Analiza Obrazów
Odczytywanie tablic rejestracyjnych z obrazów

26.01.2025

Krzysztof Jabłoński
Szymon Łabędziewski
Mateusz Wawrzyczek
Artur Saganowski

1. Wstęp

Celem projektu jest stworzenie systemu do automatycznego rozpoznawania tablic rejestracyjnych z obrazów przy użyciu technik uczenia maszynowego. System ma umożliwiać szybkie i dokładne wykrywanie oraz odczytywanie numerów rejestracyjnych, co może być wykorzystywane w monitorowaniu ruchu drogowego, systemach parkingowych czy egzekwowaniu przepisów.

Zakres projektu obejmuje wykrywanie tablic na obrazie przy użyciu modelu pretrained YOLO, segmentację znaków oraz ich klasyfikację przy użyciu sieci neuronowej. Projekt obejmuje także generowanie zestawów danych, testowanie skuteczności modelu oraz stworzenie interfejsu użytkownika umożliwiającego łatwe wprowadzanie i przetwarzanie obrazów. Dodatkowo, wdrożono automatyzację instalacji i uruchamiania aplikacji.

W projekcie wykorzystano język Python oraz biblioteki takie jak YOLO (Ultralytics) do wytrenowania modelu detekcji tablic, TensorFlow do klasyfikacji znaków, OpenCV do przetwarzania obrazów i Tkinter do stworzenia interfejsu użytkownika. Automatyzację procesu instalacji zapewnia skrypt powłoki Bash.

2. Opis projektu

Główne funkcjonalności projektu obejmują automatyczne rozpoznawanie tablic rejestracyjnych na obrazach poprzez wykrywanie ich lokalizacji, segmentację znaków oraz klasyfikację liter i cyfr. System umożliwia użytkownikowi załadowanie obrazu, przetworzenie go i wyświetlenie rozpoznanego numeru rejestracyjnego w interfejsie graficznym. Projekt wspiera generowanie własnych zestawów danych do trenowania modeli, testowanie dokładności systemu oraz automatyczną instalację wymaganych zależności za pomocą dedykowanego skryptu. Dodatkowo, aplikacja jest zoptymalizowana do pracy na procesorze, co pozwala na jej łatwe wdrożenie na różnych platformach.

Główne katalogi i pliki to:

README.md – Dokumentacja opisująca działanie projektu oraz instrukcje instalacji.

skrypt.sh – Skrypt automatyzujący instalację zależności i uruchamianie aplikacji.

main.py – Główny skrypt aplikacji z interfejsem graficznym do przetwarzania obrazów.

letter_regonizer.py – Moduł odpowiadający za wykrywanie tablic, przetwarzanie obrazu oraz rozpoznawanie znaków.

test_model.py – Skrypt testujący dokładność modelu YOLO w detekcji tablic rejestracyjnych.

model/ – Folder zawierający wytrenowane modele:

best.pt – Model YOLO do wykrywania tablic rejestracyjnych.

letter_recognition_model.h5 – Model klasyfikacji znaków.

images/ – Przykładowe obrazy używane do testowania modelu.

machine_learning/ – Skrypty związane z uczeniem maszynowym:

script.py – Pobiera zestawy danych i trenuje model YOLO.

machine_learning_letter_recognition/ – Zawiera kod do trenowania modelu rozpoznawania znaków:

create_dataset.py – Generuje dane treningowe na podstawie różnych czcionek.

train.py – Skrypt do trenowania modelu klasyfikacji liter.

fonts/ – Folder z czcionkami wykorzystywanymi do generowania danych.

test_model.py – Skrypt do weryfikacji działania wytrenowanego modelu na nowych danych.

