

**WYDZIAŁ ZARZĄDZANIA**

Katedra Informatyki Biznesowej i Inżynierii Zarządzania

Projekt dyplomowy

*Aplikacja mobilna do nauki matematyki dla szkoły podstawowej z wykorzystaniem machine learningu do rozpoznawania odpowiedzi pisanych ręcznie*

*A mobile math learning app for elementary school using machine learning to recognize handwritten answers*

Autor: *Monika Helena Etrych*

Kierunek studiów: Informatyka i Ekonometria

Opiekun pracy: *dr Beata Basiura*

Kraków, 2023 r.

Spis treści

[Spis rysunków 3](#_Toc134544162)

[Spis tabel 3](#_Toc134544163)

[Wstęp 4](#_Toc134544164)

[1. Rozpoznawanie cyfr i ich interpretacja 5](#_Toc134544165)

[1.1. Interpretacja obrazów przez komputer 5](#_Toc134544166)

[1.2. Metody uczenia maszynowego do rozpoznawania cyfr 6](#_Toc134544167)

[1.3. Obróbka obrazów z cyframi 8](#_Toc134544168)

[1.3.1. Wstępne przetwarzanie obrazów 8](#_Toc134544169)

[1.3.2. Augmentacja 9](#_Toc134544170)

[1.4. Weryfikacja modelu do rozpoznawania cyfr 10](#_Toc134544171)

[2. Model aplikacji do nauki uczenia matematyki 13](#_Toc134544172)

[2.1.1. Założenia modelu aplikacji 13](#_Toc134544173)

[2.1.2. Architektura aplikacji 14](#_Toc134544174)

[2.2. Wybrane technologie zastosowane do budowy aplikacji 15](#_Toc134544175)

[2.2.1. Android 15](#_Toc134544176)

[2.2.2. Java 15](#_Toc134544177)

[2.2.3. Python 15](#_Toc134544178)

[2.3. Wybrane metody uczenia maszynowego zastosowane w aplikacji 16](#_Toc134544179)

[2.3.1. Zestaw danych MNIST 16](#_Toc134544180)

[2.3.2. Augmentacja 17](#_Toc134544181)

[2.3.3. Model i badanie skuteczności 17](#_Toc134544182)

[2.3.4. Przygotowanie danych 17](#_Toc134544183)

[2.3.5. Podział obrazów liczb na cyfry 17](#_Toc134544184)

[2.4. Korzystanie z modelu ML poprzez API HTTP 20](#_Toc134544185)

[3. Opis implementacji i działania aplikacji 21](#_Toc134544186)

[3.1. Ekran startowy i podstawowe funkcjonalności 21](#_Toc134544187)

[3.2. Schematy, opisy poszczególnych funkcjonalności, opis kodów 22](#_Toc134544188)

[3.3. Przykłady zastosowania aplikacji w konkretnych zadaniach 23](#_Toc134544189)

[Zakończenie 24](#_Toc134544190)

[Bibliografia 25](#_Toc134544191)

[Aneks (kod, tabelka z danymi, treść ankiety) 27](#_Toc134544192)

# Spis rysunków

[Rysunek 1.1 Cztery podstawowe reprezentacje pikseli 5](#_Toc134698401)

[Rysunek 1.2 Losowy obrót obrazów 9](#_Toc134698402)

[Rysunek 1.3 Macierz błędu dla dwóch klas 11](#_Toc134698403)

[Rysunek 1.4 Macierz błędu dla wielu klas 12](#_Toc134698404)

[Rysunek 2.1 Działania 13](#_Toc134698405)

[Rysunek 2.2 Zadania tekstowe 13](#_Toc134698406)

[Rysunek 2.3 Diagram klas - Activity 15](#_Toc134698407)

[Rysunek 2.4 Diagram klas 16](#_Toc134698408)

[Rysunek 2.5 Diagram klas 16](#_Toc134698409)

[Rysunek 2.6 Przykładowe dane z zestawu MNIST 18](#_Toc134698410)

[Rysunek 2.7 Rozkład cyfr w MNIST 19](#_Toc134698411)

[Rysunek 2.8 Podział liczby na cyfry 23](#_Toc134698412)

[Rysunek 2.9 Cyfra po dopisaniu pustego pola 24](#_Toc134698413)

[Rysunek 2.10 Liczba po obróbce 24](#_Toc134698414)

[Rysunek 2.11 Schemat architektury aplikacji z modelem ML 26](#_Toc134698415)

# Spis tabel

# Wstęp

1-2 stron, cel pracy, ważny problem, może być hipoteza badawcza, napisać na koniec!

Problem rozpoznawania cyfr – problematyczne, bo każdy ma inny charakter, pochyłość, duże małe. Różnice kulturowe. Pisanie cyfry od góry do dołu lub od dołu.

# Rozpoznawanie cyfr i ich interpretacja

## Interpretacja obrazów przez komputer

Obrazy reprezentowane przez komputer składają z pikseli - kwadratów, które posiadają kilka wartości określających kolor. Wyróżnia się 4 podstawowe reprezentacje: binarną, monochromatyczną (ang. grayscale), RGB i RGBA (rys. 1.1).

Reprezentacja binarna oraz monochromatyczna składają się tylko z jednej wartości dla danego piksela. W przypadku reprezentacji binarnej wartość jest zerojedynkowa, gdzie 0 oznacza, że pixel ma kolor czarny, natomiast 1 to kolor biały. W reprezentacji monochromatycznej wartość jest z zakresu od 0 do 255. W tym przypadku 0 oznacza kolor czarny, 255 - biały, natomiast wartości pośrednie, to odcienie szarości.

RGB zawiera trzy wartości określające zawartość koloru: czerwonego, zielonego i niebieskiego. RGBA posiada dodatkowo jedną wartość, która określa poziom przezroczystości [1].

