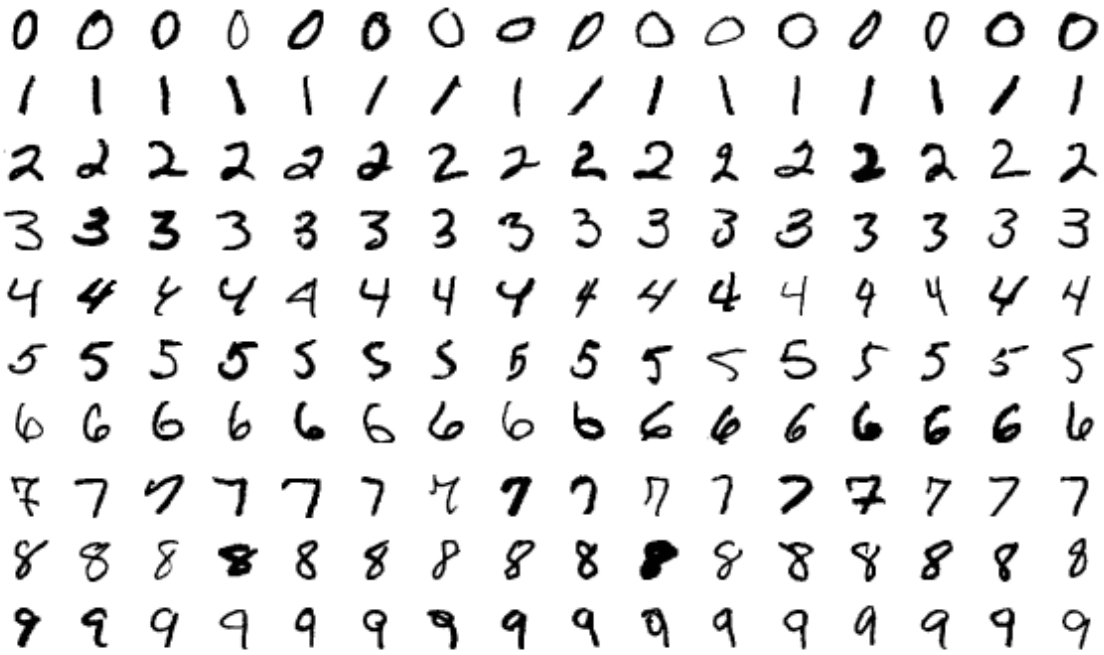
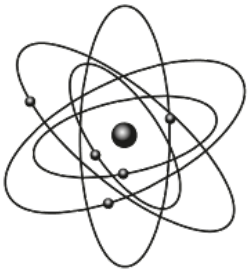
Rozpoznawanie cyfr

Projekt z Analizy Obrazów



AGH



WYDZIAŁ FIZYKI
I INFORMATYKI STOSOWANEJ

Szymon Nachlik
Kacper Starzyk

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Informatyka Stosowana
2023

1 Wstęp

Celem projektu było stworzenie aplikacji która przy użyciu kilku metod uczenia maszynowego (klasyczne uczenie maszynowe oraz konwolucyjne sieci neuronowe) będzie w stanie rozpoznać napisaną przez nas cyfrę. Jako zbioru danych wykorzystaliśmy MNIST czyli ogólnodostępną bazę obrazków w rozdzielczości 28x28 ręcznie pisanych cyfr. Zawiera ona 60,000 danych treningowych oraz 10,000 danych testowych. Przez rozdzielczość obrazów z bazy danych po każdym narysowaniu cyfry przez użytkownika program musi ją odpowiednio przeskalować.