Projekt dostępny jest w formie repozytorium na githubie pod linkiem:
https://github.com/KeyJayY/AO-Project

3. Instalacja i konfiguracja

a) Wymagania sprzętowe i programowe

Do poprawnego działania aplikacji wymagany jest komputer z systemem operacyjnym Linux (np. Ubuntu, Fedora, Arch Linux) lub Windows. Aplikacja wymaga środowiska Python w wersji 3.8–3.11, ponieważ nowsze wersje nie są jeszcze w pełni kompatybilne z biblioteką TensorFlow używaną w projekcie.

b) Proces instalacji Linux (użycie skrypt.sh)

Instalacja aplikacji odbywa się poprzez wykonanie skryptu skrypt.sh, który automatycznie instaluje wszystkie niezbędne zależności i uruchamia aplikację. Skrypt sprawdza system operacyjny i dobiera odpowiednie komendy instalacyjne dla różnych dystrybucji Linuxa. Główne kroki instalacji obejmują:

- Sprawdzenie obecności Pythona (w wersji 3.8–3.11) i menedżera pakietów pip.
- Instalację wymaganych bibliotek:
  - Os – służy do zarządzania plikami i ścieżkami
  - Cv2 – biblioteka służąca do przetwarzania obrazów
  - Datasets - biblioteka umożliwiająca łatwy dostęp i udostępnianie zestawów danych do zadań związanych z dźwiękiem, komputerowym widzeniem i przetwarzaniem języka naturalnego
  - YOLO – klasa z biblioteki ultralytics służąca do obsługi modeli YOLO
  - Tkinter – biblioteka umożliwiająca tworzenie Interface'u graficznego użytkownika
  - PIL.Image – bibliotek wykorzystywana do obsługi obrazów
- Uruchomienie głównego skryptu aplikacji main.py.
- Aby uruchomić instalację, należy wykonać w terminalu polecenie:
  - ./skrypt.sh

c) Proces instalacji Windows

Aby program działał poprawnie system Windows musi posiadać zainstalowane środowisko Python. Przed pierwszym uruchomieniem należy zainstalować wszystkie potrzebne biblioteki, najprościej można to zrobić używając instalatora pakietów python (pip) poleceniem "pip install -r requirements.txt" w głównym katalogu aplikacji.

Alternatywnie użytkownicy Windows mogą skorzystać z WSL i uruchomić skrypt skrypt.sh.

d) Uruchamianie aplikacji

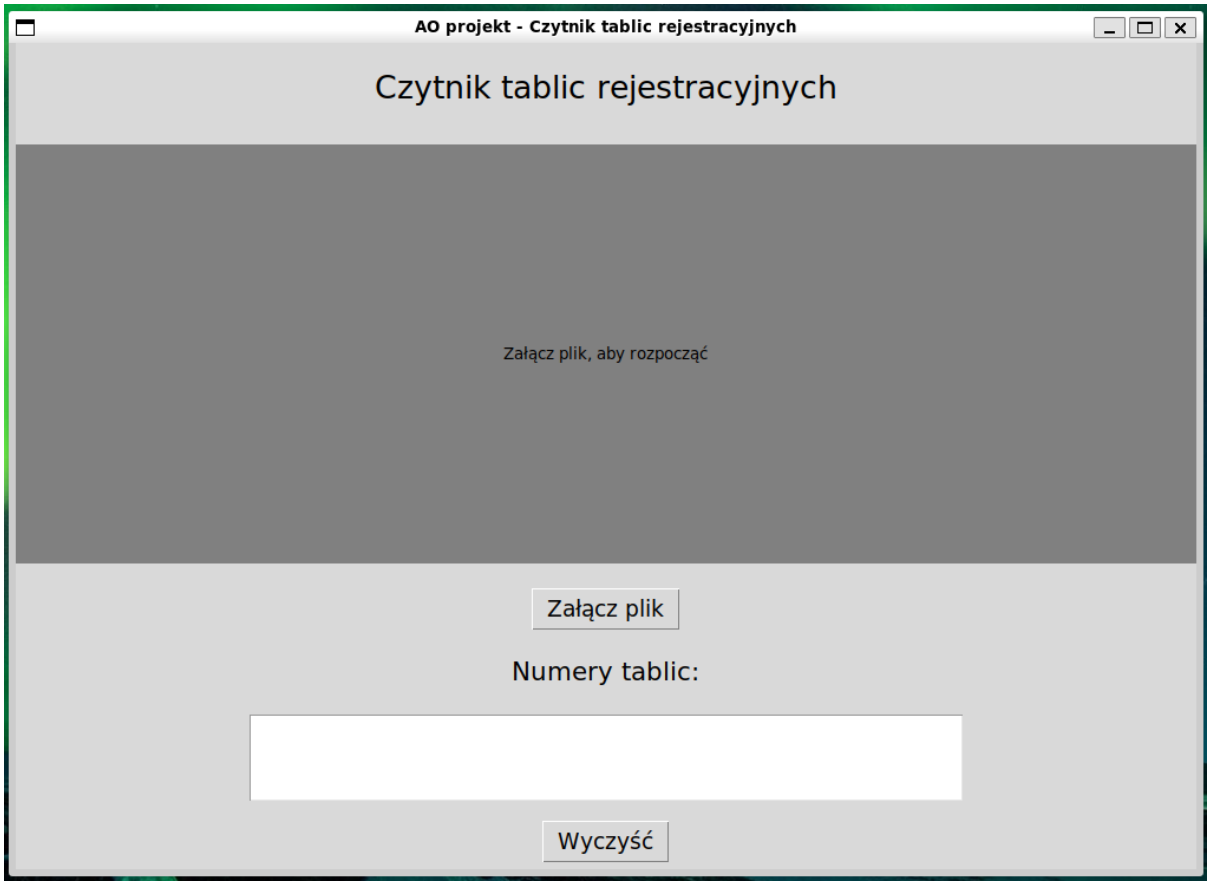
Po pomyślnej instalacji aplikację można uruchomić bezpośrednio za pomocą skryptu instalacyjnego lub ręcznie poprzez wykonanie komendy:

- python3 main.py (python main.py dla Windows)

Aplikacja otworzy interfejs graficzny, który umożliwia załadowanie obrazu tablicy rejestracyjnej i przetworzenie go w celu rozpoznania znaków. W przypadku problemów z uruchomieniem należy upewnić się, że wszystkie wymagane biblioteki zostały poprawnie zainstalowane oraz że ścieżki do plików modeli (best.pt, letter_recognition_model.h5) są prawidłowe. W katalogu images dostępne są przykładowe obrazy, których można użyć w celu przetestowania aplikacji.

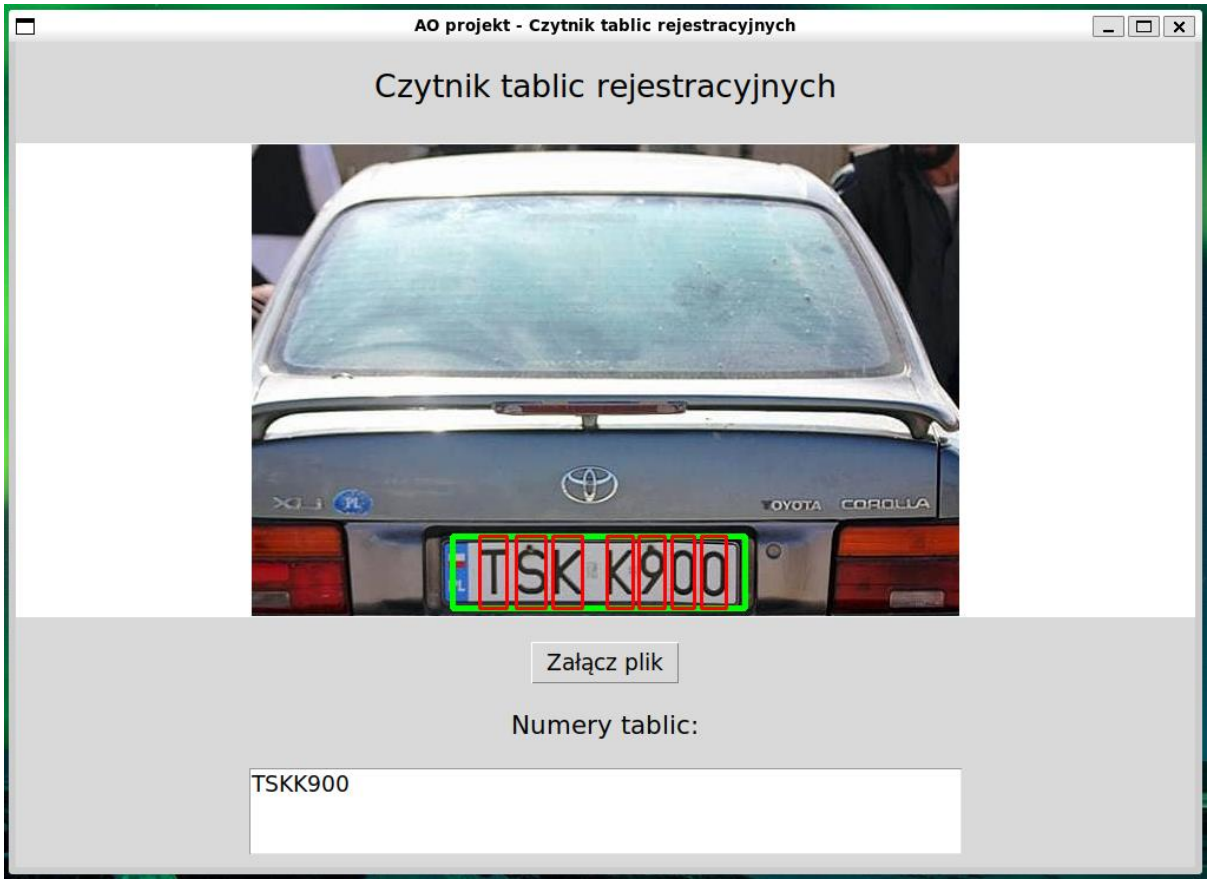
4. Graficzny Interface Użytkownika

Aplikacja posiada intuicyjny i przejrzysty interfejs graficzny zrealizowany za pomocą biblioteki **Tkinter**, który umożliwia łatwą obsługę procesu rozpoznawania tablic rejestracyjnych. Po uruchomieniu aplikacji użytkownik widzi główne okno z następującymi elementami:



- **Nagłówek:** „Czytnik tablic rejestracyjnych” – informujący o funkcji aplikacji.
- **Pole ładowania obrazu:** szare pole na środku ekranu z komunikatem „Załącz plik, aby rozpocząć”, które zmienia się po załadowaniu obrazu.
- **Przycisk „Załącz plik”** – umożliwia wybór obrazu z dysku użytkownika.

W razie problemów z wyświetlaniem podglądowych plików do wyboru, wystarczy w docelowej lokalizacji wpisać nazwę istniejącego pliku do panelu nazwy i zatwierdzić wybór.