Rysunek 1.1 Cztery podstawowe reprezentacje pikseli



źródło: <https://maxcandocia.com/article/2016/Apr/06/how-computers-recognize-images/>

Kontrast odgrywa ważną rolę w reprezentacji i rozpoznawaniu obrazów, ponieważ wpływa on na percepcję. Reprezentowany jest jako różnica pomiędzy najjaśniejszym obszarem a najciemniejszym, pozwala na wyraźne odróżnienie obiektu od tła. Wysoki kontrast ułatwia rozpoznawanie kształtów oraz detali, co jest istotne w uczeniu maszynowym. Dzięki wyraźnym obrazom modele uczenia maszynowego mogą lepiej rozpoznawać wzorce, co przekłada się na lepszą skuteczność ich działania. W przypadku niskiego kontrastu kontury dwóch elementów są zbliżonego koloru, co może prowadzić do błędnego rozpoznania dwóch elementów jako jednego [16].

Wymiary obrazu oraz rozdzielczość (ilość pikseli na cal) wpływają na fizyczny rozmiar pliku, a co za tym idzie na złożoność procesów uczenia maszynowego. Im większe obrazy i im bardziej dokładne tym więcej czasu komputer potrzebuje na ich przetworzenie. Jednak obrazy o zbyt małych wymiarach i rozdzielczości mogą tracić ważnych informacje. Dlatego ważne jest zachowanie równowagi pomiędzy minimalizacją czasu uczenia modelu oraz zachowywaniem jak największej ilości potrzebnych danych.

Fizyczny rozmiar obrazu zależy również od ilości informacji o nim, które są przechowywane. Wyróżnia się formaty bezstratne, takie jak TIFF, które przechowują wszystkie informacje o obrazie. Formaty z kompresją stratną, takie jak JPG, PNG czy GIF pozwalają na stworzenie stosunkowo dobrej jakości obrazu przy małym rozmiarze poprzez redukcję niektórych informacji [17].

## Metody uczenia maszynowego do rozpoznawania cyfr

Metody uczenia maszynowego są coraz częściej wykorzystywane do rozpoznawania cyfr w różnych dziedzinach, takich jak bankowość czy usługi pocztowe. Uczenie maszynowe pozwala na automatyczne i precyzyjne rozpoznawanie cyfr. Wiele papierowych formularzy w  sposób szybki i dokładny może zostać przeniesiona do komputera dzięki takim możliwościom. W usługach pocztowych natomiast rozpoznawanie kodów pocztowych znacząco przyspiesza proces sortowania listów i przesyłek [2].

Do rozpoznawania cyfr wykorzystuje się algorytmy uczenia nadzorowanego. W tym podejściu, dane uczące zawierają cyfry wraz z odpowiadającymi im etykietami, czyli poprawnymi wartościami cyfr. Algorytm uczony na takich danych jest w stanie nauczyć się rozpoznawać cyfry na podstawie cech wizualnych, takich jak kształt i proporcje. [1, 3]

Wśród popularnych algorytmów uczenia maszynowego stosowanych do rozpoznawania cyfr znajdują się między innymi konwolucyjne sieci neuronowe (CNN), wielowarstwowe sieci neuronowe (MLP) oraz algorytm wektorów nośnych (SVM). Zastosowanie tych algorytmów wraz z odpowiednio przygotowanymi danymi uczącymi, pozwala na uzyskanie skuteczności rozpoznawania cyfr na poziomie nawet 99% [3, 4].

Algorytm wektorów nośnych polega na maksymalizacji odległości pomiędzy hiperpłaszczyzną, a marginesem przynależności do poszczególnych klas, poprzez zmianę położenia hiperpłaszczyzny. Istnieje wiele możliwych hiperpłaszczyzn, a jej kształt zależy od złożoności problemu - liczby klas do zaklasyfikowania. Kiedy istnieją dwie klasy, to hiperpłaszczyzna może być prostą, dla trzech danych klas wejściowych jest płaszczyzną 2D itd. [5].

Wielowarstwowy perceptron w sieci neuronowej składa się z trzech podstawowych warstw: warstwy wejściowej, warstw ukrytych oraz warstwy wyjściowej. Liczba warstw ukrytych zależy od złożoności problemu. Warstwy składają się z neuronów oraz funkcji aktywacji. Wraz z kolejnymi iteracjami modelu, wagi są aktualizowane tak, aby zminimalizować błąd predykcji. Wagi te określają poziom wpływu każdego neuronu na wynik sieci. Funkcje aktywacji pozwalają na nieliniowe modelowanie złożonych zależności, co z kolei wpływa na lepszą predykcję [6].

W konwolucyjnych sieciach neuronowych można wyróżnić sześć warstw: wejściową, konwolucyjną, aktywacji, łączenia, w pełni połączoną oraz wyjściową. Warstwa konwolucyjna składa się z filtrów i map cech. Filtry są to kwadratowe wycinki obrazu, które wyodrębniają cechy charakterystyczne dla różnych obrazów. Mapa cech jest zbiorem wyodrębnionych cech. Warstwa aktywacji przekształca dane z mapy cech tak aby dodać nieliniowość. Zwykle używa wybranej funkcji np. ReLU. Warstwa łączenia generalizuje cechy, aby zapobiec przeuczeniu. Warstwa w pełni połączona jest używana do stworzenia końcowej nieliniowej kombinacji cech oraz dokonywania przewidywań przez sieć.

W ostatniej warstwie wykorzystuje się optymalizator np. Adam, który jest odpowiedzialny za minimalizację funkcji kosztu. Jest on adaptacyjny, to znaczy dopasowuje wagi podczas uczenia maszynowego [3, 6, 7].

