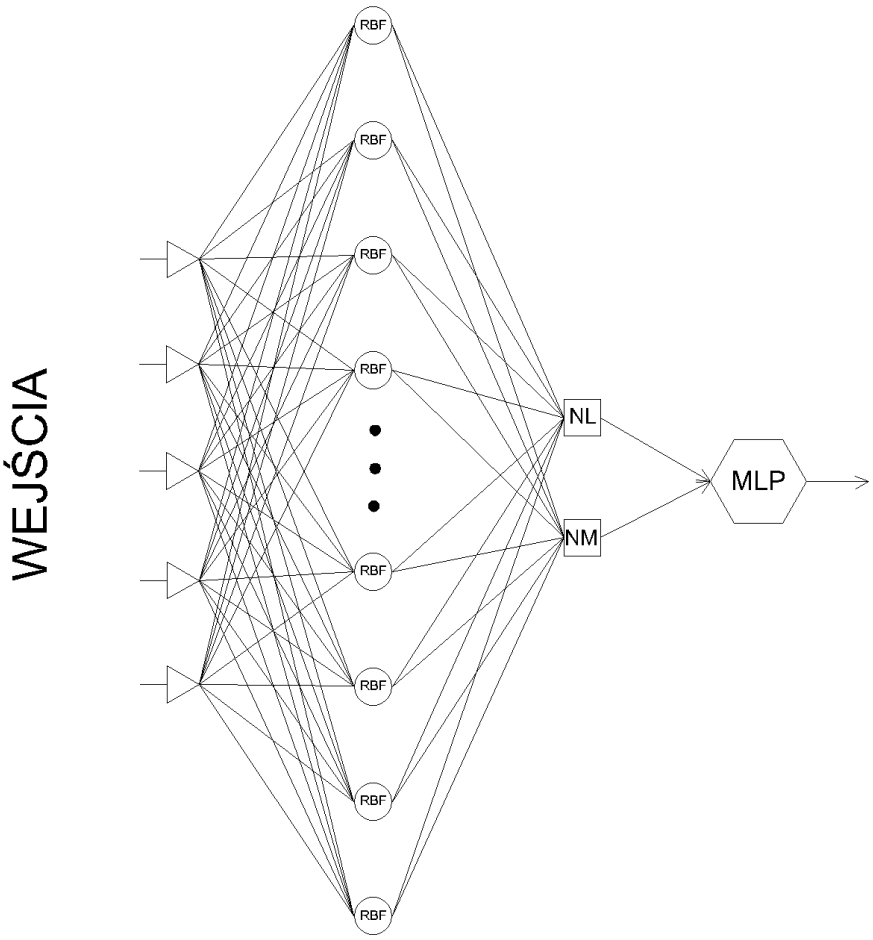
RBFN

Sieć radialna (Radial Basis Function Network) złożona tylko z jednej warstw ukrytej, na którą składają się neurony radialne występujące w dużej liczbie. Na model sieci radialnej składa się również warstwa wejściowa służąca do wprowadzania danych oraz warstwa wyjściowa, która najczęściej występuje w postaci pojedynczego neuronu liniowego.

GRNN

Sieć uogólnionej regresji (Generalized Regression Neural Network) została zaprojektowana tak by łączyć zalety poznanych powyżej sieci MLP i radialnej co przekłada się bezpośrednio na jej budowę. Oprócz warstw wejściowej i wyjściowej (pojedynczy neuron) sieć ta składa się z warstwy radialnej i regresyjnej. Warstwa radialna tak jak w sieci radialnej posiada dużą liczbę neuronów radialnych, natomiast warstwa regresyjna to jedynie dwa neurony. Dwa neurony składające się na warstwę regresyjną to neuron mianownikowy i licznikowy.

Neuron mianownikowy to neuron obliczający iloczyn wektora sygnałów wejściowych i wektora własnych wag, które ustalane są na drodze uczenia i zależne od liczby neuronów radialnych, które zostały aktywowane w trakcie uczenia sieci GRNN.



Rysunek 2.11: Model sieci uogólnionej regresji

Neuron licznikowy podobnie jak neuron mianownikowy oblicza iloczyn wektora własnych wag jednak drugim czynnikiem iloczynu jest wektor sygnałów z warstwy radialnej.

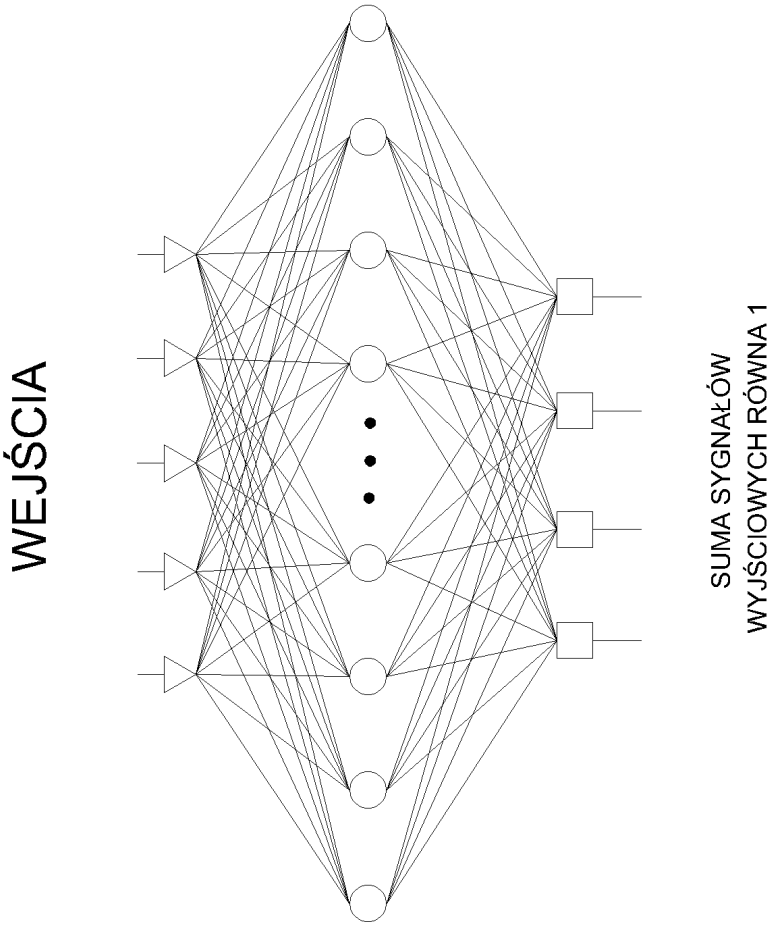
PNN

Probabilistyczna sieć neuronowa ma podstawową budowę to znaczy posiada warstwę wejściową i wyjściową oraz jedną warstwę ukrytą.

Jej cechami charakterystycznymi są:

- Warstwa ukryta składająca się z neuronów radialny w liczbie równej liczbie przypadków uczących,
- Liczba wyjść sieci równa liczbie możliwych rozwiązań zadanego problemu,
- Suma wartości wszystkich wyjść równa 1.

Kluczowe dla tej sieci jest normalizowanie wyników wyjściowych warstwy ukrytej w taki sposób aby suma wartości wyjść warstwy wyjściowej była równa 1. Dzięki



Rysunek 2.12: Model probablistycznej sieci neuronowej

temu liczby znajdujące się na wyjściach sieci PNN odpowiadają prawdopodobieństwu rozpoznania danej kategorii.

2.3.2 Rekurencyjne sieci neuronowe

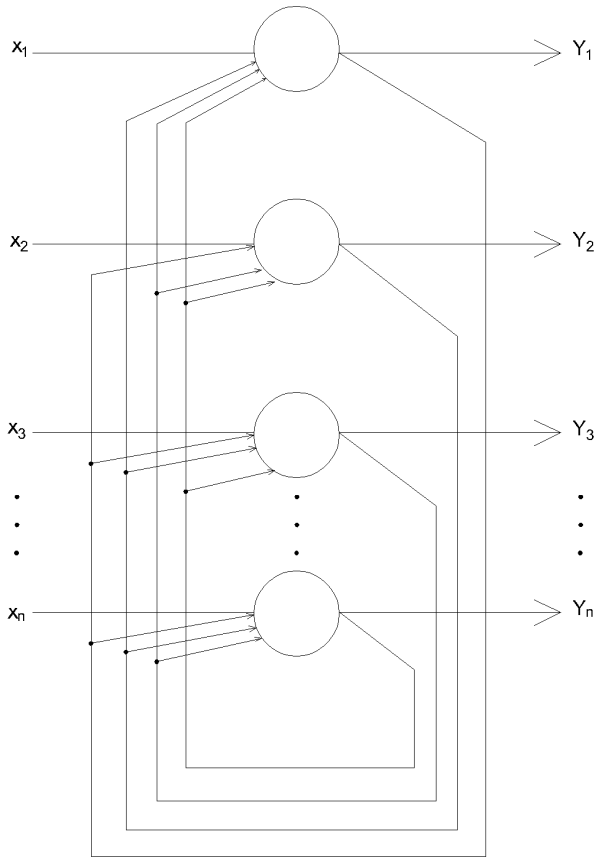
Są to sieci, w których występuje sprzężenie zwrotne w postaci podawania sygnału z wyjścia sieci na wejście. Efektem takiego połączenia są nagłe i silne pobudzenia neuronów, natomiast wynik może być różny. W zależności od budowy sieci oraz danych wejściowych, sygnały tej sieci mogą gwałtownie wzrosnąć dążąc do nieskończoności, ale także wygasić się.

W sieciach tych można zauważyć procesy związane z powstawaniem i rozwojem chaosu [2].

Sieć Hopfielda

Szczególny przypadek sieci rekurencyjnej złożonej tylko z jednej warstwy neuronów połączonych ze sobą na zasadzie każdy z każdym. Zasada łączenia zabrania jednak podawania na wejście neuronu jego wyjścia. Natomiast jeśli architektura

miałaby charakter ogólny i takie połączenie zostałoby w niej ujęte, to wtedy takiemu połączeni przypisywana jest waga 0. Kolejnym ważnym założeniem jest symetria wag połączeń, oznacza to, że wyjście i-tego neuronu podłączone do neuronu j-tego musi mieć taką samą wagę jak połączenie łączące wyjście j-tego neuronu i wejście i-tego neuronu.



Rysunek 2.13: Model sieci Hopfielda

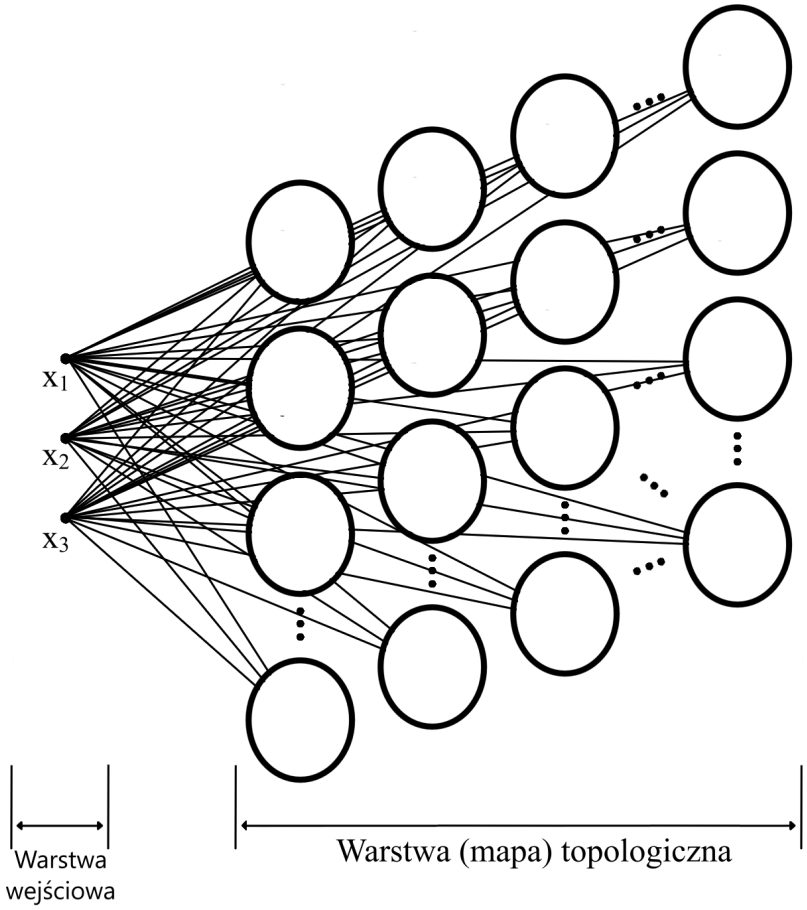
Głównym zastosowaniem sieci Hopfielda są rozwiązania optymalizacyjne, znanym przykładem jest wykorzystanie tej sieci do rozwiązania problemu komiwojażera [2].

2.3.3 Samoorganizująca się sięć neuronowa

Sieć oparta na samoorganizującym się odwzorowaniu (SOM). Ten typ sieci wykorzystuje uczenie bez nauczyciela inaczej zwane uczeniem nienadzorowanym. Pośród innych sieci wykorzystujących uczenie bez nauczyciela sieci samoorgaizujące wyróżniają się metodą zmian wag zachodzących podczas procesu uczenia.

Sieć Kohonena

Jest najpopularniejszym przykładem sieci samouczącej się. Składa się z warstwy wejściowej, która jest znana użytkownikowi oraz warstwy (mapy) topologicznej, która pozostaje ukryta, aż do zakończenia procesu uczenia. Można prześledzić sposób jej działania jednak jest to skomplikowane, ponieważ mapy te są złożone i wielowymiarowe co utrudnia ich zobrazowanie. Poniżej pokazany rysunek będzie zatem dużym uproszczeniem w stosunku do rzeczywistych modeli.



Rysunek 2.14: Model sieci Kohonena

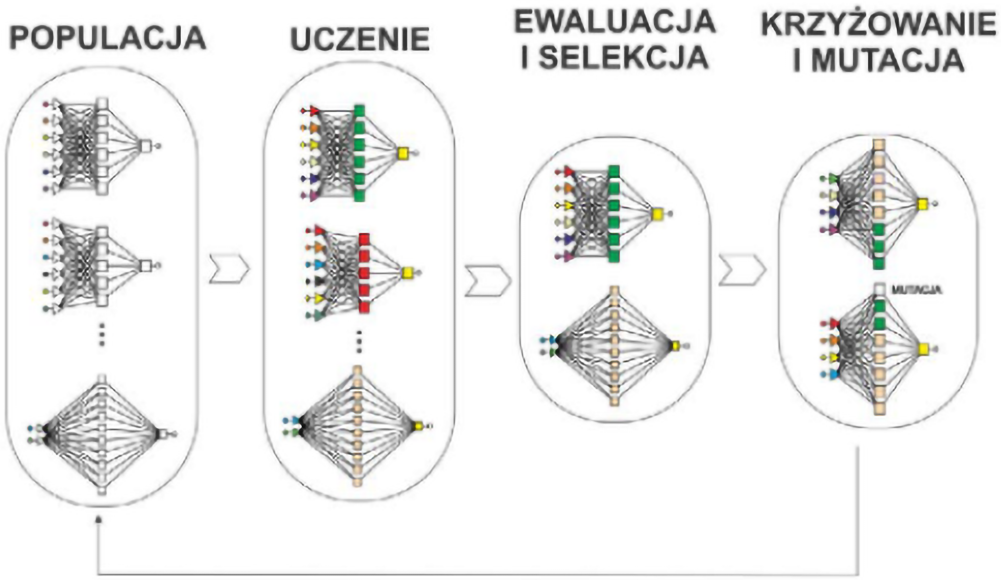
2.3.4 Sieć klasyfikacyjna

Model sieci, której zadaniem jest wybór najbardziej prawdopodobnej klasy z podanych przez użytkownika. Struktura takiej sieci jest charakterystyczna tylko pod względem liczby neuronów w warstwie wyjściowej. Ich liczba musi odpowiadać liczbie klas zwanych też kategoriami.