- **Pole wyświetlające rozpoznany numer tablicy** – znajduje się pod przyciskiem i wyświetla wynik przetwarzania obrazu.
- **Przycisk „Wyczyść”** – pozwala na wyczyszczenie wyników i załadowanie nowego obrazu.

Po załadowaniu obrazu aplikacja wyświetla zdjęcie z zaznaczoną tablicą rejestracyjną (zielona ramka) oraz wykrytymi znakami (czerwone ramki na poszczególnych literach i cyfrach).

Aplikacja umożliwia wprowadzanie danych wejściowych poprzez wybór pliku graficznego z tablicą rejestracyjną za pomocą przycisku „Załącz plik”, co otwiera okno dialogowe umożliwiające nawigację po systemie plików i wybór obrazu. Obsługiwane formaty plików to:

- **.png**
- **.jpg**
- **.jpeg**
- **.bmp**

Po wybraniu obrazu aplikacja automatycznie przetwarza zdjęcie i wykonuje następujące kroki:

1. Detekcja tablicy rejestracyjnej na obrazie przy użyciu modelu opartego o YOLO.
2. Przycięcie i segmentacja tablicy w celu wyodrębnienia znaków.
3. Klasyfikacja poszczególnych liter i cyfr.
4. Wyświetlenie przetworzonego obrazu z zaznaczonymi wykrytymi znakami oraz wyniku w polu tekstowym.

Jeśli użytkownik chce przetworzyć inny obraz, może użyć przycisku **„Wyczyść”**, który resetuje widok i umożliwia załadowanie nowego pliku.

5. Sposób działania aplikacji:

Proces rozpoznawania tablicy rejestracyjnej i odczytywania z niej tekstu podzielić można na 3 części niezależne części.

