

Dokumentacja projektu - "Detekcja Samochodów i Rozpoznawanie Tablic Rejestracyjnych"

Bartosz Biesaga,
Paweł Dyjak,
Yuliya Zviarko

Wydział Fizyki i Informatyki Stosowanej
Akademia Górnictwo-Hutnicza im. Stanisława Staszica w Krakowie

Styczeń 2025

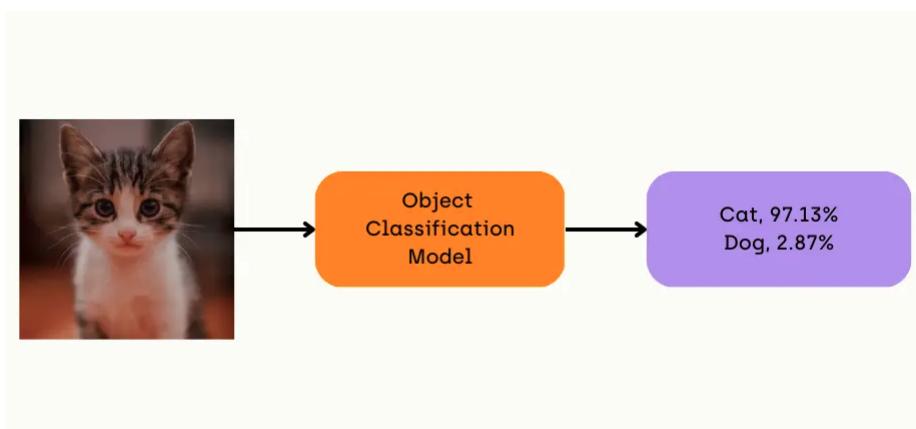
Spis treści

1 Wstęp teoretyczny	3
1.1 Czym jest klasyfikacja obiektów?	3
1.2 Czym jest detekcja obiektów?	3
1.3 Metryki oceny detekcji obiektów	4
1.4 Czym jest YOLO?	5
1.5 Algorytm YOLO: Jak to działa?	6
2 Opis projektu	7
2.1 Kluczowe elementy projektu	7
2.2 Założenia projektu	7
3 Implementacja	8
3.1 YOLO - detekcja aut i rejestracji	8
3.1.1 Filtracja i konwersja danych do formatu YOLO	9
3.1.2 Trening modelu YOLO	9
3.1.3 Postprocessing detekcji obiektów	9
3.1.4 Grupowanie wyników detekcji na klasy	9
3.1.5 Usuwanie duplikatów tablic rejestracyjnych	10
3.1.6 Przypisanie tablic rejestracyjnych do pojazdów	10
3.2 Przetwarzanie tablicy rejestracyjnej i segmentacja pojedynczych znaków	11
3.3 Model użyty do rozpoznawania wydzielonych znaków	13
3.3.1 Trening i validacja	13
3.4 Odczyt rejestracji samochodu	13
4 Interfejs użytkownika	15
4.1 Główne funkcjonalności	15
4.2 Przewodnik po użyciu aplikacji	16
5 Testowanie aplikacji na różnych zestawach zdjęć	18
5.1 Test 1: Zdjęcia dobrej jakości	18
5.2 Test 2: Brudne lub oklejone taśmą auta	19
5.3 Test 3: Różne warunki pogodowe	22
5.4 Test 4: Nieodpowiednie kolory tablic	24
6 Podział obowiązków w projekcie	26
7 Wnioski	28
8 Odnośniki	28

1 Wstęp teoretyczny

1.1 Czym jest klasyfikacja obiektów?

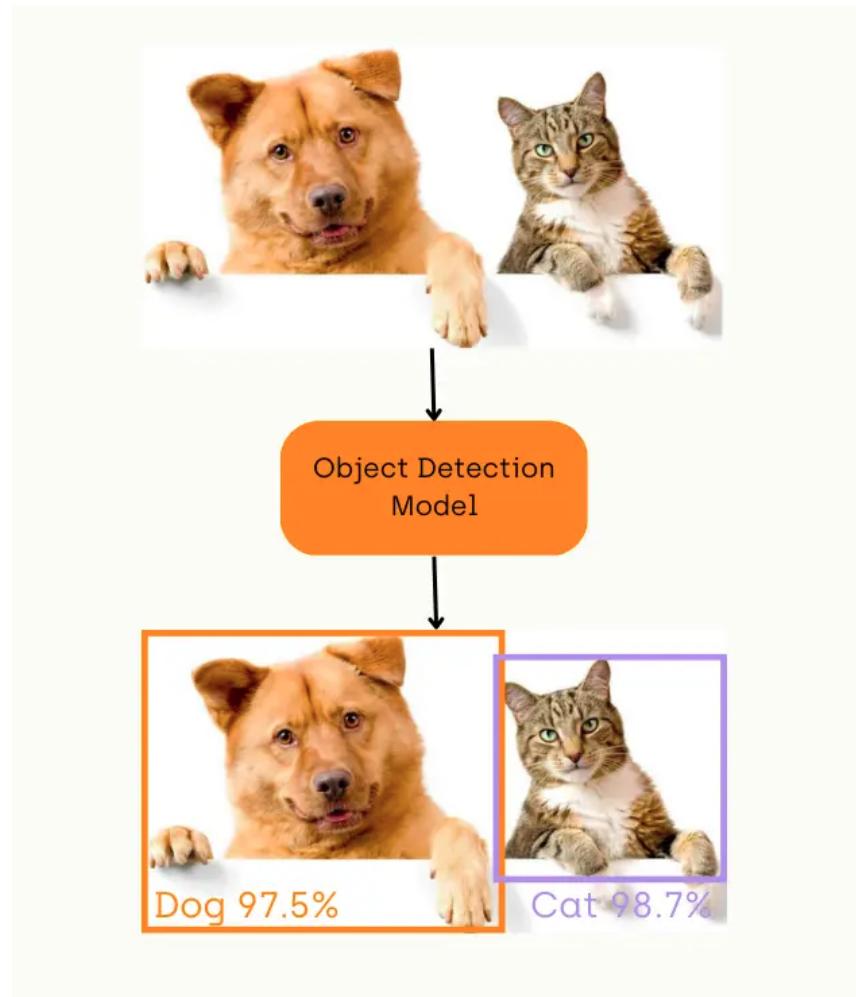
Klasyfikacja obiektów to zadanie polegające na rozpoznawaniu i przypisywaniu obiektem w obrazie odpowiednich kategorii z wcześniej zdefiniowanego zestawu klas. Proces ten wykorzystuje techniki uczenia maszynowego, w których model jest trenowany na oznakowanych zbiorach danych zawierających obrazy oraz ich etykiety. Po zakończeniu treningu model potrafi analizować nowe obrazy, przypisując im właściwą kategorię na podstawie wyuczonych cech. Przykładami zastosowania są rozpoznawanie znaków drogowych czy określanie gatunku rośliny na zdjęciu.



Rysunek 1: Przykład blokowy - Klasyfikacja Obiektów.

1.2 Czym jest detekcja obiektów?

Detekcja obiektów to zadanie z zakresu wizji komputerowej, które polega na identyfikowaniu i lokalizowaniu obiektów określonych klas w obrazach wejściowych. Z rozwojem głębokiego uczenia, detekcja obiektów osiągnęła bardzo obiecujące wyniki. Istnieje wiele algorytmów i modeli, takich jak R-CNN, Faster R-CNN, YOLO czy SSD, które zostały opracowane w celu detekcji obiektów. Modele te znajdują zastosowanie w takich dziedzinach jak pojazdy autonomiczne, systemy nadzoru czy śledzenie obiektów.



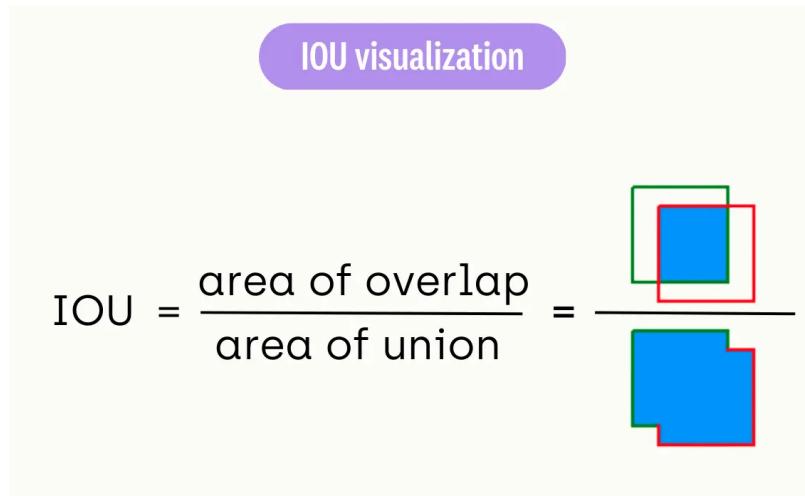
Rysunek 2: Przykład blokowy - Detekcja Obiektów.