Najczęściej wykorzystywanym rodzajem sieci klasyfikacyjnej jest sieć klasyfikująca obrazy, jest to jednocześnie sieć splotowa, która jest w stanie wydajnie uczyć się na obiektach dwuwymiarowych jakimi są obrazy.

2.3.5 GNN

Sieci genetyczne (Genetic Neural Network) są wynikiem uzyskanym na drodze połączenia optymalizacyjnych algorytmów genetycznych i modelowania przy użyciu sieci neuronowych. Daje to w efekcie sieci, które same dobierają optymalny wektor wejściowy oraz strukturę sieci. Za tymi zaletami idzie wada w postaci dużego kosztu obliczeniowego, a co za tym idzie strata czasu. Wynika to z faktu dostarczania przez sieci GNN wielu równoległych rozwiązań zapewniających porównywalny poziom jakości przy zastosowaniu różnych technik i struktur.



Rysunek 2.15: Model sieci genetycznej [2]

2.3.6 CNN

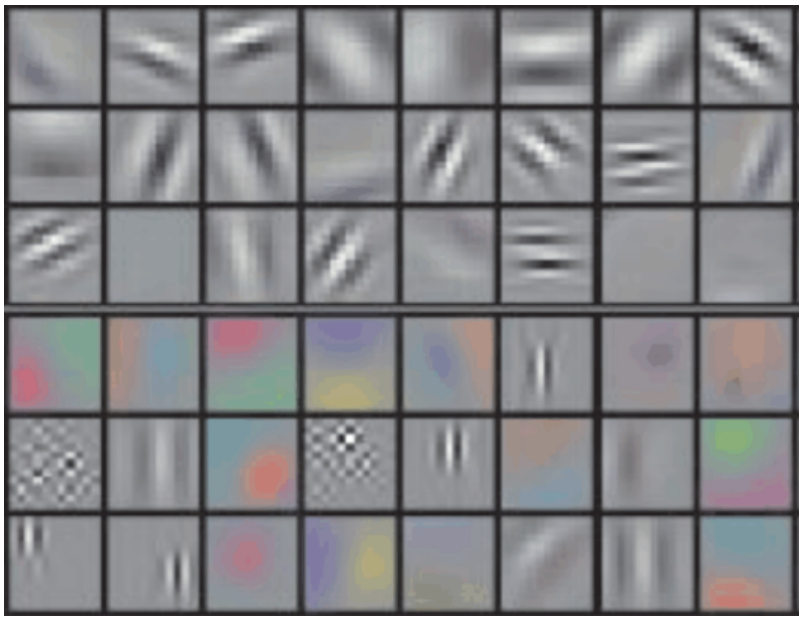
Splotowa sieć neuronowa (Convolutional Neural Network) jedna z ważniejszych sieci neuronowych pozwalających na naukę zależności w obrazach. Możliwość przetwarzania obrazów sieć ta zawdzięcza odpowiedniej warstwie wejściowej oraz strukturze wewnętrznej złożonej z różnych typów warstw. Sieci CNN są zaprojektowane z myślą o uczeniu na podstawie obrazów lub wizji.

Kluczowym elementem sieci neuronowych są operacje splotu wykonywane w warstwie splotowej (convolutioal layer), od której pochodzi nazwa sieci. Zapis matematyczny tych operacji dla dwuwymiarowych danych brzmi następująco [3]:

$$\begin{aligned} Y(i,j) = I(i,j) * K(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) = \\ = \sum_m \sum_n I(i-m, j-n)K(m,n), \end{aligned} \tag{2.6}$$

gdzie: I – macierz opisująca jasność pikseli, K – dwuwymiarowe jądro.

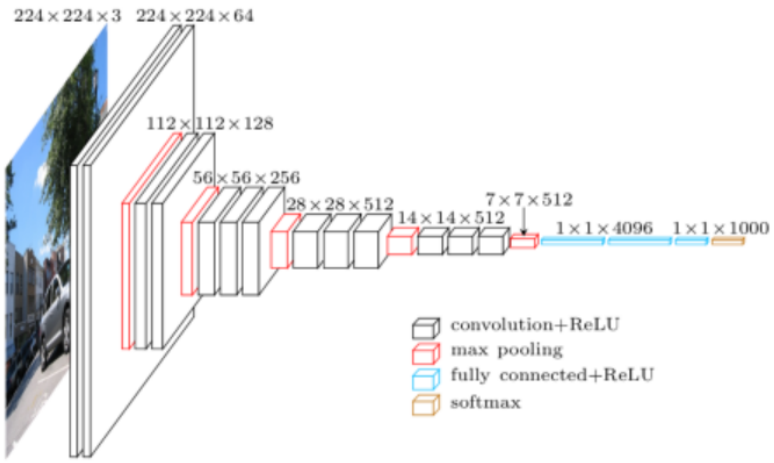
Sieć CNN jest w stanie zastąpić algorytmy widzenia komputerowego, których zadaniem jest między innymi ekstrakcja cech punktów szczególnych pod względem krawędzi i kolorów. Przewaga jaką daje sieć splotowa pozwoliła zaoszczędzić czas użytkownika na tworzeniu skomplikowanych algorytmów ekstrakcji i porównywania cech lub tworzenia sieci neuronowych, które wymagały złożonych danych wejściowych w postaci opisanych ręcznie przez twórcę przykładowych cech charakterystycznych obrazów.



Rysunek 2.16: Przykład wyników przetwarzania obrazu kolorowego w cechy diagnostyczne w jednej warstwie ukrytej o 48 neuronach przez sieć CNN [3]

Splotowe sieci neuronowe składają się z wielu (nawet kilkunastu) warstw z posiadających różne zadania oraz o znacznej różnicy liczby neuronów. W strukturze sieci CNN możemy wyróżnić dwa etapy uczenia, pierwszy etap to nauka punktów charakterystycznych, natomiast drugi etap to klasyfikacja danych. Różnorodność i charakterystyka poszczególnych warstw przedstawione są na rysunku 2.15.

- **Warstwa wejściowa** (input layer) – warstwa przyjmująca dane wejściowe w postaci obrazów lub wizji, w zależności od formatu i liczby wymiarów może mieć różną budowę,
- **Warstwa splotowa** (convolutional layer) – warstwa ta odpowiada za wykrywanie punktów charakterystycznych obrazów otrzymanych na wejściu przekształcenie ich do postaci filtru, a następnie obliczanie splotu podczas wykonywania skanu (przeczesywania) za pomocą wyuczonego filtru w celu znalezienia punktów charakterystycznych [5],



RYSUNEK 2.17: Model sieci splotowej [4]

- **Warstwa "łącząca"** (pooling layer) – występuje najczęściej pomiędzy dwoma warstwami splotowymi. Jest odpowiedzialna za zmniejszanie rozmiaru obrazu podanego na jej wejście jednocześnie zachowując wszystkie jego elementy (punkty charakterystyczne. Łączenie polega na mapowaniu obrazu za pomocą filtru o wymiarach 2x2 lub 3x3 piksele. Podczas mapowania z użyciem filtru 2x2 unika się nakładania kolejnych filtrów (przeskok o 2 piksele). Nakładanie zachodzi natomiast podczas użycia filtru 3x3, ponieważ przeskok następuje również o 2 piksele. Wyróżnia się dwie metody łączenia używane w praktyce:
  - **maksimum** (max – maximum pooling) maksymalna wartość z grupy pikseli,
  - **średnia** (avg – average pooling) średnia z grupy pikseli.

  Dzięki takiemu zabiegowi warstwa ta odpowiada również za zmniejszenie kosztu obliczeniowego i przyspieszenie uczenia jednocześnie zmniejszając szansę na przeuczenie[5],

- **Warstwa porzucenia** (dropout layer) – warstwa odpowiadająca za ochronę przed przeuczeniem sieci neuronowej. Jej działanie polega na losowym "wyłączaniu" – zerowaniu wartości sygnałów wyjściowych z określonej części neuronów. Taki proces zapobiega uczeniu się "na pamięć", czyli przeuczeniu polegający na tym, że sieć zbytnio dopasowuje się do zestawu treningowego [6],

- **Warstwa spłaszczająca** (flatten layer) – odpowiada za przekształcenie wielowymiarowych danych do postaci jednowymiarowej. Rekombinacja danych jest niezbędna ze względu na wejście warstwy w pełni połączonej, która nie przyjmuje danych wielowymiarowych z powodu aktywacji tej warstwy przez funkcję **Softmax** [6],

- **Warstwa w pełni połączona** (fully-connected layer) – Z poprzedniej warstwy nazywanej spłaszczającą zostaje przekazana jednowymiarowy wektor danych. Warstwa w pełni połączona wykonuje liniową kombinację, a następnie zwraca wektor o długości równej liczbie klas.

  Warstwa ta znajduje się zwykle na końcu sieci lub poprzedza ostatnią warstwę [5],

- **Warstwa dense** – Warstwa połączona z warstwą poprzednią na zasadzie każdy z każdym. Wykonywana jest w niej operacja przemnażania macierzy przez wektor, gdzie wektorem są wagi połączeń wejściowych do warstwy dense, natomiast macierz to zbiór sygnałów wyjściowych warstwy poprzedzającej [7].

2.4 Metody uczenia sieci neuronowych

Uczenie sieci może przebiegać na różne sposoby w zależności od posiadanych przez nas danych oraz złożoności rozwiązywanego problemu. Kluczowe okazuje się posiadanie przez nas danych wyjściowych zwanych danymi oczekiwanymi. Czasem mimo posiadania takich informacji nie korzystamy z nich by zobaczyć jak sieć poradzi sobie bez nich. Takie działania mogą doprowadzić do ciekawych i nieszablonowych lub też błędnych odpowiedzi. Przyjmuje się następujący podział metod uczenia:

2.4.1 Uczenie z nadzorem

Metoda uczenia wykorzystująca dane wejściowe w postaci przykładowych danych wejściowych i spodziewanych danych wyjściowych. Sieć ma zadanie znaleźć powiązanie między podanymi informacjami tak by była w stanie podać poprawną odpowiedź wyjściową po otrzymaniu tylko danych wejściowych. Stosuje się najczęściej podczas uczenia sieci klasyfikującej lub gdy znane są oczekiwane odpowiedzi.

2.4.2 Uczenie bez nadzoru

Uczenie to polega na wpuszczeniu na wejście sieci danych wejściowych bez podania oczekiwanego wyjścia. Oznacza to, że sieć musi sama zrozumieć zależności między danymi i podać odpowiedź na wyjście. Sposób ten jest najczęściej wykorzystywany, gdy nie znamy rozwiązania zadanego problemu na przykład podczas rozwiązywania zadania, które dotychczas pozostawało nierozwiązane przy pomocy metod deterministycznych.

2.4.3 Uczenie częściowo nadzorowane

Połączenie poprzednich metod, gdzie najpierw wykorzystujemy uczenie bez nadzoru, a następnie z nadzorem. Metoda ta daje zwykle lepsze rezultaty.

2.4.4 Uczenie ze wzmocnieniem

Polega na podawaniu na wejście sieci tylko części zestawu treningowego. Sieć stara się dostroić używając różnych zestawów danych uczących tak, aby uzyskać jak najlepszy wynik.

2.5 Algorytmy uczenia sieci neuronowych

Algorytmy można podzielić względem różnych kryteriów. Jednym z nich jest odporność na błędy w danych uczących. Błędy te mogą mieć postać:

- Niepełnych danych – przykładowo gdy dane wejściowe składają się z trzech elementów (Nazwa miasta, Liczba ludności, Cena nieruchomości) wtedy błąd będzie w postaci braku jednego z tych elementów (puste pole),
- Błędów grubych – błąd wynikający z pomyłki osoby zbierającej dane lub programu agregującego dane z różnych źródeł, powodów w tym przypadku może być wiele, a efektem będzie na przykład przesunięcie przecinka, które w efekcie może zaburzyć wynik średniej,
- Zanieczyszczenia szumem – pojawiające się zwykle podczas odczytu z innych urządzeń w szczególności analogowych, może skutkować błędami na przykład podczas zmiany kodowania, efektem będą tu dane niezdatne do odczytu na przykład szereg liter, symboli i cyfr w miejscu wartości liczbowej.

2.5.1 Podstawowe algorytmy uczenia

Algorytmy mogą dawać bardzo dobre wyniki do momentu pojawienia się danych, które nie są czyste – wolne od wad. Mimo powstania algorytmów bezpieczniejszych nadal są popularne w zastosowaniach praktycznych, może to wynikać z faktu mniejszego nakładu obliczeniowego. Na przykład gdy do dyspozycji jest mała ilość danych, którą użytkownik jest w stanie przejrzeć i naprawić w rozsądnym czasie opłaca się poświęcić czas na poprawienie błędnych danych zakładać ich odrzucenie poprzez użycie bezpiecznego algorytmu tracąc cenne dane.

Istnieją różne algorytmy, które nie uwzględniają błędów danych, oprócz wymienionych poniżej należy wspomnieć o algorytmie Lovenberga-Marquardta oraz algorytmie gradientów sprzężonych.

Algorytm wstecznej propagacji błędu

Jeden z najpopularniejszych algorytmów uczenia, jego wprowadzenie pozwoliło na efektywne uczenie wielowarstwowych sieci neuronowych. Sposób działania tego algorytmu opiera się na następujących krokach:

- Analiza sieci od wejść do wyjść, obliczanie sygnałów wyjściowych poszczególnych warstw oraz pochodnych cząstkowych funkcji aktywacji,
- Analiza sieci od wyjść do wejść, następuje zamiana obliczanie różnicy między sygnałami wyjściowymi, a zadanymi, różnica ta jest następnie podawana na wyjście, natomiast funkcje aktywacji zastępowane są pochodnymi obliczonymi w kroku poprzednim,
- Aktualizacja wag sieci, kroki będą powtarzane do momentu osiągnięcia wartości pożądanej [8].

Zmiana wag odbywa się przy użyciu wektora zmiany wag $\Delta\overrightarrow{w}$, obliczanego jak poniżej:

$$\overrightarrow{w}(t+1) = \overrightarrow{w}(t) + \Delta\overrightarrow{w} \tag{2.7}$$

gdzie: t – numer epoki.

$$\Delta\overrightarrow{w} = \eta p_t \tag{2.8}$$

gdzie: η – współczynnik uczenia, p – kierunek uczenia.