- W pierwszej kolejności na obrazie wykrywana jest lokalizacja tablicy rejestracyjnej przy użyciu modelu opartego na modelu pretrained YOLO
- Następnie wycięty fragment z tablicą poddawany jest przekształcenią mającym na celu segmentacje obrazu i wyodrębnienie poszczególnych znaków
- Następnie każdy ze znaków przekazywany jest do modelu, który klasyfikuje znaki.

Poszczególne części są dokładniej opisane w dalszej części sprawozdania.

6. Wykrywanie tablic rejestracyjnych

Aby poprawnie rozpoznawać tablice rejstracyjne, należy wytrenować model. Funkcja prepare_yolo_dataset zajmuje się pobraniem gotowego zestawu danych z Hugging Face. Zestaw ten zawiera zdjęcia samochodów oraz adnotacje w formie bounding boxów (prostokątów/ramek), które wskazują położenie tablic rejestracyjnych na obrazach.

Dane te są następnie konwertowane na format wymagany przez YOLO. Konwersja obejmuje m.in. zapis zdjęć do katalogów images/train oraz images/val (dla danych treningowych i walidacyjnych) oraz przeliczenie współrzędnych bounding boxów do formatu YOLO. Format ten reprezentuje położenie prostokątów za pomocą współrzędnych środka (x_center, y_center) oraz względnych szerokości i wysokości w stosunku do wymiarów obrazu. Ostatecznie dla każdego obrazu tworzone są pliki tekstowe .txt, które zawierają te dane w odpowiednim formacie.

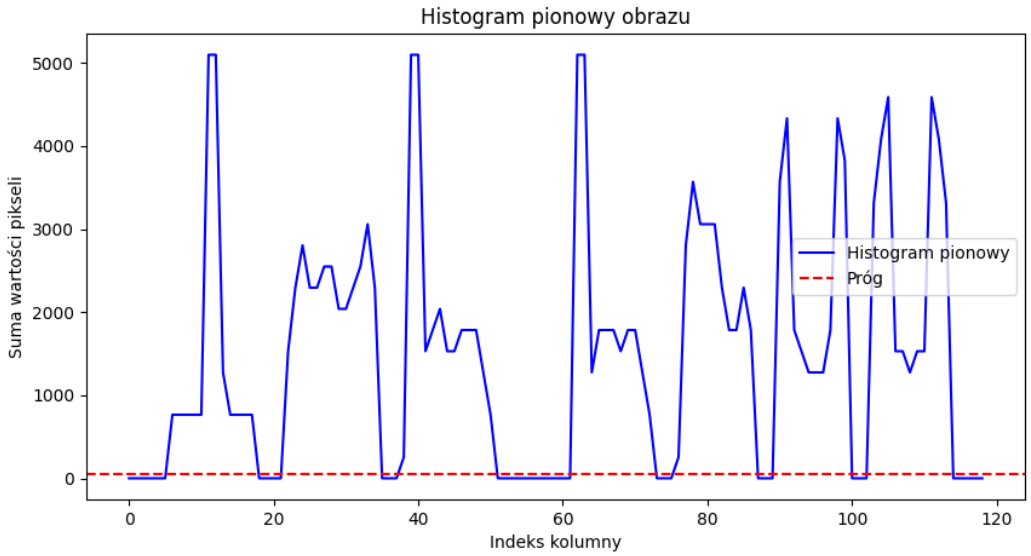
Następnie funkcja train_yolo_model zajmuje się trenowaniem modelu YOLO. W pierwszej kolejności generowany jest plik konfiguracyjny data.yaml, który określa ścieżki do danych treningowych i walidacyjnych oraz liczbę klas – w tym przypadku tylko jedną, czyli tablice rejestracyjne. Model YOLOv8 w wersji „nano” jest inicjalizowany z wykorzystaniem gotowego modelu typu pretrained. Następnie model jest trenowany przez pięć epok z wykorzystaniem przygotowanego wcześniej zestawu danych. W trakcie tego procesu model uczy się wykrywać tablice rejestracyjne na podstawie dostarczonych zdjęć i etykiet. Wytrenowany model wykorzystywany jest do wykrywania tablic rejestracyjnych na nowych obrazach. Poniżej przedstawiono przykładowe działanie modelu, współrzędne zwrócone przez model zostały naniesione na obraz.