Wymienione trzy metody zostały porównane w 2021r przez zespół R. Dixita. W  procesie wstępnego przetwarzania obrazu wykorzystano tylko podstawowe przekształcenia takie jak normalizacja wartości pikseli, co pozwoliło na zamianę wartości pikseli z przedziału 0-255 na wartości z przedziału 0-1. Ponadto, wartość liczby prezentowanej na obrazie została zamieniona na zmienną kategoryczną. Następnie wybrane metody zostały zestawione pod kątem czasu wykonania oraz dokładności dla danych treningowych i  testowych. Najwyższy wynik dokładności dla danych treningowych osiągnęła metoda SVM - 99,98%, natomiast dla danych testowych – sieci CNN - 99,31% [3].

W innym badaniu z roku 2012r przeprowadzonym przez zespół D.C. Ciresana porównano duże modele wielowarstwowe sieci neuronowych (MLP) z różnymi konfiguracjami liczby warstw oraz neuronów. W rezultacie najlepszy wynik rozpoznawania obrazów cyfr uzyskano dla MLP składającej się z ośmiu warstw i liczb neuronów w warstwach ukrytych: 2500, 2000, 1500, 1000, 500, 10. Proces uczenia modelu obejmował aż 2000 epok, a osiągnięta dokładność wyniosła 99,65%. Warto zauważyć, że cyfry, które zostały źle sklasyfikowane przez model, były trudne do określenia nawet dla człowieka [8].

W 2020r zespół S. Ahlawata podjął próbę dopasowania hiperparametrów do modelu konwolucyjnych sieci neuronowych. W przeprowadzonym badaniu sprawdzono wpływ liczby warstw, rozmiaru kroku (liczby pixeli o jaką przesuwa się filtr), pól recepcyjnych (wycinka obrazu odpowiadającego danej cesze), rozmiaru filtrów (rozmiaru macierzy z wagami), dopełnienia (paddingu), rozcieńczenia (dilution) oraz rodzajów klasyfikatorów. W ramach badań zastosowano kilka metod przygotowania danych takie jak: skalowanie, centrowanie, redukcje szumów. Najlepszy wynik, który udało się osiągnąć to dokładność na poziomie 99,89% używając optymalizatora Adam [9].

## Obróbka obrazów z cyframi

### Wstępne przetwarzanie obrazów

W procesie rozpoznawania cyfr przez model, kluczowe znaczenie ma odpowiednie przekształcenie danych reprezentujących obraz przed ich przekazaniem do modelu. Wstępne przetwarzanie obrazów ma na celu wymagane przygotowanie danych do dalszej analizy. Ponadto zmniejsza czas potrzebny do wytrenowania modelu i zwiększa szybkość otrzymania wyniku. Na przykład, kiedy wejściowy obraz jest stosunkowo duży, zmniejszenie rozmiaru macierzy danych znacząco zmniejsza czas trenowania, bez znacznej utraty dokładności modelu.

Sieci neuronowe wymagają, aby dane wejściowe miały ten sam wymiar, ponieważ sieci te korzystają z macierzy wag, które mają ustaloną liczbę kolumn i wierszy. Przetwarzanie obrazów przez ustawienie stałych wymiarów liczby pikseli jest jednym ze sposobów na dostosowanie rozmiaru obrazów do wymogów sieci neuronowej. Zmiany rozmiaru obrazu można dokonać na dwa sposoby. Pierwszy z nich polega na zmniejszeniu obrazu poprzez centrowanie oraz wycinanie pustego obszaru jeśli obraz jest za duży. Drugi sposób, stosowany w przypadku gdy obraz jest za mały, polega na zwiększeniu rozmiaru przez dodanie pustych pikseli po bokach.

Złożoność modelu można uprościć za pomocą zamiany reprezentacji obrazu na mniej złożoną. Dla kolorowych zdjęć stosuje się zamianę na skalę monochromatyczną, a dla skali szarości zamianę na reprezentację binarną. Podane przekształcenia upraszczają obróbkę jednak w zależności od celu zastosowania danych, modyfikacje obrazu mogą prowadzić do utraty informacji o odcieniach czy kolorach. Jednak w przypadku rozpoznawania kształtów zamiana reprezentacji z monochromatycznej na binarną jest korzystnym zabiegiem, ponieważ pozwala na wyraźniejsze wyróżnienie cech i konturów cyfr [10].

### Augmentacja

W sytuacji, kiedy danych treningowych jest zbyt mało, model głębokiego uczenia może ulec przeuczeniu, czyli zbytniemu dopasowaniu do danych trenujących. Przeuczenie skutkuje niskimi wynikami otrzymanymi na zbiorze testowym na nowych danych. Aby zapobiec temu problemowi, konieczne jest dostarczenie modelowi odpowiedniej liczby danych trenujących. Można to osiągnąć stosując augmentację, czyli przekształcenia istniejących obrazów za pomocą np. losowych zmian rotacji lub jasności. Na rysunku 1.1 w pierwszym rzędzie przedstawiono obrazy cyfr w normalnym położeniu oraz w drugim rzędzie pokazane zostały te same cyfry po zastosowaniu zmian rotacji o losowy kąt z zakresu ± 30 stopni. Augmentacja jest łatwym sposobem na zwiększenie różnorodności danych trenujących i  tym samym poprawę jakości modelu.

Innymi sposobami na zwiększenie liczby danych są losowe zmiany jasności i naświetlenia, czy dodawanie pikseli w losowych miejscach.

Rysunek 1.2 Losowy obrót obrazów

Obraz zawierający tekst

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Te same zabiegi przetwarzania danych mogą być stosowane zarówno w ramach wstępnego przetwarzania danych, jak i augmentacji. Jednakże, wstępne przetwarzanie danych jest stosowane zarówno na danych treningowych, jak i testowych, podczas gdy augmentacja jest stosowana wyłącznie na danych treningowych po wstępnym przetworzeniu [10].