Algorytmy gradientowe

Zestaw algorytmów, których zasada działania opiera się na rozwinięciu funkcji w szereg Taylora. Aproksymacja wartości tego szeregu opisana jest poniższym wzorem [8]:

$$E(\overrightarrow{w}+p) = E(\overrightarrow{w}) + [\nabla E(\overrightarrow{w})]^T p + \frac{1}{2}p^T H(\overrightarrow{w})p + ..., \tag{2.9}$$

Algorytmy te nie są jednak pozbawione wad, mimo ich prostoty są powolne i mają tendencję do zapętlania w miejscach zwanych minimami lokalnymi.

2.5.2 Odporne algorytmy uczenia

Szereg algorytmów niewrażliwych na zanieczyszczenie danych. Odporność tych algorytmów opiera się między innymi na zmianie sposobu obliczania funkcji błędu. Wynikiem jest mniejszy wpływ pojedynczych błędów na całe zestawy danych. [8] [9]

Oprócz przedstawionego poniżej algorytmu warte wspomnienia są: TAO – odporny algorytm propagacji wstecznej oraz algorytm propagacji wstecznej RBP.

Odporny algorytm z kryterium LMLS

LMLS (Least Mean Log Squares) jest to kryterium błędu przyjęte w tym algorytmie opisane poniższym wzorem [8]:

$$p(r_i) = log(1 + \frac{1}{2}r_i^2) \tag{2.10}$$

Jest to najprostszy i najczęściej używany odporny algorytm.

2.6 Ważne zjawiska związane z sieciami neuronowymi

2.6.1 Epoka

Jest to jednorazowe użycie wszystkich przypadków uczących (gdzie jako przypadek uczący uważamy zestaw wejścia i wyjścia będącego poprawną odpowiedzią na zadane wejście). Za każdym razem losowana jest inna kolejność wykorzystania przypadków by uniknąć zapamiętania kolejności, następuje także zmiana wag. Zarówno zbyt mała jak i zbyt duża liczba epok może mieć negatywny wpływ.

2.6.2 Przeuczenie

Efekt wywołany różnymi czynnikami polegający na zbyt mocnym dopasowaniu się modelu sieci do zestawu treningowego. Może być efektem zbyt małej liczby danych użytych wielokrotnie lub dużego zestawu, którego elementy są do siebie mocno zbliżone. Inną możliwością jest źle dobrany zestaw treningowy nie oddający prawdziwego zakresu danych. Przeuczeniem możemy też nazwać sytuację gdy sieć nauczyła się mocno reagować na drugorzędne cechy danych sprawiając błędy w podstawowych przypadkach.

2.6.3 Podział przypadków uczących na podgrupy

Jest to główne zadanie podczas przygotowania danych do uczenia sieci w metodzie uczenia z nauczycielem. Użytkownik powinien dokonać podziału na dwie lub trzy podgrupy. Są nimi kolejno:

- Zestaw treningowy – zbiór używany do uczenia sieci,
- Zestaw walidacyjny – zbiór wykorzystywany podczas uczenia do sprawdzania postępu nauki,
- Zestaw testowy – służy do ostatecznej oceny jakości modelu.

2.6.4 Hiperparametry

Wartości zmienne dla zbudowanej sieci, które użytkownik może zmieniać, aby zoptymalizować proces uczenia. Zwykle każda warstwa ma własne hiperparametry.

Rozdział 3

Algorytmy widzenia komputerowego

Szeroka dziedzina widzenia komputerowego dostarcza nam narzędzi w postaci programów i algorytmów, które umożliwiają nam manipulowanie zawartością obrazu. Obecnie algorytmy widzenia komputerowego są wykorzystywane w wielu dziedzinach życia codziennego, miedzy innymi w monitoringu, medycynie, wojsku, robotyce oraz przy produkcji filmów. Głównymi zastosowaniami tych algorytmów jest łączenie ze sobą obrazów, dopasowywanie obrazów medycznych czy dobór stereopar.

Jednym z głównych zadań algorytmów jest znalezienie jak największej liczby punktów charakterystycznych na obrazie. Każdy z punktów nazywany "keypoint"posiada dodatkowo swój deskryptor, który opisuje okolicę tego punktu na obrazie. Każdy z punktów kluczowych ma także odpowiadającą mu skalę, którą możemy zauważyć na obrazie poprzez rozmiar okręgu, którym został otoczony punkt charakterystyczny. Ostatnim ważnym elementem jest wektor kierunkowy, który daje nam możliwość porównywania obrazów mimo ich obrotu. Jest on zaznaczany w postaci promienia okręgu określającego odpowiedni punkt.

Jest wiele algorytmów widzenia komputerowego różnią się aparatem matematycznym i złożonością przez co dają różne wyniki dla tych samych danych wejściowych. Przykładami takich algorytmów są SIFT, SURF, BRIEF, BRISK, ORB oraz KAZE i AKAZE.

3.1 SIFT

SIFT (Scale Invariant Feature Transform) Algorytm zaproponowany przez David G.Lowe'a w 2004 roku. Jest on oparty na przybliżeniu Laplasjanu Gaussów (LoG) nazywanym w skrócie (DoG) Differece of Gaussians – Rożnica Gaussów [10]

Rysunek 3.1: Zaznaczenie punktów charakterystycznych przez algorytm SIFT

Równanie opisujące działanie różnicy Gaussów ma postać:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y), \tag{3.1}$$

Gdzie: G – funkcja Gaussa

Rysunek 3.2: Zaznaczenie punktów charakterystycznych przez algorytm SIFT z uwzględnieniem wektora kierunkowego oraz rozmiarem deskryptora

Zdjęcie użyte jako przykładowe posiada rozmiar 1024×663 piksele. Na pierwszym przykładzie możemy zauważyć dużą liczbę zaznaczonych punktów.

Algorytm SIFT możemy w skrócie opisać za pomocą kilku kroków:

- Detekcja ekstremów przestrzeni skali
- Lokalizacja punktów charakterystycznych
- Skojarzenie orientacji
- Deskryptor punktów kluczowych

3.2 KAZE

Algorytm widzenia komputerowego z 2012 roku przedstawiony przez Pablo F. Alcantarilla. Oparty jest na znormalizowanej skali wyznacznika macierzy Hessego

$$\frac{\partial L}{\partial t} = div(c(x, y, t).\nabla L) \tag{3.2}$$

gdzie: c – funkcja przewodnictwa, div – rozbieżność, ∇ – operator gradientu, L – luminancja obrazu.

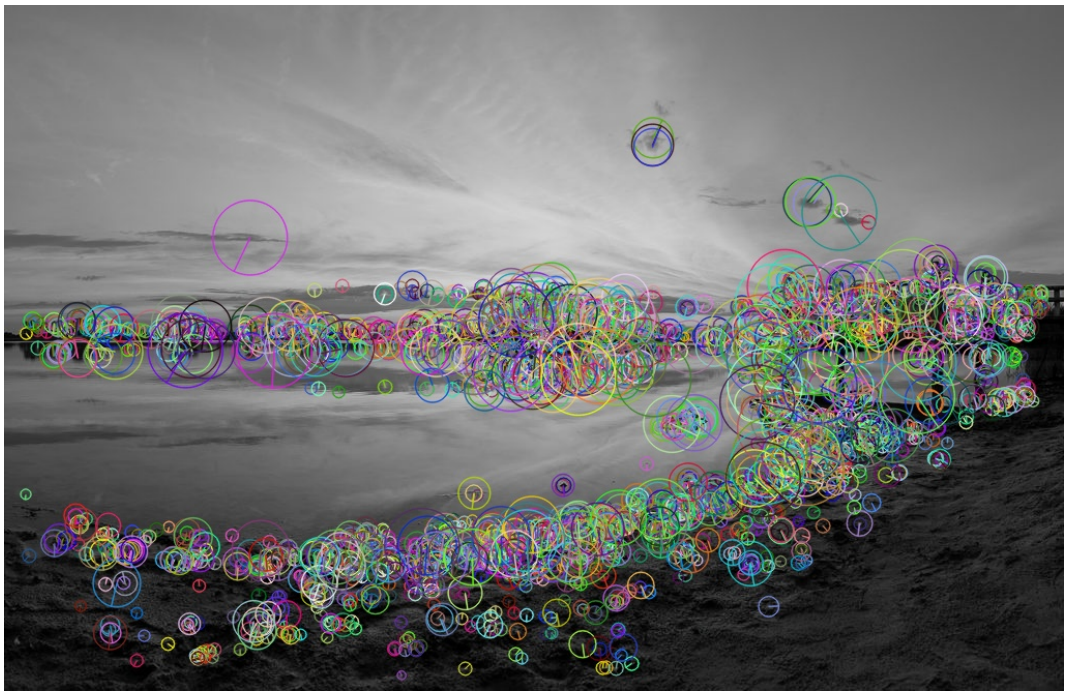
Rysunek 3.3: Zaznaczenie punktów charakterystycznych przez algorytm KAZE

Rysunek 3.4: Zaznaczenie punktów charakterystycznych przez algorytm KAZE z uwzględnieniem wektora kierunkowego oraz rozmiarem deskryptora

3.3 BRISK

Algorytm wykrywania punktów charakterystycznych odporny na zmianę skali i rotację. Nazwa pochodzi od Binary robust invariant scalable keypoints co oznacza binarne odporne niezmienne skalowalne punkty charakterystyczne. Stworzony w roku 2011 jako jego darmowa alternatywa algorytmu SIFT, który mimo początkowej dostępności stał się algorytmem licencjonowanym dla użytkowników komercyjnych. Oparty jest o algorytm przestrzenno-oktawowy FAST [11].

Rysunek 3.5: Zaznaczenie punktów charakterystycznych przez algorytm BRISK



Rysunek 3.6: Zaznaczenie punktów charakterystycznych przez algorytm BRISK z uwzględnieniem wektora kierunkowego oraz rozmiarem deskryptora

Rozdział 4

OCR

OCR to akronim, pod którym kryje się optyczne rozpoznawanie znaków (Optical Character Recognition). Rozwiązanie to może mieć wiele zastosowań użytku przemysłowego, specjalnego oraz medycznego jak i codziennego. Przykładami narzędzi używających rozpoznawanie znaków są:

- aplikacje wykrywające tekst w czasie rzeczywistym z kamery telefonu, na przykład Google Tłumacz oraz Aplikacja wbudowana w telefonach Huawei,
- programy wykrywające tekst w obrazach,
- narzędzia do wykrywania tekstu w plikach, w których tekst występuje w postaci zdjęć, bloków lub innej formie uniemożliwiającej dostęp do jego edycji.

Mimo, że systemy korzystające z algorytmów OCR istnieją już wiele lat, nadal istnieje duży niewykorzystany potencjał rozwoju tych systemów, a także sposobu ich użycia. Postępująca obecnie cyfryzacja wymaga przepisywania tekstu zawartego w książkach lub innych formach piśmiennych na komputer. Proces ten mógłby zostać zautomatyzowany poprzez zeskanowanie źródeł pisanych, a następnie użycie algorytmów rozpoznających znaki.

Kolejnym przykładem użycia OCR może być lub już jest aplikacja dla osób niewidzących. Zamontowanie kamery na przykład w okularach, a następnie połączenie ich ze słuchawką przekazującą odczytany tekst wprost do ucha.

Innym przykładem byłby system kamer miejskich kontrolujących na bieżąco położenie samochodów poprzez odczytywanie tablic rejestracyjnych i numerów znajdujących się za szybą. Takie rozwiązanie mogłoby zaoszczędzić cenny czas podczas pościgów policyjnych lub poszukiwaniu skradzionych aut.

4.1 Metoda deterministyczna

Metoda rozwiązania problemu polegającego na stworzeniu prostego programu OCR przy użyciu języka Python oraz algorytmów widzenia komputerowego omówionych w poprzednim rozdziale. Mimo, że celem pracy jest rozwiązanie problemu przy użyciu sieci neuronowej to w pierwszej kolejności została podjęta próba użycia istniejących już narzędzi i zaprzęgnięcia ich do rozwiązania zadania w celu późniejszej analizy i porównania efektywności.

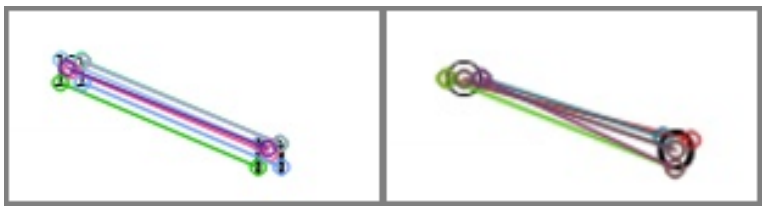
Algorytmy widzenia komputerowego są dostępne poprzez bibliotekę OpenCV, która umożliwia korzystanie z nich w językach Python oraz C++. Język Python natomiast został wybrany ze względu na popularność tego języka w dziedzinie sieci neuronowych i elastyczność przetwarzania danych.

4.1.1 Porównanie algorytmów

Pomimo dużej liczby dostępnych algorytmów oraz dostępnych dokumentacji i badań nie było jasnych przesłanek, który algorytm jest najbardziej odpowiedni do wykonania tego zadania. Dlatego też został przeprowadzony szereg badań mających na celu wybór algorytmu, a także optymalizację jego parametrów.

Badania i wyniki

Pierwsze badanie miało na celu sprawdzenie jakie algorytmy są dostępne i darmowe poprzez bibliotekę OpenCV. Dostępne algorytmy zostały sprawdzone za pomocą krótkiego testu na pięciu pierwszych literach alfabetu, test sprawdził jak radzą sobie z rozpoznawaniem znaków. Użyty został zestaw liter z zakresu a-e oraz trzech różnych fontów różniących się od siebie typem. Fonty możemy podzielić na szeryfowe (serif), bezszeryfowe, "slab serif", skryptowe oraz odręczne. Dodatkowo mogą one występować w różnych wersjach na przykład pogrubione lub zapisane kursywą, jednak to będzie przedmiotem późniejszych badań.



Rysunek 4.1: Przykład poprawnego (po lewej) i mogącego sprawić problem (po prawej) porównania

Rysunek 4.2 graficznie przedstawia sposób porównania przez algorytmy.

Tabela 4.1 przedstawia wyniki dwóch badań. Pierwsze sprawdza trafność algorytmów dla domyślnej wartości ratio-tresh. Natomiast druga część tabeli pokazuje

wynik dla najlepszej wartości tego parametru po dokonaniu badań za pomocą metody prób i błędów. Wyniki tych badań jasno sugerują, że algorytmy AKAZE oraz ORB nie radzą sobie z rozpoznawaniem tekstu dlatego zostały odrzucone podczas dalszej części badań. Z pozostałych czterech odrzucono również brief ze względu na identyczne wyniki jak w przypadku algorytmu BRISK.

Tabelę 4.1 należy czytać w następujący sposób:

- kolor zielony (1) wszystkie rozpoznane punkty sobie odpowiadały,
- kolor żółty (-1) rozpoznane punkty nie zawsze sobie odpowiadały,
- kolor czerwony (0) brak rozpoznanych punktów lub błędnie rozpoznane punkty charakterystyczne przeważały nad poprawnymi.