1.3 Metryki oceny detekcji obiektów

Do oceny wydajności modeli detekcji obiektów stosuje się kilka powszechnie używanych metryk:

- **Średnia precyzja (mAP):** Metryka uwzględniająca zarówno dokładność, jak i czułość modelu. Wyższa wartość mAP świadczy o lepszej skuteczności.
- **Precyzja średnia (AP):** Miara precyzji modelu na różnych poziomach czułości. Wyższe AP oznacza bardziej niezawodne przewidywania.
- **Przecięcie nad unią (IoU):** Wskaźnik określający stopień zgodności pomiędzy

przewidywaną a rzeczywistą ramką obiektu. Wyższe IoU wskazuje na dokładniejsze dopasowanie.



Rysunek 3: wyznaczenie "IoU".

Dodatkowo, modele detekcji obiektów mogą być oceniane na podstawie efektywności obliczeniowej, takich jak *True Positive Rate (TPR)*, *False Positive Rate (FPR)*, *F1-score* czy *Log Average Miss Rate (MR)*.

1.4 Czym jest YOLO?

YOLO (You Only Look Once) to algorytm detekcji obiektów w czasie rzeczywistym opracowany przez Josepha Redmona i Aliego Farhadiego w 2015 roku. Jest to jednofazowy detektor, który wykorzystuje **konwolucyjną sieć neuronową** (CNN) do przewidywania ramki ograniczającej i prawdopodobieństwa klasy obiektów w obrazach wejściowych. YOLO jest szybszy i bardziej efektywny niż detektory dwufazowe, takie jak R-CNN, ponieważ przetwarza cały obraz w jednej iteracji.

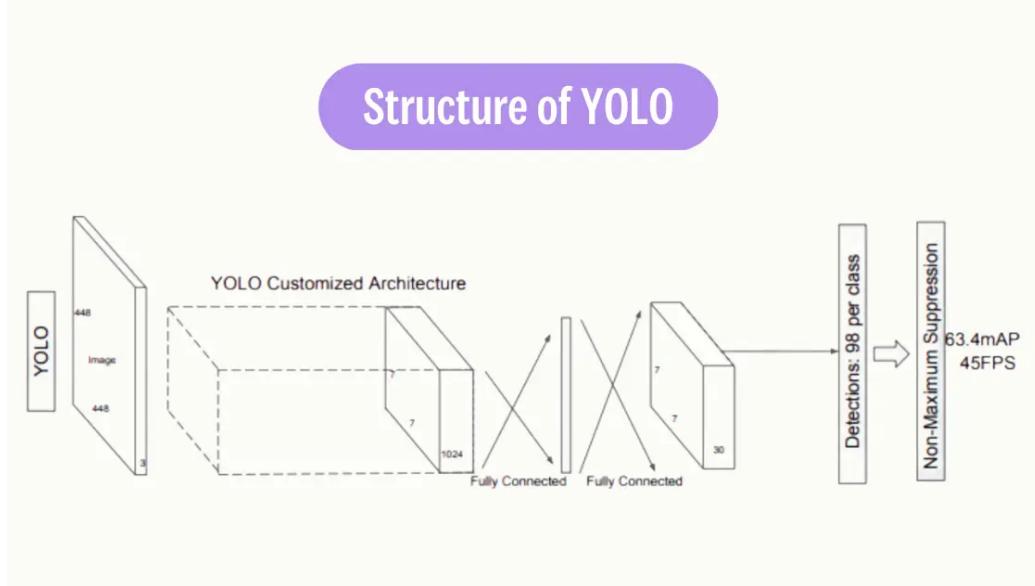
YOLO dzieli obrazy na siatkę komórek, w której każda komórka odpowiada za przewidywanie prawdopodobieństwa obecności obiektu oraz współrzędnych ramki ograniczającej. YOLO opracowano w wielu wersjach, każda kolejna wersja zawiera ulepszenia w zakresie dokładności, szybkości przetwarzania oraz obsługi małych obiektów.

1.5 Algorytm YOLO: Jak to działa?

Podstawowa idea algorytmu YOLO polega na podziale obrazu wejściowego na siatkę komórek, z których każda przewiduje prawdopodobieństwo obecności obiektu i współrzędne jego ramki. Proces ten obejmuje następujące kroki:

- Obraz wejściowy jest przepuszczany przez CNN w celu wydobycia cech obrazu.
- Cechy te są przekazywane przez w pełni połączone warstwy, które przewidują prawdopodobieństwa klas i współrzędne ramek.
- Obraz dzielony jest na siatkę komórek, z których każda odpowiada za przewidywanie zestawu ramek i prawdopodobieństw klas.
- Wynikiem jest zbiór ramek ograniczających oraz etykiet klasowych dla każdego obiektu.
- Ramki są następnie filtrowane za pomocą algorytmu postprocessingu, takiego jak *non-max suppression*, który usuwa nakładające się ramki i wybiera tą o najwyższym prawdopodobieństwie.

Jedną z kluczowych zalet YOLO jest to, że przetwarza cały obraz w jednym przebiegu, co czyni go wyjątkowo szybkim i efektywnym, jak wcześniej podkreślono.



Rysunek 4: Struktura YOLO.

2 Opis projektu

2.1 Kluczowe elementy projektu

W naszym projekcie postanowiliśmy zaimplementować aplikację umożliwiającą automatyczne przetwarzanie obrazów w celu wykrywania samochodów oraz odczytywania polskich tablic rejestracyjnych.

Proces detekcji aut oparty jest na algorytmie **YOLO**, który wykrywa samochody w danym obrazie. Następnie, obraz jest poddawany dalszej obróbce, w której wyodrębniane są interesujące nas fragmenty, a także stosowane algorytmy segmentacji w celu poprawy jakości analizy. Kolejnym krokiem jest wykorzystanie modelu rozpoznawania znaków, który umożliwia odczyt tablic rejestracyjnych. W efekcie aplikacja jest w stanie automatycznie wykrywać pojazdy oraz odczytywać ich numery rejestracyjne.

Projekt został zaprojektowany z myślą o zastosowaniu w systemach monitorowania i kontroli, które stanowią integralną część współczesnej infrastruktury miejskiej oraz systemów zarządzania ruchem. Przewidywane obszary zastosowania obejmują, ale nie ograniczają się do:

- kontrolę dostępu do stref o ograniczonym wjeździe, w tym wjazdy do obszarów zamkniętych, parkingów czy terenów przemysłowych,
- analiza zdjęć wykonanych przez fotoradar w celu automatycznego odczytania tablicy rejestracyjnej pojazdu.

2.2 Założenia projektu

Założenia projektu obejmują następujące cele i wymagania:

- Aplikacja musi umożliwiać odczytywanie zdjęć, ich przetwarzanie oraz zapis wyników.
- Przetwarzanie zdjęć obejmuje następujące etapy:
 - Algorytm wykorzystujący technologię YOLO ma za zadanie wykrywać pojazdy oraz ich tablice rejestracyjne na obrazie.
 - Skrypt opracowany przez nas ułatwi pracę z wykrytą tablicą rejestracyjną, umożliwiając dalszą obróbkę.
 - Algorytm segmentacji ma na celu wstępna obróbkę obrazu, co pozwala na ułatwienie analizy oraz podział tablicy rejestracyjnej na pojedyncze fragmenty.

- Pojedyncze fragmenty tablicy będą odczytane i przetwarzane przez model, który został nauczony rozpoznawania znaków.
- Projekt musi zwrócić wynik w postaci odczytanego numeru rejestracyjnego oraz zdjęcia pojazdu z zaznaczoną ramką wokół rozpoznanych elementów (pojazd i tablica rejestracyjna).

