

Marcin Łykowski

numer albumu: 47168

kierunek studiów: Informatyka

specjalność: Inteligencja obliczeniowa

forma studiów: studia niestacjonarne

ROZPOZNAWANIE TABLIC REJESTRACYJNYCH POJAZDÓW NA OBRAZACH Z KAMERY SAMOCHODOWEJ

RECOGNITION OF VEHICLE LICENSE PLATES IN IMAGES FROM A CAR CAMERA

praca dyplomowa magisterska

napisana pod kierunkiem:

dra hab. inż. Przemysława Klęska, prof. ZUT

Katedra Metod Sztucznej Inteligencji i Matematyki Stosowanej

Data wydania tematu pracy: 30.03.2022

Data dopuszczenia pracy do egzaminu:
(uzupełnia pisemnie Dziekanat)

Szczecin, 2022

Oświadczenie autora pracy dyplomowej

Oświadczam, że praca dyplomowa magisterska pn. *Rozpoznawanie tablic rejestracyjnych pojazdów*

na obrazach z kamery samochodowej napisana pod kierunkiem dra hab. inż. Przemysława Klęska, prof. ZUT jest w całości moim samodzielnym autorskim opracowaniem sporządzonym przy wykorzystaniu wykazanej w pracy literatury przedmiotu i materiałów źródłowych. Złożona w dziekanacie Wydziału Informatyki treść mojej pracy dyplomowej w formie elektronicznej jest zgodna z treścią w formie pisemnej.

Oświadczam ponadto, że złożona w dziekanacie praca dyplomowa ani jej fragmenty nie były wcześniej przedmiotem procedur procesu dyplomowania związanych z uzyskaniem tytułu zawodowego w uczelniach wyższych.

Podpis autora:

Szczecin, dnia:

Streszczenie

W tym miejscu trzeba napisać streszczenie pracy w języku polskim. Zawiera krótką charakterystykę dziedziny, przedmiotu i wyników zaprezentowanych w pracy. Maksymalnie 1/2 strony.

słowa kluczowe: np. informatyka, sterowanie, grafika komputerowa

Abstract

The abstract's purpose, which should not exceed 150 words, is to provide sufficient information to allow potential readers to decide on the thesis's relevance—a maximum of half the page.

keywords: e.g.: computer science, control, computer graphics

Spis treści

Wstęp	7
1 Wprowadzenie teoretyczne	9
1.1 Przegląd istniejących metod detekcji tablic rejestracyjnych	11
1.1.1 Metody oparte na krawędziach (ang. <i>edge based</i>)	11
1.1.2 Metody oparte na kolorach (ang. <i>color based</i>)	13
1.1.3 Metody oparte na teksturach (ang. <i>texture based</i>)	14
1.1.4 Klasyfikatory	14
1.1.5 Metody głębokiego uczenia (ang. <i>Deep learning</i>)	19
1.2 Przegląd istniejących metod segmentacji tablic rejestracyjnych	20
1.2.1 Binaryzacja	21
1.2.2 Metoda rzutu jasności	23
1.2.3 Pozostałe metody	23
1.3 Przegląd istniejących metod rozpoznawania tablic rejestracyjnych	24
1.3.1 Metody oparte na sieciach neuronowych	24
1.3.2 Metody oparte na wzorcach	27
2 Zebranie materiału uczącego	29
3 Opracowany algorytm	33
3.1 Proces uczenia klasyfikatora	33
3.1.1 Przygotowanie danych uczących	33
3.1.2 Uczenie klasyfikatora	36
3.2 Schemat algorytmu	38
3.2.1 Algorytm detekcji	39
3.2.2 Algorytm segmentacji i rozpoznawania znaków	41
3.3 Biblioteki użyte w programie	43
3.3.1 OpenCV	43
3.3.2 Tesseract OCR	44
3.3.3 NumPy	44
3.3.4 Pickle	44
3.3.5 Numba	45

3.3.6	Joblib	45
4	Wyniki badań	47
4.1	Wyniki detekcji tablic rejestracyjnych	47
4.2	Wyniki rozpoznawaniu znaków	49
4.3	Porównanie wydajności czasowej	50
4.4	Propozycje udoskonalenia algorytmu	51
	Podsumowanie	52
	Spis literatury	55
	Książki	55
	Artykuły	55
	źródła internetowe i inne	58

Wstęp

W dzisiejszych czasach ludzka praca stanowi jeden z największych składników kosztów dla wielu przedsiębiorstw. Taki stan rzeczy prowadzi do poszukiwania rozwiązań mających na celu zautomatyzowanie najbardziej powtarzalnych czynności. Potwierdza to wzrost zainteresowania na przestrzeni ostatnich lat zagadnieniami takimi jak uczenie maszynowe czy widzenie komputerowe. Jedną z gałęzi gospodarki, w której tego rodzaju automatyzacja jest zauważalna, nawet dla osób niezwiązanych z branżą, jest transport drogowy. Nieustannie zwiększająca się liczba aut poruszających się po drogach, wzrost sieci dróg i autostrad niejako samoistnie wymusiła próby zautomatyzowania pewnych czynności.

Jednym z najczęściej poruszanych zagadnień jest problem rozpoznawania tablic rejestracyjnych (ang. *Licence Plate Recognition* — LPR). Do zadań takich systemów należy wykrycie na obrazie obszarów, w których znajdują się tablice rejestracyjne. Następnie na zlokalizowanych fragmentach rozpoznawane są numery rejestracyjne pojazdów. Dokładność uzależniona jest od wielu czynników, takich jak jakość obrazu, prędkość pojazdu, warunki atmosferyczne lubpora dnia.

Celem niniejszej pracy jest przedstawienie tematyki rozpoznawania tablic rejestracyjnych. Wybór takiego zagadnienia w niniejszej pracy dyplomowej motywowany jest zainteresowaniami autora w zakresie uczenia maszynowego oraz widzenia komputerowego, a także branżą motoryzacyjną. W zakres pracy wchodzą:

- omówienie wybranych algorytmów z zakresu przetwarzania obrazów i uczenia maszynowego, potrzebnych do realizacji postawionego zadania,
- przygotowania odpowiedniego materiału (sekwencje wideo) na potrzeby uczenia maszynowego i testowania,
- przedstawienie ostatecznego schematu algorytmicznego dla całego procesu,
- przeprowadzenie eksperymentów, pomiary dokładności i czasów wykonania, wnioski końcowe.

W pierwszej części pracy przedstawione zostaną najczęściej wykorzystywane techniki uczenia maszynowego i widzenia komputerowego do osiągnięcia wysokiej jakości systemu automatycznego rozpoznawania tablic rejestracyjnych. W drugiej części pracy zostanie przedstawiony stworzony program komputerowy do realizacji zadania rozpoznawania tablic rejestracyjnych. Program składa się z dwóch modułów. Pierwszy z nich odpowiada za detekcję tablic rejestracyjnych w obrazie. Drugi moduł rozpoznaje znaki na fragmentach obrazów przekazanych z modułu detekcji. Część ta zawiera szczegółowy opis bibliotek wykorzystanych do realizacji przedstawionego zadania oraz implementacji opracowanego algorytmu. Przedstawiono również etapy pozyskania zbioru uczącego dla opracowanego mechanizmu. Po opisaniu opracowanego procesu, zostaną zaraportowane wyniki dla

8 Rozpoznawanie tablic rejestracyjnych pojazdów na obrazach z kamery samochodowej

wyuczonego klasyfikatora.

W przedstawionej pracy udało się zrealizować postawione zadanie. Do realizacji programu wykorzystano język programowania Python w wersji 3.8. Klasyfikator oparty został o cechy Haara (ang. *Haar-like features*). Do klasyfikacji użyto algorytm RealBoost ze słabymi klasyfikatorami realizowanymi poprzez koszykowanie wartości funkcji logit (ang. *response binning*). Praca ma charakter eksperymentalny. Z tego powodu oraz z racji ograniczonych zasobów, algorytm ma swoje niedoskonałości. W podsumowaniu pracy zaprezentowano możliwości dalszego rozwoju klasyfikatora.