Tablica 4.1: Tabela dotycząca sprawności znanych algorytmów w wyszukiwaniu liter na obrazie przy użyciu metody punktów charakterystycznych

Zestaw kontrolny	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial
Użyty algorytm	AKAZE			BRIEF			BRISK			KAZE			ORB			SIFT		
Litera a	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	1	-1	-1
Litera b	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1	0	-1
Litera c	0	0	0	-1	0	-1	-1	0	-1	-1	0	-1	0	0	0	1	-1	-1
Litera d	0	0	0	1	-1	-1	1	-1	-1	1	-1	-1	0	0	0	1	-1	-1
Litera e	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	1	1	-1	-1	-1
	z algorytmem filtrującym Loew'a oraz ratio_tresh = 0.7																	
Litera a	brak możliwości użycia algorytmu filtrującego Loewa			1	0	0	1	0	0	1	-1	1	brak możliwości użycia algorytmu filtrującego Loewa			1	0	0
Litera b				1	1	0	1	1	0	1	-1	1				-1	0	1
Litera c				1	0	0	1	0	0	1	0	1				1	0	0
Litera d				1	0	0	1	0	0	1	1	1				1	0	1
Litera e				1	0	0	1	0	0	1	0	0				-1	0	0

Na podstawie wybranych algorytmów oparte zostało trzecie badanie mające na celu ich optymalizację. Wartością, którą należało poddać usprawnieniu był dystans (distance) czyli wynik, za pomocą którego przedstawiana jest miara podobieństwa między dwoma porównywanymi punktami charakterystycznymi.

Wartość parametru distance należy rozumieć tak, że bliskość podobieństwa jest odwrotnie proporcjonalna do wartości współczynnika. Zatem najlepszym dopasowaniem punktów jest przypadek gdy parametr jest równy 0. Zadaniem badania natomiast było ustalenie maksymalnej wartości tego parametru aby uzyskać jak najwięcej trafień (znalezionych poprawnie podobieństw między literami). Wyniki najlepszego doboru dla każdego algorytmu z osobna zostały przedstawione w tabelach 4.2, 4.3 oraz 4.4.

Tabele 4.2 – 4.4 należy czytać w następujący sposób:

- kolor zielony (1) wszystkie rozpoznane punkty sobie odpowiadały,
- kolor żółty (0) rozpoznane punkty nie zawsze sobie odpowiadały,
- kolor czerwony (2) brak rozpoznanych punktów lub błędnie rozpoznane punkty charakterystyczne przeważały nad poprawnymi.

W wyniku przeprowadzonego badania odrzucono algorytm SIFT ze względu na dużą liczbę błędów mimo optymalizacji jego parametrów.

Kolejne doświadczenia zwarte w tabelach 4.5 oraz 4.6 zostały wykonane za pomocą programu wykorzystującego połączenie algorytmów KAZE oraz BRISK.

Podczas poprzedniego badania zaobserwowano, że nie zawsze dopasowanie z najmniejszą wartością parametru distance poprawnym rozwiązaniem. Dlatego też kolejne badanie miało na celu sprawdzenie działania aparatu matematycznego w postaci średniej arytmetycznej oraz geometrycznej. Wyniki tego badania można zaobserwować w tabeli 4.5, w której jest również zawarte nawiązanie do wartości minimalnej opisanej jako min.

Przyglądając się tabeli 4.5 widzimy, że najwyższe wartości osiągane są dla średniej geometrycznej dla małych populacji. Mimo, że najlepszy wynik dla populacji równej 5 i 7, to jednak ze względu na przypadek słabego wyniku dla populacji równej 6 uznano średnią geometryczną z czterech najlepszych wyników jako najlepsze bezpieczne rozwiązanie.

Gdy mówimy o algorytmach widzenia komputerowego pamiętamy, że część z nich jest odporna na rotację oraz zmianę skali. Jednak nie było pewności czy algorytm z niewrażliwy na rotację będzie w stanie odczytać litery zapisane kursywą (italic) w tekście oraz w przypadku tekstu pogrubionego (bold), które można porównać do zmiany skali, na którą te algorytmy również są odporne.

TABLICA 4.2: Tabela dotycząca sprawności algorytmu SIFT w wyszukiwaniu liter na obrazie przy użyciu metody punktów charakterystycznych po optymalizacji funkcji dystansu

	dla distance <=301								
Użyty algorytm	**SIFT**								
Font bazowy	times new roman			courier new			arial		
Font testowy	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial
a	1	2	2	2	1	2	1	2	1
b	1	2	2	2	1	1	2	2	1
c	1	2	2	2	1	2	2	1	1
d	1	2	2	2	1	1	2	2	1
e	1	2	2	1	1	1	2	2	1
f	1	2	2	2	2	2	2	2	1
g	1	2	2	2	1	2	2	2	1
h	1	2	2	2	1	2	2	2	1
i	1	2	2	2	2	2	2	2	1
j	1	2	2	2	2	2	2	2	1
k	1	2	2	2	1	2	2	2	1
l	1	2	2	2	2	2	2	2	1
m	1	1	1	1	1	2	1	2	1
n	1	2	2	2	1	2	2	2	1
o	1	1	1	1	1	1	1	1	1
p	1	2	2	2	1	2	2	1	1
q	1	2	2	2	1	2	2	2	1
r	1	2	2	2	2	2	2	2	1
s	1	2	2	2	1	1	2	1	1
t	1	2	2	2	2	2	2	2	2
u	1	1	2	2	1	2	2	2	1
v	1	2	2	2	1	2	2	2	1
w	1	2	2	2	1	2	2	2	1
x	1	2	2	2	1	2	2	2	1
y	1	2	2	2	2	2	2	2	1
z	1	2	2	2	2	2	2	2	1

Aby przekonać się jak algorytmy wyposażone w wiedzę o punktach charakterystycznych z liter podstawowych (regular) będą w stanie znaleźć trafienia w literach pisanych kursywą oraz pogrubionych. Wyniki badania, w którym wykorzystano już zoptymalizowany przez średnią geometryczną z 4 elementów, znajdują się w tabeli 4.6.

Wyniki badania uświadamiają, że wykorzystanie liter zapisanych kursywą oraz pogrubionych działa negatywnie na wyniki średniej. Jest to efektem przybliżonego opisu kursywy i pogrubienia gdyż kursywa nie jest czystą rotacją litery, a co za tym idzie niektóre punkty charakterystyczne przestają mieć znaczenie. Podobnie sytuacja wygląda przy foncie pogrubionym tu również punkty charakterystyczne mogą być całkiem inne.

Podczas prac nad ostateczną formą programu wykorzystującego metodę deterministyczną poprzez algorytmy widzenia komputerowego wykonano jeszcze szereg

TABLICA 4.3: Tabela dotycząca sprawności algorytmu KAZE w wyszukiwaniu liter na obrazie przy użyciu metody punktów charakterystycznych po optymalizacji funkcji dystansu

	dla distance <= 0.091								
Użyty algorytm	**KAZE**								
Font bazowy	times new roman			courier new			arial		
Font testowy	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial
a	1	1	1	1	1	2	2	2	1
b	1	2	2	2	1	1	1	1	1
c	1	2	2	2	1	2	2	2	1
d	1	2	2	2	1	1	2	1	1
e	1	2	2	2	1	1	2	1	1
f	1	2	2	2	1	2	2	2	1
g	1	2	2	2	1	2	2	2	1
h	1	2	2	2	1	1	2	1	1
i	1	2	2	2	1	2	2	2	1
j	1	2	2	2	1	2	2	2	1
k	1	2	2	2	1	2	2	2	1
l	1	2	1	2	1	2	1	2	1
m	1	1	2	2	1	2	2	1	1
n	1	2	2	2	1	1	2	1	1
o	1	2	2	1	1	1	1	1	1
p	1	2	2	2	1	2	2	2	1
q	1	2	2	1	1	2	1	2	1
r	1	2	1	2	1	2	1	2	1
s	1	1	1	1	1	1	1	1	1
t	1	2	2	2	1	2	2	2	1
u	1	2	2	2	1	1	2	1	1
v	1	2	1	2	1	1	1	1	1
w	1	2	1	2	1	2	0	2	1
x	1	1	1	1	1	1	1	1	1
y	1	2	2	2	1	1	2	1	1
z	1	1	2	1	1	1	1	1	1

innych badań mających na celu optymalizację metody jednak ich wkład nie był, aż tak znaczący, aby należało je tutaj przedstawić.

Jedynym parametrem, o którym warto jeszcze wspomnieć jest threshold. Jest to zmienna używana podczas przetwarzania obrazu, który zamieniony na skalę szarości jest poddawany binaryzacji. Binaryzacja polega na sprogowaniu obrazu w zależności od wartości każdego punktu obrazu i przypisaniu zera lub jedynki – odpowiednio koloru czarnego lub białego. W skrócie można ten proces opisać jako zamianę obrazu monochromatycznego na obraz czarno-biały Podczas optymalizacji wczytywanych obrazów najlepsze wyniki osiągano dla wartości 245 w skali od 0 do 255. Przy czym badanie przeprowadzono dla zbioru wartości liczb równych wielokrotnością cyfry 5, przybliżenie to wynikało z niskiej wagi problemu oraz oszczędności obliczeniowej.

TABLICA 4.4: Tabela dotycząca sprawności algorytmu BRISK w wyszukiwaniu liter na obrazie przy użyciu metody punktów charakterystycznych po optymalizacji funkcji dystansu

	dla distance <=490								
Użyty algorytm	**BRISK**								
Font bazowy	times new roman			courier new			arial		
Font testowy	times new roman	courier new	arial	times new roman	courier new	arial	times new roman	courier new	arial
a	1	1	2	1	1	2	1	1	1
b	1	1	1	1	1	1	1	1	1
c	1	2	1	2	1	1	1	1	1
d	1	1	2	1	1	2	1	1	1
e	1	1	2	1	1	1	1	1	1
f	1	2	2	1	1	2	1	2	1
g	1	2	1	1	1	1	1	2	1
h	1	1	2	1	1	1	1	2	1
i	1	1	2	2	1	2	2	2	1
j	1	2	1	2	1	1	2	2	2
k	1	1	1	1	1	2	2	1	1
l	1	2	1	1	1	2	2	1	2
m	1	1	2	1	1	2	1	1	1
n	1	2	1	1	1	1	1	1	1
o	1	1	2	1	1	2	1	1	1
p	1	2	2	1	1	1	1	2	1
q	1	1	2	2	1	2	1	1	1
r	1	2	2	1	1	2	1	2	1
s	1	1	1	1	1	1	1	1	1
t	1	2	2	2	1	1	1	1	1
u	1	1	2	2	1	1	2	1	1
v	1	1	2	1	1	1	1	2	1
w	1	2	2	1	1	1	1	1	1
x	1	1	1	1	1	1	1	1	1
y	1	2	2	2	1	2	2	2	1
z	1	1	1	1	1	1	2	1	1

Obserwacje i wnioski

Różny rozmiar tego samego fontu może dawać inny wynik z powodu zmiany skali punktów charakterystycznych.

Podobieństwo liter jest początkowym problemem dla algorytmu na szczęście można go szybko zniwelować poprzez ograniczenie "odległości" pomiędzy punktami charakterystycznymi znajdującymi się na literach do siebie porównywanych. Jednakże podczas wyboru odległości granicznej należy wziąć pod uwagę, że odległość ta nie będzie równa 0 jaki i bliska tej wartości dla różnych czcionek. Litery mogą być do siebie podobne poprzez bycie swoimi odbiciami lustrzanymi na przykład: "p" i "b", "p" i "q", "w" i "m", "b" i "d", "n" i "u", zawieranie się na przykład "c" zawarte w "o" lub "n" i "r" zawarte w "m", innym łącznikiem między literami jest zawieranie podobnych krawędzi na przykład: "c", "e" i "o" lub "l" i "t" oraz "i" i "j" jak i "x" z "y".

TABLICA 4.5: Tabela przedstawiająca porównanie różnych aparatów matematycznych uśredniających wynik parametru distance

Aparat matematyczny	Na 27 możliwych przypadków, gdzie dwa obrazki nie są literami, a brakującą literą jest o
śred. aryt. z 10	18
śred. aryt. z 5	18
śred. aryt. z 3	19
minimum	15
śred. geom. z 10	20
śred. geom. z 8	20
śred. geom. z 7	22
śred. geom. z 6	18
śred. geom. z 5	22
śred. geom. z 4	21
śred. geom. z 3	21
śred. geom. z 2	21

TABLICA 4.6: Tabela ukazująca efekty zmiany zestawu bazowego punktów charakterystycznych

Zestaw liter	Na 27 możliwych, gdzie dwa obrazki nie są literami, a brakującą literą jest o
Bez kursywy i pogrubienia	21
kursywa	20
pogrubienie	20
kursywa i pogrubienie	15

Zarówno pogrubienie jak i kursywa mogą zostać odebrane przez algorytm jako odrębne fonty w szczególności w przypadku cienkich liter oraz fontów z szeryfami. Różne fonty znacznie się od siebie różnią dlatego nie będzie zaskoczeniem problem z wykryciem liter mimo znajomości algorytmu z ich kilkoma konfiguracjami.

Innym problemem jest odległość między literami w różnych fontach. Im ona mniejsza tym większe wyzwanie będzie ona stawiać algorytmowi wyszukiwania kształtów, a co za tym idzie może spowodować niemożliwość powiązania z jakąkolwiek literą. W skrajnym przypadku cały wyraz może zostać uznany za pojedynczy kształt.

Threshold jest kluczowy już na samym początku, musi zostać dobrany tak by ujął jak najwięcej szczegółów znalezionych liter, jednak z drugiej strony powinien "odsiać" zbędne cienie wynikające na przykład ze słabego doświetlenia, zasłonięcia kartki przez osobę robiącą zdjęcie lub błędów drukarki.

Znaki specjalne mogą sprawić dużą trudność ze względu na ich występowanie całościowe lub częściowe w alfabecie. W szczególności takich językach jak polski,

czeski, niemiecki i języki im pokrewne. Przykładem takich powiązań między znakami specjalnymi a literami jest korelacja liter “i” oraz “j”, “ż” z “.” czyli znakiem kończącym zdanie lub podobieństwo znaku przecinka “,” z elementem liter “ć” i “ź”.

Jakość zdjęcia może sprawić, że kluczowe elementy używane jako punkty charakterystyczne mogą zostać utracone, a co za tym idzie szansa na poprawne odszyfrowanie litery będzie niska.

4.1.2 Budowa programu

Program oparty o algorytmy widzenia komputerowego został zaimplementowany tak aby wydobyć optymalne działanie równolegle pracujących algorytmów, które zostały uprzednio zoptymalizowane. Polega to na dodaniu wyników algorytmów KAZE i BRISK do wspólnego wektora danych, a następnie sprawdzeniu, który wynik pojawia się najczęściej. Zastosowanie narzędzia matematycznego w postaci dominanty wynika z faktu, że średnia geometryczna jest stosowana na wektorach podczas każdego porównania.