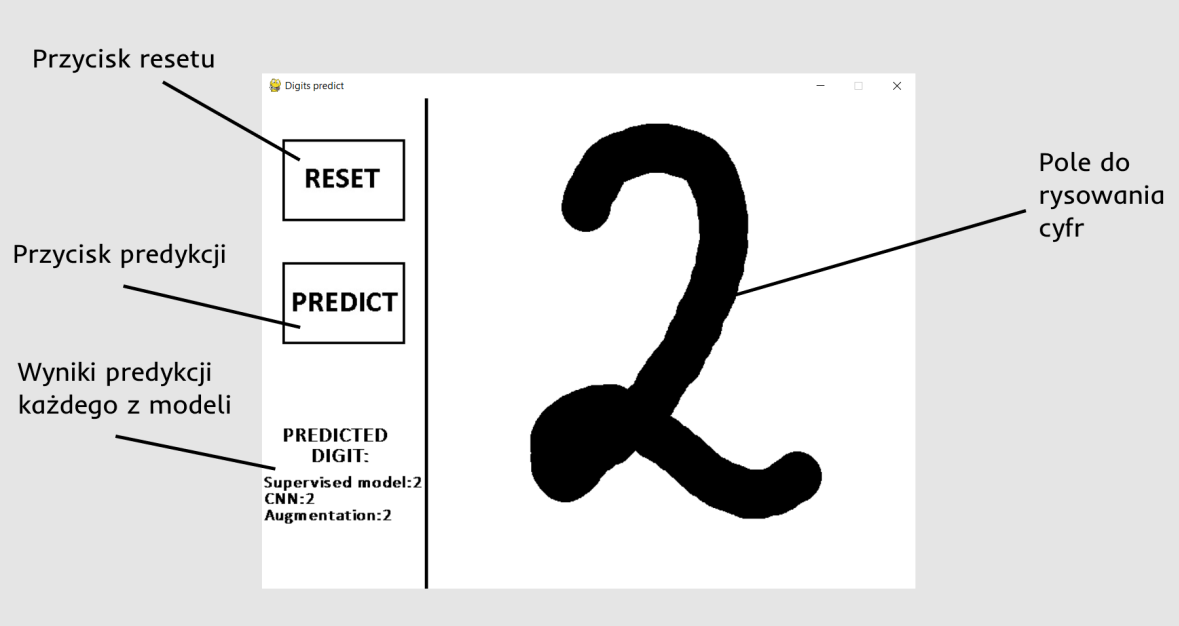
2 Wykorzystane technologie

Do projektu wykorzystaliśmy następujące biblioteki i technologie:

- python 3 - implementacja funkcjonalności, trenowanie modeli poprzez użycie bibliotek
- pygame - interfejs użytkownika i rysowanie po ekranie
- numpy - obliczenia i struktury danych przydatne w ML
- scikit-learn - klasyczne uczenie maszynowe
- tensorflow - sieci neuronowe

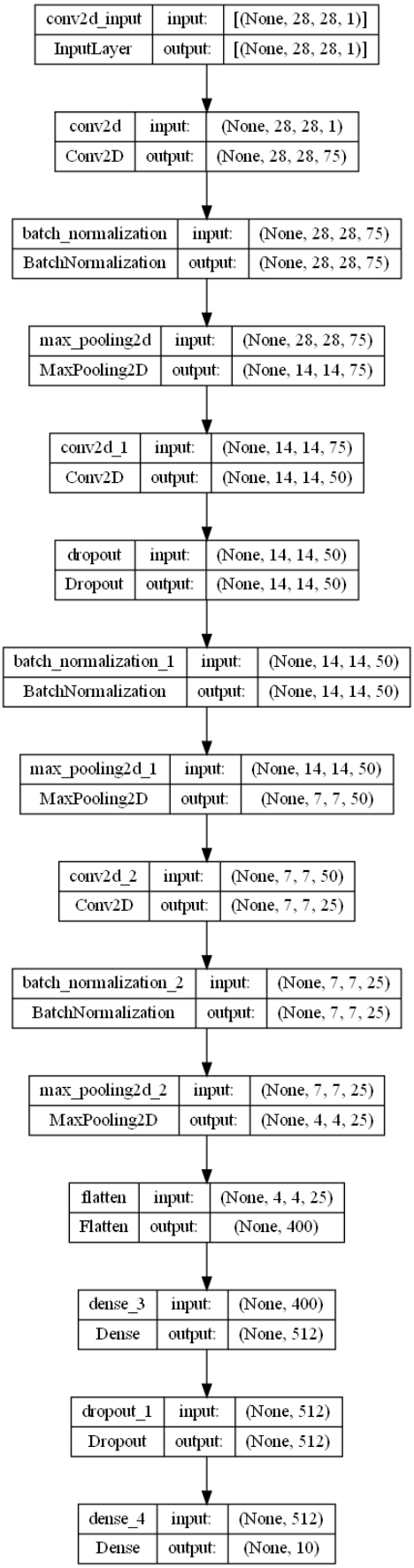
3 Opis interfejsu

Interfejs programu zawiera przyciski dzięki którym można uruchomić rozpoznawanie narysowanej przez nas cyfry, przycisk resetu, wyniki przewidywań modelu oraz pole rysowania cyfry.



