AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
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

Wstęp

Niniejsza praca prezentuje proces budowy, uczenia oraz badania jakości modelu rozpoznawania cyfr pisanych ręcznie. Praca przedstawia również zastosowanie wyuczonego modelu w sposób graficzny za pomocą aplikacji dla dzieci, która służy do nauki matematyki. Wykorzystanie w tej aplikacji funkcji rozpoznawania liczb ręcznie pisanych jest dobrym połączeniem tradycyjnego uczenia matematyki, gdzie uczeń może rozpisać dane zadanie na kroki wykorzystując pole do pisania jako brudnopis. Jednocześnie aplikacja jest stworzona na urządzenia mobilne towarzyszące większości ludzi na co dzień. Dzięki aplikacji uczniowie mają możliwość nauki w dogodnym dla nich momencie, wykorzystując nowoczesne technologie, które przyciągają ich uwagę i mogą zachęcić do nauki.

Rozpoznawanie odpowiedzi ręcznie pisanych jest procesem wymagającym uwagi. Ręczne pismo cechuje się dużą różnorodnością, można wyróżnić pismo różnej wielkości, z różnym stopniem pochylenia. Niektórzy piszą te same cyfry zaczynając od góry, podczas gdy inni od dołu. Na charakter pisma składa się wiele składowych takich jak cechy osobowości czy różnice kulturowe. Nie tylko indywidualne predyspozycje mają wpływ na sposób pisania, ale również to, w jakiej części świata chodziliśmy do szkoły. W zależności od regionu cyfry wyglądają nieco inaczej. Dla przykładu w USA jedynka często zapisuje się jako pionową kreskę.

1. Rozpoznawanie cyfr i ich interpretacja

1.1. Interpretacja obrazów przez komputer

Obrazy reprezentowane przez komputer składają się 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 [2].

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 ważne informacje. Dlatego ważne jest zachowanie równowagi pomiędzy minimalizacją czasu uczenia modelu oraz zachowaniem 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 [3].

1.2. Wielowarstwowy perceptron – MLP

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 [4].

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, 5]

Wielowarstwowy perceptron 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. Neurony są podstawowymi jednostkami przetwarzającymi informacje zawarte w danych. W warstwach wejściowych neurony odbierają dane i przekazują je do warstwy ukrytej. Każdy neuron może odpowiadać danej cesze lub atrybutowi. Neurony w warstwach ukrytych dokonują obliczeń na podstawie dostarczonych danych. Zbierają sygnały dostarczone z poprzedniej warstwy, przemnażają przez wagę, a następnie przekazują wynik do funkcji aktywacji.

Neurony uczą się dobierania odpowiednich wag, tak aby cechy bardziej istotne wpływały w większym stopniu na wynik. Do uczenia wykorzystywana jest propagacja wsteczna błędu, czyli porównywanie wyników sieci z wartościami oczekiwanymi i aktualizowaniu wag tak aby zminimalizować błąd predykcji. Kolejne iteracje tego procesu są nazywane epokami. W uczeniu maszynowym ważny jest odpowiedni dobór liczby epok, tak aby model został wystarczająco wyuczony, ale nie został przeuczony.

W uczeniu modelu istotny jest dobór rozmiaru partii, czyli liczby próbek, które będą przetwarzane przez model jednocześnie w trakcie jednej iteracji. Większe rozmiary partii wpływają na lepsze uogólnienie problemu, jednak wymagają większej ilości pamięci do dokonywania równoległych operacji. Przy większym rozmiarze partii istnieje większe prawdopodobieństwo, że wagi będą bardziej stabilne, ponieważ są wyliczane na podstawie większego zbioru danych. W przypadku mniejszego rozmiaru partii, istnieją częstsze, często chaotyczne aktualizacje wag, mniejszy zbiór danych może różnić się znacząco. Częstsze aktualizacje mogą być pomocne w wychodzeniu z minimów lokalnych, jednak mogą wymagać więcej iteracji do wyuczenie modelu.

Funkcje aktywacji pozwalają na nieliniowe modelowanie złożonych zależności, co z kolei wpływa na lepszą predykcję. Jedną z najczęściej stosowanych funkcji w warstwie ukrytej jest ReLU, a w warstwie wyjściowej Softmax. ReLU zamienia wartości ujemne na zero, a dodatnie pozostawia bez zmian. Dzięki temu nieistotne cechy nie są przekazywane dalej. Natomiast Softmax zwraca wektor prawdopodobieństwa przynależności do danej klasy [6].

W 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 [7].

1.3. Inne metody uczenia maszynowego do rozpoznawania cyfr

Wśród innych 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% [5, 8].

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. [9].

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 [5, 6, 10].

Wymienione dotąd 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% [5].

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 [11].

1.4. Obróbka obrazów z cyframi

1.4.1. 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, co 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 cyfr zamiana reprezentacji z monochromatycznej na binarną jest korzystnym zabiegiem, ponieważ pozwala na zmniejszenie złożoności obliczeniowej oraz na wyraźniejsze wyróżnienie cech i konturów. [12].

1.4.2. 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 (rys. 1.2) 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

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 [12].

1.5. Weryfikacja modelu do rozpoznawania cyfr

Poziom dokładności stanowi jedną z najprostszych metod oceny jakości modelu. Określa on stosunek poprawnie zaklasyfikowanych danych do liczby wszystkich danych. Miara ta jest adekwatna tylko, gdy zbiór danych jest zbalansowany, czyli zawiera równą liczbę próbek dla każdej klasy [13].

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 zbytnia złożoność modelu, zbyt mała liczba danych treningowych lub zbyt długie trenowanie modelu [14].

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 [15].

Podczas klasyfikacji binarnej można rozróżnić cztery przypadki zaklasyfikowania przedstawione na rys. 1.3:

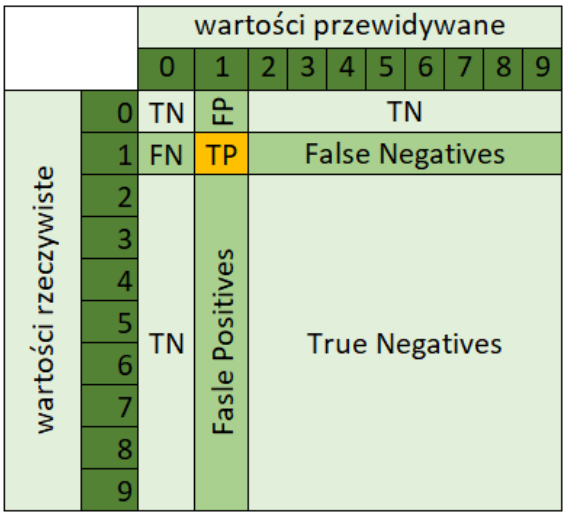
Rysunek 1.3 Macierz błędu dla dwóch klas

		wartości przewidywane	
		+	-
wartości rzeczywiste	+	przawdziwie pozytyny - TP	fałszywie pozytywny - FN
	-	fałszywie negatywny - FP	prawdziwie negatywny - TN

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) [16].

Rysunek 1.4 Macierz błędu dla wielu klas



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.

W przypadku modelu do rozpoznawania cyfr ręcznie pisanych ważnym wskaźnikiem będzie precyzja, aby sprawdzić, ile danych zaklasyfikowanych do danej klasy zostało prawidłowo zaklasyfikowane. 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.

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 [17, 18].

2. Model aplikacji do nauki matematyki

2.1.1. Założenia modelu aplikacji

Założono, że aplikacja w oparciu o stworzone pytania ma być w stanie w prosty sposób wygenerować wiele nowych pytań, tak aby zapewnić użytkownikowi urozmaicenie przez 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: $x + y$, użytkownik dostanie polecenie: $2 + 3$. Przy następnym pytaniu może dostać to samo działanie, ale z innymi liczbami.

Rysunek 2.1 Przykładowe działania

```
1  x / y
2  x * y
3  x + y
4  x - y
5  x + y + z
6  x + y - z
7  x - y + z
8  x - y - z
```

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 Przykłady zadań tekstowych

```
1  x * y / 2
2  Policz pole TRÓJKĄTA o podstawie #x i wysokości #y.
3  x * x
4  Policz pole KWADRATU o podstawie #x.
5  x * x
6  Policz pole KOŁA o promieniu #x [pi].
7  x * y / 2
8  Policz pole ROMBU o długości jednej przekątnej równej #x i drugiej przekątnej równej #y.
9  x * y / 2
10 Policz pole DELTOIDU o długości pierwszej przekątnej #x i drugiej przekątnej #y.
11 x * y
12 Policz pole PROSTOKĄTU o wymiarach #x i #y.
```

źródło: opracowanie własne

Ustalono, że odpowiedzi użytkownika będą udzielane poprzez ręczne napisanie liczby w przestrzeni do rysowania. Następnie program wyśle napisaną liczbę z zapytaniem na serwer, skąd model uczenia maszynowego zwróci zaklasyfikowaną liczbę. Program wyświetli rozpoznaną liczbę i zapyta użytkownika czy chodziło mu o nią. 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 przedstawione zadania. Ponieważ aplikacja nie ma przeznaczonej grupy wiekowej, będą do wyboru poziomy trudności polegające na wyborze przedziału, z którego będą losowane liczby w pytaniach.

2.1.2. Architektura aplikacji

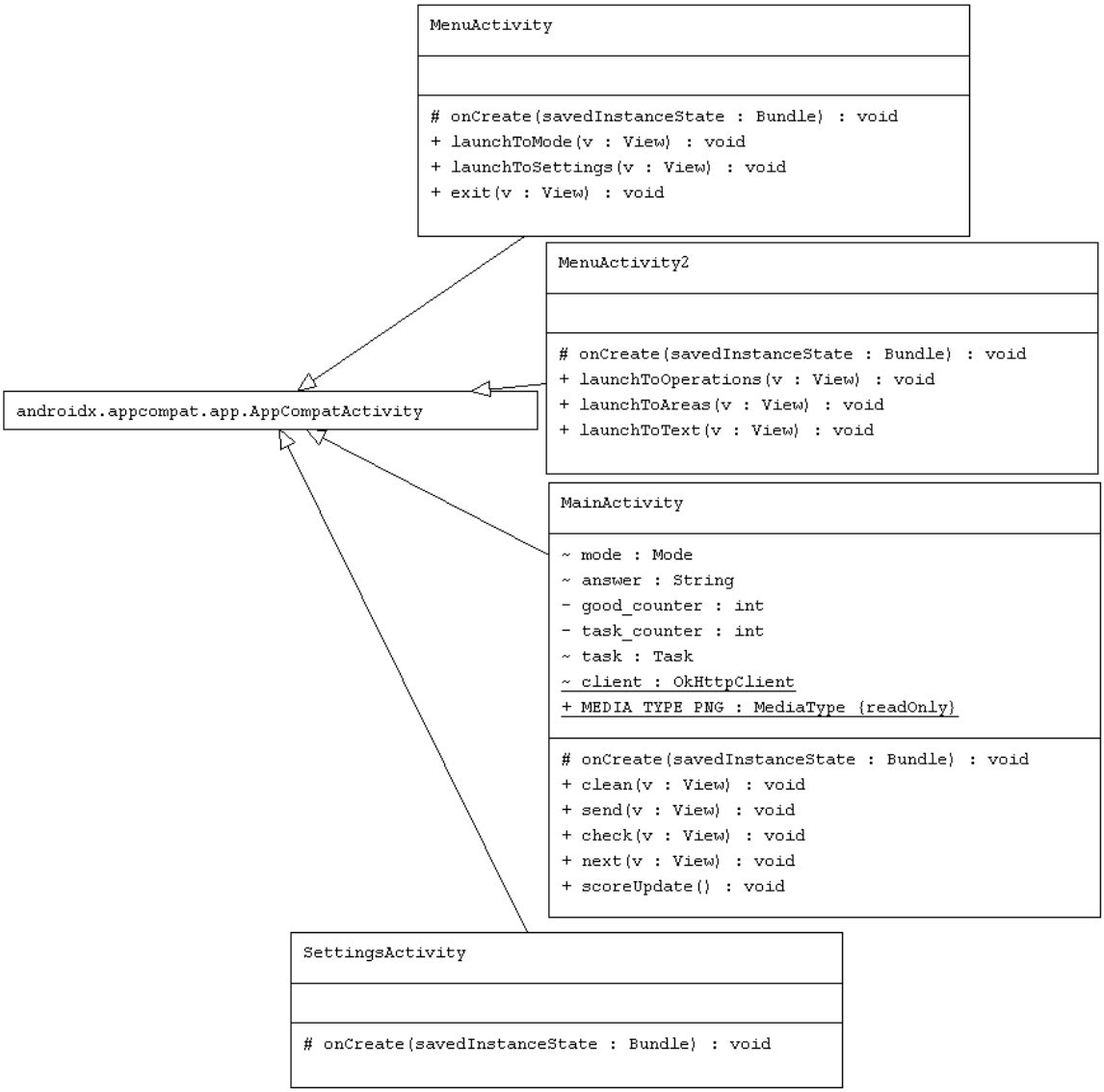
W programowaniu aplikacji mobilnych na platformę Android, aktywność (activity) oraz widok (view) to dwa kluczowe pojęcia. Aktywności reprezentują poszczególne ekrany, czyli interaktywne interfejsy użytkownika, z którymi użytkownik może wchodzić w interakcję. Z kolei widoki to elementy interfejsu użytkownika, 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 związany 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 ekranu z ustawieniami aplikacji.