Baza danych

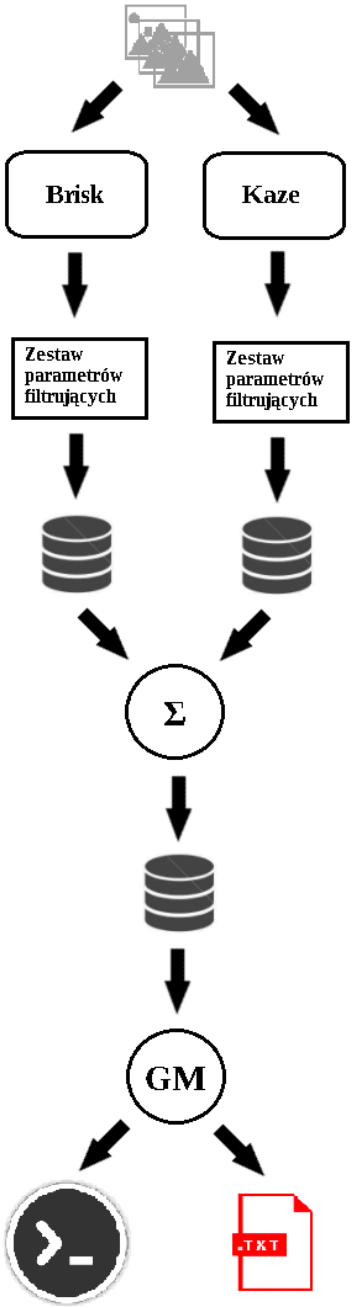
Bazą danych, o którą oparte jest działanie programu jest zbiór liter alfabetów zapisanych w rożnych fontach oraz ich wersjach w kursywie i pogrubieniu. Daje nam to zatem kilkadziesiąt różnych odpowiedzi dla pojedynczego znaku, który jest sprawdzany przez program. Aby lepiej nakreślić liczbę porównań przyjmijmy, że zostało zaimportowanych sześć alfabetów w różnych fontach i w każdej odmianie, daje nam to osiemnaście alfabetów po dwadzieścia sześć liter każdy czyli 468 liter przy czym każda litera jest porównywana przez dwa algorytmy. Czyli każda litera podana programowi na wejście jest porównywana łącznie z 936 razy. Oznacza to duży nakład obliczeniowy.

Baza liter została w całości przygotowana na potrzeby tego programu. Podczas jej tworzenia został zaprogramowany algorytm o roboczej nazwie SCaS – Search, Cut and Save (Szukaj, Wytnij i Zapisz), który stał się też częścią ostatecznego programu. Jego zadaniem jest znajdowanie liter na zadanym obrazie ich wycięcie i zapisanie każdej litery do osobnego pliku o rozszerzeniu .png.

Kod tego algorytmu jest dostępny tutaj:
https://github.com/Kinoruu/SCaS_Algorithm/blob/master/main.py

Wykorzystane biblioteki

Podczas implementacji programu wykorzystano przede wszystkim bibliotekę o nazwie OpenCV w wersji dla języka Python. Była ona niezbędna by móc korzystać z algorytmów widzenia komputerowego, a także działania na obrazach.



Rysunek 4.2: Schemat blokowy programu, gdzie GM oznacza przyjęcie wartości najczęściej występującej

Drugą biblioteką wykorzystaną w tym programie jest NumPy, w celu pracy na obrazach zapisanych w postaci macierzy. Innym zastosowaniem było tworzenie pustych macierzy i wektorów oraz wykorzystanie aparatu matematycznego wbudowanego w tę bibliotekę.

Ostatnia zewnętrzna biblioteka nosi nazwę Pillow i była ona wykorzystana głównie do zamiany przetworzonych już obrazów w postaci macierzy z powrotem do oryginalnej postaci. Funkcja wbudowana w tę bibliotekę umożliwiająca taką transformację nosi nazwę Image.

```
1  import numpy as np
2  import cv2
3  from PIL import Image
```

Rysunek 4.3: Zrzut ekranu zaimplementowanych bibliotek

```
150  gray = cv2.imread(filename='test3.png', flags=cv2.IMREAD_GRAYSCALE)
```

Rysunek 4.4: Zrzut ekranu wykorzystania biblioteki OpenCV do importowania obrazu

```
471              kaze = cv2.KAZE_create()
472              keypoints1, descriptors1 = kaze.detectAndCompute(i, None)
473              keypoints2, descriptors2 = kaze.detectAndCompute(j, None)
```

Rysunek 4.5: Zrzut ekranu wykorzystania biblioteki OpenCV do implementacji algorytmu KAZE

Kod programu

Cały program liczy sobie w momencie pisania tej pracy niecałe 700 linijek. I jest dostępny pod adresem:

https://github.com/Kinoruu/advanced_letters_recognition/blob/master/main.py

4.1.3 Wydajność metody

Duży nakład pracy nad badaniami optymalizacyjnymi oraz stworzeniem własnej biblioteki nie czyni jednak rozwiązania deterministycznego idealnym. Nie można go nawet nazwać wystarczającym gdy jego skuteczność dla pojedynczych liter alfabetu to w przybliżeniu 80% (skuteczność 21 na 26 liter). Jednakże biorąc pod uwagę względną statystykę języka polskiego i angielskiego wydajność tej metody spada do około 60%.

4.2 Metoda sieci neuronowych

4.2.1 Baza danych

Początkowo użyta została rozbudowana wersja bazy danych stworzonej dla metody deterministycznej. Kilkukrotnie ją rozszerzono (przy pomocy wspomnianego już algorytmu SCaS, natomiast fonty pochodziły z bazy wbudowanego w system Windows programu Notatnik oraz zewnętrznego programu do pisania LibreOffice Writer) z 400 do ponad 2000 liter oraz podzielono w sposób losowy na trzy ze-

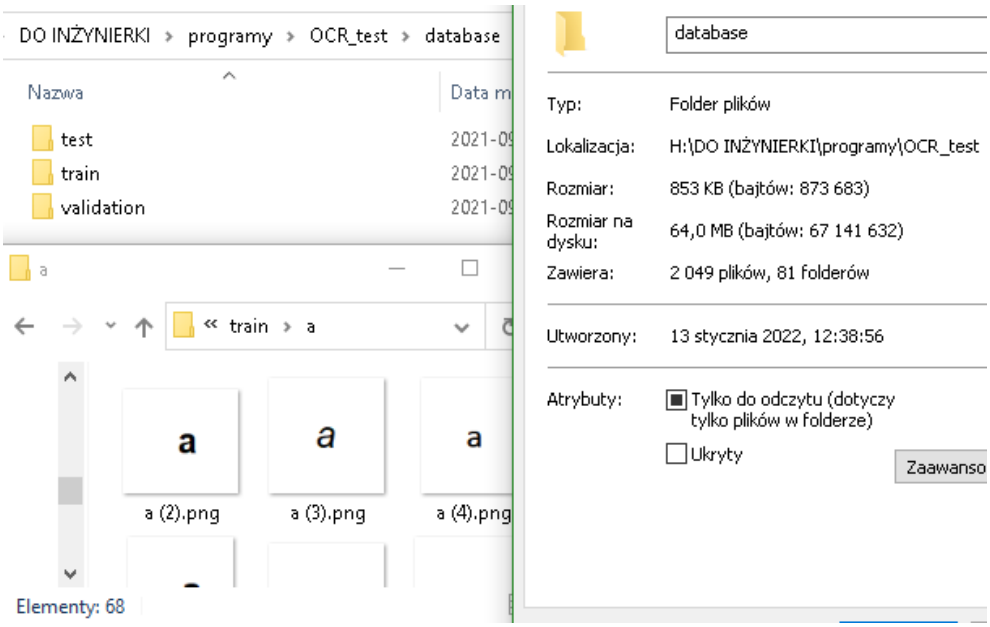
```
151    gray = np.array(gray)
```

Rysunek 4.6: Zrzut ekranu przykładowego wykorzystania biblioteki NumPy

```
372        im = Image.fromarray(im)
```

Rysunek 4.7: Zrzut ekranu przykładowego wykorzystania biblioteki Pillow

stawy w stosunku 68:10:1 (odpowiednio zestaw treningowy, walidacyjny i testowy co można zauważyć na zrzucie ekranu przedstawionym na rysunku 4.8.

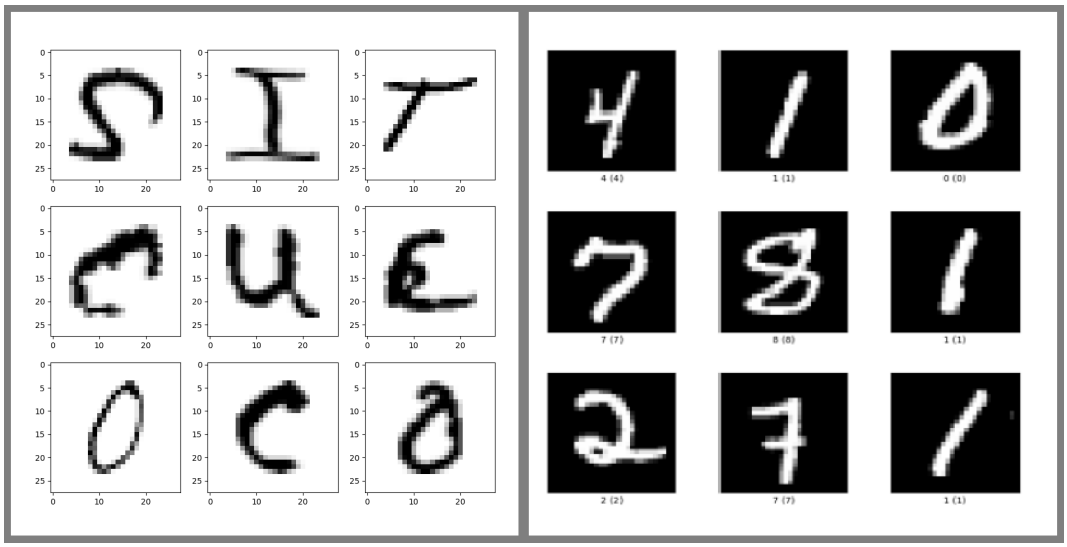


Rysunek 4.8: Wykres przedstawiający liczbę próbek dla liter i cyfr znajdujących się w bazie danych stworzonej na potrzeby tego projektu

Drugim krokiem było użycie ogólnodostępnych baz danych ręcznie pisanych liter oraz cyfr. W pierwszej kolejności podjęta została próba stworzenia sieci neuronowej i jej optymalizacja dla osobnego przypadku korzystając jedynie ze zbioru cyfr. Mimo, że do tej pory programy (wersja deterministyczna oraz sieć neuronowa dla własnej bazy danych) miały na celu rozpoznawanie liter, to ze względu na mniejszy rozmiar gotowej bazy danych podjęto próbę rozpoznawania cyfr. Gotowe zbiory danych charakteryzują się uporządkowaną budową, to znaczy zestaw składa się z obrazu oraz etykiety. Natomiast zestaw stworzony na potrzeby metody deterministycznej nie posiadał etykiet, bazował on na nazwach zdjęć.

Baza danych liter od a do z, z pominięciem polskich liter została pobrana ze strony [12] w postaci pliku .csv. Natomiast zbiór cyfr został zaimportowany z biblioteki TensorFlow z katalogu datasets dostępnego poprzez bibliotekę Keras [13].

Na rysunku 4.9 przedstawione zostały przykładowe wystąpienia z obu zbiorów danych.



Rysunek 4.9: Przykładowe próbki z zestawów liter i cyfr

Implementacja tych zbiorów w programie została ukazana na rysunkach 4.10 oraz 4.11.

```
50      data_az = pd.read_csv("A_Z Handwritten Data.csv").astype('float32')
```

Rysunek 4.10: Zrzut ekranu przedstawiający implementację bazy liter w programie

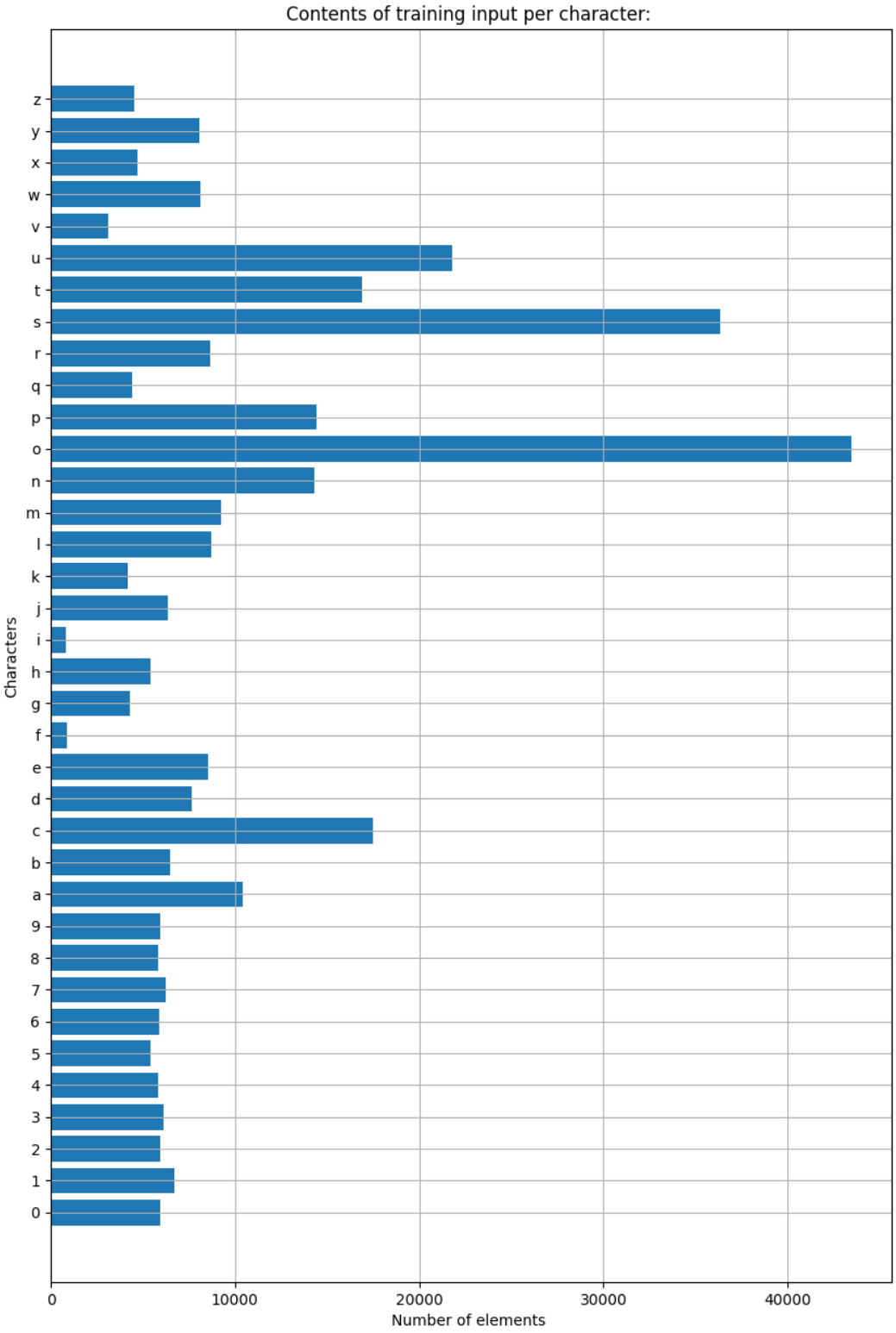
```
78      ((train_data, train_labels), (test_data, test_labels)) = mnist.load_data()
```

Rysunek 4.11: Zrzut ekranu przedstawiający implementację bazy cyfr w programie

Przybliżoną liczbę próbek dla każdej z liter i cyfr można zobaczyć na rysunku 4.12. Zauważyć możemy, że cyfry posiadają zbliżoną do siebie liczbę wystąpień, natomiast rozbieżności liczby liter są kilkukrotne. Wynikać to może z faktu statystyki języka lub trudności wyuczania rozpoznawania danych liter o czym wiedzieli twórcy tej bazy danych.