4 Opis modeli

Aplikacja korzysta z trzech modeli do predykcji napisanych cyfr.


conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 28, 28, 75)
batch_normalization | input: | (None, 28, 28, 75)
BatchNormalization | output: | (None, 28, 28, 75)
max_pooling2d | input: | (None, 28, 28, 75)
MaxPooling2D | output: | (None, 14, 14, 75)
conv2d_1 | input: | (None, 14, 14, 75)
Conv2D | output: | (None, 14, 14, 50)
dropout | input: | (None, 14, 14, 50)
Dropout | output: | (None, 14, 14, 50)
batch_normalization_1 | input: | (None, 14, 14, 50)
BatchNormalization | output: | (None, 14, 14, 50)
max_pooling2d_1 | input: | (None, 14, 14, 50)
MaxPooling2D | output: | (None, 7, 7, 50)
conv2d_2 | input: | (None, 7, 7, 50)
Conv2D | output: | (None, 7, 7, 25)
batch_normalization_2 | input: | (None, 7, 7, 25)
BatchNormalization | output: | (None, 7, 7, 25)
max_pooling2d_2 | input: | (None, 7, 7, 25)
MaxPooling2D | output: | (None, 4, 4, 25)
flatten | input: | (None, 4, 4, 25)
Flatten | output: | (None, 400)
dense_3 | input: | (None, 400)
Dense | output: | (None, 512)
dropout_1 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_4 | input: | (None, 512)
Dense | output: | (None, 10)

4.1 Model korzystający z nadzorowanego uczenia maszynowego

Model zbudowaliśmy w języku Python z użyciem biblioteki Scikit-learn i bibliotekami pomocniczymi. Po pobraniu danych treningowych i krótkim preprocessingu danych (podział na dane oraz etykiety, zamiana na odpowiedni typ danych, sprawdzenie rozkładu danych, wizualizacja, przeskalowanie oraz podział na zbiory treningowe i testowe) wytrenowaliśmy wstępnie 3 modele w celu wybrania najbardziej rokującego algorytmu. Na wstępie odrzuciliśmy algorytm kNN, ponieważ końcowy model z zastosowaniem tego algorytmu miałby zbyt duży rozmiar (kilkaset MB) jak na zastosowanie do prostej aplikacji . Z algorytmów SGD Classifier, Decision Tree Classifier oraz Random Forest Classifier najlepszym okazał się ten ostatni. Kolejnym krokiem było strojenie hiperparametrów. W celu ułatwienia i przyspieszenia procesu zbudowaliśmy prostą pipeline. Walidację dostrajanych modeli uzyskaliśmy za pomocą metody walidacji krzyżowej (cross-validation). Po znalezieniu najlepszych hiperparametrów przetrenowaliśmy model na całym zbiorze treningowym. Po ewaluacji otrzymaliśmy celność modelu na poziomie 96.5%.

4.2 Model korzystający z konwolucyjnych sieci neuronowych (CNN)

Model zbudowaliśmy w języku Python z użyciem biblioteki Tensorflow i bibliotekami pomocniczymi. Podobnie jak we wcześniejszym modelu, najpierw przeprowadziliśmy preprocessing. Następnie zbudowaliśmy sieć neuronową składającą się z wielu warstw (wizualizacja sieci w pliku cnn_model_plot.png). Po kompilacji modelu wytrenowaliśmy go na zbiorze treningowym i mierząc celność za pomocą zestawu walidacyjnego. Dzięki wbudowanym metodom Tensorflow uzyskaliśmy również krzywe uczenia w celu zobaczenie przebiegu uczenia sieci. Po ewaluacji modelu na zbiorze testowym celność wyniosła 99%.