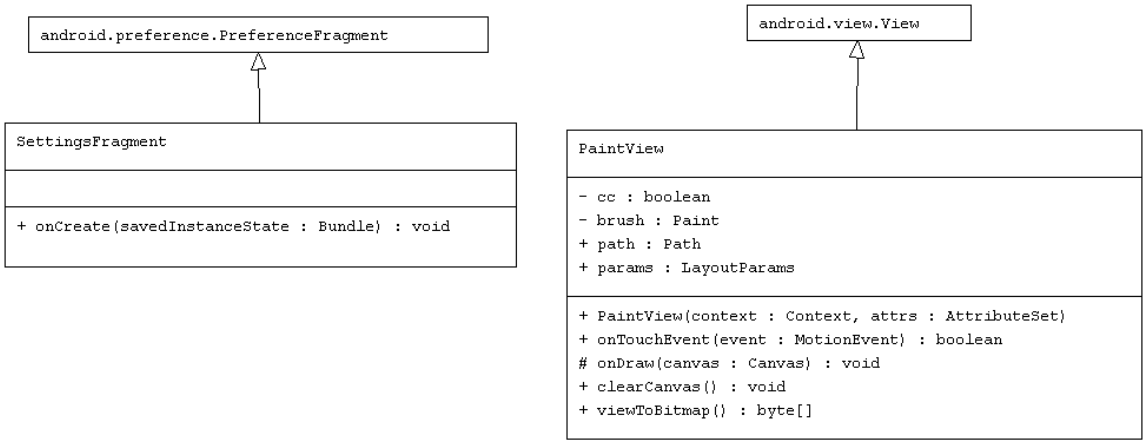
Klasa MainActivity pełni rolę obsługi ekranu z zadaniami. W zależności od wybranego przez użytkownika trybu wyświetla odpowiednie pytanie. Po otrzymaniu odpowiedzi od użytkownika, wysyła obraz liczby z zapytaniem na serwer. Program sprawdza poprawność odpowiedzi oraz informuje użytkownika o wyniku. Dodatkowo,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

Rysunek 2.3 Diagram klas aktywności - obsługa przycisków w menu (MenuActivity, MenuActivity2), aktywność z zdaniami (MainActivity), aktywność z ustawieniami (SettingsActivity)

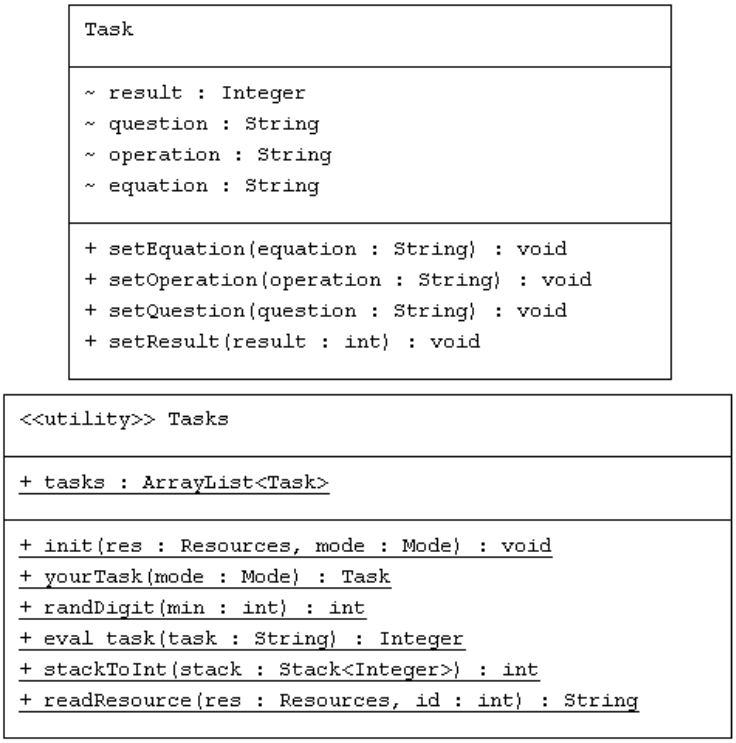


źródło: opracowanie własne

Rysunek 2.4 Diagram klas fragmentu i widoku - fragment z ustawieniami (SettingsFragment), widok do rysowania (PaintView)



Rysunek 2.5 Diagram klas



źródło: opracowanie własne

Poza tym w aplikacji znajdują się także klasy Task i Tasks (pokazane na rys 2.5), 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.

2.2. Wybrane technologie zastosowane do budowy aplikacji

Główna część aplikacji została napisana w języku Java na system Android. Ponadto rozpoznawanie liczb ręcznie pisanych zostało zaimplementowane w języku Python.

2.2.1. Android

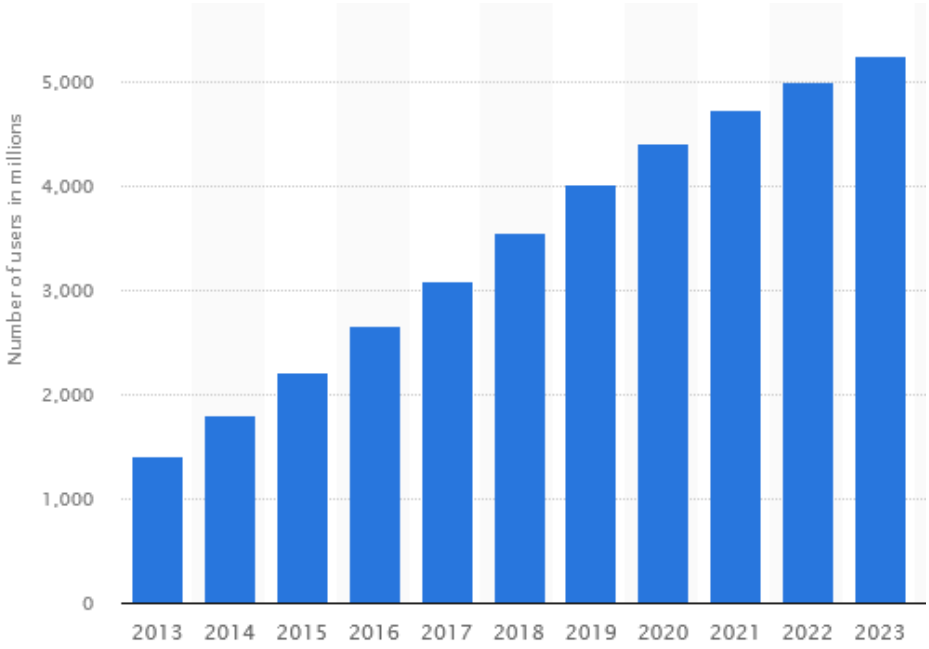
Android powstał w 2008 roku wraz z wprowadzeniem na rynek pierwszego urządzenia opartego o ten system - HTC Dream.

W pierwszym kwartale 2023 roku Android był najpopularniejszym systemem operacyjnym na urządzenia mobilne. Według Bankmycell zajmował on 71,95% rynku, natomiast na drugim miejscu znajdował się iOS z udziałem w rynku wynoszącym 27,42%. Pozostałe systemy stanowiły zaledwie 0,63% [19]. Android Swoją popularność 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 się prezentowała na każdym z urządzeń [20].

Z wymienionych wcześniej urządzeń z Androidem, smartfony są najbardziej powszechnie używane. Największą ich zaletą jest niewielki rozmiar, dzięki czemu użytkownik może je mieć zawsze ze sobą. Według badań DataReportal 96% przebadanych użytkowników posiada smartfon. Dla porównania kolejnym urządzeniem jest laptop/komputer, którego posiada jedynie 58% respondentów [21].

Wzrost popularności smartfonów jest pokazany na rys. 2.6. W 2023 roku liczba użytkowników wyniosła ponad 5 miliardów.

Rysunek 2.6 Popularność smartfonów



Źródło: https://www.statista.com/forecasts/1143723/smartphone-users-in-the-world

2.2.2. Java

Istnieją dwa najbardziej popularne języki do tworzenia aplikacji Android: Java i Kotlin. Java jest znacznie starszym językiem, który powstał w 1996 roku. Przez wiele lat Java była głównym językiem programowania do tworzenia aplikacji na platformę Android. Przez swoją długą historię posiada stabilne środowisko, duże wsparcie ze strony społeczności oraz rozbudowany zbiór bibliotek i narzędzi.

Natomiast Kotlin, młodszy język, powstał dopiero w 2011 roku, jednak szybko zyskał swoją popularność. W 2016 roku zostało oficjalnie ogłoszone wsparcie dla Kotlina jako języka programowania Android, a w 2019 roku został uznany za preferowany język przez developerów Androida [22].

2.2.3. Pozostałe narzędzia:

- Flask – framework do tworzenia serwerów
- Tensorflow – biblioteka wykorzystywana w uczeniu maszynowym i sieciach neuronowych

- Pillow, OpenCV – biblioteki do obróbki obrazów

W aplikacji wykorzystano narzędzia i biblioteki do różnych zadań. Flask został wykorzystany do stworzenia serwera, który obsługuje komunikację z użytkownikami oraz zapewnia dostęp do modelu uczenia maszynowego. Tensorflow został użyty do bodowy oraz wytrenowania modelu uczenia maszynowego.

Biblioteki Pillow i OpenCV zostały zastosowane do wstępnej obróbki danych. Obie biblioteki zostały wykorzystane do przeprowadzenia operacji takich jak przekształcenia, augmentacja danych oraz podział obrazów liczb na obrazy cyfr.