3 Implementacja

3.1 YOLO - detekcja aut i rejestracji

Początkowy etap projektu obejmował pozyskanie odpowiedniego zbioru danych oraz jego przetworzenie w celu trenowania modelu YOLO. Dane zostały pobrane z zestawu Open Images V7 i przefiltrowane, aby zawierały wyłącznie interesujące nas klasy: `Car` oraz `Vehicle registration plate`. Proces rozpoczął się od zimportowania odpowiednich bibliotek:

```

1 import fiftyone
2 import fiftyone.utils.openimages
3
4 # Pobranie danych treningowych
5 dataset_train = fiftyone.zoo.load_zoo_dataset(
6     "open-images-v7",
7     split="train",
8     label_types=["detections"],
9     classes=["Car", "Vehicle registration plate"],
10    max_samples=1500,
11    only_matching=True,
12)
13
14 # Pobranie danych walidacyjnych
15 dataset_val = fiftyone.zoo.load_zoo_dataset(
16     "open-images-v7",
17     split="validation",
18     label_types=["detections"],
19     classes=["Car", "Vehicle registration plate"],
20     max_samples=300,
21     only_matching=True,
22)

```

Listing 1: Importowanie bibliotek.

3.1.1 Filtracja i konwersja danych do formatu YOLO

Aby przygotować dane do trenowania modelu YOLO, konieczne było przetworzenie ich do odpowiedniego formatu. Proces ten obejmował filtrowanie i zapisanie obrazów oraz etykiet w strukturze katalogów zgodnej z wymaganiami YOLO. Następnie utworzono plik `data.yaml`, który zawiera ścieżki do zbiorów danych oraz definicję klas.

3.1.2 Trening modelu YOLO

Na koniec zainstalowano model YOLO oraz rozpoczęto trening:

```
1 !pip install ultralytics
2
3 from ultralytics import YOLO
4
5 model = YOLO("yolo11n.pt")
6 results = model.train(data="dataset/data.yaml", epochs=100,
    imgsz=640)
```

Listing 2: Trening modelu YOLO.

3.1.3 Postprocessing detekcji obiektów

Po zakończeniu trenowania modelu YOLO, wykonano postprocessing detekcji obiektów w celu przypisania tablic rejestracyjnych do odpowiednich pojazdów. W tym celu zaimplementowano funkcję `crop_boxes_from_image`, która przetwarza obraz wyjściowy modelu YOLO, grupując wykryte pojazdy i tablice rejestracyjne.

3.1.4 Grupowanie wyników detekcji na klasy

Wyniki przetwarzania obrazu przez model YOLO są grupowane na pojazdy i tablice rejestracyjne:

```
1 for cls, box in zip(map(int, map(torch.Tensor.tolist,
    result.boxes.cls)),
                      [list(map(int, box)) for box in
                       map(torch.Tensor.tolist,
                           result.boxes.xyxy)]):
2     if cls == 0:
3         cars.append([image[box[1]:box[3], box[0]:box[2]], box])
4     else:
5         license_plates.append([image[box[1]:box[3], box[0]:box[2]],
6                                box,
7                                (box[2] - box[0]) * (box[3] - box[1]), False])
```

Listing 3: Grupowanie wyników na klasy.

Wykryte obiekty są dzielone na klasy: `Car` (klasa 0) i `Vehicle registration plate` (klasa 1).

3.1.5 Usuwanie duplikatów tablic rejestracyjnych

Jeśli bounding boxy tablic rejestracyjnych nakładają się, wybierany jest większy obszar:

```
1 for i in range(len(license_plates)):
2     for j in range(i+1, len(license_plates)):
3         x = (min(license_plates[i][1][2], license_plates[j][1][2]) -
4               max(license_plates[i][1][0],
5                    license_plates[j][1][0]))
5         y = (min(license_plates[i][1][3], license_plates[j][1][3]) -
6               max(license_plates[i][1][1],
7                    license_plates[j][1][1]))
8         if x > 0 and y > 0:
9             if license_plates[i][2] > license_plates[j][2]:
10                 indices_to_remove.append(j)
11             else:
12                 indices_to_remove.append(i)
```

Listing 4: Usuwanie duplikatów tablic.

Porównywane są wymiary obszarów detekcji, co pozwala wyeliminować mniejsze i potencjalnie błędne wykrycia.

3.1.6 Przypisanie tablic rejestracyjnych do pojazdów

Tablice rejestracyjne są przypisywane do odpowiadających im pojazdów na podstawie wspólnego obszaru bounding boxów:

```
1 for i, car in enumerate(cars):
2     license_plate_list = []
3     for j, license_plate in enumerate(license_plates):
4         x = min(license_plate[1][2], car[1][2]) -
5             max(license_plate[1][0], car[1][0])
6         y = min(license_plate[1][3], car[1][3]) -
7             max(license_plate[1][1], car[1][1])
8         if x > 0 and y > 0 and ((x * y / license_plate[2]) >=
9             license_plate_car_ioa):
10             license_plate_list.append(license_plate[0])
11     pairs.append([car[0], license_plate_list])
```

Listing 5: Przypisywanie tablic do pojazdów.

Każda tablica rejestracyjna jest analizowana pod kątem stopnia nakładania się z wykrytym pojazdem, a wynik przypisania jest zapisywany w strukturze `pairs`.

Na koniec zwracana jest lista par: pojazd oraz lista przypisanych tablic rejestracyjnych.

3.2 Przetwarzanie tablicy rejestracyjnej i segmentacja pojedynczych znaków

Proces przetwarzania obrazu tablicy rejestracyjnej został zaimplementowany w funkcji `process_image`, która składa się z następujących kroków:

1. **Tworzenie folderu na podgląd etapów przetwarzania:** Jeśli użytkownik przekazał ścieżkę do folderu debugowania (`debug_folder`), zostaje on utworzony (jeśli jeszcze nie istnieje), aby zapisywać obrazy pośrednich etapów przetwarzania.
2. **Wczytywanie obrazu:** Obraz tablicy rejestracyjnej jest wczytywany z pliku (`image_path`) lub przekazywany jako argument (`img`). W przypadku braku obu parametrów funkcja zwraca błąd.
3. **Rozmycie obrazu:** Obraz zostaje rozmyty za pomocą filtra Gaussa (`cv2.GaussianBlur`) z jądrem o rozmiarze określonym przez `blur_ksize`. Etap ten poprawia jakość binaryzacji.
4. **Usunięcie eurobandu:** W celu eliminacji niebieskiego paska europejskiego (eurobandu):
 - Wyodrębniane są kanały niebieski (B) i czerwony (R).
 - Obliczana jest wartość bezwzględna różnicy między wartościami tych kanałów (`abs_diff`).
 - Na podstawie tej różnicy tworzona jest maska progowa (`diff_thresh`), która odwracana (`cv2.bitwise_not`) pozwala na usunięcie eurobandu.
5. **Oddzielenie znaków z tablicy:**
 - Obraz jest konwertowany do przestrzeni barw HSV, a następnie wybierany jest kanał jasności (`value`).
 - Na tym kanale wykonywana jest adaptacyjna binaryzacja (`threshold_local`), co pozwala na precyzyjne oddzielenie znaków od tła ze względu na to, że znaki na polskich rejestracjach są znacznie ciemniejsze od tła.

- Obraz zostaje negatywowany (`cv2.bitwise_not`), aby znaki były białe.
- Zastosowanie maski eurobandu:** Binaryzowany obraz zostaje połączony z maską usuwającą euroband (`cv2.bitwise_and`), co pozwala na wyodrębnienie znaków z tablicy bez zakłóceń.
 - Zapis obrazów pośrednich:** Jeśli użytkownik przekazał ścieżkę do folderu debugowania, na każdym etapie przetwarzania zapisywane są pośrednie wyniki w odpowiednich plikach graficznych.

Efektem końcowym funkcji jest przetworzony obraz tablicy rejestracyjnej, który będzie użyty w kolejnym etapie, czyli segmentacji znaków, zaimplementowanej w funkcji `get_characters_images`, obejmującej następujące kroki:

- Tworzenie folderu debugowania:** Jeśli użytkownik podał ścieżkę do folderu debugowania (`debug_folder`), zostaje on utworzony, a pośrednie wyniki przetwarzania zapisywane są jako pliki graficzne.
- Wykrywanie konturów:** Kontury obiektów na obrazie tablicy rejestracyjnej są wykrywane za pomocą funkcji `cv2.findContours`. Umożliwia to identyfikację potencjalnych znaków.
- Filtracja konturów:** Każdy wykryty kontur jest analizowany pod kątem spełniania następujących warunków:
 - Wysokość konturu mieści się w zakresie określonym przez parametry `min_char_height_factor` i `max_char_height_factor`.
 - Szerokość konturu spełnia kryteria określone przez `min_char_width_factor` i `max_char_width_factor`.
 - Stosunek wysokości do szerokości (`aspect_ratio`) mieści się w zadanym zakresie.
 - Kontur nie doryka granic obrazu (`touches_border` zwraca `False`).