1. Wprowadzenie teoretyczne

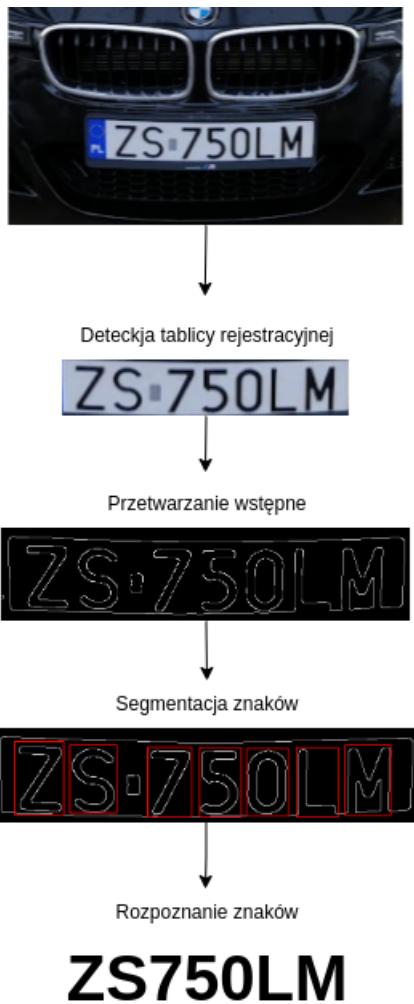
Na przestrzeni ostatnich lat stosowanie Systemów Automatycznego Rozpoznawania Tablic Rejestraacyjnych (ARTR) (ang. *Automatic Licence Plate Recognition* — ALPR) stało się znacznie bardziej powszechnie. W większości dużych miast istnieją parkingi, gdzie po umieszczeniu opłaty za postój, przy zbliżeniu się do wyjazdu, szlaban otwiera się automatycznie po rozpoznaniu numeru rejestracyjnego pojazdu, w którym się poruszamy. W obecnych czasach wszystkie nowoczesne systemy do zarządzania i sterowania ruchem drogowym oparte są o technologie ARTR. Instytucje takie jak służby drogowe, dzięki rejestrowanym i przetwarzanym w czasie rzeczywistym ogromnym ilościom danych, są w stanie odpowiednio szybko reagować na wydarzenia na drogach takie jak kolizje, korki lub innego rodzaju utrudnienia. Innym z możliwych przykładów zastosowania wspomnianych systemów są odcinkowe pomiary prędkości, opłaty za przejazd płatnymi drogami lub wykrywanie kierowców łamiących przepisy. Dzięki nieustannemu rozwojowi technologii i coraz wydajniejszym komputerom, systemy stają się tańszymi i łatwiej dostępną alternatywą dla systemów opartych na RFID (ang. *Radio-frequency identification*), które to wymagają specjalnej etykiety do prawidłowego działania.

Rozpoznawanie tablic rejestracyjnych jest techniką polegającą na wykryciu i odczytaniu znaków z tablicy rejestracyjnej na podstawie zarejestrowanego obrazu. Do tego celu wykorzystywany jest aparat o wysokiej rozdzielcości oraz odpowiedni program komputerowy. Oprogramowanie otrzymuje na wejściu cyfrową reprezentację obrazu. Dla zdjęć kolorowych każdy piksel opisany jest wartościami z palety barw RGB reprezentującymi jego barwę oraz współrzędnymi umiejscowienia w obrazie. Dla zdjęć monochromatycznych barwy opisywane są najczęściej za pomocą wartości luminacji obrazu.

W procesie automatycznego rozpoznawania tablic rejestracyjnych pozyskany obraz jest odpowiednio przetwarzany. Przed przejściem do rozpoznawania, obraz często jest konwertowany do skali szarości i filtrowany za pomocą filtrów (np. Gaussa lub średnio-przepustowego) w celu redukcji szumu. W procesie tym można wyróżnić trzy etapy [4]:

- **detekcję** — określenie położenia tablicy rejestracyjnej w analizowanym obrazie,
- **segmentację** — wyodrębnienie pojedynczych znaków na fragmencie obrazu ze zlokalizowaną tablicą,
- **identyfikację** — rozpoznanie każdego ze znaków i przedstawienie ich w formie tekstowej, którą można później wykorzystać do dalszych działań w zależności od przeznaczenia systemu.

Na Rysunku 1.1 przedstawiono graficzną reprezentację powyższego procesu.



Rysunek 1.1: Etapy procesu automatycznego rozpoznawania tablic rejestracyjnych (źródło: opracowanie własne).

Kolejne etapy korzystają z wyników uzyskanych w poprzednich krokach, co oznacza, że błąd powstajeły we wcześniejszej fazie, będzie rzutował na jakość działania całego systemu. W wielu systemach zanim dojdzie do rozpoznawania tablicy rejestracyjnej, obraz jest w pierwszej kolejności odpowiednio przetwarzany. Powszechnie stosowanymi czynnościami są: skalowanie obrazu, modyfikacje jasności oraz redukcja zakłóceń. W zależności od wymagań stawianych przed danym mechanizmem i środowiskiem jego działania, czynności te mogą znacznie się od siebie różnić. Najbardziej podstawowe systemy wymagają, aby pojazd znajdował się nieruchomo w określonym miejscu. Tego typu rozwiązania najczęściej stosowane są na parkingach, gdzie szlaban otwiera się po odczytce numerów rejestracyjnych pojazdu i potwierdzeniu opłaty za postój w zewnętrznej bazie danych. Takie systemy pracują w środowisku o niskim poziomie zakłóceń wynikających z warunków atmosferycznych i oświetlenia. Obecnie na rynku znajduje się wiele komercyjnych rozwiązań, które oferują wysoką dokładność (powyżej 95%) dla tego rodzaju detekcji. Taki rodzaj systemów ARTR nazywany systemami statycznymi. Dużo

większą złożonością charakteryzują się systemy dynamiczne, w których znacznie większą rolę odgrywają zakłócenia wynikające ze zmiennych warunków oświetlenia. W obecnych czasach stworzenie dynamicznego systemu ARTR o wysokiej dokładności wciąż stanowi wyzwanie i jest tematem wielu prac naukowych. Celem niniejszej pracy jest analizowanie obrazów pochodzących z kamery samochodowej, co zdecydowanie sprawia, że jest to system dynamiczny. Poniżej przedstawiono najczęściej stosowane metody widzenia komputerowego w systemach ARTR.

1.1 Przegląd istniejących metod detekcji tablic rejestracyjnych

Zgodnie ze słownikiem języka polskiego, definicja tablicy rejestracyjnej brzmi następująco:

Definicja 1.1.1 — Tablica rejestracyjna. Płytką zawierającą numery identyfikacyjne pojazdu, umieszczana z przodu i z tyłu pojazdu.

Dla programu komputerowego powyższe zdanie jest niezrozumiałe. W zadaniu detekcji tablicy rejestracyjnej, wymagane jest, aby maszyna „zrozumiała” jakich obiektów należy szukać. W tym kontekście, za definicję można uznać „prostokątny obszar, z dużym zagęszczeniem horyzontalnych i wertykalnych krawędzi” [45]. W oparciu o powyższe cechy zaprezentowano wiele algorytmów do rozwiązywania zadania wykrywania tablic rejestracyjnych. Część z nich wywodzi się z tradycyjnych metod widzenia komputerowego i metod głębokiego uczenia. Każda z metod ma swoje zalety, ale również często ograniczenia. W związku z tym, trudno jednoznacznie stwierdzić, która z metod jest najbardziej efektywna.

Detekcja numerów rejestracyjnych jest wyzwalającym zadaniem ze względu na poniższe czynniki:

- zajmowanie niewielkiego obszaru na zdjęciu przez tablicę rejestracyjną,
- istnienie ogromnej liczby formatów tablic rejestracyjnych (w zależności od kraju rejestracji lub rodzaju pojazdu),
- słabe oświetlenie, rozmazany obraz, refleksy świetlne,
- ruch pojazdu, zabrudzone tablice.

Tradycyjne metody widzenia komputerowego oparte są na cechach takich jak kształt, kolor, symetria, tekstury itp.[37]. W celu uzyskania lepszych wyników, spotyka się rozwiązania, w których łączy się wiele technik. Poniżej wyróżniono najczęściej stosowane metody w detekcji tablic rejestracyjnych.

1.1.1 Metody oparte na krawędziach (ang. edge based)

W większości krajów tablice rejestracyjne posiadają prostokątny kształt. Dla poprawienia widoczności numerów pojazdów, stosuje się kolory o wysokim kontraste dla czcionki i tła. Dzięki temu, tablice rejestracyjne zawierają wiele równoległych i prostopadłych linii. Z uwagi na to, znaczna część badań bazuje na podejściu opartym o wykrywanie krawędzi. W większości przypadków kolor tablicy rejestracyjnej jest różny od koloru pojazdu. Dzięki temu, granice tablicy zostają uznane za krawędzie. Wiele metod wykorzystuje filtr Sobela. Jego działania polega na dyskretnym różniczkowaniu i aproksymacji pochodnych kierunkowych intensywności obrazu. Filtr ten składa się z dwóch macierzy o wymiarach

3×3 (1.1) służących do detekcji krawędzi horyzontalnych i wertykalnych.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (1.1)$$

Zaletą takiego podejścia jest niewątpliwa łatwość użycia, natomiast jedną z głównych wad jest jego wrażliwość na szum.

Często wykorzystywana metodą do wykrywania krawędzi obiektów w obrazach jest *Binary Image Processing* [27]. Technika ta polega na sprowadzenia obrazu do postaci, w której kolory pikseli przyjmują tylko dwie wartości - czarną lub białą. Osiąga się to za pomocą ustalenia progu, który determinuje kolor piksela. Próg wyznaczany jest na podstawie histogramu obrazu w odcieniach szarości. Metoda ta jest użyteczna, ze względu na fakt łatwego odseparowania obiektu od tła. Wykorzystuje ona założenie, że krawędzie tablicy są proste i poziome. Przy zdeformowanych lub zabrudzonych tablicach, algorytm ten nie osiąga zadowalających wyników.