7. Wyodrębnianie znaków z obrazu tablicy rejestracyjnej

Po wycięciu fragmentu zawierającego tablicę rejestracyjną otrzymujemy następujący obraz.

Następnie obraz progowany jest z wykorzystaniem metody Otsu oraz usuwane są zaburzenia z brzegów tablicy.

Korzystając z tak przetworzonego obrazu sumując ilości białych pikseli w kolumnach uzyskać możemy histogram pionowy obrazu



Analizując powyższy histogram łatwo możemy określić, w których miejscach znajdują się litery, a w których puste przestrzenie. Poniżej przedstawiony został obraz z nałożonymi granicami znaków.

Tak wyodrębnione znaki przekazywane są do modelu, który je klasyfikuje.

8. Trenowanie modelu rozpoznawania znaków

Proces trenowania sieci konwolucyjnej rozpoczęto od przygotowania środowiska, gdzie określane są podstawowe parametry, takie jak ścieżki do dwóch zbiorów danych – jeden zawierający rzeczywiste zdjęcia znaków z tablic rejestracyjnych, drugi zawierający sztucznie wygenerowane obrazy czarnych znaków na białym tle – oraz wielkość obrazów, liczba epok i rozmiar batcha.

Pierwszym krokiem jest wczytywanie i wstępne przetwarzanie danych. Funkcja preprocess_image przekształca każdy obraz w zestawie danych na skalę szarości, binaryzuje go przy użyciu progowania Otsu, a następnie znajduje największy kontur na obrazie, który odpowiada interesującemu obiektowi, czyli literze lub cyfrze. Wykorzystując współrzędne konturu, obraz jest przycinany do obszaru obiektu, a następnie przetwarzany przez operację zamykania morfologicznego, aby usunąć drobne szumy. Ostatecznie obraz zostaje zmieniony na ustalony rozmiar, co zapewnia jednolity format wejściowy dla sieci.

Funkcja load_dataset iteruje po katalogach zbioru danych, wczytując obrazy i etykiety dla każdej klasy. Obrazy są przetwarzane przez preprocess_image, a następnie zapisywane w liście jako tablica, podczas gdy ich etykiety są reprezentowane jako numery klas odpowiadające nazwom katalogów. Dane z obu zbiorów są łączone w jedną strukturę.

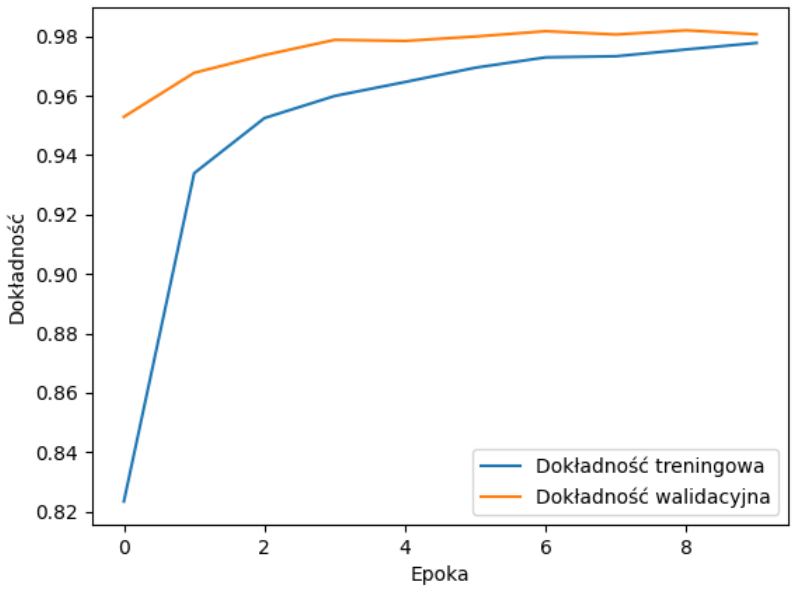
Po wczytaniu danych są one normalizowane, czyli wartości pikseli są skalowane do przedziału [0, 1]. Aby sieć konwolucyjna mogła przetwarzać obrazy, dodawany jest kanał głębi, zmieniając

wymiary danych na (64, 64, 1). Etykiety są przekształcane na format one-hot encoding, co pozwala na ich wykorzystanie w klasyfikacji wieloklasowej. Dane są następnie dzielone na trzy zestawy: treningowy (70%), walidacyjny (15%) i testowy (15%).