4.3 Model korzystający z konwolucyjnych sieci neuronowych (CNN) z augmentacją danych

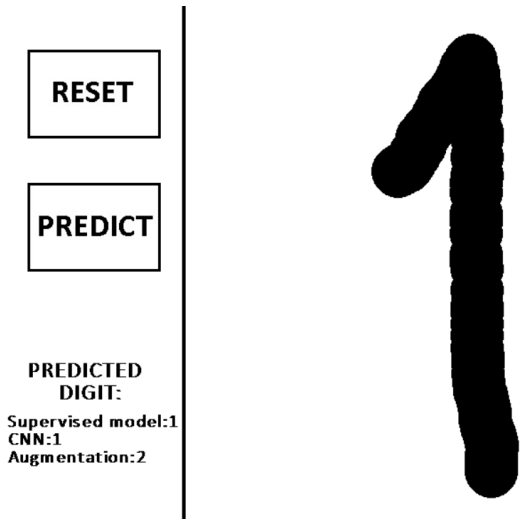
Model zbudowaliśmy w języku Python z użyciem biblioteki Tensorflow i bibliotekami pomocniczymi. Model ten nie różni się od poprzedniego preprocessingiem i samą strukturą sieci neuronowej, jednak została użyta metoda augmentacji danych, czyli generowania nowych danych. Każdy obraz ze zbioru danych treningowych jest:

- obracany o losowy kąt
- przybliżany i oddalany o losową wartość
- przesuwany poziomo i pionowo o losową wartość

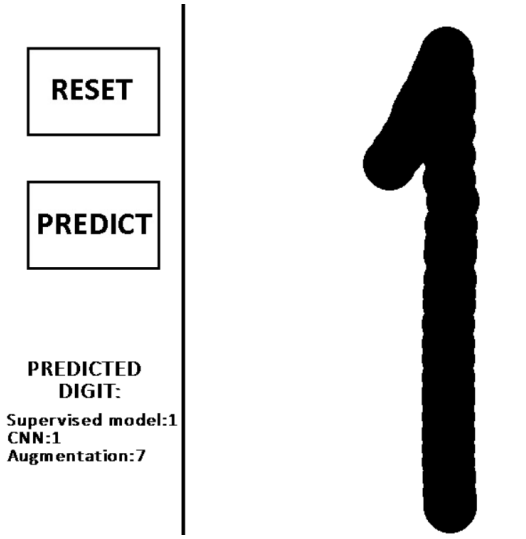
Następnie taki obraz jest użyty w uczeniu sieci neuronowej. Celem takiego zabiegu jest nie tylko dogenerowanie danych ale również zwiększenie wariancji, czyli uogólnienia modelu na nowe przykłady. Po ewaluacji uzyskaliśmy celność modelu na poziomie 99.39% czyli nieco lepiej niż poprzedni model.

5 Wnioski

W pewnych sytuacjach wyniki predykcji poszczególnych modeli są niezgodne - jest to oczywiste ponieważ korzystają one z różnych technik a każda z nich ma swoje wady i zalety. Duży wpływ na wyniki ma też zbiór wykorzystanych danych do wytrenowania modeli. MNIST jest zbiorem obrazów cyfr napisanych "w sposób amerykański". Z tego też powodu cyfra 1 będzie pionową kreską w przeciwienstwie do tego jak w Polsce piszemy tę cyfrę wykorzystując dodatkową kreskę. Z tego też powodu jeśli cyfrę zapiszemy polskim sposobem może okazać się, że zostanie ona pomylona z 7 lub w rzadszych przypadkach z 2. Co ciekawe ten błąd tyczy się głównie modelu CNN z augumentacją danych.