Kontury spełniające te kryteria są rysowane na masce znaków (`chars_mask`).

- Wyodrębnienie znaków:** Obraz tablicy rejestracyjnej jest mnożony przez maskę znaków (`cv2.bitwise_and`), co pozwala na usunięcie zbędnych konturów i pozostawienie jedynie potencjalnych znaków.
- Wydzielenie pojedynczych znaków:** Kontury na przetworzonym obrazie są ponownie wykrywane (`cv2.findContours`), a następnie sortowane według współrzędnej `x`, aby uzyskać odpowiednią kolejność. Każdy znak jest wycinany z obrazu i zapisywany w tablicy `characters`.

6. **Znajdowanie największej przerwy między znakami:** Współrzędne lewego i prawego brzegu każdego znaku są zapisywane w tablicy `char_locations`. Na tej podstawie obliczana jest największa przerwa między znakami, co pozwala na określenie miejsca podziału pomiędzy wyróżnikiem miejsca a wyróżnikiem pojazdu.
7. **Zapis wyników:** Jeśli podano ścieżkę do folderu debugowania, maska znaków, przetworzony obraz oraz wycięte znaki są zapisywane jako pliki graficzne w odpowiednich etapach przetwarzania.

Funkcja zwraca listę wyciętych znaków (`characters`) oraz indeks największej przerwy między znakami (`max_gap_index`), co pozwala na dalsze przetwarzanie i klasyfikację znaków.

3.3 Model użyty do rozpoznawania wydzielonych znaków

W ramach projektu wykorzystano gotowy model, który został wczytany i dostosowany do zadania klasyfikacji pojedynczych znaków z tablic rejestracyjnych.

Model `efficientnet_b1` został wybrany ze względu na jego niewielki rozmiar, wysoką szybkość działania oraz bardzo dobre wyniki osiągnięte na zbiorze, na którym był wcześniej trenowany. Aby dostosować model do bieżącego zadania, ostatnia warstwa została podmieniona, umożliwiając klasyfikację na dwie klasy.

Wykorzystano framework PyTorch do implementacji i dostosowania modelu

3.3.1 Trening i walidacja

Do trenowania modelu wykorzystano dane zawierające wycięte znaki z obrazów tablic rejestracyjnych, które zostały odpowiednio przygotowane poprzez skalowanie do jednolitego rozmiaru. Trening modelu odbywał się z użyciem **algorytmu optymalizacji Adam** oraz funkcji **strat Cross-Entropy** typowej dla tego zadania. Model był walidowany na odrębnej próbce danych, co pozwoliło na monitorowanie jego wydajności oraz wybranie wag, dla których osiągał najlepsze rezultaty.

Po zakończeniu procesu trenowania, model może być używany do klasyfikacji nowych znaków z obrazów polskich tablic rejestracyjnych.

3.4 Odczyt rejestracji samochodu

W tym etapie wykorzystujemy opracowane przez nas funkcje do obróbki obrazu oraz jego segmentacji, a także wytrenowany model:

`process_license_plate()` zarządza całym procesem przetwarzania obrazu tablicy rejestracyjnej, wykonując:

- **Przetwarzanie obrazu wejściowego:** Obraz jest wczytywany z pliku lub przekazywany bezpośrednio jako argument.
- **Segmentacja znaków:** Obraz tablicy jest analizowany, a poszczególne znaki (litery i cyfry) są wyodrębniane.
- **Rozpoznawanie znaków:** Każdy wyodrębniony znak jest przekazywany do funkcji `recognize_characters()` w celu rozpoznania liter i cyfr.
- **Obsługa zakazanych liter:** W drugiej części tablicy, niektóre znaki są zabronione (**B**, **D**, **I**, **O**, **Z**). Funkcja ta identyfikuje takie znaki i proponuje alternatywne rozpoznania będące kolejnymi najbardziej prawdopodobnymi predykcjami modelu nie będącymi zakazanymi literami. Zakazane litery są powiązane z możliwymi pomyłkami w rozpoznaniu, ponieważ ich kształt jest podobny do cyfr:
 - **B** może zostać pomylona z **8**,
 - **D** może zostać pomylona z **0**,
 - **I** może zostać pomylona z **1**,
 - **O** może zostać pomylona z **0**,
 - **Z** może zostać pomylona z **2**.

Te zakazy mają na celu minimalizowanie ryzyka błędnego rozpoznania tablicy rejestracyjnej.

Na końcu funkcja zwraca rozpoznany tekst, który stanowi wynik procesu.

```
1 def process_license_plate(image_path=None, image=None, model=None,
2                           preprocess=None):
3     try:
4         if image is None:
5             if image_path is None:
6                 raise ValueError("Either 'image_path' or
7                                   'processed_image' must be provided.")
8             else:
9                 processed_image =
10                    process_image(image_path=image_path)
11         else:
12             processed_image = process_image(img=image)
```

```

11     char_images, space_index =
12         get_characters_images(processed_image)
13     recognized_text, outputs = recognize_characters(model,
14         preprocess, char_images, space_index)
15
16     forbidden_chars = {'B', 'D', 'I', 'O', 'Z'}
17     parts = recognized_text.split(' ')
18     if len(parts) > 1:
19         first_part, second_part = parts[0], parts[1]
20         corrected_second_part = ''
21         for idx, char in enumerate(second_part):
22             if char in forbidden_chars:
23                 alternative_predictions = []
24                 output = outputs[space_index + 1 + idx]
25                 for char_idx, score in enumerate(output[0]):
26                     predicted_char =
27                         list(CHARS.keys())[char_idx]
28                     if predicted_char not in forbidden_chars:
29                         alternative_predictions.append(
30                             (predicted_char, score.item()))
31                 alternative_predictions.sort(key=lambda x:
32                     x[1], reverse=True)
33                 if alternative_predictions:
34                     corrected_second_part +=
35                         alternative_predictions[0][0]
36                 else:
37                     corrected_second_part += char
38             recognized_text = f"{first_part}"
39             {corrected_second_part}"
40
41     return recognized_text
42 except Exception as e:
43     print(f"Error processing license plate: {e}")
44     return "[PROCESSING ERROR]"

```

Listing 6: Odczyt rejestracji samochodu.

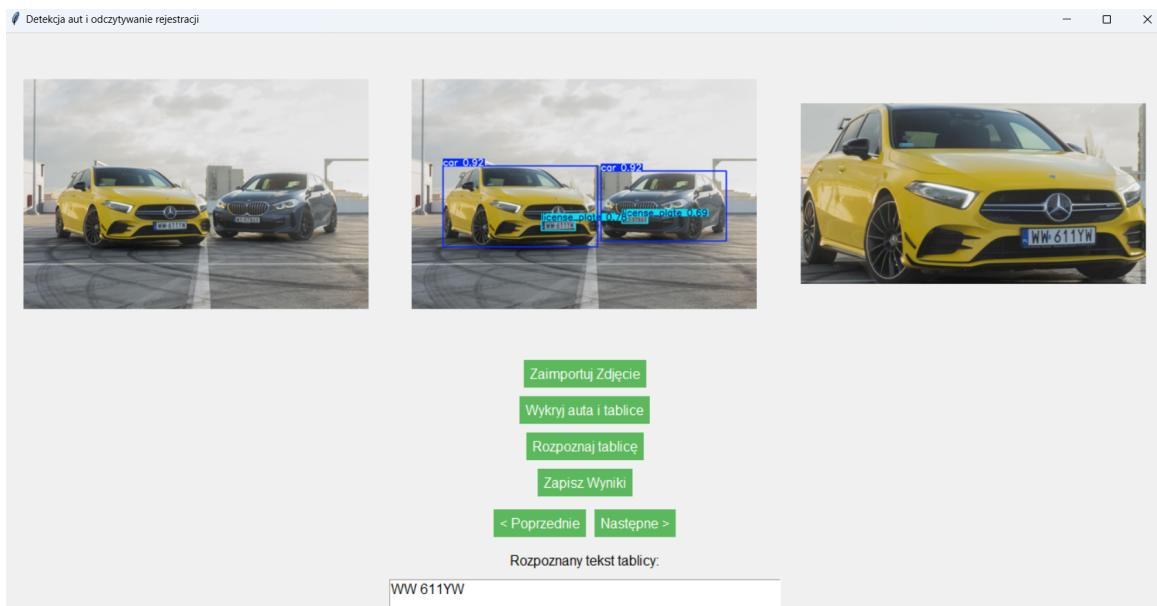
4 Interfejs użytkownika

4.1 Główne funkcjonalności

- Importowanie obrazów w wielu formatach, między innymi .jpg, .png, .jpeg.
- Wykrywanie pojazdów oraz ich tablic rejestracyjnych za pomocą modelu YOLO.
- Wyświetlanie wykrytych obiektów na obrazie.

