AKADEMIA ŁOMŻYŃSKA
WYDZIAŁ NAUK INFORMATYCZNO-TECHNOLOGICZNYCH


# ROZPOZNAWANIE RĘCZNIE PISANYCH CYFR

**Wydziałowy projekt zespołowy**

**Zespół**

Jakub Duda

Paweł Goliński

Oskar Cezary Krajza

**Prowadzący ćwiczenia**

Dr inż. Janusz Rafałko

Informatyka

Studia stacjonarne I stopnia, rok IV, semestr VII

Rok akademicki: 2024/2025

# Spis treści

## Opis projektu

Celem projektu jest opracowanie programu, który będzie w stanie rozpoznawać ręcznie pisane cyfry. Wykorzystamy do tego zbiór danych MNIST oraz SVHN, które posłużą do trenowania modelu. Projekt będzie bazować na technikach głębokiego uczenia, co umożliwi uzyskanie wysokiej dokładności rozpoznawania cyfr. Docelowo do programu będziemy mogli wprowadzać własne cyfry.

## Podział zadań

Projekt będzie realizowany przez zespół składający się z trzech członków:

1. Jakub Duda: Przygotowanie instrukcji użytkownika, dokumentacja metod oraz stworzenie pseudokodu. Przygotowanie ilustracji. Opracowanie wniosków.

2. Paweł Goliński: Odpowiedzialny za implementacje modelu TensorFlow oraz dobór odpowiednich architektur sieci neuronowych. Przygotowanie GUI, ocena efektywności metod. Pomoc w opracowaniu wniosków.

3. Oskar Cezary Krajza: Testowanie programu, Opis podstawowej wiedzy teoretycznej. Wykonanie eksperymentów takich jak dokładność. Pomoc w opracowaniu wniosków.

## Język programowania

- Python

Narzędzia do przetwarzania danych:

- Zbiory danych: MNIST (ręcznie pisane cyfry) oraz SVHN (cyfry na zdjęciach z ulic).
- Biblioteki: TensorFlow, Keras, NumPy, Matplotlib, PyQt, Scikit-Learn
- Metody przetwarzania: Normalizacja obrazów, augmentacja danych

## Instrukcja użytkownika

Uruchamianie programu do rozpoznawania ręcznie pisanych cyfr

Aby uruchomić program, wykonaj poniższe kroki:

**Krok 1:** Instalacja Pythona

Przejdź na oficjalną stronę Pythona: https://www.python.org.
Pobierz najnowszą wersję Pythona odpowiednią dla Twojego systemu operacyjnego (Windows, macOS, Linux).
Podczas instalacji upewnij się, że zaznaczyłeś opcję Add Python to PATH.
Po zakończeniu instalacji sprawdź, czy Python działa poprawnie, wpisując w terminalu/poleceniu:
bash
Skopiuj kod
python –version

**Krok 2:** Instalacja niezbędnych bibliotek

W terminalu lub wierszu poleceń wpisz kolejno poniższe komendy, aby zainstalować wymagane biblioteki:

Zainstaluj bibliotekę PyQt5:
*pip install PyQt5*

Zainstaluj TensorFlow:
*pip install tensorflow*

Zainstaluj bibliotekę OpenCV:
*pip install opencv-python*

Zainstaluj bibliotekę pillow:
*pip install pillow*

Zainstaluj bibliotekę scikit-learn:
*pip install scikit-learn*

Zainstaluj bibliotekę scipy:
*pip install scipy*

Jeśli program wymaga dodatkowych bibliotek, takich jak NumPy czy Matplotlib, upewnij się, że są one również zainstalowane:
*pip install numpy matplotlib*

**Krok 3:** Pobranie i uruchomienie programu

Otwórz folder, w którym znajduje się program i wytrenowany model (plik modelu: model_Svhn.keras).
Uruchom główny plik programu z rozszerzeniem .py

**Krok 4:** Korzystanie z programu

Po uruchomieniu programu otworzy się graficzny interfejs użytkownika (GUI).
W GUI możesz rysować cyfry i zatwierdzać je do rozpoznania.
Program automatycznie przetworzy wprowadzone dane i wyświetli przewidywaną cyfrę.

**Uwagi dodatkowe:**

Jeśli pojawią się jakiekolwiek błędy podczas instalacji lub uruchamiania, sprawdź, czy używasz kompatybilnej wersji Pythona (zalecana wersja: 3.12.7).
W przypadku problemów z bibliotekami spróbuj zaktualizować pip:
*python -m pip install --upgrade pip*

Teraz program jest gotowy do użycia!

**Podstawy teoretyczne i źródła**

Aplikacja opiera się na technikach głębokiego uczenia, w szczególności wykorzystuje konwolucyjne sieci neuronowe (CNN) do klasyfikacji obrazów.
Sieci konwolucyjne są idealne do przetwarzania danych obrazowych, ponieważ uwzględniają lokalne zależności między pikselami (np. krawędzie, tekstury).
Przykłady warstw w CNN: warstwy konwolucyjne, warstwy pooling (maksymalne lub uśredniające), warstwy w pełni połączone.