Dzięki połączeniu tych narzędzi i bibliotek możliwe było zbudowanie aplikacji, która wykorzystuje model uczenia maszynowego do analizy i rozpoznawania liczb na obrazach.

2.3. Wybrane metody uczenia maszynowego zastosowane w aplikacji

2.3.1. Zestaw danych MNIST

Model został wytrenowany na podstawie zestawu danych MNIST, który zawiera obrazy cyfr ręcznie pisanych przez uczniów szkoły średniej oraz pracowników Bureau of the Census (amerykańska agencja zajmująca się spisami ludności). Zestaw danych jest najczęściej dzielony w proporcji 85:15, gdzie 85% stanowi 60’000 danych uczących, a 15% to 10’000 danych testowych [23]. Przykładowe dane z zestawu MNIST przedstawiono na rys. 2.7.

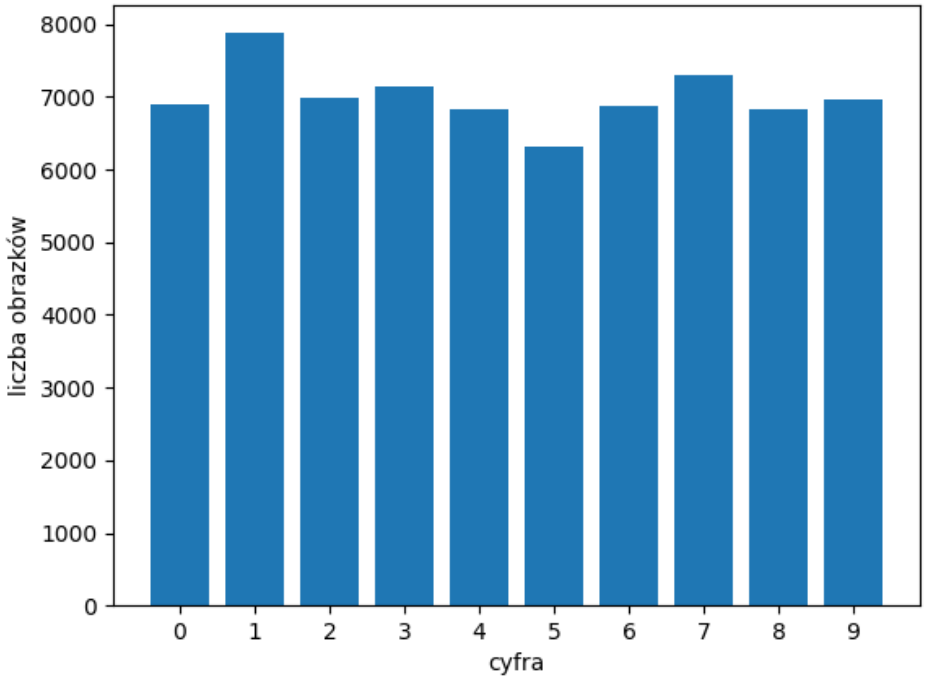
Rysunek 2.7 Przykładowe dane z zestawu MNIST

źródło: opracowanie własne

Zestaw danych widoczny na rys. 2.8 nie jest zbalansowany, różnica między liczbą wystąpień „1” i „5” wynosi około 2000. Po wykonaniu testu chi-kwadrat o hipotezie zerowej mówiącej o zbalansowaniu danych i hipotezie alternatywnej mówiącej o niezbalansowaniu danych. P-value okazało się znacznie mniejsze niż 0.05, Dlatego odrzucono hipotezę zerową na rzecz hipotezy alternatywnej. Stwierdzono, że dane nie są zbalansowane.

Jednak ten problem nie ma wpływu na zaproponowany algorytm uczenia maszynowego, ponieważ zgodnie z założeniami sieci neuronowej dane powinny zawierać mniej więcej tyle samo obserwacji w każdej z klas, ale nie muszą koniecznie być zbalansowane. Niezbalansowanie danych (zwłaszcza silne) może niekorzystnie wpływać na model, jednak jak później zostanie pokazane, w tym przypadku nie wpływa niekorzystnie [24].

Rysunek 2.8 Rozkład cyfr w MNIST



źródło: opracowanie własne

MNIST jest zestawem danych dobrze przygotowanym do uczenia maszynowego. Obrazy zostały znormalizowane do wymiarów 20x20 pikseli, przy zachowanych proporcjach. Poprzez zastosowany antyaliasing, obrazy są w skali szarości, a nie czarnobiałe. Następnie obrazy zostały umieszczone w centrum obrazów o wymiarach 28x28 pikseli [23].

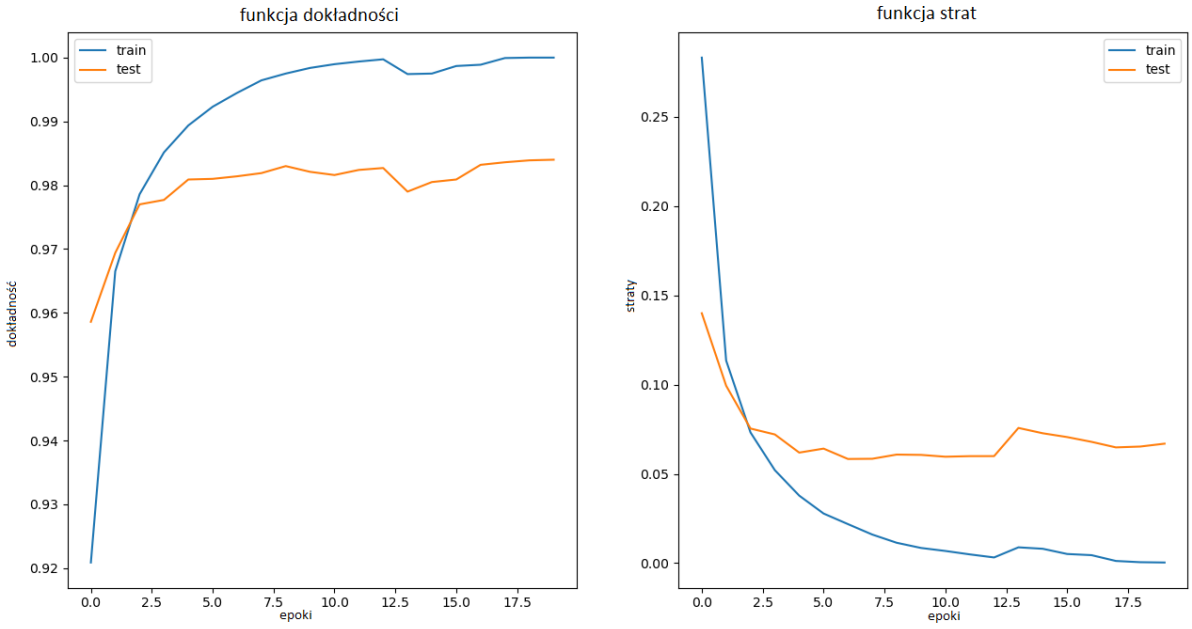
2.3.2. Augmentacja – podejście praktyczne

Na początku użyto najczęściej stosowanego podziału danych według proporcji 85:15 danych uczących do danych testowych, jednak po zbudowaniu opisanego dalej modelu MLP, okazało się, że zbiór danych uczących jest niereprezentatywny, to znaczy dane uczące są łatwiejsze do rozpoznania niż dane testujące. Świadczy o tym duża odległość pomiędzy

krzywymi train i test na wykresie funkcji strat na (rys. 2.9). Dlatego postanowiono zastosować augmentację by stworzyć zestaw danych uczących, który będzie większy i trudniejszy do rozpoznania.

Przygotowanie zestawu MNIST rozpoczęto od podziału danych według proporcji 50:50 danych uczących do danych testowych. Taki podział wynika z tego, że po dorobieniu danych uczących, będzie ich więcej i proporcje zmienią się do 60:40.

Rysunek 2.9 Wykresy funkcji dokładności i funkcji strat dla zbiorów danych uczących (train) i testowych (test) przedstawiające niereprezentatywność zbioru uczącego



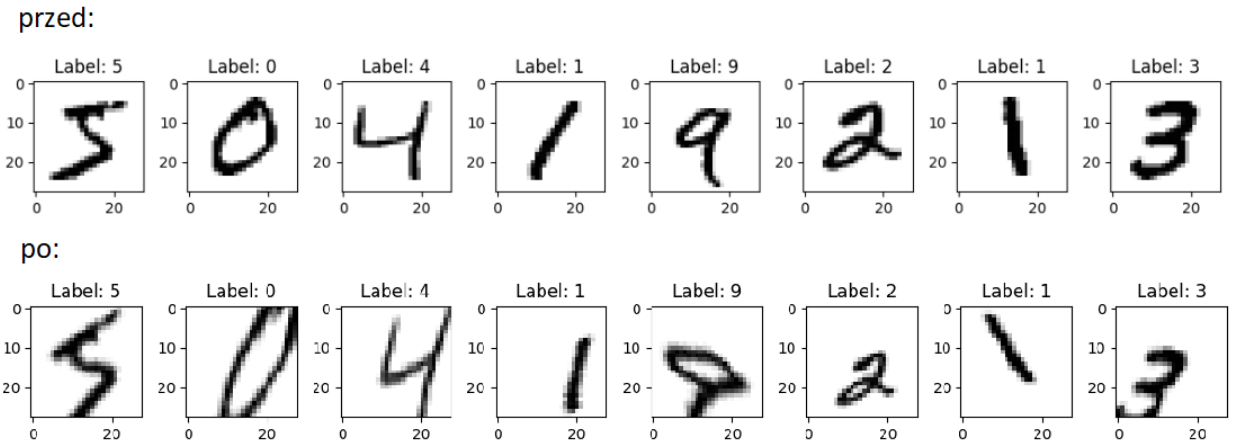
źródło: opracowanie własne

Po sprawdzeniu różnej liczby stworzenia nowych danych, okazało się, że model najlepiej uczy się przy stworzonych dodatkowych 20’000 nowych obrazów. Obrazy powstały poprzez przekształcenia takie jak obracanie, zmiany przybliżenia, przesuwanie i pochylenie. Pochylenie jest zmianą położenia pikseli wzdłuż danej linii, w efekcie obraz wygląda na rozciągnięty, natomiast obrót zachowuje obraz niezmieniony.

Nowe obrazy powstały poprzez obrócenie o kąt z zakresu ±30 stopni. Losowe przekształcenia skali (zmniejszenie lub powiększenie) z zakresu od 0,5 do 1,5. To znaczy, że obrazom zostało zmienione przybliżenie o maksymalnie 50%. Zastosowane zostało przesunięcie pionowe i poziome o 25% wysokości i szerokości, natomiast pochylenie o ±45

stopni. Na rys. 2.10 porównano wybrane obrazy z zestawu MNIST oraz powstałe po przekształceniach.