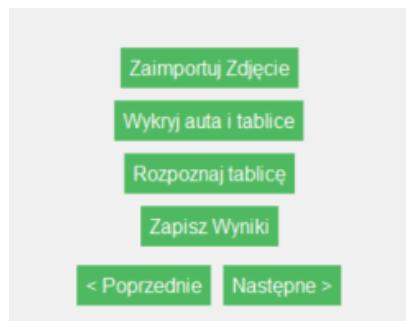
- Rozpoznawanie tekstu tablic rejestracyjnych z użyciem modelu klasyfikującego.
- Eksport wyników w formie tekstowej i graficznej.

4.2 Przewodnik po użyciu aplikacji



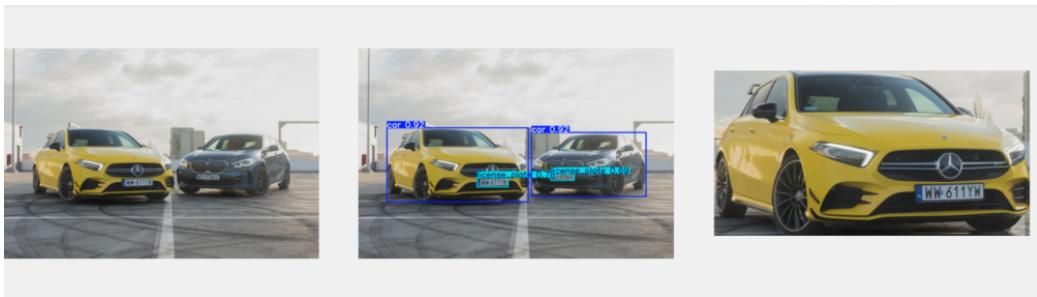
Rysunek 5: Interfejs użytkownika.

Aplikacja oferuje prosty i intuicyjny interfejs, umożliwiający użytkownikowi łatwe korzystanie z jej funkcjonalności. Za pomocą przycisku **Zainportuj zdjęcie** można wczytać obraz, który następnie będzie poddawany analizie.

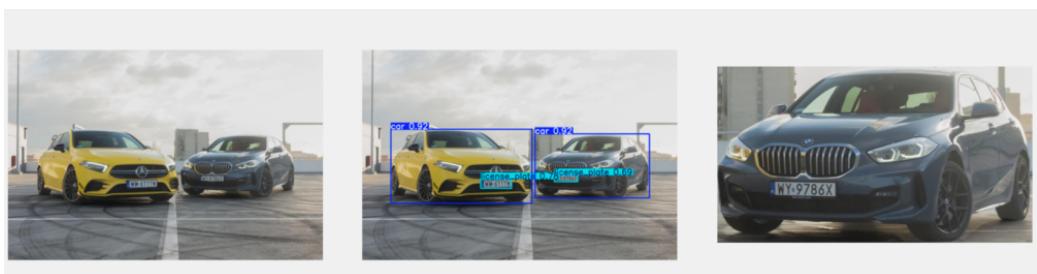


Rysunek 6: Przyciski funkcjonalne aplikacji.

Po zimportowaniu obrazu użytkownik ma możliwość uruchomienia algorytmu YOLO za pomocą przycisku **Wykryj auto i tablice**. Algorytm analizuje obraz, wykrywając na nim samochody oraz ich tablice rejestracyjne. Aplikacja została zaprojektowana do obsługi wielu obiektów na jednym obrazie, co ilustruje przykład z dwoma pojazdami. Dodatkowo, przyciski **Poprzednie** i **Następne** umożliwiają przeglądanie szczegółowych wyników dla poszczególnych wykrytych obiektów.

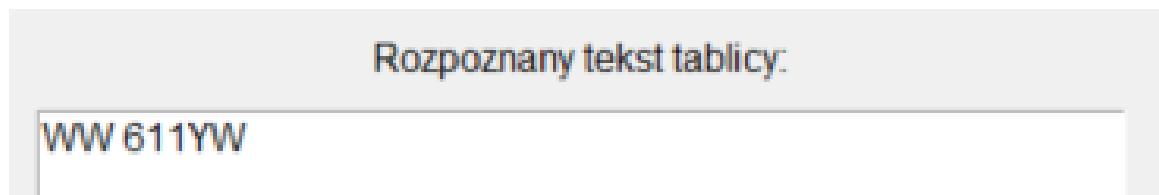


Rysunek 7: Wykryty samochód widoczny po lewej stronie obrazu.



Rysunek 8: Wykryty samochód widoczny po prawej stronie obrazu.

Kolejnym krokiem jest rozpoznawanie tablic rejestracyjnych, co można uruchomić za pomocą przycisku **Rozpoznaj tablice**. Algorytm segmentuje tablicę na pojedyncze znaki, a następnie rozpoznaje je, generując tekstowy wynik.



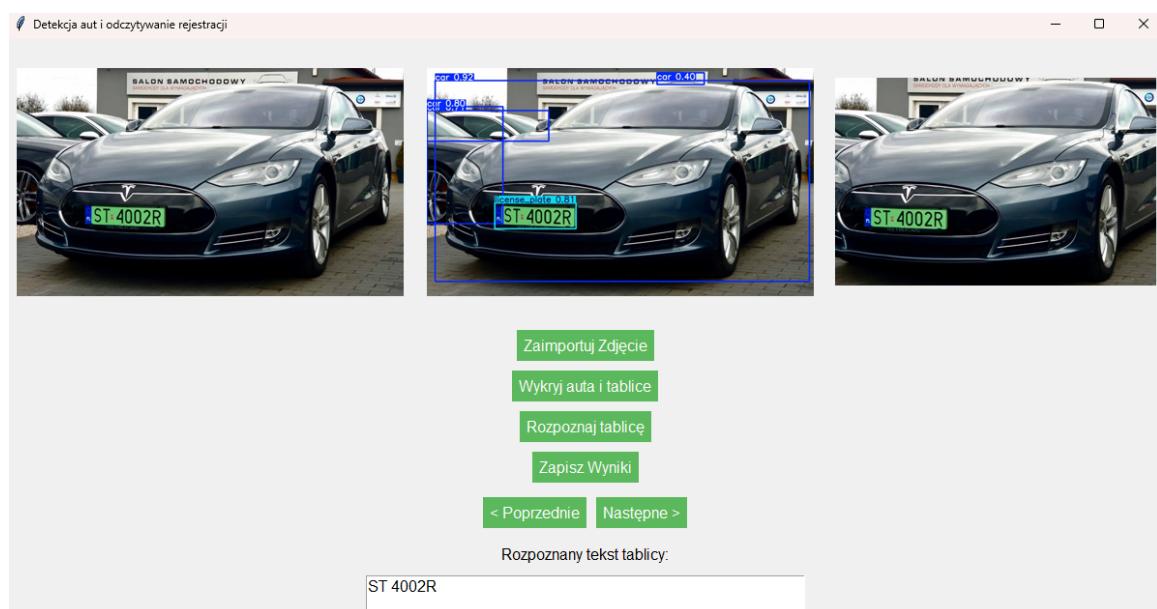
Rysunek 9: Tekstowy wynik odczytu tablicy rejestracyjnej.