## Weryfikacja modelu do rozpoznawania cyfr

Poziom dokładności stanowi jedną z najprostszych metod oceny jakości modelu. Określa on stosunek poprawnie sklasyfikowanych danych do liczby wszystkich danych. Miara ta jest adekwatna tylko, gdy zbiór danych jest zbilansowany, czyli zawiera równą liczbę próbek dla każdej klasy [11].

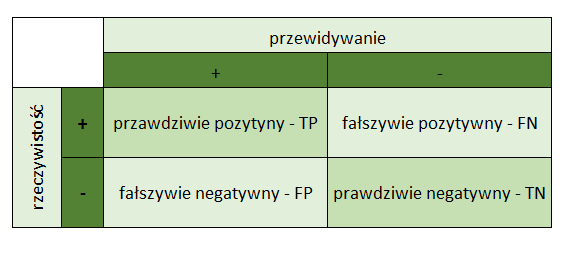
Innym sposobem weryfikacji modelu uczenia maszynowego jest sprawdzenie na jakim etapie nauczenia znajduje się tworzony model. Taki model pod wpływem czasu może być kolejno: niedouczony, wyuczony odpowiednio i przeuczony. Jednak, jeśli model jest źle skonstruowany może nigdy nie osiągnąć późniejszych etapów uczenia.

Problem niedouczenia polega na tym, że hipotezy modelu są nieodpowiednie i zbyt proste w porównaniu do złożoności danych. Przez to sam model nie tłumaczy wystarczająco dobrze danych. Natomiast przeuczenie polega na dopasowaniu modelu do specyficznych zachowań zawartych w danych treningowych nie pozwala uogólnić wyników na nowe przypadki danych. Przyczynami mogą być zbyt zbytnia złożoność modelu, zbyt mała liczba danych treningowych lub zbyt długie trenowanie modelu [12].

Dwa wykresy: wykres dokładności i wykres strat, mogą dostarczyć informacji o stanie wyuczenia modelu. Funkcja dokładności przedstawia poziom dokładności modelu w kolejnych epokach uczenia. Tak długo jak obydwie te krzywe dla danych uczących i testujących rosną model polepsza swoją skuteczność ale model nie jest jeszcze wystarczająco wyuczony. Kiedy funkcje stabilizują się, nie rosną tak gwałtownie mamy model odpowiednio wyuczony. Jednak, gdy funkcja strat na zestawie danych testujących po pewnym czasie zacznie maleć, to oznacza, że model został przeuczony. Celem w uczeniu maszynowym jest maksymalizacja współczynnika dokładności i minimalizacja strat, błędu [13].

Podczas klasyfikacji binarnej można rozróżnić cztery przypadki zaklasyfikowania (rys. 1.3):

Rysunek 1.3 Macierz błędu dla dwóch klas



*źródło: opracowanie własne*

W przypadku gdy klas jest więcej, tak jak w przypadku klasyfikacji dziesięciu cyfr (rys. 1.4), podstawowe miary klasyfikacji są obliczane dla każdej klasy osobno, w rezultacie jest otrzymywanych dziesięć wyników: TP, TN, FP, FN. Przypadki zaklasyfikowania prawdziwie negatywne (TN), fałszywie pozytywne (FP) i fałszywie negatywne (FN) są obliczane jako suma dla wszystkich klas poza prawdziwie pozytywną (TP) [18].

Rysunek 1.4 Macierz błędu dla wielu klas

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Wyróżnia się procentowe wskaźniki, które bazują na wskazanych powyżej (rys. 1.3) wartościach: czułość, specyficzność oraz precyzja. W zależności od oczekiwań danego modelu powinno się maksymalizować wybrany współczynnik. Nie jest możliwe otrzymanie wszystkich współczynników na wysokim poziomie lub na niskim poziomie.

Czułość jest to stosunek poprawnie zaklasyfikowanych wartości jako pozytywne (TP) do sumy wartości prawdziwie pozytywnych (TP) i fałszywie negatywnych (FN): TP/(TP+FN). Jest wykorzystywana, gdy niepożądane jest wystąpienie nieprawdziwie negatywnych wyników klasyfikacji.

Specyficzność to stosunek wyników prawdziwie negatywnych (TN) do sumy prawdziwie negatywnych (TN) i fałszywie pozytywnych (FP): TN/(TN+FP). Jest używana, kiedy ważne jest zwrócenie uwagi na prawdziwie negatywne przypadki.

Precyzja to stosunek prawdziwie pozytywnych (TP) do sumy prawdziwie pozytywnych (TP) i fałszywie pozytywnych (FP): TP/TP+FP. Znajduje ona zastosowanie, gdy ważne są prawdziwie pozytywne wyniki sklasyfikowania [14, 15].

# Model aplikacji do nauki uczenia matematyki

### Założenia modelu aplikacji

Założono, że w oparciu o stworzone pytania program ma być w stanie w prosty sposób wygenerować wiele nowych pytań, tak aby zapewnić użytkownikowi zabawę na dłuższy czas. Postanowiono zastosować metodę polegającą na stworzeniu działań z użyciem zmiennych. Kiedy program ma wyświetlić pytanie użytkownikowi, losuje jedno z działań (rys. 2.1), a następnie losuje każdą ze zmiennych z zadanego zakresu, następnie podstawia liczby w miejsca zmiennych. W ten sposób zamiast: , użytkownik dostanie polecenie: . Przy następnym pytaniu może dostać to samo działanie ale z innymi liczbami.

Rysunek 2.1 Działania

Obraz zawierający tekst, strzała/wskazówka

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

W przypadku zadań z treścią zostanie zastosowana podobna metoda polegająca na rozpisaniu zadania w dwóch linijkach. W jednej zostanie podana treść zadania, w której w miejscu liczb zostanie podana zmienna poprzedzona znakiem ‘#’. W drugiej linii zostanie podane działanie odpowiadające treści zadania (rys. 2.2).