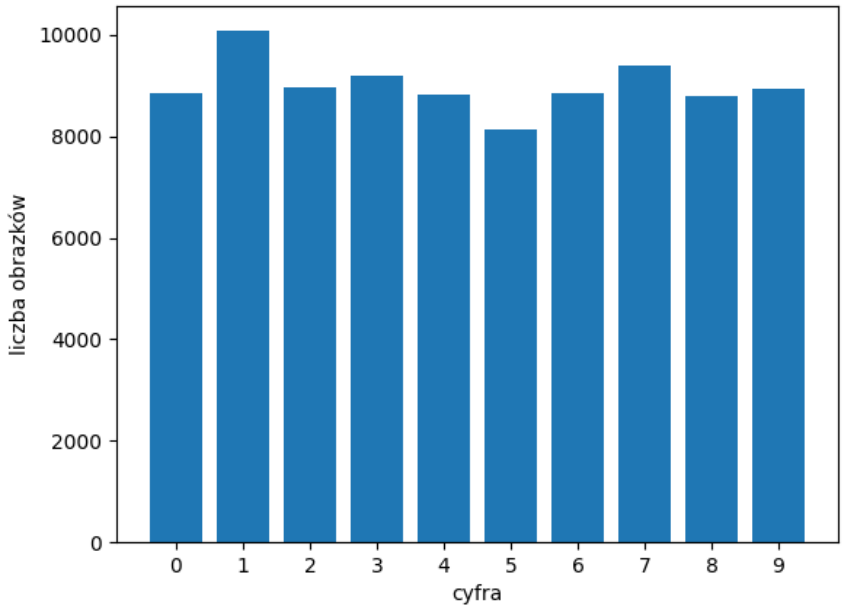
Rysunek 2.10 Wyniki augmentacji



źródło: opracowanie własne

Rozkład cyfr widoczny na rysunku rys. 2.11 nie zmienił się po augmentacji, nadal są zachowane takie same proporcje, jednak danych w każdej z klas przybyło po równo.

Rysunek 2.11 Rozkład cyfr po augmentacji



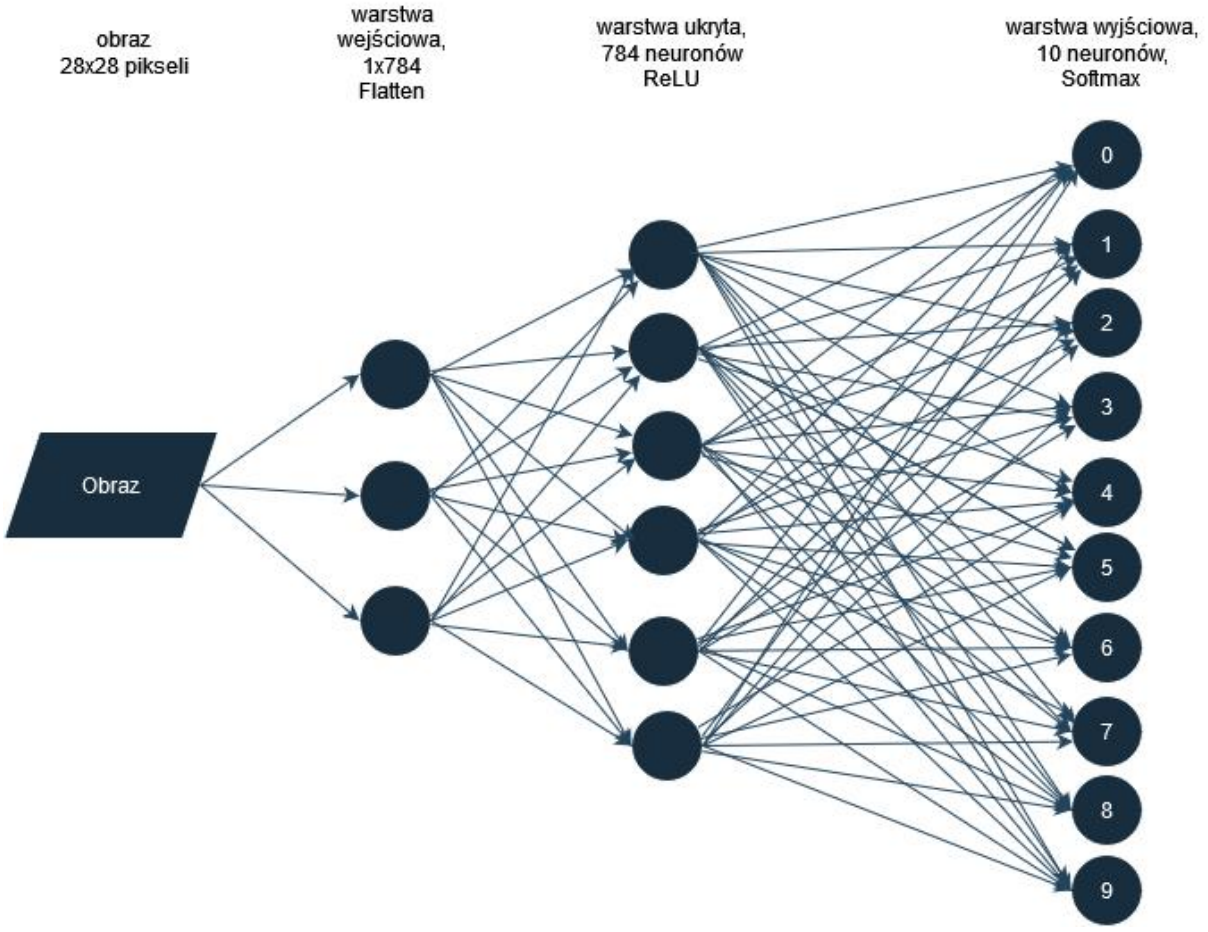
źródło: opracowanie własne

2.3.3. Model i badanie skuteczności

Zanim przystąpiono do uczenia modelu przygotowano dane poprzez zamianę skali szarości na reprezentację binarną. Zgodnie z uzasadnieniem podanym w podrozdziale 1.3 „Wstępne przetwarzanie danych”.

Proponowany model przedstawiony na rys. 2.12, składa się z jednej warstwy ukrytej. Warstwa wejściowa to flatten, która spłaszcza obraz dwuwymiarowy 28x28 pikseli do jednowymiarowego wektora z wartościami pikseli. Warstwa ukryta składa się z 784 neuronów, wartość ta odpowiada liczbie pikseli na obrazie o wymiarach 28x28. Wykorzystano funkcję aktywacji ReLU. Zgodnie z uzasadnieniem podanym w podrozdziale 1.2 „Wielowarstwowy perceptron - MLP”. Warstwa wyjściowa zawiera 10 neuronów, po jednym na każdą cyfrę. Wykorzystano w niej funkcję Softmax, do określenia przynależności do każdej z klas. Model został wytrenowany w trakcie 18 epok. Rozmiar partii wynosił 256.

Rysunek 2.12 Diagram zastosowanej sieci neuronowej

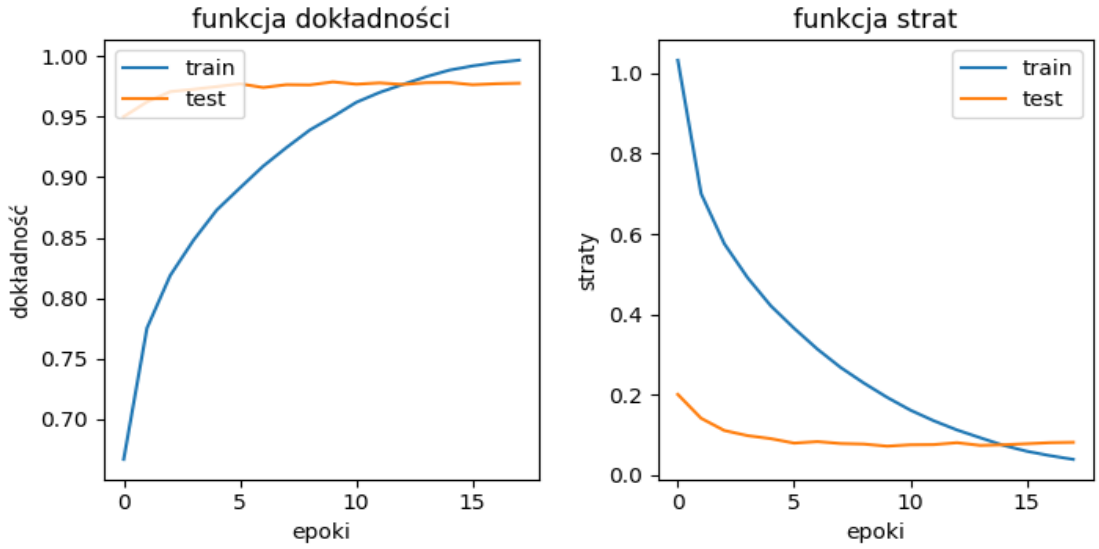


źródło: opracowanie własne

Zaproponowany model osiąga dokładność na poziomie 97,28%. Wynik ten jest zadawalający pod kątem rozpoznawania cyfr.

Na rys. 2.13 została przedstawiona funkcja strat, gdzie krzywa dla danych treningowych maleje i stabilizuje się, natomiast krzywa dla danych testowych maleje i bardzo szybko stabilizuje się, znajduje się poniżej krzywej dla danych treningowych. Świadczy to o niereprezentatywności danych testowych. Zbiór danych testowych jest dla modelu łatwiejszy do przewidzenia niż zbiór danych treningowych. Jest to zrozumiały proces, ponieważ na danych treningowych zastosowano augmentację. Wartości funkcji dokładności są stabilne, co oznacza, wyniki modelu są powtarzalne [15].

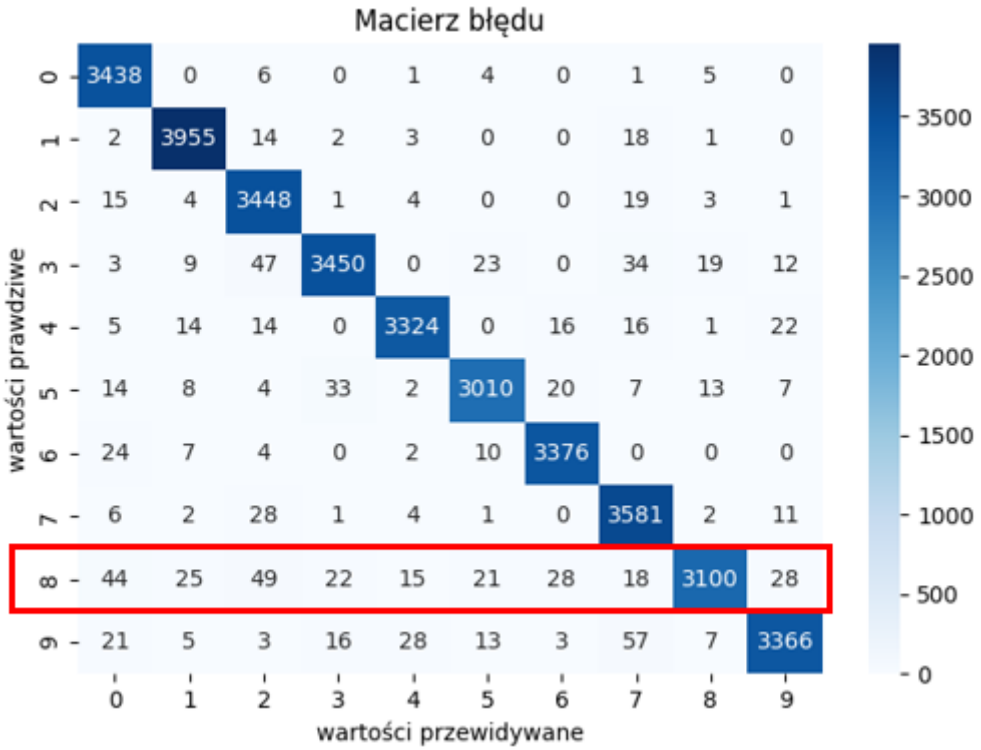
Rysunek 2.13 Wykresy funkcji dokładności i funkcji strat dla zbiorów danych uczących (train) i testowych (test)



źródło: opracowanie własne

Na rys. 2.14 macierzy błędu można zauważyć, że model przewiduje większość przypadków poprawnie. Warto zwrócić uwagę, że najwięcej, bo aż 250 ósemek na 3350 zostało sklasyfikowane niepoprawnie.