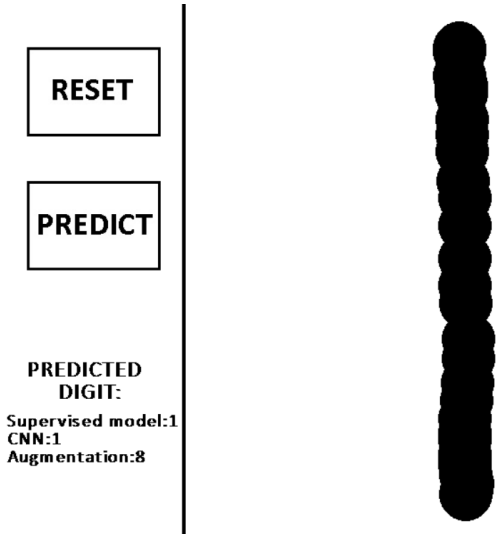


Rysunek 1: Błędna predykcja jedynki - zamiast tego augumentacja przewidziała cyfrę 2



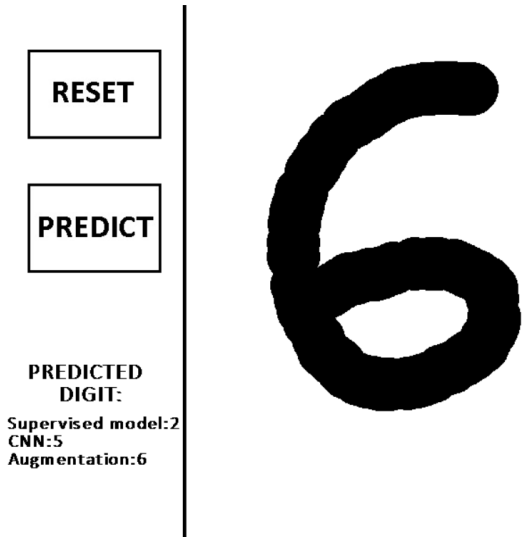
Rysunek 2: Błędna predykcja jedynki - zamiast tego augumentacja przewidziała cyfrę 7

Jeśli natomiast zapiszemy jedynkę w sposób amerykański to może pojawić się kolejny problem - tym razem ze strony biblioteki pygame która pozwala nam na rysowanie cyfr. Jeśli zbyt szybko narysujemy linię okazuje się, że uwidocznione zostają okręgi pędzla z których składa się rysunek. Przez to model błędnie interpretuje "krągłości"jako ósemkę.



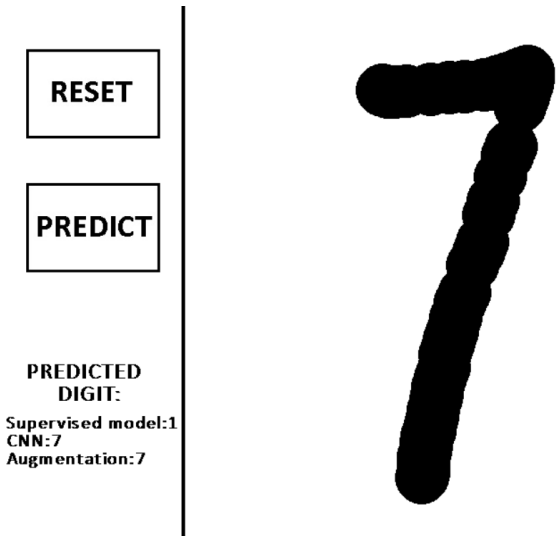
Rysunek 3: Błędna predykcja jedynki - zamiast tego augumentacja przewidziała cyfrę 8

Cyfry 2, 3, 4 oraz 5 są w znacznej większości przypadków rozpoznawane poprawnie przez wszystkie modele. Dopiero po rysowaniu cyfr obróconych pod lekkim kątem modele zaczynają mieć lekkie problemy. Najgorzej wypada tutaj nadzorowane uczenie maszynowe. Szóstka najlepiej rozpoznawana jest przez model z augumentacją. Gorzej wypadają dwa pozostałe modele - supervised ML często myli ją z 2, 3, 8 natomiast CNN w wielu przypadkach pokazuje piątkę.



Rysunek 4: Błędna predykcja szóstki przez dwa pierwsze modele

Siódemka często mylona jest w klasycznym uczeniu maszynowym z jedynką i dwójką.



Rysunek 5: Błędna predykcja siódemki przez dwa pierwsze modele

W rozpoznawaniu cyfry 8 i 9 najlepiej sprawdza się augumentacja. Dwie pozostałem metody często pokazują liczbę 3.