4.2.2 Wykorzystane biblioteki

Podstawową biblioteką niezbędną do stworzenia sieci neuronowej jest TensorFlow. Na jej miejsce mogła zostać wybrana równie popularna biblioteka o nazwie PyTorch, jednak została odrzucona przez jej mniejszą znajomość. Biblioteka ta udostępnia kluczowy dla uczenia maszynowego katalog Keras, z którego można pobrać



Rysunek 4.12: Rysunek przedstawiający rozkład liczby próbek dla poszczególnych znaków uczących – zbiór po połączeniu omówionym w rozdziale 4.2.4

potrzebne warstwy, z których złożone są sieci. Z niej też pochodzi zestaw danych cyfr.

Biblioioteki NumPy oraz Pandas stanowią podstawę matematyczną programu pozwalając na odczytywanie zestawów danych oraz ingerencję w ich format (liczba wymiarów oraz rozmiar – skracanie, wydłużanie oraz łączenie i dzielenie)

Matplotlib oraz seaborn odpowiadają za tworzenie wykresów, seaborn dodatkowo dzięki bibliotece scikit-learn (sklearn) i wbudowanej w niej funkcji pozwalającej tworzenie macierzy błędu (confusion matrix – macierze te są bardzo przydatne podczas przedstawiania skuteczności działania sieci klasyfikujących) jest w stanie przedstawić graficznie w postaci mapy ciepła jakościową tabelę wyuczenia sieci.

4.2.3 Badania

Sieci neuronowe mimo swojej złożonej architektury nie nadają się do wykonywania różnych zadań, to znaczy, jeśli sieć została zaprojektowania do pewnego celu to nie oznacza, że po podaniu na jej wejście innych danych uczących będzie w stanie nauczyć się poprawnie na nie odpowiadać. Jednak często wystarczy zmienić kilka parametrów tej samej architektury by móc wykorzystać ją do innego, ale zbliżonego zadania. Dlatego też ten rozdział przedstawi pracę nad wyborem i optymalizacją dwóch użytych modeli sieci neuronowej.

Wybór sztucznej sieci neuronowej

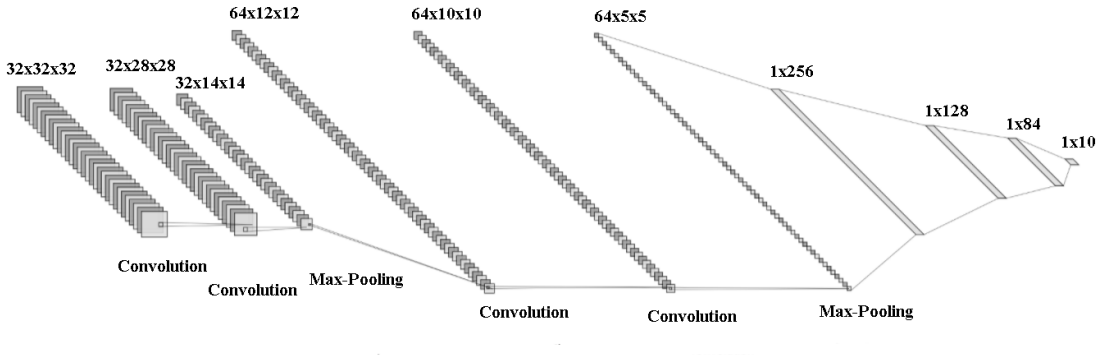
Wiedząc, że zadaniem sieci będzie rozpoznawanie znaków alfanumerycznych na obrazach, wybór był oczywisty mianowicie sieć klasyfikująca splotowa. Ten rodzaj sieci jest przeznaczony w szczególności do nauki na podstawie obrazów i przypisanych im etykiet.

Początkowo stworzona została podstawowa sieć CNN składająca się z 10 warstw (wliczając warstwę wejściową i wyjściową). Model ten został wykorzystany do wyuczenia osobno sieci rozpoznającej cyfry i litery. I mimo optymalizacji nie dawał idealnych rezultatów. W tym momencie pojawiły się dwie opcje, pierwszą było połączenie dwóch wyuczonych modeli w jeden, natomiast drugą było połączenie dwóch baz w jeden zbiór i przy jego pomocy wyuczenie nowego modelu. Metodę łączenia gotowych modeli wykorzystuje się najczęściej do łączenia modeli wyuczonych w tym samym celu i na tym samym zbiorze danych wejściowych. Mimo braku informacji na temat łączenia dwóch różnych modeli, podjęta została próba badawcza. Skończyła się ona niepowodzeniem z powodu błędu związanego z nakładaniem połączeń i ich wag. W związku z tym nie metoda łączenia modeli została porzucona i uznana z niepowodzenie.

Dlatego też wybrana została bezpieczniejsza metoda połączenia baz danych oraz zbudowania nowego modelu. Uzyskany efekt był podobny pod względem jakości do

każdej z poprzednich sieci z tą różnicą, że teraz cała logika była zawarta w jednym modelu sieci neuronowej.

Kolejnym krokiem wynikającym z chęci poprawienia tworzonego projektu było poszukiwanie gotowych sieci CNN ogólnego przeznaczenia lub takich stworzonych specjalnie do rozwiązania zadanego problemu. W ten sposób znaleziona i wykorzystana została sieć LeNet-5 w wersji zaktualizowanej [4]. Mimo wiedzy na temat nowszych sieci splotowych o wysokich dokładnościach dla dowolnych zestawów danych, badania zostały wykonane na zmodernizowanej sieci pochodzącej oryginalnie z 1998. Schemat jej działania jest przedstawiony na rysunku 4.13.



Rysunek 4.13: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

Wybór metody uczenia

Ze względu na typ sieci neuronowej to jest sieci splotowej klasyfikującej, najlepszym wyborem była metoda uczenia z nauczycielem. Wybór ten opiera się na tym, że w sieciach klasyfikujących znamy oczekiwane wyjście zatem zastosowanie uczenia nienadzorowanego (bez nauczyciela) mogłoby okazać się nieskuteczne, a odpowiedzi niespodziewane i nie trafione względem zadanego problemu jakim jest rozpoznanie znaków na obrazie.

Dodatkowo stworzona na potrzeby projektu baza danych jak i gotowe zbiory danych uczących posiadały etykiety (labels) pozwalające na wykorzystanie uczenia nadzorowanego.

Dobór parametrów uczenia

Optymalizacja gotowej architektury sieci neuronowej polega na dobraniu algorytmów wspomagających uczenie oraz dobraniu wartości hiperparametrów oraz innych parametrów sieci.

Dobór optymalizatora, czyli algorytmu aktualizującego wagi po każdej epoce w celu otrzymania jak najmniejszych różnic między wynikiem sieci, a wartością ocze-

TABLICA 4.7: Tabela przedstawiająca wyniki badania porównującego działanie algorytmów optymalizujących

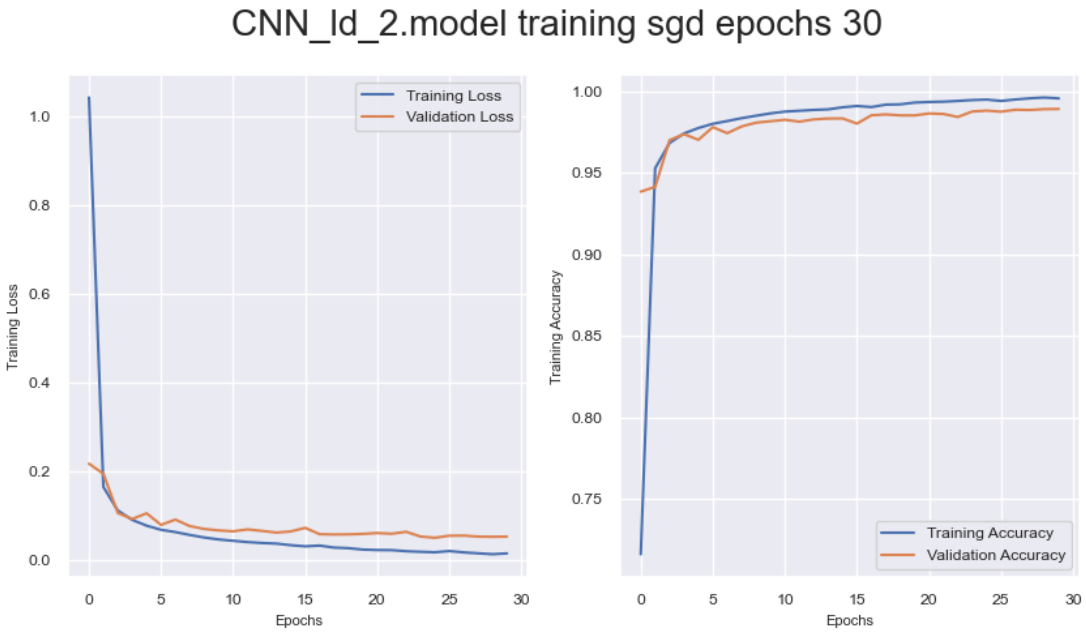
	Algorytmy optymalizujące						
Liczba epok	Adam	SGD	Adadelta	Adagrad	Adamax	Nadam	ftrl
10					98.43	98.41	26.94
20	98.39 (14)	98.66	31.66	87.21	98.87	98.36 (16)	64.61
25	97.92 (13)	98.46		88.53	99.16	98.53 (12)	67.52
30	98.18 (15)	90.80	54.59	89.69	99.07	98.34 (15)	65.25
40			67.95				
50			69.81				
60			74.96				
100			84.57				

kiwaną na wyjściu. Dodatkowo ich zadaniem jest zabezpieczenie przed przeuczeniem oraz działanie mające na celu umożliwienie działania sieci również dla danych wejściowych różniących się od tych zawartych w zbiorze uczącym. Biblioteka TensorFlow oferuje kilka takich algorytmów między innymi: Adam, SGD, AdaDelta, AdaGrad, AdaMax, Nadam, Ftrl.

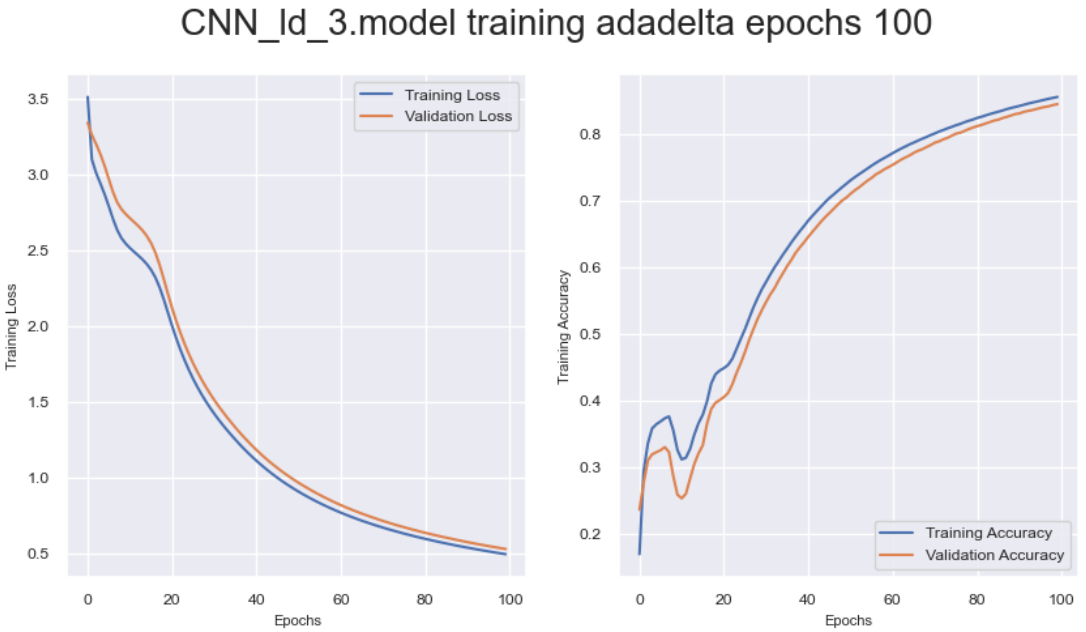
Aby wybrać najlepszy z nich przeprowadzone zostało badanie , wyuczono tę samą architekturę sieci przy użyciu różnych optymalizatorów dla różnych wag. Wyniki zostały przedstawione w tabeli 4.7. Oprócz wyników tabelarycznych możemy też zaobserwować różnicę wyuczania na grafach ukazujących funkcję błędu oraz wzrost dokładności dla każdej kolejnej epoki.