Rysunek 2.14 Macierz błędu



źródło: opracowanie własne

Tabela 2.1 Zsumowane wartości tablicy błędów na zbiorze testowym wymagane do wyznaczenia wskaźników jakości klasyfikacji: czcionka na żółtym tle dla największych wartości FP, czcionka czerwona dla największych wartości FN

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
TP	3438	3448	3448	3450	3324	3010	3376	3581	3100	3366
TN	31411	30931	31336	31328	31529	31810	31510	31194	31599	31400
FP	134	74	169	75	59	72	67	170	51	81
FN	17	40	47	147	88	108	47	55	250	153

źródło: opracowanie własne

Tabela 2.1 zawiera wartości opisane w rozdziale 1.5. „Weryfikacja modelu do rozpoznawania cyfr" pozwalające wyznaczyć wymagane wskaźniki jakości klasyfikacji na zbiorze testowym przedstawione w tabeli 2.2. Przedstawione tabele służą do sprawdzania jak dobrze model radzi sobie z identyfikacją poszczególnych klas oraz oceny jakości wyników.

Tabela 2.2 Czułość, specyficzność i precyzja: czcionka na żółtym tle dla najmniejszej wartości czułości, czerwona czcionka dla najmniejszych wartości precyzji

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Czułość	1	0.99	0.99	0.96	0.97	0.97	0.99	0.98	0.93	0.96
Specyficzność	1	1	0,99	1	1	1	1	0,99	1	1
Precyzja	0,96	0,98	0,95	0,98	0,98	0,98	0,98	0,95	0,98	0,98

źródło: opracowanie własne

W podanym problemie ważne jest zwrócenie uwagi na liczby w tabeli 2.1 zakreślone żółtym kolorem, oznaczają, że dla klas dwójek i siódemek mają najwyższą liczbę wyników fałszywie pozytywnych. Oznacza to, że najwięcej danych jest mylnie klasyfikowane jako te cyfry. Potwierdza to najniższy poziom precyzji na poziomie 0.95 w tabeli 2.2.

Natomiast cyfra 8 wyróżniająca się w tabeli 2.1 największą wartością niepoprawnych klasyfikacji (FN) ma czułość na poziomie 0.93 (tabela 2.2). Oznacza, to że ósemki są najczęściej klasyfikowane jako inne cyfry.

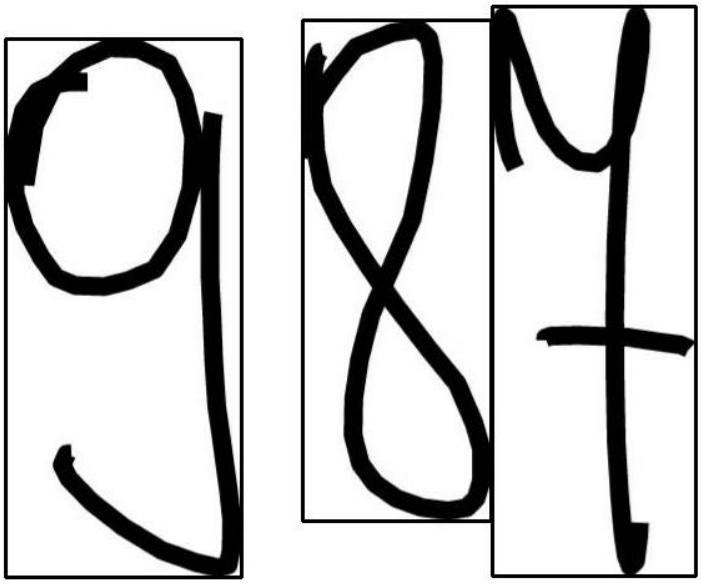
Warto zwrócić również uwagę, że pomimo niezbilansowanych danych, nie ma to wpływu na model. Klasa, która miała najmniej obserwacji, czyli cyfra „5” (rys. 2.11), jest klasyfikowana równie dobrze jak pozostałe klasy, które miały więcej obserwacji. Ponadto inne cyfry nie są rozpoznawane jako jedynki, których obserwacji było najwięcej.

2.3.4. Podział obrazów liczb na cyfry oraz przygotowanie danych

Przygotowanie danych zastosowano również dla obrazów, które przychodzą z aplikacji do zaklasyfikowania. Te obrazy są różnych wymiarów, dlatego ważne jest przygotowanie danych, tak aby przypominały te stworzone w zestawie MNIST.

Obrazy trafiające do modelu klasyfikującego cyfry zawierają liczby. Program rozdziela liczby na cyfry (rys. 2.15), które kolejno przechodzą etap wstępnego przygotowania.

Rysunek 2.15 Podział liczby na cyfry

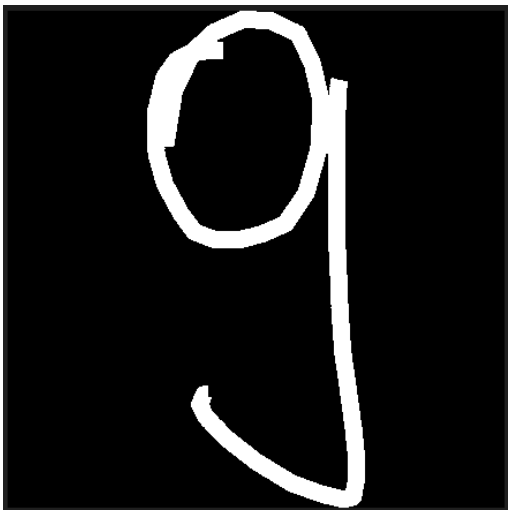


źródło: opracowanie własne

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 rezultacie tego zabiegu powstaje obraz z obróconą kolorystyką, zamiast czarnych elementów na białym tle są białe elementy na czarnym tle. Dzięki odwróconym kolorom funkcja służąca do podziału liczb na cyfry, 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 cyfry tak jak w przypadku na rys. 2.15 jest wyższy niż szerszy, to taki obraz jest uzupełniany pustą przestrzenią po bokach tak aby utworzyć kwadrat (rys. 2.16).

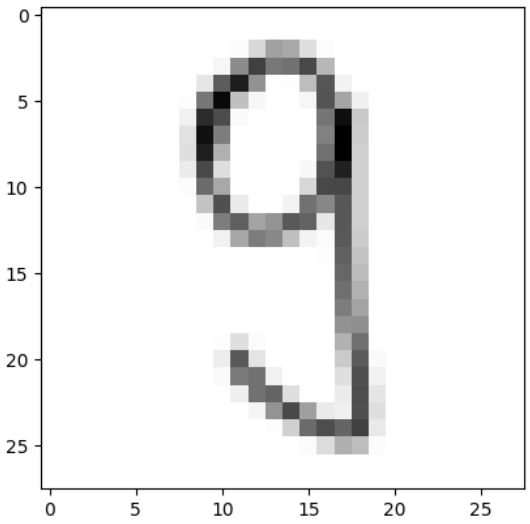
Rysunek 2.16 Cyfra po dopisaniu pustego pola



źródło: opracowanie własne

Następnie obraz jest uzupełniany 50 pikselowym pustym polem z każdej strony, tak aby cyfra znajdowała się w centrum. Na końcu obraz jest przeskalowywany do wymiarów 28x28. Po przygotowaniu obraz wygląda jak rysunek przedstawiony na rys. 2.17.

Rysunek 2.17 Liczba po obróbce



źródło: opracowanie własne

2.4. Korzystanie z modelu ML poprzez API HTTP

Aplikacja wykorzystuje komunikację z serwerem, aby rozpoznać liczby napisane przez użytkownika (rys. 2.18). 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 na przykład „21”.

Rysunek 2.18 Schemat architektury aplikacji z modelem ML

źródło: opracowanie własne

3. Opis implementacji i działania aplikacji

3.1. Ekran startowy i podstawowe funkcjonalności

W pierwszym menu wyboru (rys. 3.1), użytkownik ma możliwość rozpoczęcia, przejścia do ustawień lub zakończenie aplikacji.

Rysunek 3.1 Ekran startowy

źródło: opracowanie własne

Jeżeli użytkownik wybierze opcję Start, zostanie mu przedstawione menu wyboru trybu nauki (rys. 3.2). W aplikacji zostały zaporoponowane 3 tryby nauki: działania, pola figur oraz zadania tekstowe.

Rysunek 3.2 Ekran wyboru trybu nauki

źródło: opracowanie własne

Po wyborze, każdego z wybranych trybów użytkownikowi pokaże się ten sam ekran co po wyborze trybu Działania (rys. 3.3) jednak z innymi pytaniami.

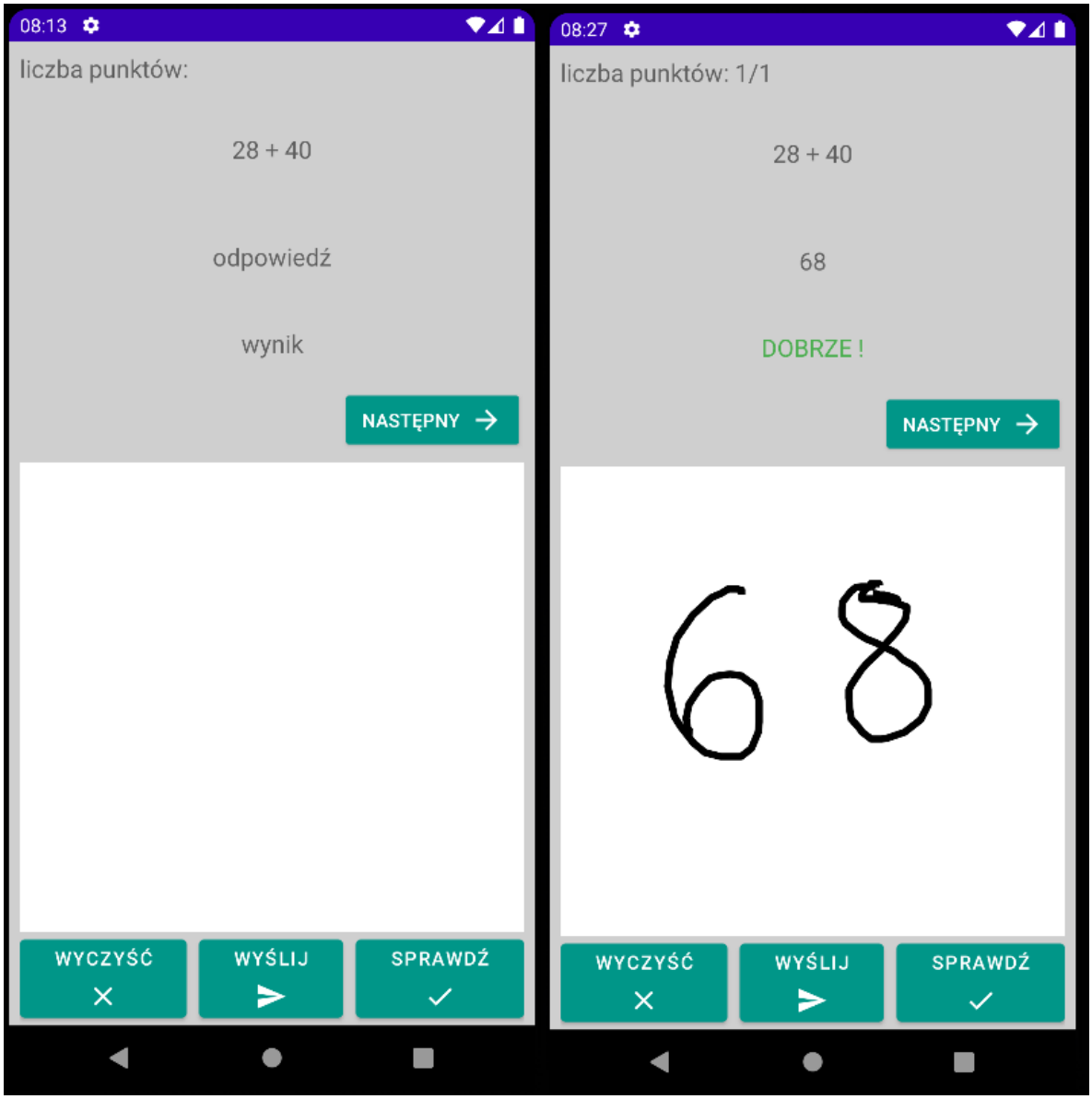
W ramach aplikacji stworzono zestaw zadań składający się z: ośmiu zadań obliczeniowych, ośmiu zadań z polami figur oraz dwudziestoma zadaniami tekstowymi. Aplikacja wczytuje trzy pliki txt z zadaniami zbudowanymi według wzorców opisanych wcześniej w podrozdziale 2.1.1. „Założenia aplikacji”.