5 Testowanie aplikacji na różnych zestawach zdjęć

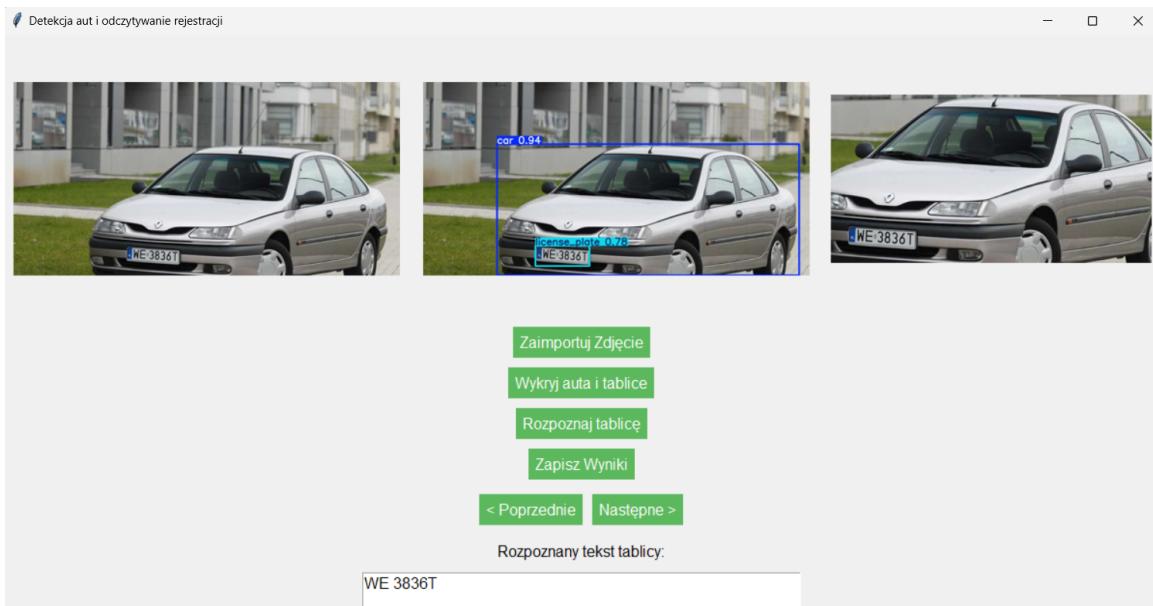
Aplikacja w wielu przypadkach pokazała dobre lub zbliżone wyniki, szczególnie gdy testowane zdjęcia były wysokiej jakości. W takich przypadkach udawało się uzyskać prawidłowe rezultaty. Co więcej, w niektórych bardziej wymagających przykładach aplikacja również potrafiła zidentyfikować rejestrację poprawnie.

5.1 Test 1: Zdjęcia dobrej jakości

W przypadku zdjęć dobrej jakości model osiągał bardzo wysoką skuteczność. Nie jest to zaskakujące, ponieważ model był trenowany na podobnych danych, co zapewniło wysoką precyzję w odczycie tablic rejestracyjnych.



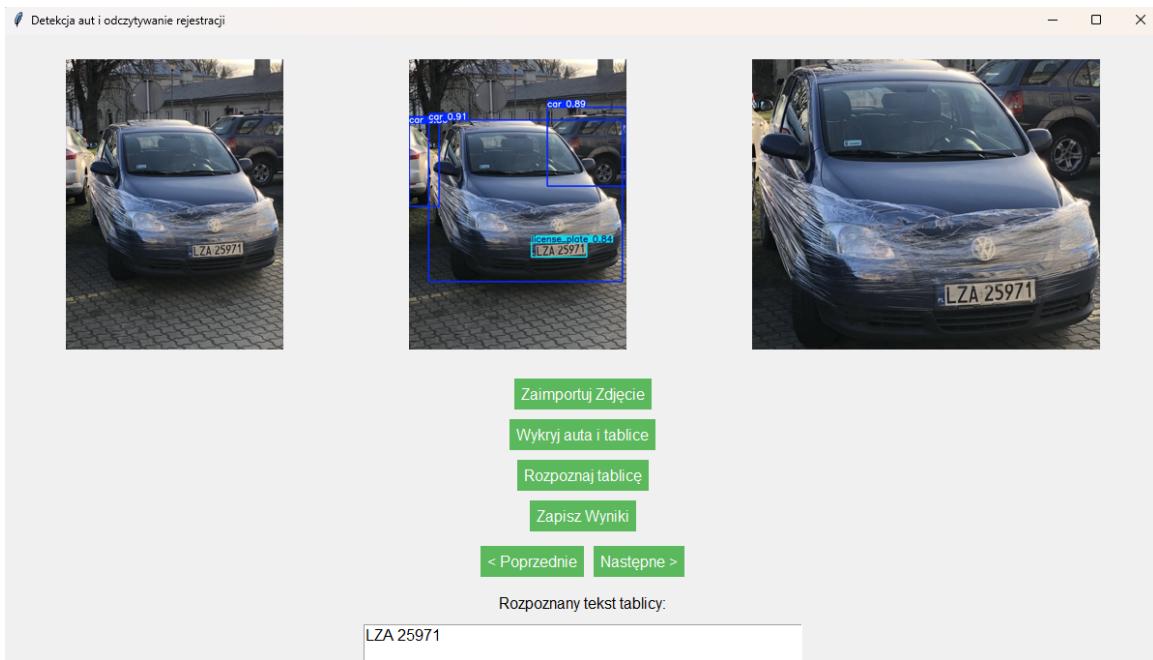
Rysunek 10: Zdjęcie wysokiej jakości - poprawny odczyt tablicy rejestracyjnej.



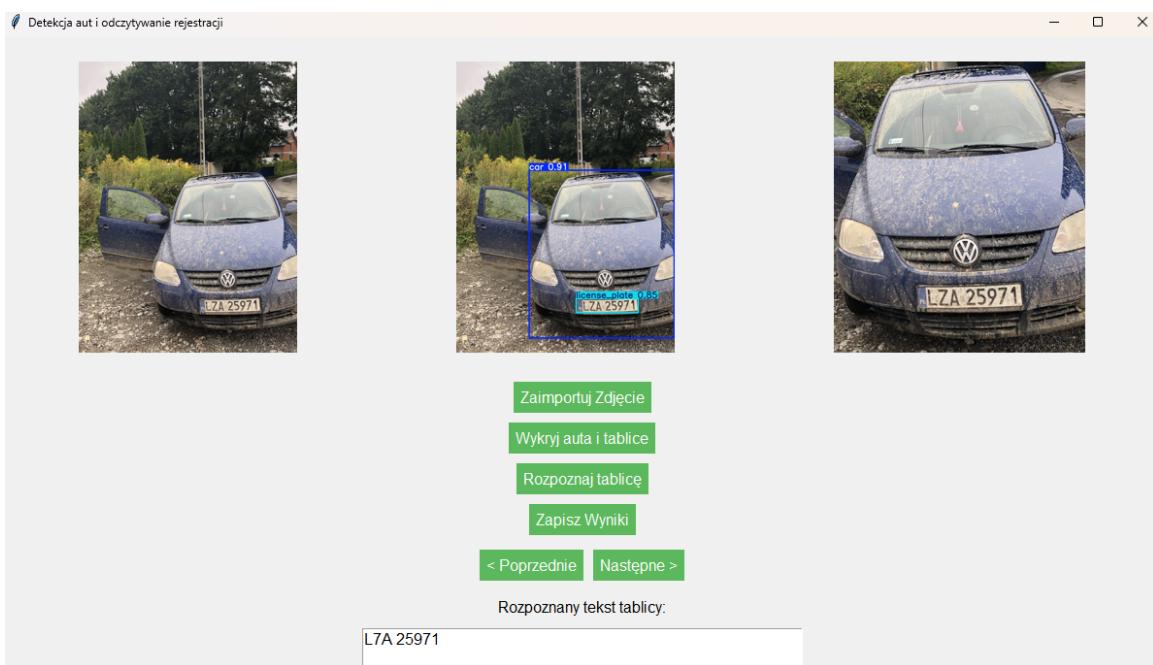
Rysunek 11: Kolejny przykład zdjęcia wysokiej jakości z poprawnym wynikiem.

5.2 Test 2: Brudne lub oklejone taśmą auto

W niektórych przypadkach aplikacja była w stanie poprawnie odczytać tablice rejestracyjne, nawet gdy samochód był brudny lub częściowo zaklejony taśmą.