Rysunek 2.2 Zadania tekstowe

Obraz zawierający tekst

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Ustalono, że odpowiedzi użytkownika będą udzielane poprzez ręczne napisanie liczby w przestrzeni do malowania. Następnie program wyśle napisaną liczbę z zapytaniem na serwer, skąd model uczenia maszynowego zwróci napisaną liczbę. Program wyświetli rozpoznaną liczbę i zapyta użytkownika czy chodziło mu o tą liczbę. W przypadku błędu użytkownik będzie mieć możliwość poprawy napisanej liczby i ponownego przesłania. W przypadku akceptacji program sprawdzi czy podany wynik przez użytkownika zgadza się z wynikiem obliczonym przez program oraz wyświetli stosowny komunikat.

Dodatkową funkcjonalnością będzie licznik, który będzie wskazywać ile odpowiedzi zostało udzielonych poprawnie na wszystkie zadane zadania. Ponieważ aplikacja nie ma przeznaczonej grupy wiekowej, będą do wyboru poziomy trudności polegające na wyborze przedziału, z którego będą losowane liczby w pytaniach.

Ważnym wymaganiem będzie również skalowalność aplikacji, tak aby mogła być zastosowana na różnych urządzeniach z Androidem.

### Architektura aplikacji

W programowaniu aplikacji mobilnych na platformę Android, aktywność (activity) oraz widok (view) to dwa kluczowe pojęcia. Aktywności reprezentują poszczególne ekrany, czyli interaktywne interfejsy użytkownika, z którymi użytkownik może wchodzić w interakcję. Z kolei widoki to elementy interfejsu użytkownika, takie jak przyciski, pola tekstowe czy obrazy, które są umieszczane na ekranie. Dodatkowo warto wspomnieć o fragmentach (fragments), które są „pod-ekranami” aplikacji. Można je umieszczać wewnątrz aktywności.

W aplikacji można wymienić podstawowy podział na klasy związane z wyświetlanymi elementami takie jak: MainActivity, MenuActivity, Menu2Activity, SettingsActivity, SettingsFragment, PaintView (rys. 2.3 i 2.4). Istnieją również klasy niezwiązane bezpośrednio z ekranami, takie jak Task i Tasks (rys. 2.5).

Klasy MenuActivity oraz Menu2Activity zajmują się obsługą przycisków w menu, natomiast klasy SettingsActivity oraz SettingsFragment odpowiada za obsługę okna z ustawieniami aplikacji.

Klasa MainActivity pełni rolę obsługi ekranu z zadaniami. W zależności od wybranego przez użytkownika trybu wyświetla odpowiednie pytanie. Po otrzymaniu odpowiedzi od użytkownika, wysyła obraz liczby z zapytaniem na serwer. Program sprawdza poprawność odpowiedzi oraz informuje użytkownika o wyniku. Dodatkowo, klasa zlicza poprawne odpowiedzi użytkownika, co umożliwia śledzenie postępów i wyników.

Rysunek 2.3 Diagram klas - Aktywności

Obraz zawierający tekst, diagram, Czcionka, linia

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Rysunek 2.4 Diagram klas – fragment i widok

Obraz zawierający tekst, zrzut ekranu, Czcionka, paragon

Opis wygenerowany automatycznie

Rysunek 2.5 Diagram klas

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Poza tym w aplikacji znajdują się także klasy Task i Tasks, które odpowiadają za przechowywanie pytania, równania, operacji oraz wyniku. Równanie zawiera wiersz ze zmiennymi oraz znakami jakie operacje zachodzą. Operacja zawiera równanie z podstawionymi liczbami zamiast zmiennych. Pytanie jest to treść wyświetlana użytkownikowi, natomiast wynik jest obliczany na podstawie operacji w celu sprawdzenia poprawności wyniku użytkownika.

## Wybrane technologie zastosowane do budowy aplikacji

### Android

W pierwszym kwartale 2023 roku, Android był najpopularniejszym systemem operacyjnym na urządzenia mobilne, zajmującym 71,95% rynku. Na drugim miejscu znajdował się iOS z udziałem w rynku wynoszącym 27,42%, a pozostałe systemy stanowiły zaledwie 0,63% [1]. Swoją popularność Android zawdzięcza temu, że można korzystać z niego na wielu urządzeniach takich jak: smartfony, tablety, smartwatche, telewizory, samochody, czy chromebooki. Z tego powodu ważnym celem podczas tworzenia aplikacji mobilnych na Androida jest responsywność, tak aby aplikacja dobrze wyglądała na wielu urządzeniach [2].

W trakcie pandemii Covid-19 wzrósł popyt na tablety, nieco zapomniane stały się znowu użyteczne. Przez to teraz wiele dzieci w Polsce może korzystać z nich. Dlatego ta apka jest zajebista [3]. xDDDD

Rysunek 2.6 Rynek tabletów

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Źródło: <https://www.canalys.com/newsroom/canalys-worldwide-tablet-pc-market-Q2-2020>

[1] <https://www.bankmycell.com/blog/android-vs-apple-market-share/>

[2] <https://developer.android.com/multidevice>

[3] <https://www.tabletowo.pl/tablety-sprzedaz-globalna-q2-2020-koronawirus-cov/>

Dużo wersji androida, popularność androida, dużo urządzeń

Android - problemy występujące przy wykonywaniu asynchronicznych operacji – wysyłanie na serwer