Ekran przedstawia pytanie (działanie lub treść), pola tekstowe odpowiednio na odpowiedź oraz na wynik. Największe białe pole to PaintView, czyli pole do rysowania., służy użytkownikowi do zapisywania odpowiedzi.

Ponadto istnieją cztery przyciski: Wyczyść, Wyślij, Sprawdź oraz Następny. Przycisk Wyczyść służy do całkowitego usunięcia zawartości pola do pisania. Przycisk Wyślij wysyła zawartość PaintView do serwera. Aplikacja zapisuje zwróconą odpowiedź do pola tekstowego. Przycisk Sprawdź powinien zostać kliknięty, jeżeli użytkownik uzna, że aplikacja prawidłowo rozpoznała zapisaną przez niego liczbę. Program porówna odpowiedź zapisaną w polu tekstowym z obliczonym przez niego wynikiem, a następnie wyświetli rezultat w polu Wynik.

W lewym górnym rogu znajduje się licznik punktów, który wskazuje, ile poprawnych odpowiedzi udzielił użytkownik. Dopóki nie zostanie sprawdzona żadna odpowiedź, licznik się nie wyświetla.

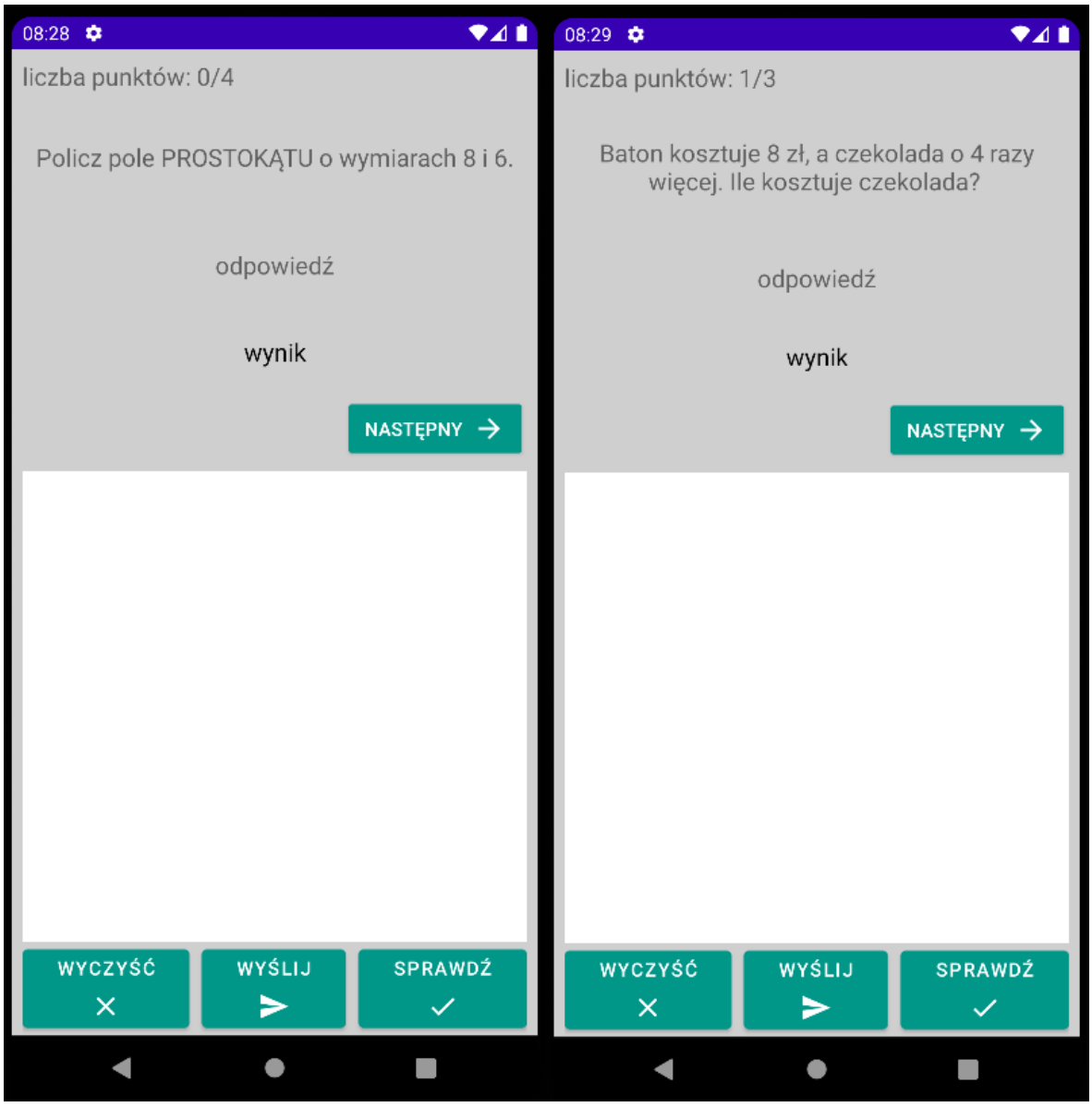
Rysunek 3.3 Główny ekran



źródło: opracowanie własne

Na rys. 3.4 zostały przedstawione pozostałe tryby: po lewej stronie został zaprezentowany tryb pola figur, a po prawej tryb zadań tekstowych.

Rysunek 3.4 Pozostałe tryby: po lewej pola figur, po prawej zadania tekstowe



źródło: opracowanie własne

3.2. Przykłady zastosowania aplikacji w konkretnych zadaniach

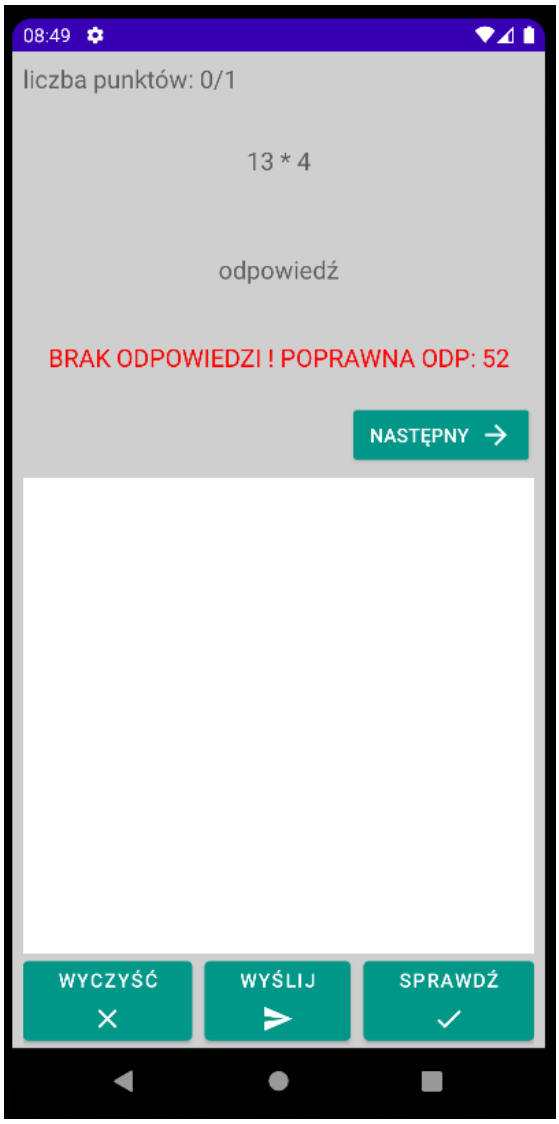
Aplikacja prezentuje pytania i sprawdza poprawność odpowiedzi użytkownika. Wyświetla stosowne komunikaty (rys. 3.5).

Rysunek 3.5 Sprawdzanie poprawności odpowiedzi

źródło: opracowanie własne

W przypadku, gdy zostanie wysłany do serwera pusty ekran, odpowiedź zostanie potraktowana, tak jakby użytkownik nie znał odpowiedzi na pytanie, ale chciał otrzymać odpowiedź na postawione pytanie (rys. 3.6).

Rysunek 3.6 Brak odpowiedzi



źródło: opracowanie własne

Rozpoznawane liczby są w kolejności od lewej do prawej, natomiast wysokość położenia nie jest brana pod uwagę. Wykrywanie kolejności opiera się na znajdowaniu najmniejszej wartości współrzędnej x. Często zdarza się napisać kolejne znaki niżej lub wyżej o kilka pikseli, dlatego rozpoznawanie wysokości w tym przypadku jest nieuzasadnione. Takie zastosowanie jest szczególnie korzystne w sytuacji, w której użytkownik posiada pochyłe pismo (rys. 3.7).

W zaproponowanych trybach nauki takie rozwiązanie jest wystarczające, jednak nie umożliwia ono rozpoznawania rozwiązań, w których istotna jest wysokość danych elementów, na przykład w przypadku ułamków.