W kolejnym etapie budowany jest model sieci neuronowej przy użyciu funkcji build_model. Model składa się z trzech warstw konwolucyjnych, które stosują filtry do wykrywania charakterystycznych cech obrazu, takich jak krawędzie i tekstury. Po każdej warstwie konwolucyjnej następuje pooling maksymalny, który zmniejsza wymiar danych i poprawia wydajność modelu. Po wydobyciu cech dane są spłaszczane do jednowymiarowego wektora, a następnie przechodzą przez gęstą warstwę z 128 neuronami, gdzie odbywa się dalsza analiza. Wykorzystano również warstwę Dropout, aby zredukować problem nadmiernego dopasowania. Ostateczna warstwa modelu to warstwa softmax, która zwraca prawdopodobieństwa przynależności obrazu do każdej z klas.

Model jest kompilowany z optymalizatorem Adam oraz funkcją straty categorical_crossentropy, co sprawia, że jest odpowiedni dla problemów klasyfikacji wieloklasowej. Następnie model jest trenowany na danych treningowych przez 10 epok, a wyniki walidacji są monitorowane na końcu każdej epoki.

Po zakończeniu treningu model jest oceniany na zbiorze testowym, a dokładność testowa jest wypisywana. Wytrenowany model jest zapisywany w pliku letter_recognition_model.h5, co umożliwia jego późniejsze wykorzystanie bez konieczności ponownego treningu. Poniżej przedstawiono zmianę dokładności treningowej i walidacyjnej na przestrzeni epok.



Kod zawiera także funkcję predict_image, która pozwala na wykorzystanie zapisanego modelu do klasyfikacji nowych obrazów. Funkcja ta przetwarza obraz w sposób analogiczny do etapu wczytywania danych, a następnie wykonuje predykcję na podstawie wytrenowanego modelu. Wynik predykcji to klasa o najwyższym prawdopodobieństwie, która zostaje zwrócona użytkownikowi.

9. Generowanie zbioru danych służącego do uczenia modelu.

Do wygenerowania obrazów znaków wykorzystany został kod generujący obrazy liter alfabetu (A-Z) i cyfr (0-9). Kod tworzy główny katalog “letters”. Znajdować się w nim będą podfoldery dla każdego znaku (A, B, C itd.).

Rysowanie znaku odbywa się przy użyciu biblioteki Pillow. Czcionka jest wybierana losowo spośród dostępnych, a jej rozmiar losowany jest z przedziału od 20 do 80 pikseli. Pozycja tekstu na obrazie również jest losowa, ale tak obliczana, aby znak zmieścił się w granicach obrazu. Następnie tekst jest rysowany na obrazie, który jest reprezentowany jako macierz wartości odpowiadających jasności pikseli w skali szarości.

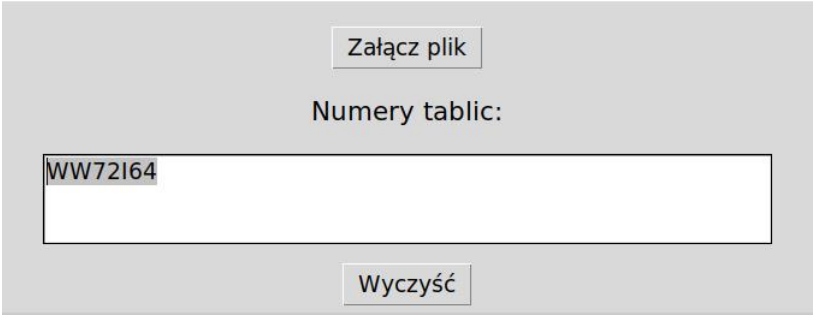
Po narysowaniu tekstu obraz jest obracany o losową warość z przedziału -30 do 30 stopni. Rotacja odbywa się wokół środka obrazu, a obszary poza obrazem wypełniane są białym kolorem. Następnie obraz zapisywany jest w odpowiednim katalogu przypisanym do konkretnej litery/cyfry.