**Zbiory danych:**

- SVHN: Zbiór danych przedstawiający cyfry wycięte ze zdjęć ulicznych, idealny do trenowania modelu rozpoznawania cyfr.
- MNIST: Zbiór danych przedstawiający ręcznie pisane cyfry.

Dane są normalizowane i przetwarzane na obrazy o rozmiarze 32x32 pikseli.

**Technologie:**

- TensorFlow i Keras: Do budowy i trenowania modeli.
- PyQt: Do stworzenia graficznego interfejsu użytkownika (GUI).
- OpenCV i NumPy: Do przetwarzania obrazów.

## Źródła:

*Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow.*
*François Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras.*

## Opis algorytmów i funkcji

### Główne metody i funkcje

Przetwarzanie obrazu z płótna użytkownika
Celem tej funkcji jest przekształcenie rysunku użytkownika na obraz 32x32, który można wprowadzić do modelu.
Pseudokod:

```
INPUT: Obraz z płótna użytkownika (320x320 pikseli)
KONWERSJA:
    Pobierz dane obrazu w formacie RGBA
    Konwertuj obraz na RGB
    Zmień rozmiar na 32x32 piksele
    Normalizuj piksele dzieląc przez 255.0
PRZEKSZTAŁĆ:
    Dodaj wymiar, aby utworzyć tablicę 4D (1, 32, 32, 3)
WYJŚCIE: Tablica 4D obrazu gotowego do analizy przez model
```

Kod funkcji: process_drawing

### Predykcja klasy cyfry przez model

Funkcja przewiduje cyfrę na podstawie wprowadzonego obrazu, wykorzystując wcześniej wczytany model TensorFlow.

Pseudokod:

```
INPUT: Przetworzony obraz (1, 32, 32, 3)
PRZEWIDYWANIE:
    Przeprowadź predykcję za pomocą modelu
    Wyznacz etykietę klasy z najwyższym prawdopodobieństwem
    Wyznacz pewność predykcji (maksymalna wartość prawdopodobieństwa)
WYJŚCIE: Etykieta klasy i pewność predykcji
```

Kod funkcji: process_drawing

Wczytywanie modelu
Funkcja ładuje wytrenowany model sieci neuronowej zapisany w formacie TensorFlow/Keras.
Pseudokod:

```
INPUT: Plik modelu (np. model_Svhn.keras)
WCZYTAJ MODEL:
    Załaduj model przy użyciu TensorFlow
WYJŚCIE: Obiekt modelu gotowy do użycia
```

Kod funkcji: create_model

**GUI:**

Funkcje związane z obsługą płótna: mousePressEvent, mouseMoveEvent, mouseReleaseEvent.
Funkcje zarządzania płótnem: clear_canvas, update_canvas.

## Studium przypadków

### Przypadek 1: Rozpoznawanie cyfry "5"

Wejście: Użytkownik rysuje cyfrę "5" na płótnie.
Przetwarzanie:
Obraz 320x320 pikseli jest zmniejszany do 32x32 i normalizowany.
Model przewiduje etykietę klasy jako "5".
Wynik:
Wyświetlana etykieta: Rozpoznano cyfrę: 5 z pewnością: 95.67%.

### Przypadek 2: Rozpoznawanie cyfry "0"

Wejście: Użytkownik rysuje okrąg symbolizujący cyfrę "0".
Przetwarzanie:
Obraz przechodzi przez tę samą ścieżkę przetwarzania.
Model przewiduje etykietę klasy jako "0".
Wynik:
Wyświetlana etykieta: Rozpoznano cyfrę: 0 z pewnością: 98.12%.

## Opis wykorzystanych zbiorów danych

### Zbiór MNIST:

Zawiera 60 000 obrazów treningowych i 10 000 testowych.
Obrazy to ręcznie pisane cyfry w skali szarości o wymiarach 28x28 pikseli.
Dane te są powszechnie stosowane jako benchmark w zadaniach klasyfikacji cyfr.
W projekcie MNIST był analizowany, ale nie został wybrany ze względu na niższą złożoność wizualną cyfr w porównaniu do SVHN.

**Zbiór SVHN (Street View House Numbers):**

Składa się z ponad 600 000 obrazów cyfr wyciętych z numerów domów na zdjęciach ulicznych.
Każdy obraz ma wymiary 32x32 piksele i zawiera więcej szczegółów niż cyfry w MNIST.
Dane zostały przetworzone przez normalizację pikseli (zakres 0–1) oraz zamianę etykiety klasy "10" na "0".

Aby ocenić skuteczność modelu, przeprowadzono testy na zestawie testowym SVHN oraz na danych własnych (cyfry rysowane przez użytkownika).