Rysunek 3.7 Rozpoznawanie liczby od lewej do prawej

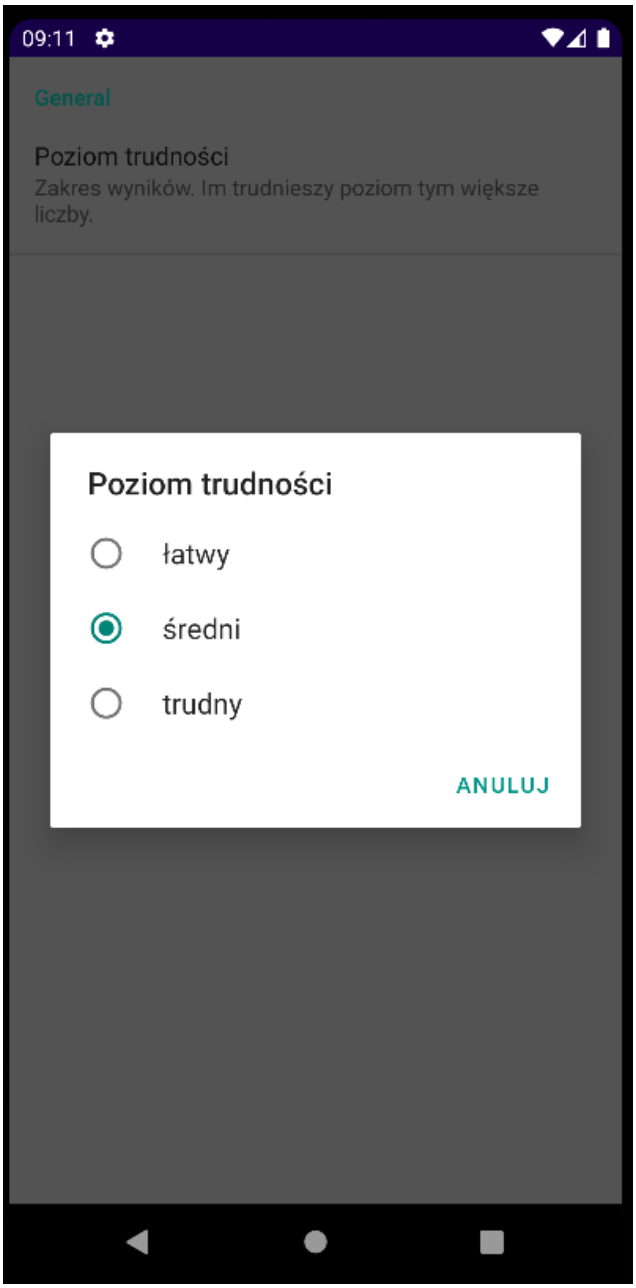
źródło: opracowanie własne

3.3. Schematy, opisy poszczególnych funkcjonalności, opis kodów

3.3.1. Poziomy trudności

W ustawieniach aplikacji (rys 3.8) istnieje możliwość zmiany poziomu trudności pytań. Zostały opracowane trzy poziomy: łatwy, średni i trudny. Poziom trudności zmienia zakres liczb, które są losowane i podstawiane do pytań. Dodatkowo przy losowaniu liczb jest uwzględniany typ działań, tam, gdzie występuje mnożenie lub dzielenie zakres liczb jest mniejszy.

Rysunek 3.8 Ekran ustawień



źródło: opracowanie własne

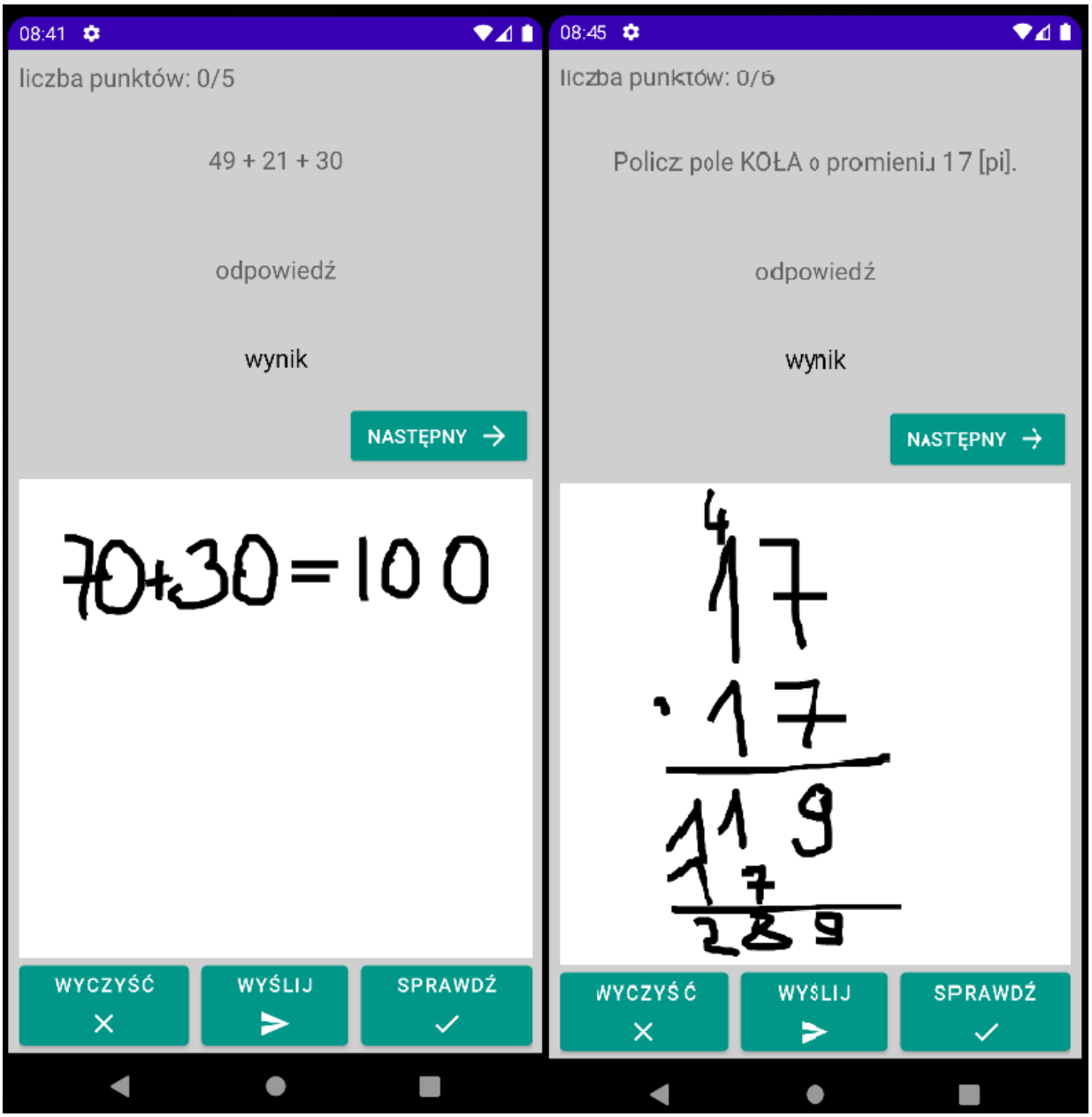
3.3.2. Dodatkowe zalety wykorzystania ekranu do pisania

W aplikacjach do nauki matematyki powszechnie stosuje się klawiaturę do wpisywania odpowiedzi. Takie rozwiązanie jest mniej naturalne, ponieważ uniemożliwia użytkownikowi rozpisania zadania na mniejsze, prostsze kroki, czego często wymaga nauka matematyki. W zadaniach trudnych i złożonych powinno się dążyć do uproszczenia problemu, co jest możliwe

dzięki ekranowi do pisania (rys. 3.9). Użytkownik ma możliwość rozpisania zadania na ekranie, a gdy będzie już znać odpowiedź może wymazać brudnopis i zapisać prawidłowy wynik. W przypadku gdy użytkownik prześle rozpisane zadanie zamiast samej odpowiedzi, wynik może zostać źle rozpoznany.

Dodatkowym atutem zaproponowanego rozwiązania jest ćwiczenie pisma u dzieci. Model nie rozpozna odpowiedzi, jeśli zostanie napisania nieczytelnie, dlatego dzieci muszą dbać o wyraźne pismo, co jest częstym problemem u nich.

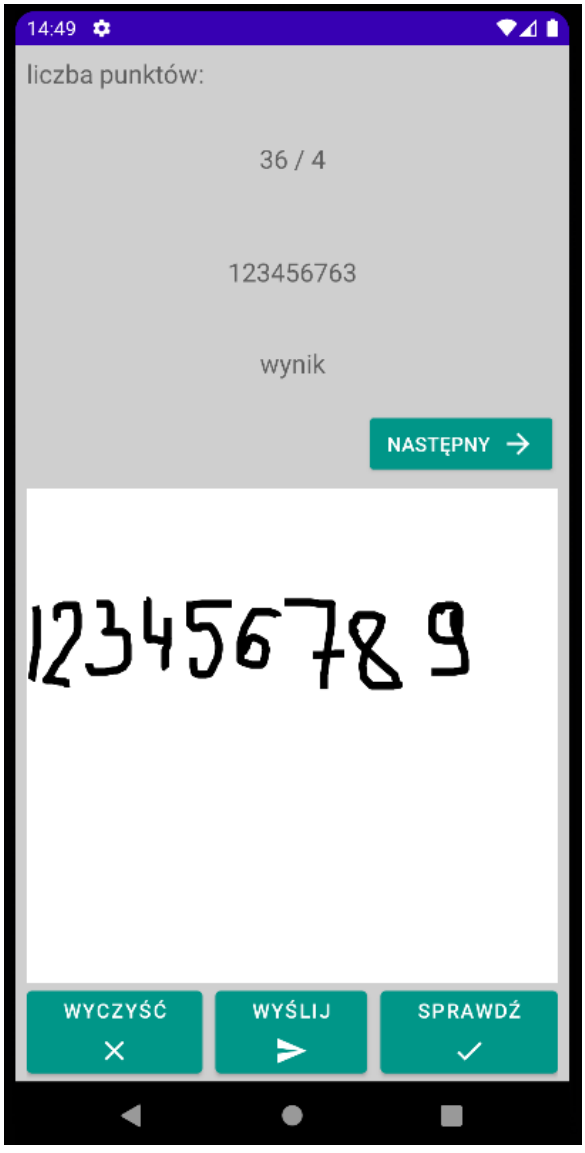
Rysunek 3.9 Wykorzystanie ekranu do pisania jako brudnopisu



źródło: opracowanie własne

Warto zwrócić uwagę, że model nie zawsze rozpoznaje poprawnie wszystkie cyfry nawet przy dość czytelnym piśmie (rys. 3.10). Jest to problem, który nie został rozwiązany w ramach tego projektu. W przyszłości istnieje możliwość rozbudowy aplikacji tak, aby w przypadku źle zaklasyfikowanej cyfry użytkownik wpisywał na klawiaturze o jaką liczbę mu chodziło. W ten sposób udałoby się zwiększyć liczbę danych, z którymi model sobie nie radzi i model mógłby się uczyć nowych wzorców. Obecnie użytkownik ma jedynie możliwość wyczyszczenia ekranu i ponownego zapisania odpowiedzi.

Rysunek 3.10 Przykład niepoprawnego rozpoznania wyniku, błędne cyfry 8 i 9



źródło: opracowanie własne

3.3.3. Opisy wybranych kodów

Działania, które aplikacja oblicza, aby sprawdzić poprawność wyniku użytkownika wyglądają na przykład „11+22+33”. W wielu językach chcący obliczyć takie wyrażenie użyto by wbudowanej funkcji *eval*. Jednak w języku Java nie ma zaimplementowanej takiej metody. W ramach budowy aplikacji napisano taką funkcję opartą o stos (rys. 3.11), jednak jej możliwości są nieco mniejsze niż klasycznej funkcji. Nie ma na przykład zaimplementowanej obsługi nawiasów, przez co nie ma możliwości obliczania wyrażeń, gdzie jest istotna kolejność wykonywania działań.