### Java

Java a Kotlin

### Pozostałe narzędzia:

* Flask –
* Tensorflow -

Android: <https://developer.android.com/multidevice>

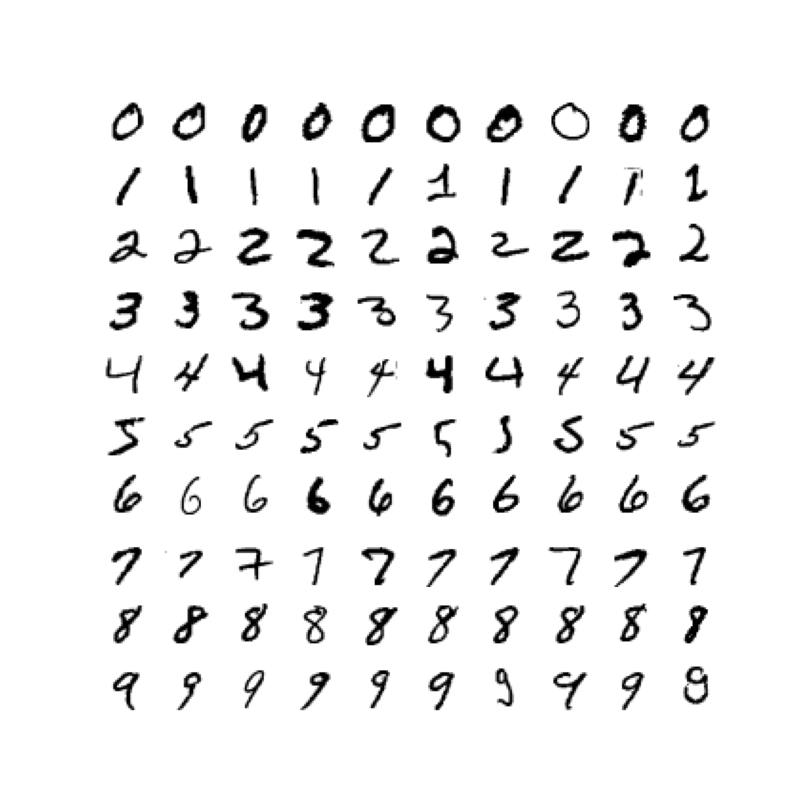
Statystyki o androidzie urządzeniach mobilnych: <https://www.statista.com/topics/1003/operating-systems/#topicOverview>

## Wybrane metody uczenia maszynowego zastosowane w aplikacji

### Zestaw danych MNIST

Model został wytrenowany na zestawie danych MNIST, który zawiera obrazy cyfr ręcznie pisanych przez uczniów oraz pracowników Bureau of the Census(amerykańska agencja zajmująca się spisami ludności). Zestaw składa się z 60’000 danych uczących oraz 10’000 danych testowych [16].

Rysunek 2.7 Przykładowe dane z zestawu MNIST



*źródło: opracowanie własne*

Zestaw danych nie jest zbalansowany, różnica między liczbą wystąpień „1” i „5” wynosi około 2000. Po wykonaniu testu chi-kwadrat p-value wyszło znacznie mniejsze niż 0.05, więc odrzuciłam H0, stwierdziałam, że dane są nie zbilansowane. Jednak to chyba nie ma wpływu na zestaw danych, bo jest wiele innych klas.

Rysunek 2.8 Rozkład cyfr w MNIST

Obraz zawierający wykres

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

### Augmentacja

Zanim augmentacja, to podział danych według proporcji pół na pół danych uczących i testowych. Taki podział wynika z tego, że po dorobieniu danych uczących, będzie ich więcej i proporcje będą 60:40. W ramach augmentacji stworzono 20’000 nowych obrazach poprzez przekształcenia takie jak obracanie, zoomowanie, przesuwanie i shear.

Obrazy są obracane o kąt z zakresu +-30 stopni. Zoom

rotation\_range\_val = 30 # max angle value  
width\_shift\_val = 0.25  
height\_shift\_val = 0.25  
shear\_range\_val = 45  
zoom\_range\_val = [0.5, 1.5]

Augmentacja: rotate, zoom, shift, shear

<https://medium.com/the-data-science-publication/how-to-augment-the-mnist-dataset-using-tensorflow-4fbf113e99a0>

### Model i badanie skuteczności

epoch = 18, batch\_size = 256

zamiana z grayscale na binary representation

rysunek sieci neuronowej: warstwa wejściowa flatten (28, 28)

warstwa ukryta 784, relu

warstwa wyjściowa 10, softmax

Confiusion matrix, accuracy, czułość, specyficzność, precyzja

Rysunek 1.2 Funkcja dokładności i funkcja strat

Obraz zawierający wykres

Opis wygenerowany automatycznie

*źródło: opracowanie własne*

Na (rys. 1.2.) został przedstawiony model odpowiednio wyuczony. Funkcja dokładności osiąga wyższe wartości dla danych uczących niż dla danych testowych. Funkcja strat maleje wraz z upływem czasu, jednak odległość pomiędzy prostymi dla danych uczących i testowych nie zmniejsza się, co świadczy o zbyt małej reprezentatywności danych treningowych. Model zawiera zbyt mało danych uczących [13].