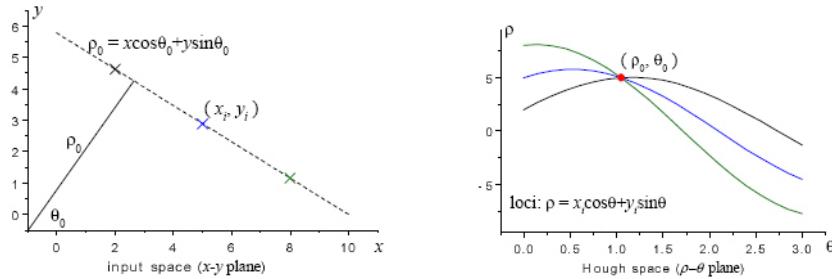
Inną stosowaną metodą do wykrywania linii na obrazach binarnych jest transformata Hougha [11]. Motywacją do jej opracowania była metoda siłowa (ang. *brute force*), która jest jednak znacznie bardziej zasobozerna. Złożoność algorytmu siłowego wynosi $O(n^3)$, gdzie n oznacza liczbę niezbędnych operacji do wykonania. Transformata Hougha polega na twierdzeniu, że każda prosta może być jednoznacznie przedstawiona za pomocą dwóch parametrów. Przestrzeń tych parametrów to właśnie przestrzeń Hougha. Najczęściej używanymi parametrami są współczynniki ρ i α z równania prostej w postaci normalnej (1.2)

$$x \cos \alpha + y \sin \alpha = \rho. \quad (1.2)$$

W powyższym równaniu ρ jest promieniem wodzącym, natomiast α kątem tworzonym przez ρ z osią X. W związku z powyższym, jest to algorytm o liniowej złożoności obliczeniowej. Można wykazać następujące własności transformacji Hougha:

Twierdzenie 1.1.1 Prostej przestrzeni kartezjańskiej odpowiada w przestrzeni Hougha punkt, natomiast punktowi przestrzeni kartezjańskiej odpowiada w przestrzeni Hougha sinusoidalna krzywa. Punkty leżące na tej samej prostej korespondują z sinusoidami przechodzącymi przez wspólny punkt w przestrzeni Hougha [40].

Zasadę transformacji ilustruje Rysunek 1.2.



Rysunek 1.2: Transformata Hougha (źródło: [20]).

Innym spotykanym podejściem [13] jest stosowanie dwóch algorytmów. Pierwszy z nich ma za zadanie wyodrębnić odcinki linii i pogrupować je na podstawie wcześniej ustalonego zbioru warunków geometrycznych. Drugi znajduje obszary o najwyższym zagęszczeniu pionowych krawędzi. Dzięki takiemu spojrzeniu na przedstawiony problem, uzyskane wyniki mają wysoką dokładność, szczególnie dla pojazdów znajdujących się w ruchu. Metody oparte na krawędziach są stosowane w wielu rozwiązańach ze względu na ich szybkość działania i prostotę. Jednakże, rozwiązania te są silnie wrażliwe na niepożądane krawędzie i nie sprawdzają się w rozmytych i złożonych obrazach.

1.1.2 Metody oparte na kolorach (ang. *color based*)

Metody oparte na kolorach bazują na fakcie, że kolor tablicy jest różny od koloru tła pojazdu. Dla tej grupy rozwiązań, zamiast modelu barw RGB, stosuje się model HSL oparty o nasycenie koloru. Model ten jest jednak wrażliwy na szum.

Często metody wykorzystujące kolor tablicy rejestracyjnej są używane do wyselekcjonowania kandydatów. Innymi słowy, oznacza to wybrania obszarów obrazu, w których może znajdować się tablica rejestracyjna. Technika ta łączona jest z innymi algorytmami, które na kolejnych etapach decydują, czy wskazany obszar rzeczywiście zawiera poszukiwany obiekt. Do tego typu metod wykorzystywany jest m. in. algorytm *Mean shift* [17] i logika rozmyta [42].

Opisywana grupa metod może zostać użyta do detekcji zdeformowanych i pochylonych tablic. Rzadko występują one osobno w metodach detekcji, głównie ze względu na ich dużą czułość na zmiany naświetlenia. Dodatkowo w zależności od kraju oraz przeznaczenia pojazdu, kolory tablic mogą się znacznie różnić. Przykładowo obecnie w Polsce tablice aut elektrycznych mają kolor zielony, a samochodów zabytkowych żółty, patrz Rys. 1.3.



Rysunek 1.3: Tablice rejestracyjne w Polsce dla aut elektrycznych i zabytkowych (źródło: opracowanie własne).

1.1.3 Metody oparte na teksturach (ang. *texture based*)

Metody oparte na teksturach wykorzystują fakt znajdowania się znaków na tablicach rejestracyjnych. Znaki na tablicy mają z reguły czarny kolor i znajdują się na jasnym tle tworząc duży kontrast. Powyższa grupa algorytmów wykorzystuje wysokączęstość zmiany kolorów w obszarze występowania tablic rejestracyjnych. W [45] autorzy zaproponowali metodę lokalizacji tablic wykorzystując algorytm kwantowania wektorowego (ang. *Vector Quantization - VQ*). W przeciwieństwie do innych metod, które wykorzystywały krawędzie lub kontrast, metoda VQ wykorzystuje aktualną zawartość tablicy rejestracyjnej. Autorzy wykazali bardzo wysoką skuteczność rozwiązania na poziomie 98%.

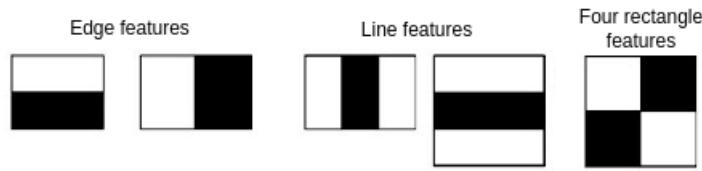
W analizie tekstur, często stosuje się filtr Gabora. Jest to filtr liniowy, pozwalający na przefiltrowanie obrazu z precyzyjnie dobranym zakresem częstotliwości. Reprezentacje częstotliwości i orientacji filtrów Gabora są uważane przez wielu współczesnych naukowców zajmujących się widzeniem komputerowym za podobne do tych z ludzkiego układu wzrokowego [24]. W [6] zaprezentowano algorytm wykorzystujący filtr Gabora. Jest to jednak metoda czasochłonna i nie znajduje zastosowania w systemach, w których szybkość działania jest jednym z najistotniejszych czynników.

Wszystkie metody oparte na teksturach są odporne na deformacje tablic. Jest to kluczowa zaleta ich stosowania. Mimo to, metody te wymagają skomplikowanych obliczeń i nie dają zadowalających efektów w złożonych środowiskach z różnymi warunkami oświetlenia.

1.1.4 Klasyfikatory

Wiele badań wykorzystuje cechy Haara razem z algorymem AdaBoost (ang. *Adaptive Boosting — AdaBoost*) do wyuczenia kaskady klasyfikatorów [37]. Podejście takie zostało zaproponowane po raz pierwszy w [41]. Algorytm Violi-Jonesa został zaprezentowany w 2001. Wykorzystuje on algorytm AdaBoost, który został zaproponowany w pracy [12]. Pomimo upływu ponad 20 lat, dalej jest on powszechnie stosowany, co świadczy o jego ponadczasowości i uniwersalności. Autorzy zaprojektowali go jako detektor twarzy, jednak jego funkcjonalność pozwala na wykrywanie dowolnych obiektów, np. tablic rejestracyjnych, przy odpowiednim wyuczeniu. Aby przedstawić, jak działa algorytm Violi-Jonesa, należy najpierw zrozumieć, czym są cechy Haara.

Cechy Haara często są przedstawiane jako skalowalne, prostokątne szablony (Rysunek 1.4), używane do porównania zależności pomiędzy pikselami. Reprezentują one zgrubne kontury obiektu. Obraz skanowany jest oknem przesuwnym o rozmiarze zbliżonym do poszukiwanego obiektu. Dla każdego okna obliczane są wartości cech Haara, na podstawie, których klasyfikator podejmuje decyzję. Wartość pojedynczej cechy obliczana jest jako różnica pomiędzy średnią jasnością pikseli w zbiorze „białym” i średnią jasnością pikseli w zbiorze „czarnym” [50].



Rysunek 1.4: Cechy Haara używane w algorytmie Violi-Jonesa (źródło: https://docs.opencv.org/4.x/d2/d99/tutorial_js_face_detection.html).

Im większa będzie liczba okien w procedurze skanującej, tym większa będzie niezbędna liczba cech do wyliczenia. Zauważono, że cechy mogą być jednak wyznaczane w czasie stałym, niezależnym od rozmiaru okna. W tym celu należy przygotować przed procedurą detekcji dodatkową tablicę zwaną obrazem całkowym (ang. *integral image*). Obrazem całkowym nazywamy tablicę dwuwymiarową o rozmiarze obrazu źródłowego, w której każdy element w i-tym wierszu i j-tej kolumnie przechowuje sumę pikseli z tej części obrazu, której prawym dolnym wierzchołkiem jest piksel (i, j) . Matematycznie obraz całkowy $ii(x, y)$ przedstawiamy jako:

$$ii(x, y) = \sum_{1 \leq j \leq x} \sum_{1 \leq k \leq y} i(j, k). \quad (1.3)$$

Obliczanie obrazu całkowego z definicji (1.3) dla każdego punktu w obrazie jest czasochłonne i charakteryzuje się złożonością obliczeniową $O(n_x^2 n_y^2)$. Sposobem na szybsze obliczenie obrazu całkowego może być wykorzystanie wzorów rekurencyjnych [9, 41]:

$$s(j, k) = s(j, k - 1) + f(j, k), \quad (1.4)$$

$$ii(j, k) = ii(j, k - 1) + s(j, k), \quad (1.5)$$

gdzie $s(j, k)$ to skumulowana suma w wierszu, $f(j, k)$ to wartość obrazu w punkcie (j, k) , natomiast $s(j, -1) = 0$ oraz $ii(-1, k) = 0$ [41]. Dla $j = k = 1$ zachodzi równość $s(1, 1) = f(1, 1)$. Dzięki takiemu rozwiązaniu, wyznaczanie obrazu całkowego charakteryzuje się złożonością obliczeniową $O(n_x n_y)$.