Metoda polega na czytaniu kolejno znaków działania i w zależności czy dany znak jest znakiem operacji „+-*/” czy cyfrą wykonywaniu odpowiednich poleceń. Tak długo jak są kolejne cyfry są one dodawane do stosu. W momencie wystąpienia znaku operacji, cyfry są zdejmowane ze stosu i zamieniane na zmienną typu integer. Kiedy iterowanie dojdzie do momentu, że algorytm posiada dwie wartości *operand* oraz znak operacji dochodzi do obliczenia operacji oraz dodanie wyniku na stos. Te operacje są wykonywane do momentu skończenia iterowania po stringu zawierającym wyrażenie.

Rysunek 3.11 Metoda eval służąca do obliczenia wyniku operacji

```java
public static Integer eval_task(String task) {
    String expression = task;
    Stack<Integer> stack = new Stack<>();
    int len = expression.length();
    char sign = 0;
    Integer operand1 = null;

    for (int i = 0; i < len; i++) {
        char ch = expression.charAt(i);

        // Jeżeli znak jest znakiem operacji
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            sign = ch;
            operand1 = stackToInt(stack); // zamienia stos cyfr na jedną liczbę typu int
        }
        // Jeżeli znak jest cyfrą
        else if (Character.isDigit(ch)) {
            stack.push(Character.getNumericValue(ch));
        }

        // Jeśli są dwie liczby i znak operacji oblicza wynik operacji
        if (i == len-1 || ch == ' ' && (sign != 0  && stack.size() >= 1)){
            int operand2 = stackToInt(stack);
            int result = 0;
            switch (sign) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    result = operand1 / operand2;
                    break;
            }

            stack.push(result);
        }
    }
    return stackToInt(stack);
}
```

źródło: opracowanie własne

Model uczenia maszynowego jest przedstawiony na rys. 3.12. Dane są normalizowane ze skali 0-255 do 0-1. Następnie budowane są warstwy sieci neuronowej zgodnie ze strukturą sieci zaprezentowaną na rys. 2.12. Ustawienia modelu zostają skonfigurowane i uczenie zostaje rozpoczęte.

Rysunek 3.12 Model uczenia maszynowego

```python
def model(x_train, y_train, x_test, y_test):
    # normalizacja wartości koloru z 0-255 do 0-1
    x_train = x_train / 255.0,
    x_test = x_test / 255.0

    # budowa warstw sieci neuronowej
    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(784, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    # konfiguracja modelu
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    # trening modelu
    history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=18, batch_size=256)
```

źródło: opracowanie własne

Fragment kodu w Flasku implementuje metody typu GET (rys. 3.13) oraz POST (rys. 3.14) w celu obsługi żądań HTTP przysyłanych przez klienta do serwera.

W kodzie zaprezentowanym na (rys. 3.13), funkcja *index* jest przypisana do adresu URL '/' i obsługuje żądanie GET. Metoda GET służy do pobierania danych z serwera. Po wysłaniu takiego żądania klient otrzyma wiadomość typu string o treści „Machine Learning”.

Rysunek 3.13 Implementacja metody HTTP GET w Flasku

```python
@app.route('/', methods=['GET'])
def index():
    return 'Machine Learning'
```

źródło: opracowanie własne

Funkcja *infer_image* (rys. 3.14) jest przypisana do adresu URL „/predict” i obsługuje żądanie POST. Metoda sprawdza czy wraz z żądaniem został przesłany plik. Następnie metoda odczytuje przekazany obraz i uruchamia funkcje do rozdzielania obrazu liczby na cyfry, a później używa modelu do przewidzenia liczby.

Rysunek 3.14 Implementacja metody HTTP POST w Flasku

```python
@app.route('/predict', methods=['POST'])
def infer_image():
    # sprawdzenie czy przekazano plik z obrazem
    if 'file' not in request.files:
        return "Please try again. The Image doesn't exist"

    # odczyt obrazu i przekazanie go do funkcji dzielącej liczby na cyfry,
    # a następnie do funkcji wykorzystującej model do rozpoznawania cyfr
    img = Image.open(request.files['file'].stream)
    img = np.array(img)
    result = extract_digits(img)

    return jsonify(result)  # zwrócenie wyniku w formacie JSON
```

źródło: opracowanie własne

Platforma Heroku zapewnia wstrzymanie działania serwera, kiedy nie jest on wykorzystywany, niestety jego ponowne włączenie zajmuje kilkanaście sekund. Dlatego zaproponowana metoda *warmup* jest wykorzystywana do uruchomienia serwera. W momencie włączenia przez użytkownika ekranu z zadaniami jest wysyłane asynchronicznie żądanie HTTP GET, aby skrócić czas oczekiwania na pierwszą odpowiedź z serwera. Dzięki użyciu operacji asynchronicznej, aplikacja nie blokuje innych działań w oczekiwaniu na odpowiedź z serwera.

Rysunek 3.15 Metoda warmup wykorzystująca metodę HTTP GET

```java
public void warmup(){
    // stworzenie zapytania HTTP typu GET oraz wysyłanie go do serwera
    Request request = new Request.Builder()
            .url("https://guarded-island-03261.herokuapp.com")
            .get()
            .build();

    // operacja asynchroniczna
    client.newCall(request).enqueue(new Callback(){
        @Override
        public void onResponse(@NonNull Call call, @NonNull Response response) {}
        @Override
        public void onFailure(@NonNull Call call, @NonNull IOException e) {}
    });
}
```

źródło: opracowanie własne

Metoda *send* jest wykorzystywana do przesłania żądania HTTP POST zawierającego obraz png. Metoda tworzy zawartość zapytania, następnie tworzy zapytanie i asynchronicznie przesyła go do serwera. Dalej metoda sprawdza, czy informacja zwrotna została otrzymana z serwera. W przypadku otrzymania odpowiedzi, wynik jest ustawiany w polu tekstowym *text_answer* w UI aplikacji. W przypadku braku odpowiedzi, jest wyrzucany wyjątek.

Rysunek 3.16 Metoda send wysyłająca zapytania do serwera wykorzystująca metodę HTTP POST

```
75    public void send (View v){
76        PaintView x = findViewById(R.id.paintView);
77        byte[] bmp = x.viewToBitmap();
78
79        // stworzenie zawartości przesyłanego zapytania
80        RequestBody requestBody = new MultipartBody.Builder()
81                .setType(MultipartBody.FORM)
82                .addFormDataPart( name: "file",  filename: "number.png",
83                        RequestBody.create(bmp, MEDIA_TYPE_PNG))
84                .build();
85
86        // stworzenie zapytania HTTP typu POST oraz wysyłanie go do serwera
87        Request request = new Request.Builder()
88                .url("https://guarded-island-03261.herokuapp.com/predict")
89                .post(requestBody)
90                .build();
91
92        // operacja asynchroniczna
93        client.newCall(request).enqueue(new Callback() {
94            @Override
95            public void onResponse(@NonNull Call call, @NonNull Response response)
96                    throws IOException {
97                try (ResponseBody responseBody = response.body()) {
98                    if (!response.isSuccessful())
99                        throw new IOException("Unexpected code " + response);
100                   String str = responseBody.string();
101                   answer = str.replace( target: "\"",  replacement: "")
102                           .replace( target: "\n",  replacement: "").trim();
103
104                   // przekazanie rozpoznanej liczby do interfejsu użytkownika
105                   runOnUiThread(new Runnable() {
106                       @Override
107                       public void run() {
108                           TextView y = findViewById(R.id.text_answer);
109                           y.setText(answer);
110                       }
111                   });
112               }
113           }
```

źródło: opracowanie własne

Zakończenie

W ramach projektu został osiągnięty cel stworzenia podstawowej wersji aplikacji do nauki matematyki dla dzieci. Program zadaje pytania matematyczne, użytkownik ma możliwość odpowiadania za pomocą ręcznego pisma. Użytkownik ma miejsce do zapisywania swoich obliczeń. Większość odpowiedzi jest poprawnie rozpoznawana.

Aplikacja spełnia podstawowe założenia, jednak, aby przyciągnąć młodych użytkowników w przyszłości powinno się poszerzyć zakres materiału, który mogą ćwiczyć w aplikacji. Mogłyby się znaleźć kolejne tryby nauki takie jak: obwody, objętości, liczby ujemne, liczby dziesiętne, ułamki, kolejność wykonywania działań, równania. Ponadto można dodać możliwość rozdzielnia trybu działania, tak aby użytkownik mógł wybrać, czy chce poćwiczyć tylko dodawanie i odejmowanie czy mnożenie i dzielenie również. Dzięki temu aplikacja mogłaby trafić do szerszej grupy odbiorców. Podobnie w trybie pola figur wybór tylko tych figur, które dzieci poznały w danej klasie (wskazówki, które figury powinny być zaznaczone od której klasy).

W przypadku poszerzenia aplikacji o tryby z liczbami ujemnymi czy ułamkami, konieczne byłoby udoskonalenie modelu, tak aby rozpoznawał również takie odpowiedzi.

Nauka dzieci odbywa się głównie w szkołach, dlatego dobrym rozwiązaniem w aplikacji byłaby możliwość rywalizacji pomiędzy koleżankami i kolegami z klasy oraz możliwość monitorowania postępów uczniów przez nauczyciela.

Model został nauczony na podstawie obrazów cyfr pisanych przez amerykanów, dlatego ważnym usprawnieniem byłoby powiększenie zestawu danych o bardziej polskie cyfry. Na przykład dodanie większej liczby jedynek nie przypominających jedynie pionowych kresek albo dziewiątek z ogonkami. W przypadku danych z MNIST dziewiątki składają się z kółka i pionowej kreski.

Dodatkową funkcjonalnością byłyby statystki, gdzie użytkownik mógłby sprawdzić na ile pytań z danej kategorii odpowiedział poprawnie, z jakiej dziedziny musi jeszcze poćwiczyć. Ponadto przed ćwiczeniem danego zagadnienia, dzieci powinny mieć możliwość przypomnienia bądź zapoznania się z częścią teoretyczną danego zagadnienia np. zobaczyć wzory na pola figur.

Bibliografia

[1] M. Candocia, *A Simple Explanation of how Computers Recognize Images*, Analysis for Many Audiences, 2016, url: https://maxcandocia.com/article/2016/Apr/06/how-computers-recognize-images/, dostęp: 23.02.2023

[2] Wikipedia, url: https://en.wikipedia.org/wiki/Contrast_(vision), dostęp: 5.05.2023

[3] 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

[4] 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

[5] 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

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

[7] D.C. Ciresana, 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

[8] 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

[9] 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

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

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

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

[13] 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

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

[15] 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

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

[17] 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

[18] 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

[19] Bankmycell, 2023, *Android vs. Apple Market Share: Leading Mobile Operating Systems (OS) (May 2023), url:* https://www.bankmycell.com/blog/android-vs-apple-market-share/, dostęp: 18.05.2023

[20] Android Developers, *Android devices*, url: https://developer.android.com/multidevice, dostęp: 18.05.2023

[21] Oberlo, 2023, *Most Popular Electronics Worldwide,* url: https://www.oberlo.com/statistics/most-popular-electronics, dostęp: 18.05.2023

[22] G. Girecko, *Kotlin vs. Java: All-purpose Uses and Android Apps,* url: https://www.toptal.com/kotlin/kotlin-vs-java, dostęp: 24.05.2023

[23] Deeplake, *MNIST*, url: https://datasets.activeloop.ai/docs/ml/datasets/mnist/, dostęp: 18.05.2023

[24] Z. Huang, Y. Sang, Y. Sun, J. Lv, *A neural network learning algorithm for highly imbalanced data classification,* Information Sciences vol. 612, str. 496

Spis rysunków

Spis tabel