Obraz zawierający wykres

Opis wygenerowany automatycznie Obraz zawierający wykres

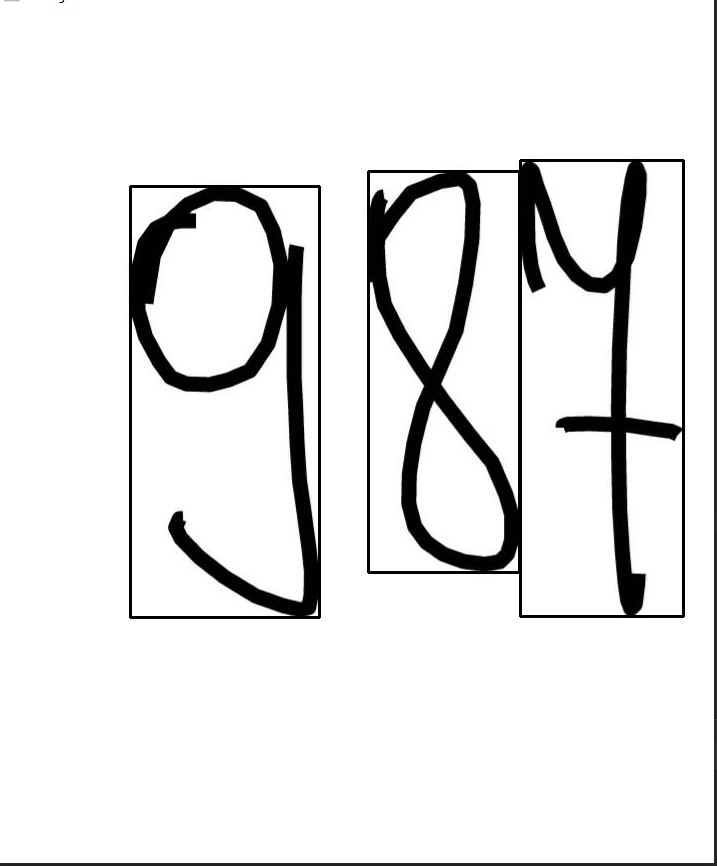
Opis wygenerowany automatycznie

### Podział obrazów liczb na cyfry oraz przygotowanie danych

Przygotowanie danych zastosowano do obrazów, które przychodzą z aplikacji do zaklasyfikowania. Obrazy, które przychodzą są różnych rozmiarów, dlatego ważne jest przygotowanie danych, tak aby przypominały te stworzone w zestawie MNIST. W zaproponowanym projekcie zastosowano trzy funkcje.

Obrazy trafiające do modelu klasyfikującego cyfry zawierają liczby. Program rozdziela liczby na cyfry (rys. 2.4), które kolejno przechodzą etap wstępnego przygotowania.

Rysunek 2.9 Podział liczby na cyfry

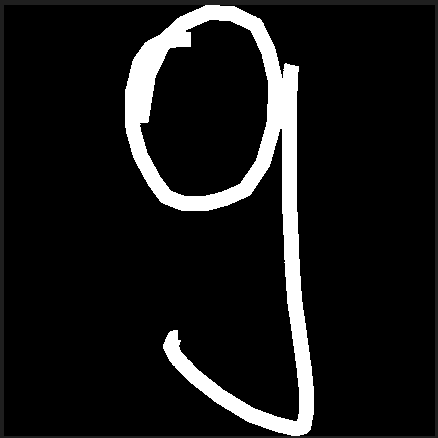


*źródło: opracowanie własne*

Obrazy mają zamienianą reprezentację z RGB najpierw na skalę szarości, a następnie na reprezentację binarną, poprzez użycie metody progowania. Piksele, które mają wartość większą od progu są zamieniane na piksele o kolorze czarnym, a piksele o wartości niższej od progu na kolor biały. W trakcie tego zabiegu zamieniany jest kolor czarnych elementów na białym tle. Po tym zabiegu powstają białe elementy na czarnym tle, jest to wykorzystywane w etapie podziału liczb na cyfry. Dzięki odwróconym kolorom funkcja lepiej znajduje kontury cyfr. Program znajduje najbardziej wysunięte brzegi elementów i następnie rysuje prostokąty. Według prostokątów dzielone są liczby na cyfry.

Następnie sprawdzane są wymiary obrazów cyfr, jeżeli obraz tak jak w przypadku rys(2.3) jest wyższy niż szerszy, to obraz jest uzupełniany pustą przestrzenią po bokach tak aby utworzyć kwadrat.

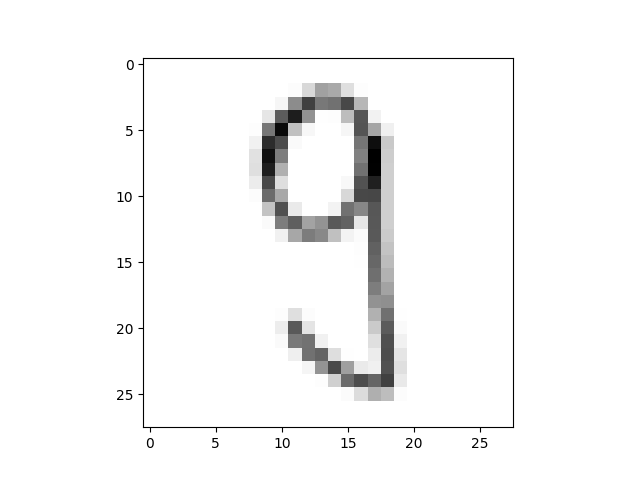
Rysunek 2.10 Cyfra po dopisaniu pustego pola



*źródło: opracowanie własne*

Następnie obraz jest uzupełniany 50 pikselowym pustym polem z każdej strony, tak aby obraz znajdował się w centrum. Na końcu obraz jest przeskalowywany do rozmiaru 28x28. Po przygotowaniu wyglądają jak przedstawione na (rys. 2.5).

Rysunek 2.11 Liczba po obróbce



*źródło: opracowanie własne*

(NOTATKI)

MAE, RMSE, RAE, RRAE, ROC i AUC

GŁÓWNE PYTANIA;

1. Czy można zostawić przeuczony model
2. Czy muszę zrobić crossward validation
3. Co zrobić z danymi nie zbilansowanymi i czy faktycznie są niezbilansowane

## Korzystanie z modelu ML poprzez API HTTP

Aplikacja wykorzystuje komunikację z serwerem aby rozpoznać liczby napisane przez użytkownika. Klient wysyła żądanie HTTP zawierające obraz do rozpoznania oraz informację o potrzebie jego zaklasyfikowania do API udostępnionego na platformie Heroku. Korzysta z metody POST protokołu HTTP do wysyłania danych. API (Flask) otrzymuje żądanie i przekazuje obraz modelowi ML. Następnie model zwraca rozpoznaną liczbę. API odpowiada na żądanie aplikacji przesyłając liczbę w formacie JSON (rys. 2.3).