Losowe rozmiary/jakość znaków, różne czcionki oraz rotacje, mają upodobnić wytworzone obrazy do tych rzeczywistych, z którymi do czynienia będzie miał model.

10. Ograniczenia aplikacji

Podczas testowania aplikacji napotkano kilka wyzwań związanych z dokładnością rozpoznawania tablic rejestracyjnych. Główne trudności obejmują:

1. **Błędne rozpoznawanie znaków na tablicach rejestracyjnych**
    a. W niektórych przypadkach aplikacja błędnie odczytuje cyfry lub litery z tablicy rejestracyjnej.



   b. Przykładem jest tablica widoczna na załączonym obrazie, gdzie rzeczywisty numer to **WW7286A**, natomiast aplikacja błędnie rozpoznała numer jako **WW72164**, co świadczy o nieprawidłowej klasyfikacji cyfry „8” jako „1” oraz litery „A” jako „4”.
   c. Możliwe przyczyny błędu:
      i. Nieodpowiednia jakość obrazu (szumy, niski kontrast).
      ii. Niedostateczna różnorodność danych treningowych, skutkująca trudnościami w klasyfikacji podobnych znaków.
      iii. Zbyt agresywne przetwarzanie obrazu (np. binaryzacja lub segmentacja).

2. **Problemy z detekcją tablic na niektórych zdjęciach**
   a. W niektórych przypadkach aplikacja nie wykrywa tablicy rejestracyjnej, szczególnie gdy znajduje się ona pod dużym kątem lub jest częściowo zasłonięta.
   b. Detekcja tablic może zawodzić w warunkach słabego oświetlenia lub gdy obraz ma niską rozdzielczość.
3. **Długi czas przetwarzania dużych obrazów**
   a. Dla zdjęć o wysokiej rozdzielczości czas analizy jest zauważalnie dłuższy, co może wpływać na wygodę użytkowania.

11. Potencjalne ulepszenia

Aby poprawić skuteczność i wydajność systemu, zaproponowano następujące ulepszenia:

1. **Rozbudowa zbioru danych treningowych**

   a. Użycie większej liczby obrazów tablic rejestracyjnych o różnym oświetleniu, kątach i jakości, co poprawi odporność modelu na rzeczywiste warunki.
   b. Dodanie większej liczby przykładów dla podobnie wyglądających znaków, takich jak „8” i „B”, „1” i „I”.
2. **Optymalizacja preprocessing'u obrazu**
   a. Ulepszenie algorytmów przetwarzania wstępnego obrazu, np. dynamiczne dostosowywanie progowania w celu lepszego odseparowania znaków od tła.
   b. Zastosowanie bardziej zaawansowanych technik filtracji szumów.
3. **Wykorzystanie bardziej zaawansowanego modelu rozpoznawania znaków**
   a. Przetestowanie nowszych architektur sieci neuronowych, takich jak CNN z transfer learning z większych modeli rozpoznawania znaków.
   b. Rozważenie użycia technologii OCR (np. Tesseract OCR) jako wsparcia dla istniejącego rozwiązania.
4. **Dodanie możliwości poprawy wyników przez użytkownika**
   a. Wprowadzenie funkcji edycji odczytanego numeru rejestracyjnego przez użytkownika w interfejsie aplikacji.
   b. Zapisywanie poprawionych wyników do przyszłego treningu modelu.
5. **Poprawa szybkości działania aplikacji**
   a. Optymalizacja kodu przetwarzania obrazu w celu zmniejszenia czasu przetwarzania dla dużych obrazów.
   b. Wykorzystanie przetwarzania równoległego do szybszego wykrywania tablic i klasyfikacji znaków.

12. Role w projekcie:

Krzysztof Jabłoński: generowanie danych treningowych, trenowanie modeli klasyfikacji znaków, przetwarzanie obrazu, segmentacja i ekstrakcja znaków, integracja procesów z aplikacją

Szymon Łabędziewski: automatyzacja instalacji aplikacji; dokumentacja; sprawozdanie

Mateusz Wawrzyczek: sprawozdanie, dokumentacja

Artur Saganowski: graficzny interfejs użytkownika