Kiedy obraz całkowy został wyznaczony, obliczenie sumy jasności dla dowolnego fragmentu obrazu jest operacją o stałej złożoności obliczeniowej $O(1)$. Aby osiągnąć złożoność niezależną od rozmiaru okna, sumę oblicza się odejmując od sumy wartości obrazu całkowego dla prawego dolnego i lewego górnego wierzchołka sumę wartości obrazu dla pozostałych wierzchołków analizowanego prostokąta. Poniżej przedstawiono obliczenie sumy dla prostokąta rozpiętego między punktami (x_1, y_1) , a (x_2, y_2) .

$$\sum_{x_1 \leq x \leq x_2} \sum_{y_1 \leq y \leq y_2} i(x, y) = ii(x_2, y_2) - ii(x_1 - 1, y_2) - ii(x_2, y_1 - 1) + ii(x_1 - 1, y_1 - 1) \quad (1.6)$$

Na podstawie powyższego wzoru można zauważać, że wystarczą operacje tylko na 4 punktach obrazu całkowego [50]. Dla obliczenia różnicy pomiędzy sumami dwóch dowolnych prostokątów wymagane jest pobranie wartości dla 8 punktów z obrazu całkowego. Poprzez zastosowanie skalowania szablonów i zakotwiczanie ich w różnych miejscach badanego okna, możliwe jest uzyskanie dużej (tzn. rzędu 10^3 lub 10^4) liczby cech Haara

bez konieczności skalowania obrazu. Tak duża liczba cech wymagana jest na etapie uczenia klasyfikatora. Dzięki zastosowaniu boostingu, algorytm dokonuje selekcji znacznie mniejszej liczby cech, które zapewnią dobry opis rozpatrywanych obiektów w danym zagadnieniu detekcji.

AdaBoost (Adaptive Boosting) to jedna z technik boostingu. Takie podejście zostało zaprezentowane po raz pierwszy w [12]. Boostingiem nazywamy algorytm, którego zadaniem jest stworzenie mocnego klasyfikatora na podstawie wielu słabych klasyfikatorów. Słabym klasyfikatorem jest klasyfikator, który osiąga niską dokładność, jednak wyższą od losowych wyników. Za przykład może posłużyć rozpoznawanie płci na podstawie wzrostu. Słaby klasyfikator mógłby bazować na założeniu, że każda osoba o wzroście 175cm lub wyższym jest mężczyzną. Pozostała grupa osób jest kobietami. Wiele osób ze zbioru testowego zostanie określonych błędnie, jednak dokładność klasyfikatora będzie wyższa niż 50%. Na poniższych schemacie przedstawiono pseudokod algorytmu uczącego AdaBoost.

Algorytm 1 Algorytm uczący klasyfikatora AdaBoost.

```

1: procedure DISCRETEADABoost(( $x_i, y_i$ ))  $\triangleright i = 1, \dots, m, y_i \in \{-1, +1\}$ 
2:   Rozpocznij od jednostajnego rozkładu wag:  $w_i := \frac{1}{m}, i = 1, \dots, m.$ 
3:   for  $t := 1, \dots, T$  do
4:     Naucz klasyfikator  $f_t(x) \in \{-1, +1\}$  na danych uczących używając wag  $w_i$ .
5:     Oblicz błąd uczący:
6:      $\epsilon_t := \sum_{i=1}^m w_i [f_t(x_i) \neq y_i]$ .
7:     Oblicz wagę klasyfikatora:
8:      $\alpha_t := \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$ .
9:     Aktualizuj wagi przykładów wg:
10:     $Z_t := \sum_{i=1}^m w_i \exp(-\alpha_t f_t(x_i) y_i)$ ,
11:     $w_i := \frac{w_i \exp(-\alpha_t f_t(x_i) y_i)}{Z_t}$ .
12:   end for
13:   Zwróć zbiorowy klasyfikator  $F(x) := \sum_{t=1}^T \alpha_t f_t(x)$ .
14: end procedure

```

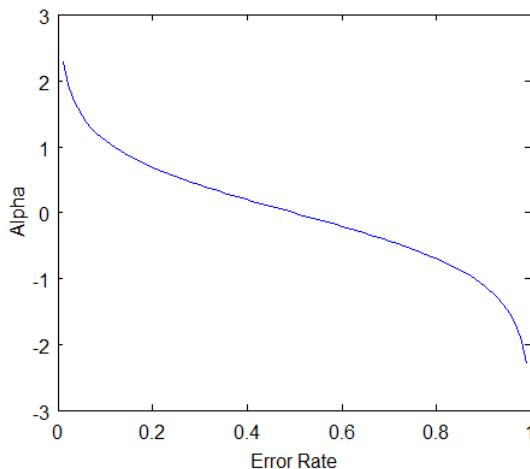
Algorytm przyjmuje na wejściu zbiór uczący wraz z etykietami opisującymi poszczególne elementy zbioru. Do każdej pary uczącej (x_i, y_i) przypisywana jest waga równa w_i . Na początku uczenia, wszystkie wagi przykładów mają taką samą wartość, a ich suma jest równa 1. Na koniec każdej pętli uczącej dochodzi do reważenia przykładów, a wagi błędnie sklasyfikowanych próbek zostają zwiększone. Analogicznie dla poprawnie sklasyfikowanych przykładów wagi zostają zmniejszone. Dzięki temu, w następnej iteracji uczenia kolejny klasyfikator będzie mógł lepiej rozpoznawać niepoprawnie oznaczone jednostki treningowe w poprzednim kroku. Waga słabego klasyfikatora, wyliczona przez AdaBoost, została określona symbolem α_t . Jest ona obliczana na podstawie błędu

klasyfikacji ϵ_t . Ostateczną odpowiedź klasyfikatora można opisać wzorem (1.7).

$$F(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t f_t(x) \right) \quad (1.7)$$

Finalny klasyfikator zawiera T słabych klasyfikatorów. Każdy ze słabych klasyfikatorów zwraca jedną z dwóch odpowiedzi $\{-1, 1\}$. Ostateczny rezultat powyższego równania można opisać jako liniowa kombinacja słabych klasyfikatorów.

Współczynnik α_t jest skorelowany z poziomem błędu klasyfikatora ϵ_t , który to jest liczony jako liczba błędnie sklasyfikowanych próbek dzielona przez rozmiar zbioru uczącego. Rysunek 1.5 przedstawia powyższą zależność.



Rysunek 1.5: Zależność współczynnika α od błędu klasyfikatora (źródło: https://chrisjmccormick.files.wordpress.com/2013/12/adaboost_alpha.png).

Z Rysunku 1.5 można odczytać, że waga rośnie dla klasyfikatorów o wysokiej dokładności. Dla klasyfikatora o poziomie błędu 0.5 waga jest równa zero. Oznacza to, że taki klasyfikator jest równie dokładny co losowe zgadywanie, dlatego jest on ignorowany. Dla klasyfikatorów o większym poziomie błędu, waga przyjmuje wartość ujemną. Jeśli taki klasyfikator uzna próbę za negatywną, ostatecznie zostanie ona oznaczona jako pozytywna.

AdaBoost jest techniką, która może zostać łączona z dowolnym algorytmem klasyfikującym, jednak nie może zostać wykorzystany jako samodzielny klasyfikator. AdaBoost występuje w połączeniu z popularnymi wariantami słabych klasyfikatorów [50]:

- AdaBoost + decision stump,
- AdaBoost + drzewka decyzyjne,
- AdaBoost + klasyfikator liniowy (np. SVM),
- AdaBoost + naiwny Bayes.

Głównymi zastosowaniami tej techniki jest wybór najlepszych cech z punktu widzenia klasyfikacji oraz dobór wag dla klasyfikatorów wchodzących w skład kaskady. Dzięki selekcji cech najlepiej opisujących dany obiekt, jest odporna na zakłócenia.

Poza algorytmem AdaBoost, stosowane są również inne metody boostingu. Jedną z nich jest Real AdaBoost, zwany również RealBoostem, zaprezentowany po raz pierwszy

w [34]. Główną różnicą w stosunku do AdaBoosta, jest założenie, że słabe klasyfikatory są rzeczywistoliczbowe, a nie binarne [50]. Na poniższych schemacie przedstawiono pseudokod algorytmu uczącego RealBoost.