RYSUNEK 4.14: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

Tabela 4.7 oraz rysunki 4.14 – 4.20 jednoznacznie wskazują, że najlepszym wybo-

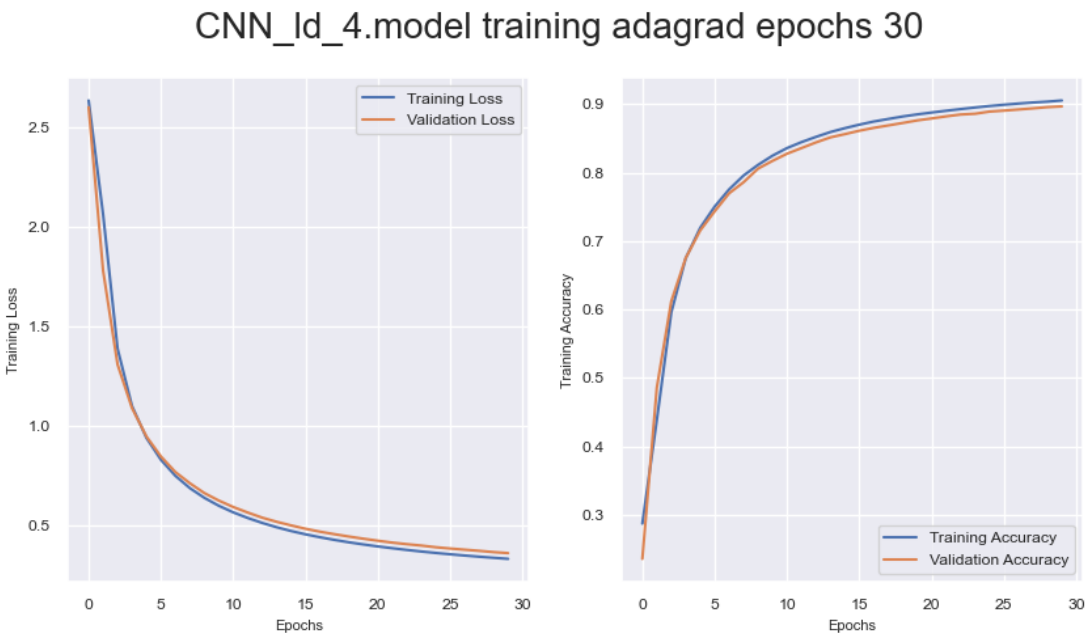


Rysunek 4.15: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]



Rysunek 4.16: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

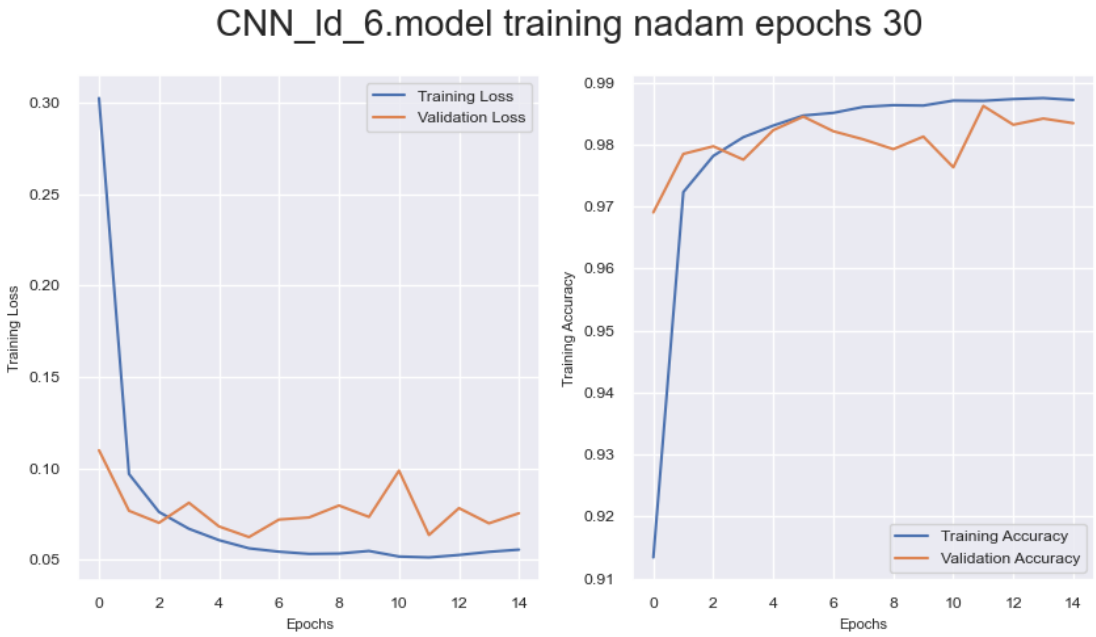
rem byłby algorytm optymalizujący Adamax. Obiecujące wyniki uzyskane zostały również dla algorytmów Adam i Nadam, jednak algorytm reagujący na przeuczenie zakończył proces nauki na epokach widocznych w nawiasach. Jako, że epoka numer 30 ze skutecznością rzędu 99.1% nie oznaczała szczytu możliwości dla tego optymalizatora zdecydowano sprawdzić wyuczanie dla większej liczby epok. Dokładne wyniki zostały przedstawione w tabeli 4.8.



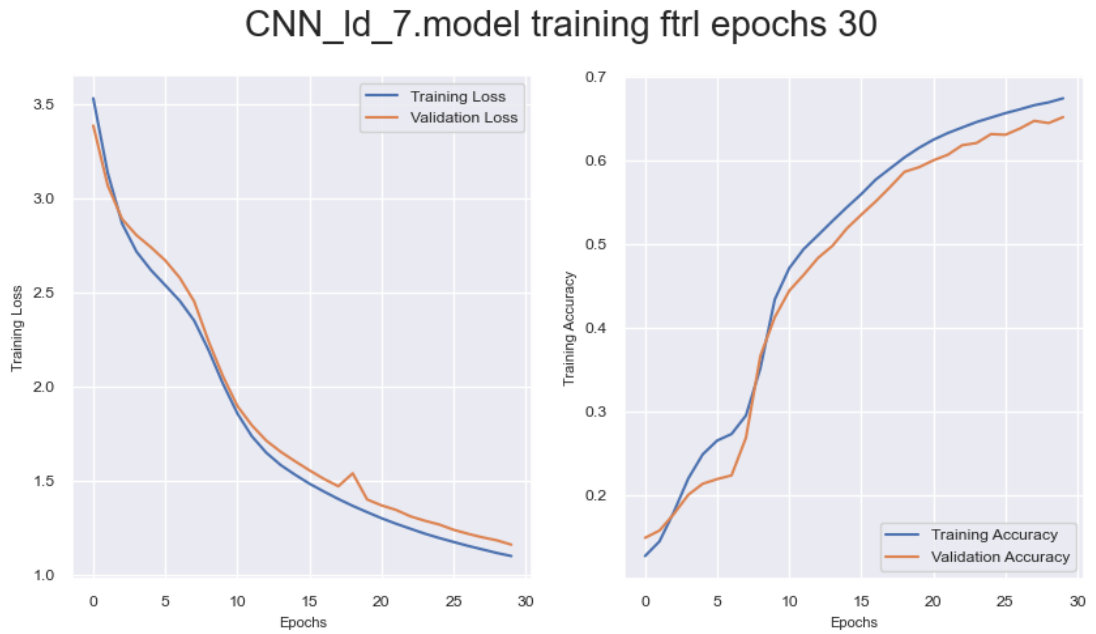
Rysunek 4.17: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

Rysunek 4.18: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

Możemy zauważyć, że odpowiednią liczbą epok jest liczba z zakresu 43-45, jednak przy użyciu algorytmu kończącego naukę po zauważeniu przeuczenia użytkownik może zwiększyć margines błędu. Rozbieżności wyników są efektem losowości wag jednak możemy zauważyć, że zwiększenie liczby epok o 10-15 podniosło skuteczność średnio o około 0.1% (można to zauważyć porównując tabelę 4.7 i 4.8 dla algorytmu Adamax dla liczy epok 25-30 oraz 40-45), co oznacza, że szansa na błąd wynosi



Rysunek 4.19: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]



Rysunek 4.20: Rysunek przedstawiający architekturę zmodernizowanego modelu LeNet-5 [4]

0.8%. Nadal nie jest zadowalający efekt, gdyż taki błąd niesie za sobą 80 błędnych liter na 10000 występujących w przeciętnym artykule. Niezadowalający wynik oraz fakt, że autor tematu [4] w swojej pracy podał wyniki zbliżone do 99.5% skłoniły do dalszych badań optymalizacyjnych.

Autor artykułu źródłowego podaje sposób na zwiększenie osiągów modelu poprzez dodanie w dwóch warstwach splotowych (pierwszej i trzeciej) parametru ker-

TABLICA 4.8: Tabela przedstawiająca wyniki badania mającego na celu znalezienie optymalnej liczby epok wyuczania

	Algorytm optymalizujący
Liczba epok	Adamax
35	99.17
40	99.23
45	99.22
50	99.14 (44)
55	99.17 (43)
60	99.10 (45)

nel_regularizer i ustawienie dla niego algorytmu L2 oraz wartości parametru lambda równego 0.0005. Jego implementację możemy prześledzić na rysunku numer 4.21.

```
187     # Layer 1
188     model_ld.add(Conv2D(filters=32, kernel_size=5, strides=1, activation="relu", input_shape=(28, 28, 1),
189                         kernel_regularizer=regularizers.l2(0.0005)))
190     # Layer 2
191     model_ld.add(Conv2D(filters=32, kernel_size=5, strides=1, use_bias=False, activation="relu"))
192     # Layer 3
193     model_ld.add(BatchNormalization())
194     model_ld.add(MaxPooling2D(pool_size=2, strides=2))
195     model_ld.add(Dropout(0.25))
196     # Layer 3
197     model_ld.add(Conv2D(filters=64, kernel_size=3, strides=1, activation="relu",
198                         kernel_regularizer=regularizers.l2(0.0005)))
```

RYSUNEK 4.21: Zrzut ekranu przedstawiający część architektury sieci neuronowej po implementacji algorytmu kernel_regularizer

Efektem dodania algorytmu kernel_regularizer, którego zadaniem jest regulacja wag, aby zniwelować możliwość przeuczenia było osiągnięcie wyników jakości uczenia na poziomie 99.5%. Z wyników tego doświadczenia wynika, że algorytmy tego typu pełnią ważną funkcję zapewniającą duży margines błędu dobierania liczby epok i mimo osiągnięcia maksymalnej wartości sieć neuronowa się nie przeucza, a jednocześnie dokładność nie spada. Potwierdzeniem takiej tezy jest zrzut ekranu przedstawiający wyniki wypisane przez program w terminalu widocznym na rysunku 4.22.

Osiągnięcie tak zadowalającego wyniku, oznaczającego zaledwie 50 błędów na 10000, co jest wynikiem bardzo wysokim jak dla sieci neuronowych, o których wiemy, że nie są nieomylne i przy obecnej technologii niemożliwym jest osiąganie 100% skuteczności, w szczególności w zakresie przetwarzania obrazów. Przykładem pochodzącym z uczenia innych sieci klasyfikujących jest przypadek rozpoznawania zwierząt. Gdy na zdjęcie dowolnego zwierzęcia nałożymy filtr imitujący skórę słonia to sieć rozpozna, że na obrazie jest słoń mimo, że kształt będzie należał na przykład do lwa lub królika. Powodem jest sposób działania sieci splotowych, a także fakt, że nadal mamy do czynienia ze słabą sztuczną inteligencją, czyli taką, która działa zgod-

```
90 Epoch 45/50
91 10605/10605 [==============================] - 360s 34ms/
   step - loss: 0.0278 - accuracy: 0.9932 - val_loss: 0.0255
    - val_accuracy: 0.9948
92 Epoch 46/50
93 10605/10605 [==============================] - 361s 34ms/
   step - loss: 0.0275 - accuracy: 0.9932 - val_loss: 0.0289
    - val_accuracy: 0.9939
94 Epoch 47/50
95 10605/10605 [==============================] - 362s 34ms/
   step - loss: 0.0278 - accuracy: 0.9931 - val_loss: 0.0248
    - val_accuracy: 0.9954
96 Epoch 48/50
97 10605/10605 [==============================] - 363s 34ms/
   step - loss: 0.0270 - accuracy: 0.9936 - val_loss: 0.0250
    - val_accuracy: 0.9954
98 Epoch 49/50
99 10605/10605 [==============================] - 367s 35ms/
   step - loss: 0.0268 - accuracy: 0.9934 - val_loss: 0.0252
    - val_accuracy: 0.9952
100 Epoch 50/50
101 10605/10605 [==============================] - 365s 34ms/
   step - loss: 0.0261 - accuracy: 0.9935 - val_loss: 0.0244
    - val_accuracy: 0.9953
```

Rysunek 4.22: Zrzut ekranu przedstawiający wyniki uczenia sieci po zastosowaniu algorytmu kernel_regularizer

nie z określonymi przez człowieka zasadami, natomiast silna sztuczna inteligencja kierowałaby się swoimi regułami a także byłaby zdolna do rozwiązywania różnych problemów, a nie tylko zadań, do których została stworzona.

Pomimo uzyskania wyniku zgodnego z opisem znajdującym się w przytaczanym wcześniej artykule, podjęta została decyzja o wykonaniu ostatniego badania optymalizującego liczbę próbek uczących i jednocześnie czas uczenia wraz z próbą uproszczenia modelu wynikającego z mniejszej ilości danych wejściowych. Doświadczenie to polegało na wykonaniu dwóch czynności – przycinaniu oraz powielaniu. W pierwszej kolejności wykonano przycinanie do największej możliwej, równej liczny próbek dla każdego znaku. Po sprawdzeniu okazało się, że jest to liczba 800 próbek na znak. Kolejnym krokiem było przycinanie tej serii próbek co 100 w dół aż do liczby 100 próbek na znak. Następnie wykonano przycinanie bez wypełniania brakujących elementów dla maksymalnej liczby próbek 1000, 2000, 5000 oraz 10000 (opisane w taeli 4.9 nawiasem z opisem "bez"). Ostatnim krokiem było wykonanie przycinania z wypełnieniem dla tych samych wartości granicznych długości zbiorów oraz dla przypadku równego 900 (opisane w tabeli 4.9 nawiasem z dopiskiem "z"). Tabela ta przedstawia porównanie liczby epok, wydajności metody, oraz czasu wyuczania.

Tablica 4.9: Tabela przedstawiająca wyniki badania porównującego działanie przycinania i powielania próbek ze zbiorów uczących

Liczba próbek na znak	all	10000 (z)	10000 (bez)	5000 (z)	2000 (z)	2000 (bez)	1000 (z)	1000 (bez)	900 (z)	800	700	600	500	400	300	200	100
Sprawność (najlepszy moment bez Przeuczenia)	99,50	99,38	99,36	99,23	98,88	98,65	98,50	98,49	98,32	98,35	98,05	97,95	97,65	97,50	97,39	96,81	94,07
Liczba epok	50	45	50	48	49	41	47	44	42	47	50	50	48	44	47	46	50
Czas wyuczania[s]	18600	19063	13695	9983	4523	3791	2881	2520	2340	2346	2347	2135	1880	1555	1500	1297	1219
Czas wyuczania[min]	310,00	317,72	228,25	166,38	75,38	63,18	48,02	42,00	39,00	39,10	39,12	35,58	31,33	25,92	25,00	21,62	20,32

Tabela ta jasno daje do zrozumienia, że zarówno przycinanie jak i powielanie nie przynosi efektów dla zadanego zbioru, jednak jeśli nie zależy nam na dużej dokładności to możemy zmniejszyć koszty obliczeniowe na rzecz niewielkiego spadku jakości modelu. Dlatego włączając ostatnie badanie uznaje się, że metoda z użyciem całego zbioru uczącego daje najlepsze rezultaty.

4.2.4 Kod programu

Kod programu podzielony jest na kilka sekcji, ale znajduje się w jednym pliku.