Miary efektywności na zestawie SVHN:
Dokładność (accuracy):

$$\text{Accuracy} = \frac{\text{Liczba poprawnych klasyfikacji}}{\text{Całkowita liczba próbek}}$$

Wynik: 92.3%.

Macierz pomyłek (confusion matrix): Analiza pomyłek pokazała, że cyfry "1" i "7" były najczęściej mylone ze względu na podobieństwo kształtów w danych wizualnych.

| Rzeczywista \ Predykcja | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 580 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 510 | 3 | 5 | 0 | 0 | 0 | 12 | 1 | 0 |
| 2 | 0 | 6 | 540 | 8 | 1 | 1 | 2 | 3 | 10 | 0 |
| 3 | 0 | 7 | 4 | 530 | 0 | 10 | 1 | 3 | 8 | 2 |
| 4 | 1 | 2 | 0 | 0 | 550 | 1 | 4 | 5 | 2 | 1 |
| 5 | 0 | 1 | 1 | 11 | 0 | 530 | 2 | 3 | 6 | 1 |
| 6 | 1 | 1 | 1 | 0 | 3 | 2 | 540 | 1 | 0 | 0 |
| 7 | 0 | 11 | 2 | 5 | 1 | 1 | 0 | 520 | 2 | 3 |
| 8 | 2 | 3 | 6 | 8 | 1 | 3 | 1 | 3 | 520 | 2 |
| 9 | 0 | 2 | 1 | 4 | 10 | 2 | 0 | 6 | 5 | 520 |

## Hipotezy badawcze

### Hipoteza

Najlepsza skuteczność modelu została osiągnięta przy 5 próbach trenowania na danych SVHN. Zwiększenie liczby prób do 10 pogorszyło wyniki, a przy 50 próbach dokładność modelu spadła znacząco do 0.17.

### Obserwacje eksperymentalne:

1. Przy 5 próbach: Model osiągnął najwyższą dokładność wynoszącą 92.8%.
2. Przy 10 próbach: dodano samą augmentacja danych, nastąpiło pogorszenie wyników – dokładność spadła do około 85%.
3. Przy 50 próbach: dodano Dropout + augmentacja danych (Dropout to technika regularyzacyjna w sieciach neuronowych, polegająca na losowym "wyłączaniu" neuronów podczas treningu, aby zapobiec przeuczeniu i wymusić bardziej uniwersalne wzorce). Dokładność modelu była bardzo niska, na poziomie 17%, co sugeruje znaczną degradację skuteczności.

### Interpretacja wyników:

- Pogorszenie wyników przy 10 i 50 próbach wskazuje na brak balansu między złożonością modelu, technikami regularyzacji oraz liczbą prób treningowych. Model wydaje się być wrażliwy na zbyt dużą ilość losowości w procesie uczenia, co może być efektem niewłaściwego dostosowania hiperparametrów w eksperymentach z Dropoutem i augmentacją danych.

## Wnioski

W ramach projektu zespołowego "Rozpoznawanie ręcznie narysowanych cyfr", skupiliśmy się na porównaniu zestawów danych MNIST i SVHN.
Nasze testy wykazały, że zestaw MNIST, z 60 000 obrazami, w manualnych testach dawał skuteczność około 30%. Dla porównania, SVHN zawierający 531 131 zdjęć osiągnął ponad 90% skuteczności. Choć SVHN oferuje znacznie lepsze wyniki, jest większy (1,3 GB) i wymaga dłuższego czasu trenowania modelu.
Równocześnie pracowaliśmy nad GUI do rysowania cyfr przez użytkownika. Początkowo wybraliśmy Tkinter, jednak napotkaliśmy problem z zapisem obrazów do plików .png przy użyciu funkcji ImageGrab.grab(bbox=(x, y, x1, y1)), co działało poprawnie tylko na jednej z maszyn. Mimo że Tkinter jest prostszy, zdecydowaliśmy się na PyQt, aby uniknąć tych problemów. Odrzuciliśmy także rozwiązanie wymagające instalacji Ghostscript.
Ostatecznie zdecydowaliśmy, że GUI będzie stworzone w PyQt, a model zostanie trenowany na zestawie danych SVHN, który zapewnia lepsze wyniki w rozpoznawaniu cyfr

## Bibliografia

1. Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow. Wydanie III
Publikacja ta koncentruje się na zastosowaniu TensorFlow i Keras do rozpoznawania obrazów. Zawiera przykłady praktyczne, które mogą pomóc w implementacji modelu do rozpoznawania ręcznie pisanych cyfr.
2. François Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras
W książce znajdują się praktyczne przykłady implementacji modeli przy użyciu Pythona i Keras, co umożliwia czytelnikom szybkie wdrożenie nabytej wiedzy w projektach. Chollet szczegółowo omawia różne architektury sieci neuronowych, techniki optymalizacji, a także metody przetwarzania danych