Rysunek 2.12 Schemat architektury aplikacji z modelem ML



*źródło: opracowanie własne*

# Opis implementacji i działania aplikacji

## Ekran startowy i podstawowe funkcjonalności

Tryby areas itp.

Łatwy, średni, trudny

Pierwszym ekranem z którym użytkownik ma styczność jest

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Prostokąt

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Prostokąt

Opis wygenerowany automatycznie

Obraz zawierający tekst, gadżet, Komunikator, Urządzenie przenośne

Opis wygenerowany automatycznie

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

## Schematy, opisy poszczególnych funkcjonalności, opis kodów

responsywność

Przyznawanie punktów, wybór liczby pytań

Tryb ciemny

## Przykłady zastosowania aplikacji w konkretnych zadaniach

Wpisanie odpowiedzi

# Zakończenie

Wyniki, rezultaty, udowodnienie tezy, co na przyszłość rozwinąć

Zwiększenie trybów gry

Rozpoznawanie liczb ujemnych, dziesiętnych

Kolejność wykonywania działań – nawiasy

Doszkalanie modelu – pytanie użytkownika o jaką liczbę mu chodziło

Statystyki

Część teoretyczna przed grą – np. wzory na pola

Dorobić danych z bardziej polskimi jedynkami i dziewiątkami

# Bibliografia

[1] M. [Candocia](https://www.linkedin.com/in/max-candocia-a270b574/" \t "_blank)*, A Simple Explanation of how Computers Recognize Images,* [Analysis for Many Audiences,](https://maxcandocia.com/)  2016, url: <https://maxcandocia.com/article/2016/Apr/06/how-computers-recognize-images/>, dostęp: 23.02.2023

[2] S. M. Shamim, M. B. A. Miah, A. Sarker, M. Rana, A. A. Jobair, 2018, *Handwritten Digit Recognition Using Machine Learning,* Algorithms Indonesian Journal of Science & Technology (1) vol. 3, str.: 29-39 ISSN 2528-1410, ISSN 2527-8045

[3] R. Dixit, R. Kushwah, S. Pashine, *Handwritten Digit Recognition using Machine and  
Deep Learning Algorithms,* Artykuł Naukowy 2021, url: <https://arxiv.org/pdf/2106.12614.pdf>, dostęp: 1.03.2023

[4] M. Javed, The Best Machine Learning Algorithm for Handwritten Digits Recognition, 2020, url:<https://towardsdatascience.com/the-best-machine-learning-algorithm-for-handwritten-digits-recognition-2c6089ad8f09>, dostęp: 1.03.2023

[5] R. Gandhi, *Support Vector Machine — Introduction to Machine Learning Algorithms*, 2018, url: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>, dostęp: 1.03.2023

[6] J. Brownlee, *Develop Deep Learning Models on Theano and TensorFlow Using Keras,* 2016, str: 37-42*,* 116 – 134

[7] M. Mamczur, *Jak działają konwolucyjne sieci neuronowe (CNN)?,* 2021, url: <https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/>, dostęp: 14.03.2023

[8] D.C. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber, *Deep Big Multilayer Perceptrons For Digit Recognition,* 2012, w: G. Montavon, G.B. Orr, KR. Müller, *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science*, vol 7700, str: 581-598, Springer, Berlin, Heidelberg

[9] S. Ahlawat, A. Choudhary, A. Nayyar, S. Singh, B. Yoon, 2020, *Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN)*

[10] J. Nelson*, What is Image Preprocessing and Augmentation?,* Roboflow, 2020, url: <https://blog.roboflow.com/why-preprocess-augment/>, dostęp: 28.02.2023

[11] N. Bressler, *How to Check the Accuracy of Your Machine Learning Model,* 2022, url: <https://deepchecks.com/how-to-check-the-accuracy-of-your-machine-learning-model/>, dostęp: 23.03.2023

[12] D. Nikolaiev *Overfitting and Underfitting Principles,* 2021,url: *<https://towardsdatascience.com/overfitting-and-underfitting-principles-ea8964d9c45c>,* dostęp: *14.03.2023*

[13] J. Brownlee, *How to use Learning Curves to Diagnose Machine Learning Model Performance, 2019,* url: <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>, dostęp: 9.03.2023

[14] A. Lekhtman, 2019, *Data Science in Medicine — Precision & Recall or Specificity & Sensitivity?,* url: <https://towardsdatascience.com/should-i-look-at-precision-recall-or-specificity-sensitivity-3946158aace1>, dostęp: 29.03.2023

[15] S. Ghoneim, 2019, *Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?,* url: <https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124>, dostęp: 29.03.2023

[16] url: <https://en.wikipedia.org/wiki/Contrast_(vision)>, dostęp: 5.05.2023

[17] T. Kowal, 2014, *Ach te piksele! Teoria obrazu,* url: <https://www.tomaszkowal.pl/index.php?option=com_content&view=article&id=204:ach-te-piksele-czyli-o-rozdzielczosciach-ekranach-i-cyfrowych-obrazach&catid=11&Itemid=57&lang=pl>, dostęp: 6.05.2023

[18] I. Chelliah, 2022, *Confusion Matrix for Multiclass Classification,* url: <https://medium.com/mlearning-ai/confusion-matrix-for-multiclass-classification-f25ed7173e66>, dostęp: 6.05.2023

# Aneks (kod, tabelka z danymi, treść ankiety)

The original images from MNIST were size-normalized to fit a 20×20 pixel box, while the aspect ratio was preserved. As a result, the images contain grey levels (i.e., pixels don’t simply have a value of black and white, but a level of greyness from 0 to 255) due to the anti-aliasing technique used by the normalization algorithm.

Next, the images were centered in a 28×28 pixel image by computing the center of mass of the pixels and translating the image to position this point at the center of the 28×28 field.

<https://datasets.activeloop.ai/docs/ml/datasets/mnist/>