Algorytm 2 Algorytm uczący klasyfikatora RealBoost.

```

1: procedure REALADABOOST( $(x_i, y_i)$ )  $\triangleright i = 1, \dots, m, y_i \in \{-1, +1\}$ 
2:   Rozpocznij od jednostajnego rozkładu wag:  $w_i := \frac{1}{m}, i = 1, \dots, m.$ 
3:   for  $t := 1, \dots, T$  do
4:     Naucz klasyfikator  $f_t(x) \in \mathbb{R}$  na danych uczących używając wag  $w_i$ , tak aby  $f_t$  minimalizowało kryterium wykładnicze  $\sum_{i=1}^m w_i \exp(-f_t(x_i)y_i)$  lub równoważnie aby  $f_t$  było przybliżeniem połowy przekształcenia logit:
5:     
$$f_t(x) = \frac{1}{2} \ln \frac{\hat{P}_w(y = 1|x)}{\hat{P}_w(y = -1|x)}.$$

6:     Aktualizuj wagi przykładów wg:
7:     
$$Z_t := \sum_{i=1}^m w_i \exp(-f_t(x_i)y_i),$$

8:     
$$w_i := \frac{w_i \exp(-f_t(x_i)y_i)}{Z_t}.$$

9:   end for
10:  Zwróć zbiorowy klasyfikator  $F(x) := \sum_{t=1}^T f_t(x).$ 
11: end procedure
  
```

Schemat ten jest bardzo podobny do schematu algorytmu AdaBoost. W przeciwieństwie do AdaBoost, słabe klasyfikatory nie posiadają wag. Mechanizm ważenia słabych klasyfikatorów jest niejako wpleciony w same odpowiedzi rzeczywistoliczbowe [50]. Odpowiedź słabego klasyfikatora jest zwykle ustalana jako przybliżenie połowy przekształcenia logit:

$$f_t(x) = \frac{1}{2} \ln \frac{\hat{P}_w(y = 1|x)}{\hat{P}_w(y = -1|x)}, \quad (1.8)$$

gdzie $\hat{P}_w(y = \pm 1|x)$ stanowi oszacowanie rozkładu klas warunkowego na x z wykorzystaniem aktualnych wag w_i . Ostateczną odpowiedź klasyfikatora można opisać wzorem (1.9).

$$F(x) = \text{sign} \left(\sum_{t=1}^T f_t(x) \right) \quad (1.9)$$

Algorytm RealBoost posiada silne podobieństwa do techniki regresji logistycznej. Schemat reważenia w boostingu pracuje w sposób pokrewny do metody rezyduów błędów (ang. *error residuals*).

Słabe klasyfikatory stosowane w algorytmach boostingowych mogą być realizowane na wiele sposobów. Stosuje się np. płytke drzewa decyzyjne lub tzw. decision stumps. Innym spotykanym podejściem jest koszykowanie wartości funkcji logit (ang. *response binning*). Ten typ klasyfikatora został zaprezentowany w [31]. Przedstawiony sposób działania polega na przybliżaniu rozkładów warunkowych przez funkcje kawałkami stałe [50].

Przed uczeniem algorytmu wyznaczana jest liczba koszy, oznaczana literą B , o równej szerokości. Oznaczmy przez chwilę przedział zmienności dowolnej cechy jako $[a_1, a_2]$. Wówczas indeks kosza, $\beta(x) \in \{1, \dots, B\}$, do którego należy x obliczany jest na podstawie wzoru (1.10).

$$\beta(x) = \begin{cases} B(x - a_1)/(a_2 - a_1), & \text{dla } a_1 \leq x \leq a_2 \\ 1, & \text{dla } x \leq a_1 \\ B, & \text{dla } a_2 < x \end{cases} \quad (1.10)$$

Odpowiedź słabego klasyfikatora dla wybranej j^* -tej cechy przedstawia wzór (1.11), gdzie $\hat{P}_w(y = -1, j^* \text{ jest } w \ b) = \sum_{\{i: y_i = -1, \beta(x_{ij^*}) = b\}} w_i$ oznacza szacowane prawdopodobieństwo zdarzenia, że przykład jest negatywny, a jego j -ta cecha należy do kosza b .

$$f_t(x; j^*) = \frac{1}{2} \ln \frac{\hat{P}_w(y = 1, j^* \text{ jest } w \ \beta(x_{j^*}))}{\hat{P}_w(y = -1, j^* \text{ jest } w \ \beta(x_{j^*}))} \quad (1.11)$$

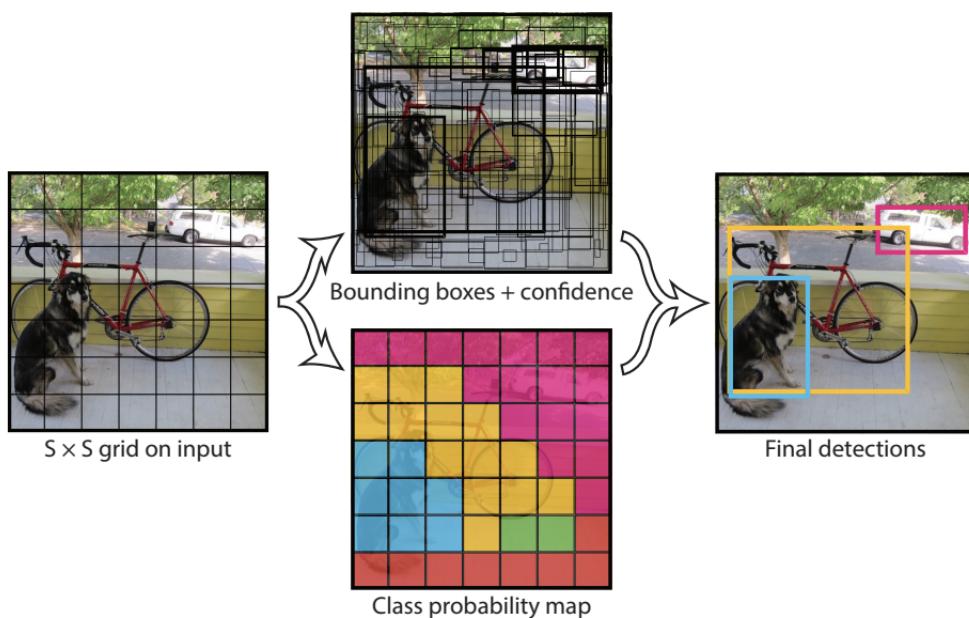
Autorzy algorytmu zalecają stosowanie dużej liczby koszy dla gładkiego histogramu rozkładu cech. Jeżeli histogram zawiera wiele maksimów lokalnych, należy zmniejszyć liczbę koszyków. W przeciwnym razie zbyt duża rozdrobniość może wprowadzić szum do klasyfikatora, natomiast zbyt niska liczba koszy może pozbawić klasyfikatora istotnych cech [31].

We wspomnianym wcześniej algorytmie Violi-Jonesa wykorzystano boosting techniką AdaBoost. Autorzy zaproponowali użycie kaskady klasyfikatorów. Takie podejście bazuje na obserwacji, że okna pozytywne stanowią średnio 0.01% wszystkich okien. Rozpatrywane okno w pierwszej kolejności badane jest przez słabsze klasyfikatory, które bazują na mniejszej liczbie cech. Dzięki takiemu podejściu algorytm stał się bardziej wydajny czasowo. Każdy kolejny klasyfikator w kaskadzie oblicza większą liczbę cech. Jeżeli na którymś etapie klasyfikator zwróci odpowiedź negatywną, proces jest przerwany. Do osiągnięcia pozytywnego wyniku, wymagane jest zwrócenie przez wszystkie klasyfikatory odpowiedzi pozytywnej. W algorytmie Violi-Jonesa kaskada składa się z 32 klasyfikatorów, które badają od 2 do 200 cech. Łącznie liczone jest 4297 cech, co daje średnio 8 cech na klasyfikator. Algorytm jest ceniony za szybkość detekcji. Pomimo faktu, że jego dokładność jest niższa od dokładności nowoczesnych metod opartych na sieciach neuronowych, stosowany jest on w rozwiązaniach o ograniczonej mocy obliczeniowej. Wynika to z faktu, że algorytm Violi-Jonesa jest bardzo wydajny oraz bazuje na stosunkowo niewielkiej liczbie cech.

1.1.5 Metody głębokiego uczenia (ang. *Deep learning*)

W związku z rozwojem dziedziny widzenia komputerowego oraz wzrostem mocy komputerów na przestrzeni ostatnich lat, wiele metod statystycznych zostało zastąpionych przez sieci neuronowe z powodu ich wysokiej skuteczności w rozpoznawaniu obiektów. Jednym z zaproponowanych podejść jest użycie sieci splotowych (ang. *Convolutional Neural Network* — CNN) [35]. Składają się one z jednej lub wielu warstw splotowych (typowych dla kroku próbkowania, określającego subwzorce) oraz z jednej lub wielu w pełni połączonych warstw tak jak w klasycznej wielowarstwowej sieci. Ten rodzaj sieci neuronowych jest predestynowany do obliczeń na strukturach 2D (tj. obrazy) [49].