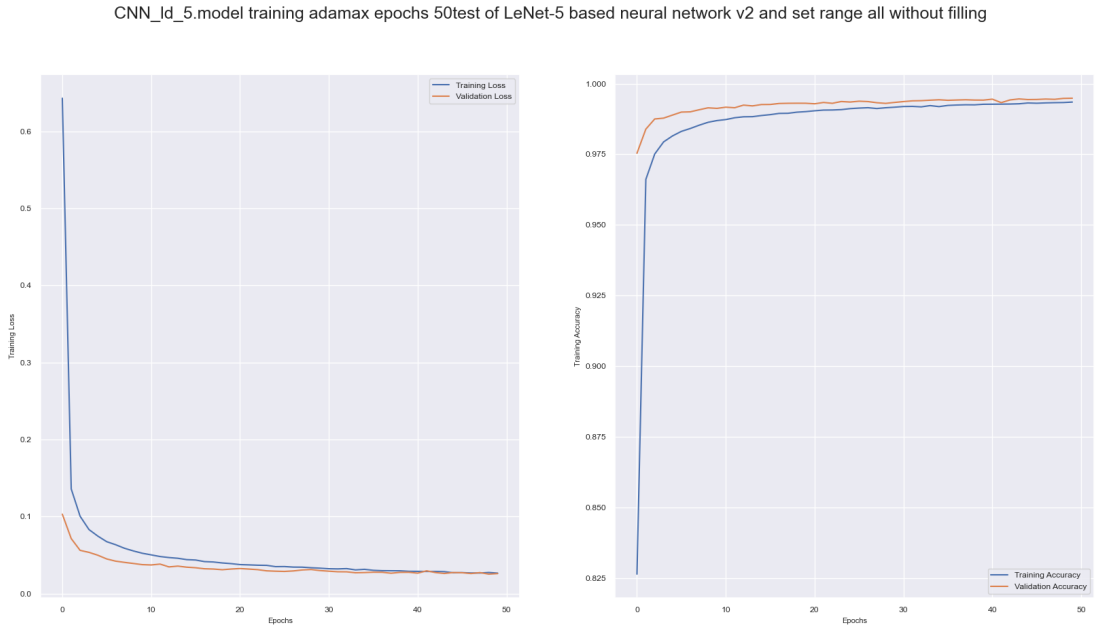
- Pobieranie bibliotek – zawartość opisana już w rozdziale 4.2.3, ten blok kodu znajduje się w linijkach 1-39,
- funkcja wczytująca zestaw uczący złożony z liter, nosi nazwę: load_az_dataset i zawiera się między liniami 46, a 71, nie przyjmuje żadnych atrybutów, zwraca cztery zbiory dwa treningowe oraz dwa testowe, po jednym z obrazami i etykietami,
- funkcja wczytująca zestaw uczący złożony z cyfr, nosi nazwę: load_zero_nine_dataset i zawiera się między liniami 74, a 84, nie przyjmuje żadnych argumentów, zwraca cztery zbiory dwa treningowe oraz dwa testowe, po jednym z obrazami i etykietami,
- sekcja kodu odpowiadająca za agregację zbiorów w jeden zbiór uczący, zajmuje 17 linijek począwszy od linii 88,
- funkcję odpowiedzialną za przycinanie ilości danych (wspomniane w rozdziale 4.2.4), nosi nazwę cut_contents_of_each_element, przyjmuje trzy atrybuty: parę zbiorów próbek (obrazów i etykiet) oraz wartość liczbową określającą rozmiar, do którego mają zostać przycięte zestawy wejściowe, funkcja zajmuje linijki od 114 do 123 i zwraca parę zbiorów,
- wypełnianie (powielanie próbek w zestawach danych – wspomniane w badaniach opisanych w rozdziale 4.2.4), funkcja zawierająca się w linijkach 126-151, odpowiedzialna za zwiększanie rozmiaru zbiorów uczących poprzez losowe powielanie odpowiednich próbek w tych zbiorach, przyjmuje takie same atrybuty jak funkcja odpowiedzialna z dodatkiem dwóch zbiorów będących wynikiem funkcji przycinającej, funkcja ta zwraca również parę zbiorów,
- funkcja odpowiedzialna za tworzenie grafu przedstawiającego liczbę próbek uczących dla każdego znaku w zbiorze, znajduje się w linijkach 154-167,
- funkcja, w której zwarta jest architektura sieci wraz z wywołaniem jej uczenia oraz tworzenia graficznych odzwierciedleń procesu uczenia,

- na końcu znajduje się wywołanie funkcji uczącej oraz linie kodu odpowiedzialne za pomiar czasu wyuczania niezbędny do przeprowadzenia badania przedstawionego w tabeli 4.8.

4.2.5 Test zbudowanej sieci

Wyniki graficzne

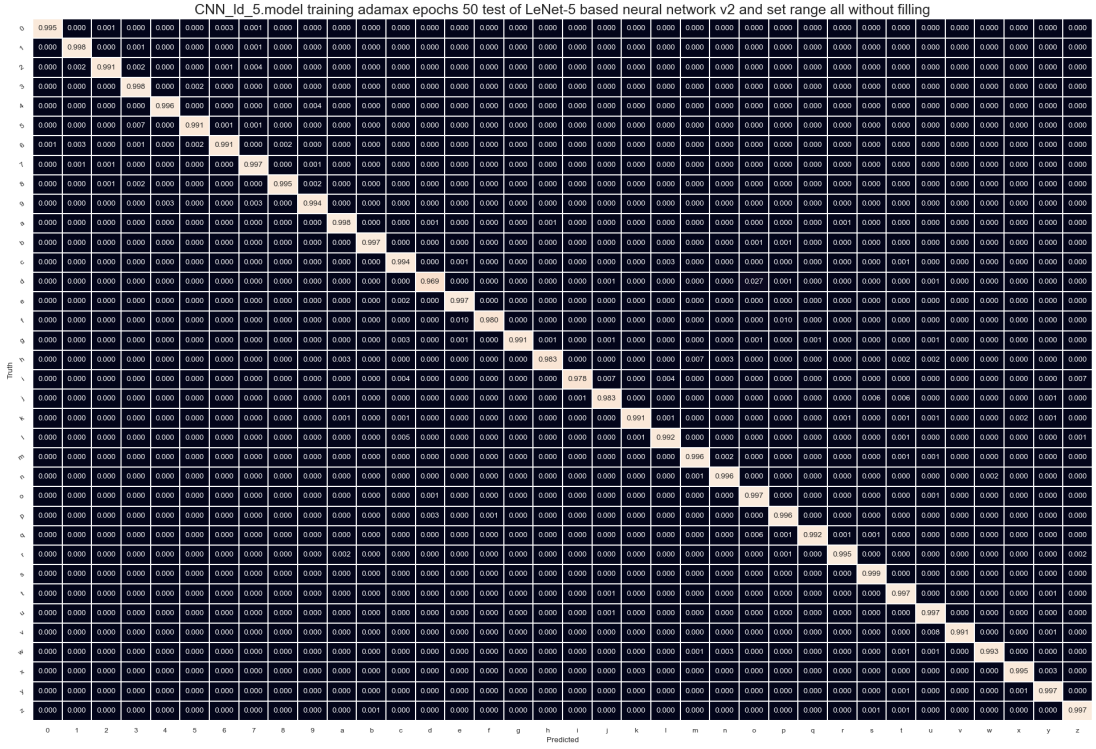
Wyniki podczas uczenia sieci neuronowych można przedstawiać na wiele sposobów. Jednymi z nich są wykresy funkcji strat i dokładności oraz mapy ciepła. Te ostatnie są szczególnie użyteczne podczas testowania sieci klasyfikujących. Wykresy Zostały przedstawione na rysunku 4.23, natomiast mapa ciepła na rysunku 4.24.



Rysunek 4.23: Zrzut ekranu przedstawiający wykresy uczenia ostatecznego modelu sieci neuronowej

Wnioski

Sieć została wyuczona poprawnie, a wynik walidacji (99.5%) jest bardziej niż zadowalający. Dodatkowo w celu oceny faktycznej skuteczności (nie tylko dla wyuczonej bazy danych), został stworzony program testujący model sieci w faktycznym użyciu. Program został napisany w języku Python z myślą o tym by działał jak skrypt zaimplementowany do aplikacji napisanej w języku C# omówionej w rozdziale 5. Wyniki skuteczności dla losowych zbiorów danych znaków alfanumerycznych są niskie (niższe niż zakładane) i wynoszą do 40% skuteczności. Wynika to z faktu, że siec neuronowa była wyuczona na znakach pisanych ręcznie charakteryzujących się cienką linią oraz faktem, że litery drukowane mogą mieć inną budowę.



Rysunek 4.24: Zrzut ekranu przedstawiający mapę ciepła opisującą skuteczność wyuczenia ostatecznego modelu sieci neuronowej

Większość tych trudności została już opisana w rozdziale 4.1.1 w części o nazwie "Obserwacje i wnioski".

Zatem nasuwa się pytanie czy ma sens uczenie sieci neuronowych rozpoznawać tekst jeśli mimo maksymalnej skuteczności dla zbioru uczącego, sieć nie radzi sobie dobrze dla ogólnych przypadków. Odpowiedzią jest "tak", należałoby zwiększyć zestaw uczący o litery drukowane w różnych fontach. Mogłoby się wydawać, że tak słaby wynik jest efektem przeuczenia sieci jednak zastosowane środki zapobiegawcze nie przemawiają za taką konkluzją. Dlatego też najważniejszym wnioskiem płynącym z całości tego badania jest fakt, że sieci neuronowe są bardzo dobrym narzędziem do rozpoznawania obrazów, w tym tekstu na obrazach, jednak są ograniczone poprzez ilość informacji otrzymanych od użytkownika. Jest to też podstawa definicji mocnej i słabej sztucznej inteligencji, przy czym ta pierwsza nie została jeszcze stworzona, natomiast druga jest zależna od wiedzy udzielonej przez człowieka. Co można podsumować słowami: człowiek jest obecnie najsłabszym punktem słabej sztucznej inteligencji.

Rozdział 5

Aplikacja

Program napisany w języku C# przy pomocy środowiska Visual Studio 2019. Głównym jego zadaniem jest obróbka obrazu przy pomocy aparatu matematycznego, filtrów oraz innych form kompozycji obrazu. Wybór języka był kierowany znajomością tego języka oraz łatwością tworzenia panelu głównego programu (Layoutu).

5.1 Layout i funkcjonalność

Program posiada około osiemdziesięciu funkcji związanych z przetwarzaniem obrazu. Podstawowymi funkcjami nie licząc otwierania i zapisywania plików są działania na wszystkich pikselach (punktach obrazu), możemy tego dokonać za pomocą suwaków w oknie RGB.

Wygląd okna programu składa się patrząc od góry z paska narzędzi (ukryta jest tam większość funkcji aplikacji) oraz okna, w którym wczytywany jest obraz otoczonego trzema mniejszymi okienkami zawierającymi suwaki. Dodatkowo na głównym panelu po lewej stronie znajdziemy też suwaki odpowiadające za takie funkcje jak jasność, kontrast i współczynnik gamma. Z prawej strony natomiast skorzystać możemy z takich działań na obrazach jak blur (rozmazywanie), threshold (binaryzacja obrazu), spłycanie kolorów (color shallow – zmniejszanie palety barw), W dolnym lewym rogu znajduje się małe okno podglądu obrazu oryginalnego, natomiast prawy dolny róg umożliwia czyszczenie nałożonych filtrów (None) oraz usuwanie obrazu z podglądu (Clear). Dolny pasek oferuje dwie funkcje, pierwsza jest widoczna po załadowaniu obrazu i wskazuje na rozdzielczość obrazu oraz opisuje piksel wskazany kursorem poprzez jego współrzędne i wartości w skali RGB. Po prawej stronie podczas wykonywania operacji na obrazach widoczny jest pasek postępu koloru zielonego.

Aby w skrócie przybliżyć funkcjonalność aplikacji wymienione zostaną ogólni-

Rysunek 5.1: Zrzut ekranu przedstawiający wygląd głównego okna programu – wygląd po uruchomieniu

kowo najważniejsze dostępne w niej funkcje:

- Grayscale – funkcja zamiany oryginalnego obrazu na odcienie szarości,
- Negatyw – odwrócenie kolorów,
- Sepia – symulacja sepii fotograficznej, czyli tonowania zdjęcia barwą brązową,
- Wykrywanie krawędzi – zamiana oryginalnego obraz na wersję, w której pozostają jedynie krawędzie,
- Lustrzane odbicia – odbijanie połowy obrazu w celu uzyskania symetrii,
- Rotacja – obroty w lewo i prawo o 90 stopni oraz obrót wertykalny i horyzontalny,
- Sklejanie – łączenie ze sobą dwóch wybranych obrazów niezależnie od ich wielkości,
- Algorytm pozostawiający wybrany kolor w formacie oryginalnym, natomiast reszta obrazu zostaje zamieniona na odcienie szarości,
- Możliwość tworzenia podstawowych rysunków przy pomocy myszki,

Dokładniejsze dane na temat aplikacji wraz z przykładami wizualnymi zostały przedstawione w dokumentacji programu, będącej osobnym dokumentem. Dodatkowo kod aplikacji z załączona dokumentacja są dostępne pod adresem:

https://github.com/Kinoruu/Photo_editor

Rozdział 6

Podsumowanie

Celem pracy była implementacja i optymalizacja algorytmu rozpoznawania znaków – OCR przy pomocy dwóch metod i ich porównanie. Założone cele zostały zrealizowane, to znaczy program napisany metodą deterministyczną oraz wyuczenie modelu sieci neuronowej. Wykonane zostały badania optymalizacyjne, których rezultaty były bardziej niż zadowalające. Udało się zrealizować wyuczanie sieci neuronowej przy pomocy podstawowego założonego zakresu znaków. Oznacza to, że projekt ten ma bardzo duże możliwości rozwojowe. Są nimi między innymi nowe zestawy znaków takie jak znaki specjalne oraz interpunkcyjne, litery alfabetu polskiego, alfabety azjatyckie oraz inne istniejące obecnie jak i wymarłe (nieużywane) takie jak pismo klinowe. Jako, że udało się zrealizować rozpoznawanie znaków na obrazie statycznym, jest możliwość rozwinięcia oprogramowania do przechwytywania tekstu w obrazie ruchomym nagranym lub w wersji na żywo (obrazu źródłowego pochodzącego prosto z kamery laptopa lub telefonu. Kolejnym krokiem w stronę większej wygody użytkowania byłoby połączenie skryptu wykonywalnego w języku Python do aplikacji okienkowej napisanej na przykład w języku C# lub w języku Python z wykorzystaniem biblioteki Tkinter umożliwiającej tworzenie programu okienkowego.

Literatura

[1] "Neuron biologiczny." https://fizjoterapeuty.pl/wp-content/uploads/2017/04/uklad-nerwowy-podstawowe-informacje.jpg.

[2] R. Tadeusiewicz and M. Szaleniec, *Leksykon sieci neuronowych*. Projekt Nauka. Fundacja na rzecz promocji nauki polskiej, 2015.

[3] S. Osowski, "Głębokie sieci neuronowe i ich zastosowania w eksploracji danych," *Przegląd Telekomunikacyjny+ Wiadomości Telekomunikacyjne*, 2018.

[4] J. Gupta, "Going beyond 99% — MNIST handwritten digits recognition," May 2020.

[5] K. Smeda, "Understand the architecture of CNN," January 2022.

[6] M. Mamczur, "Jak działają konwolucyjne sieci neuronowe (CNN)," January 2022.

[7] Y. Verma, "A complete understanding of dense layers in neural networks," January 2022.

[8] A. Rusiecki, "Algorytmy uczenia sieci neuronowych odporne na błędy w danych," 2007.

[9] I. Białobrzewski, "Porównanie algorytmów uczenia sieci neuronowej jednokierunkowej, z czasowym opóźnieniem, wykorzystanej do predykcji wartości temperatury powietrza atmosferycznego," *Inżynieria Rolnicza*, vol. 9, 2005.

[10] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018.

[11] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, pp. 2548–2555, 2011.

[12] S. Patel, "A-Z handwritten alphabets in .csv format," February 2018.

[13] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

Spis rysunków

Spis tablic

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.