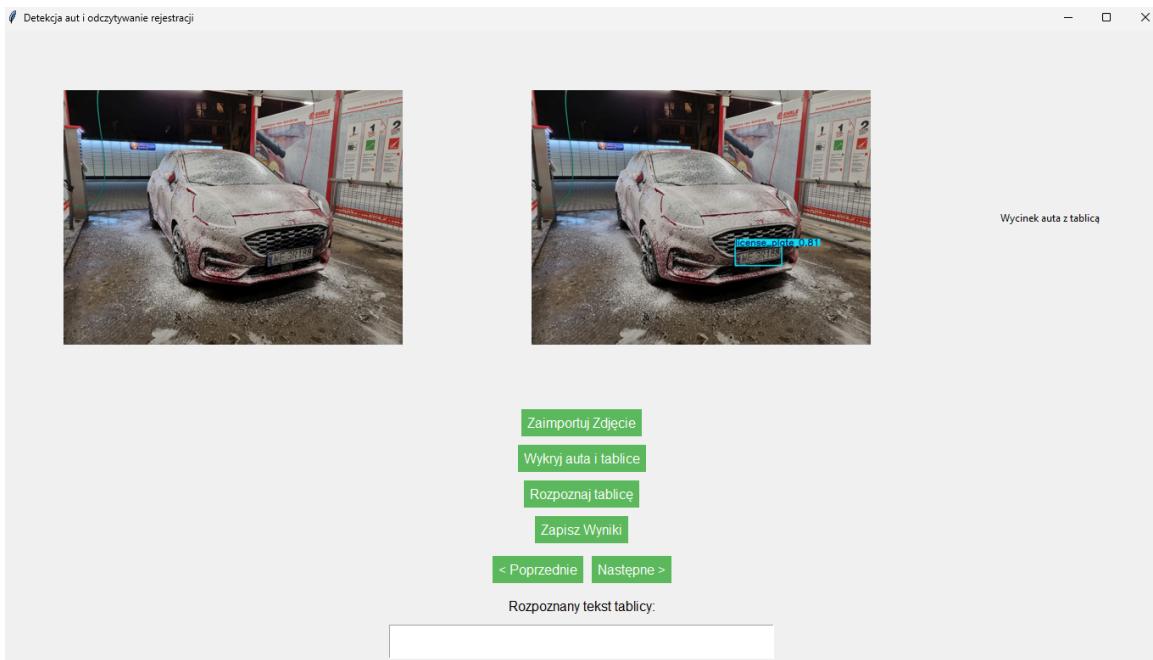


Rysunek 12: Samochód oklejony taśmą - poprawny odczyt rejestracji.



Rysunek 13: Brudny samochód - poprawny odczyt rejestracji.

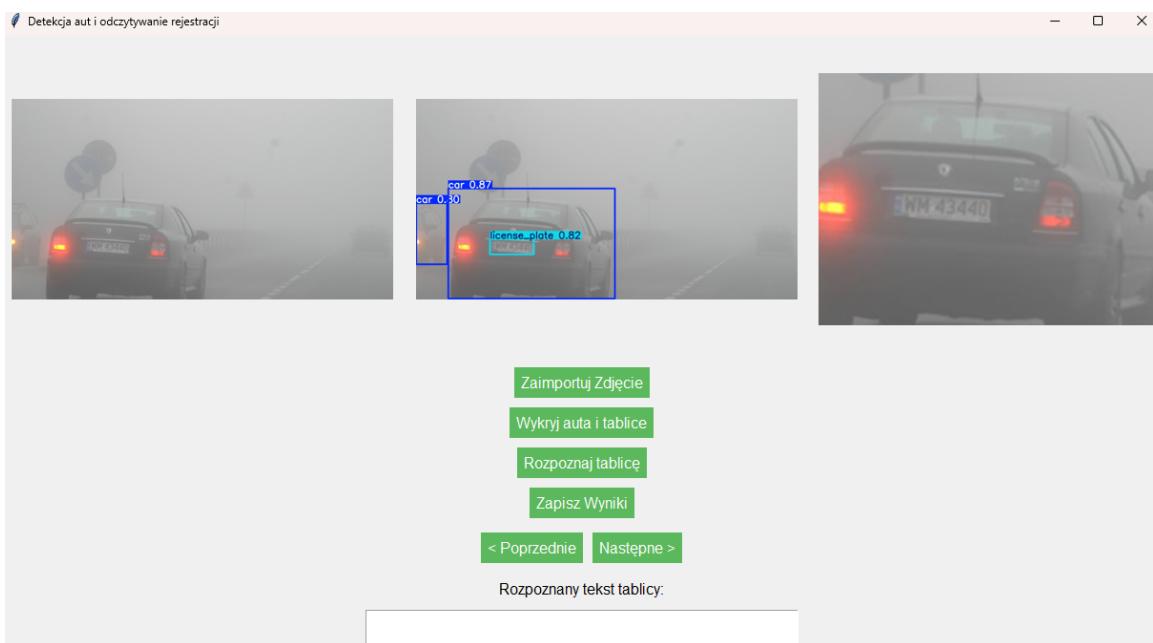
Jednakże w niektórych przypadkach, gdy samochód był bardzo brudny lub wyglądał nienaturalnie, model nie potrafił zidentyfikować tablic rejestracyjnych ani nawet wykryć samochodu.



Rysunek 14: Samochód pokryty pianą - brak poprawnego wyniku.

5.3 Test 3: Różne warunki pogodowe

W trudnych warunkach pogodowych, takich jak mgła, aplikacja miała problemy z odczytem tablic rejestracyjnych ze względu na niską jakość obrazu.



Rysunek 15: Zdjęcie w warunkach mgły - brak poprawnego wyniku.

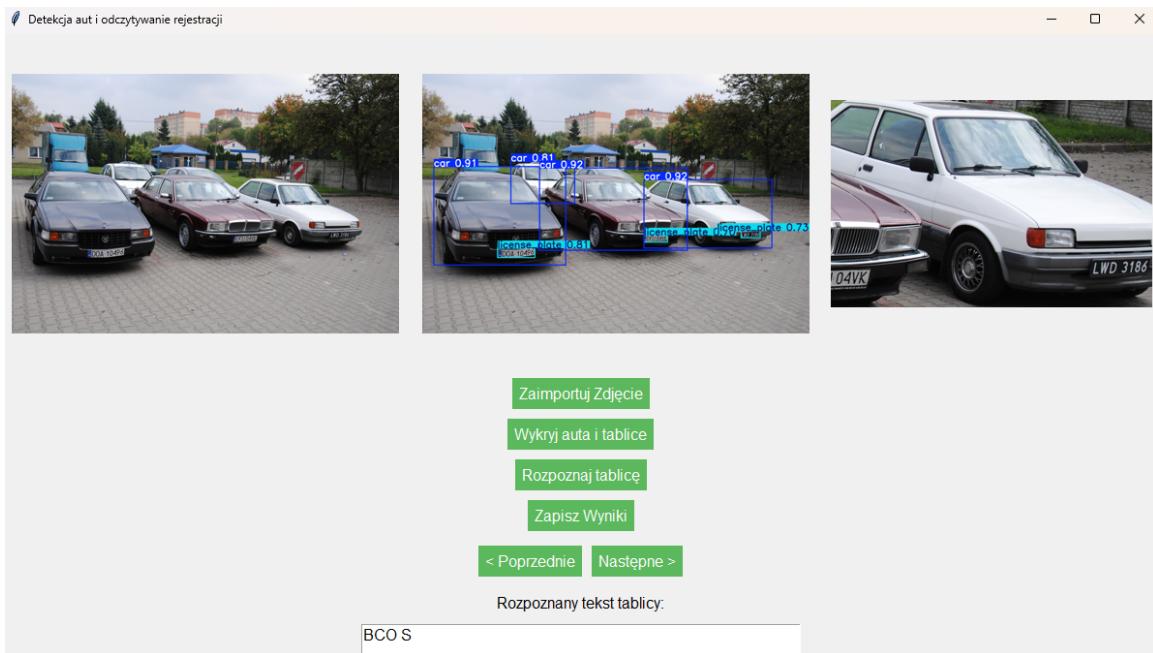
W przypadku zdjęć z kroplami wody aplikacja radziła sobie nieco lepiej, ale zdawały się drobne błędy w odczycie. Przykładowo, model błędnie rozpoznał literę "VV" jako "WV".



Rysunek 16: Krople wody na obrazie - błąd w odczycie "VV" jako "WV".