Jednym z najnowocześniejszych systemów detekcji w czasie rzeczywistych jest algorytm YOLO (ang. *You only look once*) [32]. Algorytm jest bardzo wydajny. Bazuje on na odpowiednio zaprojektowanym zadaniu regresji do predykcji tzw. bounding boxów obiektów. Autorzy zapewniają o możliwości przetwarzania 45 klatek na sekundę, a dla wersji szybszej, lecz o niższej dokładności, ponad 150 klatek na sekundę. W przeciwieństwie do tradycyjnych metod z oknem przesuwnym, YOLO podczas procesu uczenia i testowania otrzymuje na wejściu cały obraz. Na obraz wejściowy nałożona zostaje siatka o rozmiarze $S \times S$, która tworzy obwiednie, a następnie wykorzystuje je do rozpoznawania szukanych obiektów. Dla każdej uzyskanej ramki, mechanizm wyznacza prawdopodobną klasę. Ramki o prawdopodobieństwie wyższym od ustalonego progu, zostają nałożone na wejściowy obraz, najczęściej w postaci prostokątów otaczających obiekt. Rysunek 1.6 przedstawia opisany schemat. Ograniczenia YOLO wynikają z ograniczeń przestrzennych algorytmu. W zależności od wielkości siatki, system może mieć problem z wykryciem mniejszych obiektów takich jak np. stado ptaków. Algorytm jest stale udoskonalany przez autorów. Na moment pisania pracy, udostępniona została trzecia wersja systemu.



Rysunek 1.6: Algorytm YOLO (źródło: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>).

1.2 Przegląd istniejących metod segmentacji tablic rejestracyjnych

Drugim etapem w większości systemów Automatycznego Rozpoznawania Tablic Rejestracyjnych jest rozpoznanie znaków na tablicy rejestracyjnej zlokalizowanej w poprzednim kroku. Zanim dojdzie do klasyfikacji znaków, często są one najpierw segmentowane. Jest to szczególny przypadek optycznego rozpoznawania znaków [37]. Wiele krajów posiada ścisłe regulacje na temat czcionki i kolorów tablicy rejestracyjnej. Przepisy te mają

na celu zwiększenie czytelności znaków identyfikujących każdy pojazd. W Polsce, jak i w większości krajów na świecie, tablice pokryte są specjalną warstwą refleksyjną odbijającą światło. Działanie takie zwiększa ich czytelność dla ludzkiego oka w gorszych warunkach oświetleniowych. Efektem takiej cechy tablic, może być ich nieczytelność na zdjęciu wykonanym aparatem fotograficznym. Tablica odbija na tyle dużą ilość światła, co skutkuje powstaniem na zdjęciu białego prostokąta w miejscu tablicy. Przykład takiego obrazu przedstawia Rysunek 1.7.

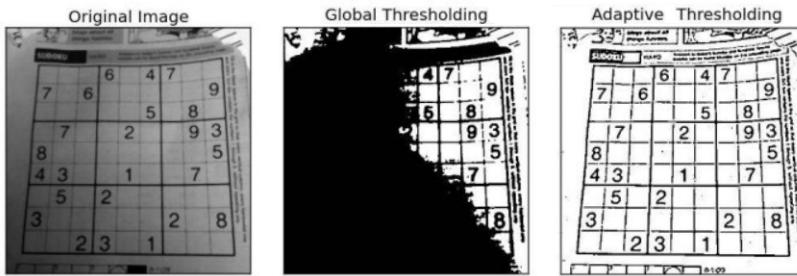


Rysunek 1.7: Refleks świetlny występujący w niewystarczających warunkach oświetleniowych (źródło: opracowanie własne).

Poza powyższymi trudnościami, tablice mogą być obrócone lub uszkodzone. Tablice często zawierają również śruby mocujące, które mogą utrudnić proces segmentacji. Aby zwiększyć w jak największym stopniu prawdopodobieństwo prawidłowego odczytania znaków, używa się szeregu czynności na obrazie wejściowym. Na przykład dla obróconych obiektów stosuje się transformację biliniową, zwaną również metodą Tustina [43].

1.2.1 Binaryzacja

W wielu klasycznych metodach widzenia komputerowego, w celu segmentacji znaków, stosuje się binaryzację. W zbinaryzowanym obrazie łatwiej jest rozdzielić znaki w porównaniu do obrazu kolorowego lub w skali szarości. Podstawowym problemem tej techniki jest odpowiedni dobór wartości progu. Wybór ten jest dokonywany w oparciu o różne metody, które można podzielić na zmiennoprogowe i automatyczne. Jednakże, wyznanie progu binaryzacji musi zostać wykonane właściwie, w celu uniknięcia połączenia znaków lub scalenia ich z obramowaniem tablicy rejestracyjnej [10]. Jedną z częściej stosowanych metod binaryzacji jest progowanie adaptacyjne (ang. *adaptive thresholding*). Stosuje się je w momencie kiedy różne obszary obrazu mogą charakteryzować się odmiennymi warunkami oświetlenia i stosowanie stałej wartości nie dałoby zadowalających efektów. Algorytm wyznacza próg dla konkretnego piksela na podstawie niewielkiego regionu wokoło. Dla tego samego obrazu wyliczane są różne wartości progu dla różnych regionów, co prowadzi do uzyskania lepszych wyników niż w przypadku stałego progu binaryzacji. Na Rysunku 1.8 przedstawiono przykłady binaryzacji obrazu stałym progiem globalnym i progiem wyznaczonym adaptacyjnie.



Rysunek 1.8: Przykład binaryzacji obrazu przy niejednorodnym oświetleniu za pomocą globalnego i lokalnego progu (źródło: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html).

Algorytm progowania adaptacyjnego otrzymuje najczęściej na wejściu obraz w skali szarości lub kolorowy. Próg jest wyliczany dla wszystkich pikseli w obrazie z osobna. Jednym ze znanych podejść jest to zaproponowane przez Chow i Kaneko [8]. W tym przypadku obraz dzielony jest na zbiór nakładających się fragmentów. Optymalny próg wyznaczany jest dla konkretnego fragmentu na podstawie analizy histogramu. Próg dla każdego piksela wyliczany jest na podstawie interpolacji wyników dla odpowiedniego fragmentu. Wadą tej metody jest jej złożoność obliczeniowa. Jest to jeden z głównych powodów, dlaczego algorytm ten nie znalazł zastosowania w aplikacjach działających w czasie rzeczywistym.

Alternatywnym podejściem do znalezienia lokalnego progu jest statystyczne zbadanie intensywności sąsiedztwa każdego piksela. Wybór metody statystycznej jest skorelowany z obrazem wejściowym. Najczęściej używa się jednej z następujących wielkości: średniej, mediany, średniej z wartości maksymalnej i minimalnej badanego otoczenia lub funkcji Gaussa. Dobór wielkości sąsiedztwa ma duży wpływ na wyliczenie progu. Rozmiar musi być na tyle duży, aby zawierać zarówno piksele tła jak i badanego obiektu. Z drugiej strony, wybór zbyt dużych obszarów może naruszyć założenie o równomiernym oświetleniu rozpatrywanego fragmentu. Ten rodzaj progowania adaptacyjnego jest bardziej wydajny od algorytmu Chow i Kaneko. Charakteryzuje się wysoką jakością oddzielania obiektów od tła i znajduje szerokie zastosowanie w wielu aplikacjach.

Innym znanim algorymem stosowanym do wyznaczania progu binaryzacji jest algorytm Otsu [25]. Metoda Otsu jest techniką opartą na wariancji w celu znalezienia wartości progowej, przy której ważona wariancja między pikselami pierwszego planu i tła jest najmniejsza [55]. Kluczową ideą jest tutaj iteracja przez wszystkie możliwe wartości progu i pomiar rozproszenia pikseli tła i pierwszego planu. Następnie znajdowany jest próg, w którym rozproszenie jest najmniejsze. Algorytm iteracyjnie wyszukuje próg, który minimalizuje wariancję wewnętrz klasy, zdefiniowaną jako ważona suma wariancji dwóch klas (tła i pierwszego planu). Wzór na znalezienie wariancji wewnętrz klasowej przy dowolnym progu t jest określony wzorem (1.12)

$$\sigma^2(t) = \omega_{bg}(t)\sigma_{bg}^2(t) + \omega_{fg}(t)\sigma_{fg}^2(t), \quad (1.12)$$

gdzie $\omega_{bg}(t)$ i $\omega_{fg}(t)$ reprezentują prawdopodobieństwa liczby pikseli dla każdej klasy przy progu t , natomiast σ^2 oznaczono wariancję wartości kolorów. Wyliczanie wariancji

dla konkretnej klasy pikseli przedstawiono na wzorze (1.13).

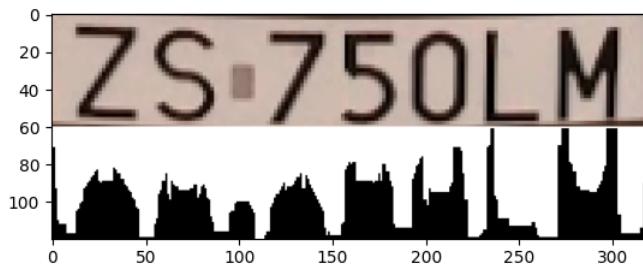
$$\sigma_{bg|fg}^2(t) = \frac{\sum_i (x_i - \hat{x})^2}{N - 1} \quad (1.13)$$

Wartość piksela odpowiedniej klasy reprezentuje symbol x_i , natomiast \hat{x} reprezentuje średnią pikseli rozpatrywanej klasy. Liczbę wszystkich pikseli zapisano literą N .

Metoda Otsu implementowana jest przez wiele środowisk obliczeniowych (np. MATLAB) [21]. Algorytm szczególnie dobrze sprawdza się w przypadkach, gdy liczby pikseli tła i obiektów pierwszego planu są zbliżone [36].

1.2.2 Metoda rzutu jasności

W kontekście segmentacji znaków, metoda rzutu jasności wykonywana jest na fragmencie obrazu potencjalnie zawierającym tablicę rejestracyjną. W celu obliczenia rzutu jasności, sumowane są wartości pikseli w kolumnach dla obrazu w skali szarości. Możliwe też jest generowanie rzutów jasności w poziomie, natomiast taka technika ma większe zastosowanie w detekcji tablic. Rysunek 1.9 przedstawia działanie metody pionowego rzutu jasności.



Rysunek 1.9: Pionowy rzut jasności dla obrazu tablicy rejestracyjnej (źródło: opracowanie własne).

Dolna część rysunku przedstawia rozkład liczby ciemnych pikseli w danej kolumnie obrazu. Dla wyższej liczby ciemnych pikseli, linie rzutu jasności przyjmują wyższe wartości. Przerwy pomiędzy znakami są wyraźnie widoczne na wykresie. Na tej podstawie, możliwe jest osiągnięcie segmentacji znaków. Powyższy przykład zawierał jednak idealnie wyciętą tablicę rejestracyjną. W rzeczywistych systemach często zdarza się, że potencjalne fragmenty zawierające tablice mają większy rozmiar niż sama tablica. Może to wynikać z powiększania zlokalizowanego obszaru w celu uniknięcia przycięcia tablicy przez algorytm detekcji. Ograniczeniem tej metody jest wymaganie równoległości tablicy do krawędzi obrazu. Dodatkowo, dla niektórych liter, widoczne są minima lokalne wynikające z ich budowy, co może prowadzić do błędnej interpretacji wyników [28].

1.2.3 Pozostałe metody

Wyodrębnienie znaków od tła jest problemem rozważanym w wielu pracach. Wciąż poszukiwane są sposoby segmentacji o wyższej skuteczności niż te obecnie stosowane. W [5] autorzy skanowali w poziomie obraz binarny, w celu znalezienia obszarów, w których stosunek pikseli tła do pikseli znaków przekraczał predefiniowany próg. Jeżeli

takie miejsce znaleziono, uznawano je za początek znaku. Dla odnalezienia końca znaku wykonywano odwrotną procedurę.

W pracy [26] zaproponowano podejście bazujące na cechach geometrycznych tablicy rejestracyjnej. Odnalezioną tablicę skalowano do stałego rozmiaru. Dzięki znajomości wymiarów tablicy, stosunku wysokości znaków i tablicy oraz miejsca, w którym rozpoznaje się numer rejestracyjny, próbowano dokonać segmentacji znaków. Taki sposób jest łatwy w implementacji, natomiast jest ściśle powiązany z konkretnym formatem tablic i nie nadaje się do użycia dla różnych rodzajów.

W [44] zaprezentowano technikę segmentacji opartą o algorytm k -średnich [18]. Główna idea wyglądała następująco:

1. Wybranie w obrazie k środków losowo,
2. Przypisanie każdego z pikseli w obrazie do jednego z klastrów na podstawie odległości (np. euklidesowej),
3. Przekalkulowanie środków klastrów poprzez uśrednienie wszystkich pikseli w klastrze,
4. Powtórzenie kroku 2 i 3, do momentu kiedy piksele przestaną zmieniać klastry.

Dokładność takiego rozwiązania jest mocno skorelowana z odpowiednim wyborem k oraz początkowym zbiorem klastrów. W celu właściwego wyboru k , autorzy użyli algorytmu SIFT (ang. *Scale Invariant Feature Transform — SIFT*) [22]. Algorytm ten jest odporny na skalę i rotację obrazu. Zaproponowana technika uzyskała wysoką dokładność segmentacji na poziomie 98.82%.

1.3 Przegląd istniejących metod rozpoznawania tablic rejestracyjnych

W wyniku operacji segmentowania znaków, otrzymywane są obrazy z oddzielonymi znakami od tła. Kolejną częścią automatycznego systemu rozpoznawania tablic rejestracyjnych jest odczytanie numeru identyfikującego dany pojazd. Jest to ostatni etap w systemie. Rezultat tego działania przeważnie wysyłany jest już do innego modułu lub systemu odpowiedzialnego za logikę biznesową danego rozwiązania.

Optyczne rozpoznawanie znaków (ang. *Optical Character Recognition — OCR*) jest obszerną dziedziną algorytmów przetwarzających obraz na tekst. Systemy te potrafią zarówno odczytywać tekst drukowany jak i pisany odręcznie. Zeskanowanie dokumentów i przetworzenie ich na tekst jest już dzisiaj dość powszechną operacją. Dzieje się tak, ponieważ proces ten zachodzi w kontrolowanych warunkach. W dynamicznym systemie rozpoznawania numerów rejestracyjnych prawidłowe odczytanie znaków jest znacznie trudniejsze. Jest to zagadnienie pojawiające się w wielu pracach naukowych. Poniżej przedstawiono najczęściej stosowane metody do realizacji tego zadania.

1.3.1 Metody oparte na sieciach neuronowych

W pracy [4] autorzy przedstawili system OCR oparty o wykorzystanie probabilistycznej sieci neuronowej (ang. *Probabilistic Neural Network — PNN*). Sieć PNN jest rozwiązaniem pamięciowym identyfikacji obiektów [29]. Oznacza to, że sieć zapamiętuje pewną liczbę obrazów ze zbioru uczącego reprezentujących identyfikowany obiekt. Klasyfikacja

obiektów przy pomocy sieci PNN realizowana jest za pomocą funkcji decyzyjnej [38]:

$$f(x) = \frac{1}{n} \sum_{i=1}^n \exp \left[- \left(\frac{d(x, x_i)}{p\rho} \right)^2 \right], \quad (1.14)$$

gdzie:

- d jest odległością euklidesową pomiędzy wektorami identyfikowanego obiektu x o i-tego przykładu uczącego x_i ,
 - n jest liczbą wektorów uczących danej kategorii,
 - p jest rozmiarem wektora x oraz wszystkich wektorów x_i ,
 - ρ jest parametrem kształtu.

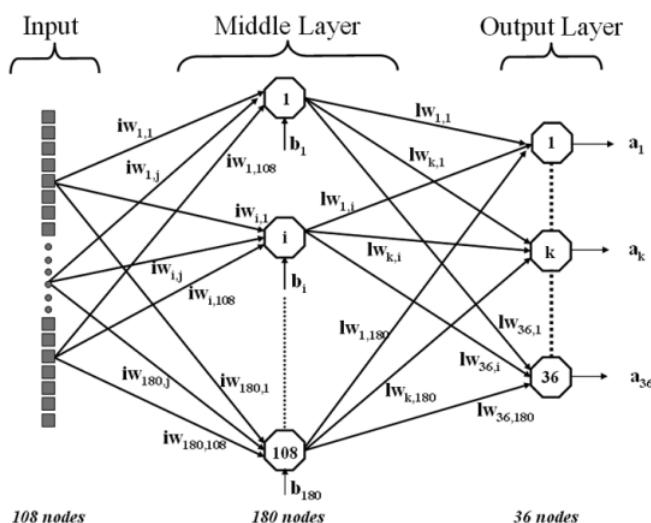
Ostateczna decyzja o przyporządkowaniu obiektu reprezentowanego przez wektor x do kategorii j-tej opiera się na relacji przedstawionej na wzorze (1.15) [29]

$$h_j c_j f_j(x) > h_k c_k f_k(x) \quad \forall j \neq k, \quad (1.15)$$

gdzie:

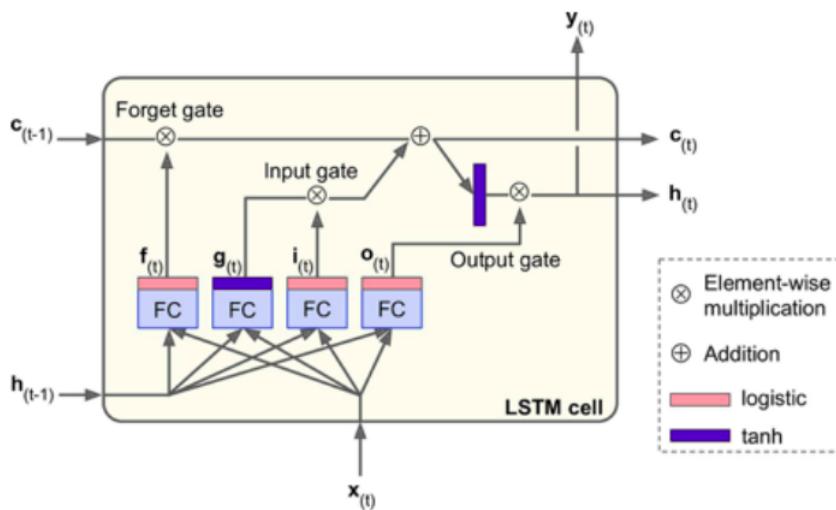
- h_j oznacza prawdopodobieństwo a priori przynależności obiektu do j-tej klasy,
 - c_j oznacza koszt błędnej klasyfikacji obiektu należącego do j-tej kategorii.