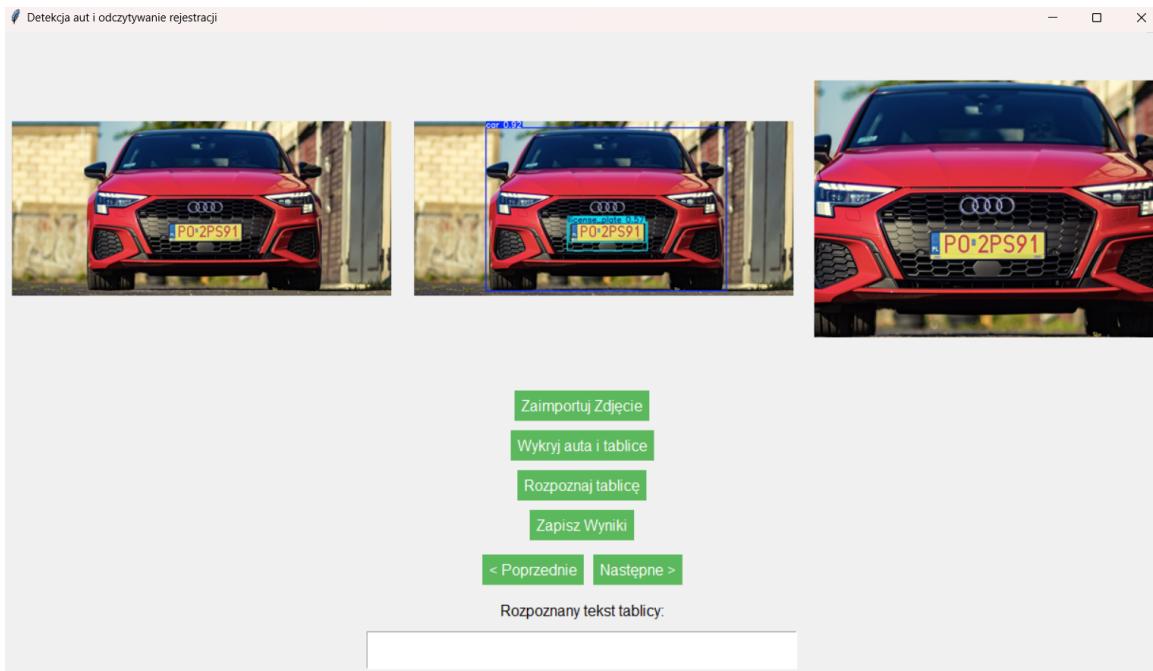
5.4 Test 4: Nieodpowiednie kolory tablic

Model miał trudności z odczytem tablic rejestracyjnych w specyficznych kolorach. Przykładowo, czarne litery na czarnym tle były niemal niemożliwe do odczytania.



Rysunek 17: Czarne litery na ciemnym tle - brak poprawnego wyniku.

Podobne problemy pojawiały się, gdy litery na tablicach były czerwone. Model w wielu przypadkach nie był w stanie ich prawidłowo rozpoznać.



Rysunek 18: Czerwone litery na tablicy - trudności z odczytem.

6 Podział obowiązków w projekcie

W trakcie realizacji projektu wykorzystaliśmy platformę GitHub, pracując na osobnych branchach, co umożliwiło sprawną organizację pracy i efektywną współpracę zespołu.

The screenshot shows the GitHub 'Branches' page with a dark theme. At the top right is a green 'New branch' button. Below it is a navigation bar with tabs: Overview (selected), Yours, Active, Stale, and All. A search bar follows, with placeholder text 'Search branches...'. The main area is divided into three sections: 'Default', 'Your branches', and 'Active branches', each containing a table of branches.

Branch	Updated	Check status	Behind	Ahead	Pull request
<code>main</code>	last month	(Default)	0	...	

Branch	Updated	Check status	Behind	Ahead	Pull request
<code>plate-recognition</code>	last week	0 / 45	3	...	

Branch	Updated	Check status	Behind	Ahead	Pull request
<code>gui_linux</code>	12 hours ago	0 / 60	...		
<code>extract_plates</code>	yesterday	0 / 11	...		
<code>dev</code>	last week	0 / 46	...		
<code>plate-recognition</code>	last week	0 / 45	3	...	
<code>YOLO</code>	3 weeks ago	0 / 4	1	...	

Rysunek 19: Struktura branczy projektu na GitHub.

Podział obowiązków w zespole wyglądał następująco:

- Bartosz - był odpowiedzialny za wstępna obróbkę YOLO – trenowanie modelu oraz napisanie odpowiednich funkcji; opracowanie etapu logicznego, obejmującego segmentację znaków i odczyt tablic rejestracyjnych w całości. Przygotował również model do odczytu oraz przeprowadził jego trening.
- Paweł - Był odpowiedzialny za przygotowanie interfejsu graficznego (GUI) oraz testowanie różnych przypadków dla zdjęć, które zostały opisane w dokumentacji, a także pracował przy trenowaniu YOLO i pomógł w tworzeniu zbioru treningowego dla modelu rozpoznającego oraz do testowania segmentacji.
- Julia - zajmowała się, wspólnie z Bartoszem, opracowaniem etapu logicznego, obejmującego segmentację znaków i odczyt tablic rejestracyjnych w całości, a także przygotowaniem dokumentacji projektu, a także pracowała przy trenowaniu YOLO.

W przypadku trudności czy problemów pracowaliśmy wspólnie nad ich rozwiązaniem oraz podejmowaniem decyzji, co umożliwiło sprawną realizację projektu.

7 Wnioski

Projekt zakończył się sukcesem. Osiągnięte wyniki potwierdzają, że udało się zrealizować wszystkie założenia, które postawiliśmy przed sobą. Dzięki zastosowaniu algorytmu YOLO oraz przeprowadzeniu segmentacji znaków, system skutecznie odczytywał tablice rejestracyjne na obrazach w dobrym jakościowo formacie.

Jednak zauważaliśmy pewne ograniczenia, które warto uwzględnić. Okazuje się, że jakość zdjęcia ma duży wpływ na skuteczność detekcji oraz odczytywania znaków – na przykład, zdjęcia o niższej jakości, takie jak zrzuty ekranu, mogą uniemożliwić prawidłowe rozpoznanie tablic rejestracyjnych. Prawdopodobnie wynika to z faktu, że nasz model rozpoznający był trenowany na stosunkowo wąskim oraz niezrównoważonym zbiorze danych – np. litera „D” pojawiła się w zbiorze treningowym jedynie raz, a wszystkie dane, na których przeprowadziliśmy trening, były w wysokiej jakości. Jeśli chodzi o segmentację, to odbywa się ona najlepiej dla jednorodnie oświetlonych tablic rejestracyjnych z białym bądź zielonym tłem oraz gdy tablica rejestracyjna jest pod niewielkim kątem względem ziemi.

W przyszłości warto rozważyć rozszerzenie zbioru danych o różne jakościowo zdjęcia oraz o szerszym zakresie tablic rejestracyjnych, ponieważ aktualnie zdecydowana większość pochodziła z województwa śląskiego, co powinno poprawić generalizację modelu i jego zdolność do odczytywania tablic rejestracyjnych w mniejszych idealnych warunkach. Jeśli chodzi o zbiór treningowy dla YOLO, to również nie był on najwyższej jakości – przy przeglądaniu przykładowych zdjęć z narysowanymi bounding boxami zauważaliśmy, że zdarzało się, że na zdjęciu było wiele aut, ale tylko jedno z nich było zaznaczone, co z pewnością nie wpływało dobrze na naukę modelu, który mógł wykryć prawidłowo wszystkie auta, ale według adnotacji powinien tylko jedno.

8 Odnośniki

- **YOLO:**

Przykładowe zdjęcia z datasetu dostępne są pod linkiem: [\[1\]](#). W projekcie dane zostały wczytane przy pomocy biblioteki `fiftyone` i przefiltrowane. Ostateczny efekt dostępny jest pod linkiem: [\[2\]](#). Szczegóły implementacji znajdują się w pliku: `models/YOLO/YOLO.ipynb`.

- **Segmentacja:**

Algorytm segmentacji znaków z wyciętej tablicy rejestracyjnej testowano na zbiorze danych: [\[3\]](#). Szczegóły implementacji znajdują się w pliku: `models/char_recognition/character_recognition_model.ipynb`. Sam algorytm był częściowo inspirowany artykułem [\[4\]](#).

- **Rozpoznawanie znaków:**

Znaki wycięte przy pomocy naszego algorytmu segmentacji z zbioru danych do testowania segmentacji zostały odpowiednio przekształcone i wykorzystane do nauki modelu rozpoznającego znaki. Szczegóły implementacji dostępne są w pliku: `models/char_recognition/character_recognition_model.ipynb`, a powstały zbiór danych dostępny pod linkiem: [\[5\]](#).

- **Wstęp teoretyczny:**

Do wstępu teoretycznego użyto artykułu dostępnego pod linkiem: [\[6\]](#).