W sieci PNN wyróżnia się trzy warstwy. Pomiędzy warstwami wejściowym i wyjściowym, znajduje się warstwa radialna, zawierająca neurony radialne. Każdy neuron można określić jako funkcję radialną z centrum nad swoim przypadkiem uczącym [29]. Warstwa zawiera liczbę neuronów równej liczbie wzorców w ciągu uczącym. Na wyjściu sygnały pochodzące z neuronów są sumowane. Sygnał wyjściowy jest normalizowany, w taki sposób, iż suma na wszystkich wyjściach sieci jest równa 1. Wartość pojedynczego wyjścia odpowiada prawdopodobieństwu kategorii przypisanej do tego wyjścia [3]. Rysunek 1.10 przedstawia architekturę zaproponowanej sieci PNN w [4]. Autorzy zaraportowali dokładność rozpoznawania znaków na poziomie 89.1%



Rysunek 1.10: Architektura sieci PNN (źródło: [4]).

Innym rodzajem sieci wykorzystywanych w celu rozpoznawania znaków jest sieć LSTM (ang. *Long Short-Term Memory* — LSTM) [14]. Jest to szczególny rodzaj rekurencyjnych sieci neuronowych (ang. *Recursive Neural Networks* — RNNs). Bazują one na wykorzystaniu informacji sekwencyjnych. W tradycyjnych sieciach wszystkie wejścia i wyjścia są od siebie niezależne. W sieciach rekurencyjnych dane wyjściowe zależą od poprzednich obliczeń. Przykładowo chcąc przewidzieć kolejny wyraz w zdaniu, istotne jest wiedzieć, które wyrazy pojawiły się do tej pory [51]. Sieci RNN posiadają zdolność „pamiętania” wcześniejszych obliczonych informacji [51]. Natomiast im upłynął dłuższy czas od przetworzenia pewnej informacji, komórki sieci stopniowo „zapominają” tą informację. Sieci LSTM zostały zaprojektowane do rozwiązywania problemu zaniku pierwszych wejść. Rysunek 1.11 przedstawia architekturę komórki sieci LSTM.



Rysunek 1.11: Architektura komórki sieci LSTM (źródło: [1]).

Kluczową ideą sieci jest jej możliwość nauczenia się co warto przechowywać w stanie długoterminowym, co z niego usunąć i co z niego czytać [51]. Stan komórki sieci LSTM jest podzielony na dwa wektory $h^{(t)}$ i $c^{(t)}$, odpowiadające odpowiednio stanowi krótko- i długoterminowemu. Literą t oznaczono daną chwilę czasu. Wyróżnia się trzy bramki: wejściową, wyjściową oraz „zapomnij”. Stan długoterminowy $c^{(t-1)}$ przemierza sieć od lewej do prawej, porzucając niektóre wspomnienia w bramce „zapomnij”, a następnie dodając nowe. Wspomnienia dodawane są przez bramkę wejściową. Po operacji dodawania, stan długoterminowy jest kopowany i przechodzi przez funkcję tanh. Ostatecznie wynik jest filtrowany przez bramkę wyjściową. Stan krótkoterminowy $h^{(t)}$ powstaje jako stan równy wyjściu dla kroku czasowego $y^{(t)}$. Wektor wejściowy $x^{(t)}$ oraz poprzedni stan krótkoterminowy $h^{(t-1)}$ przekazywane są do warstwy głównej oraz trzech warstw kontrolerów bramkowych. Warstwy te są w pełni połączone. Kontrolery bramkowe używają logistycznej funkcji aktywacji. Na ich wyjściu wartości są z zakresu od 0 do 1.

W zastosowaniach OCR, ten rodzaj sieci pracuje na poziomie znaku. Oznacza to, że sieci LSTM mogą być wykorzystywane do rozpoznawania tekstu niezależnie od użytego języka. Na przestrzeni ostatnich lat pojawiło się wiele opracowań wykorzystujących sieci LSTM do realizacji systemów OCR [16, 23]. Warto również wspomnieć, że jedna z

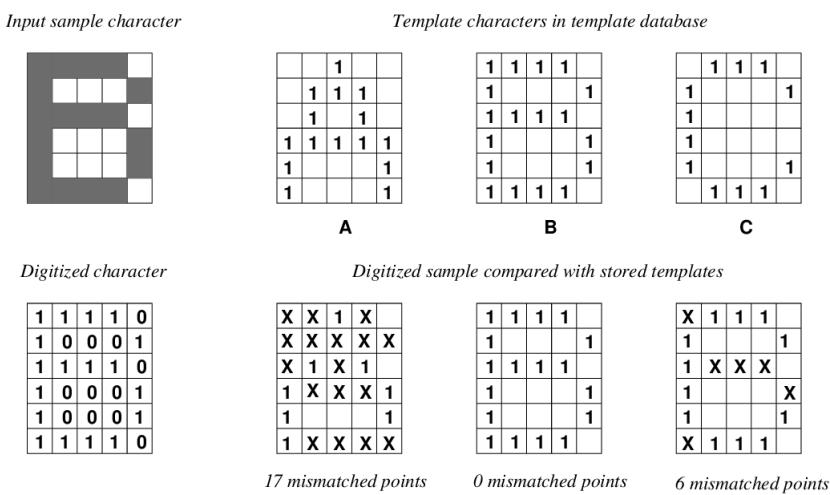
najpopularniejszych bibliotek programistycznych Tesseract [56] od wersji 4 wykorzystuje sieci LSTM.

1.3.2 Metody oparte na wzorcach

Jedną z popularnych technik klasyfikacji znaków jest metoda oparta na wzorcach (ang. *template matching*) [30, 33]. Bazuje ona na założeniu, że czcionka oraz rozmiar znaków na tablicy jest znany. Dopasowanie wzorca odbywa się z reguły na zbinaryzowanym obrazie. Dla każdego możliwego znaku, tworzony jest odpowiedni wzorzec. Podczas procesu rozpoznawania, każdy wzorzec badany jest pod kątem podobieństwa do rozpatrywanego znaku. Do rozpatrzenia prawdopodobieństwa lub obliczenia odległości używa się najczęściej następujących funkcji [37]:

- odległość Mahalanobisa,
- indeks Jaccarda,
- metryka Hausdorffa,
- odległość Hamminga,
- korelacja wzajemna.

Rysunek 1.12 przedstawia schemat opisywanej techniki na przykładzie litery *B*.



Rysunek 1.12: Rozpoznanie litery B za pomocą wzorców (źródło: [19]).

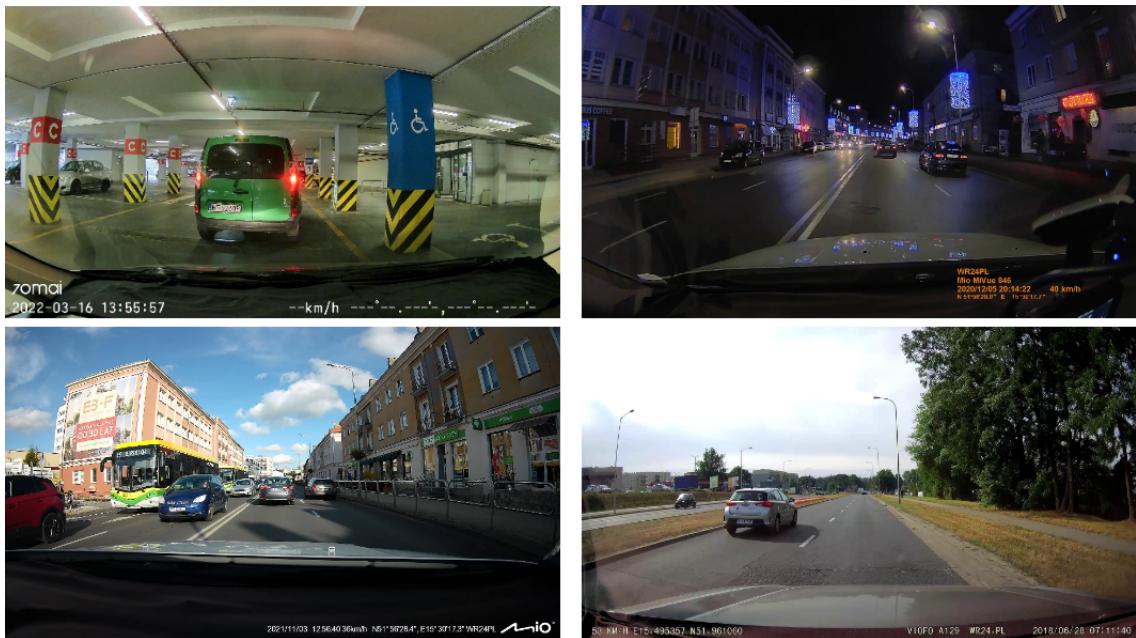
Cyfrową reprezentacją znaku jest tablica, gdzie 1 to punkt wchodzący w skład znaku, a 0 to tło. Tablica musi mieć ten sam rozmiar co tablica wzorca, więc obrazy są najczęściej skalowane. Dla litery *A* wykryto 17, a dla litery *C* 6 niepoprawnych punktów. Metoda ta jest stosunkowo prosta w implementacji. Problematyczne stają się różne czcionki. Aby umożliwić rozpoznawanie wielu formatów oraz zniwelować rotację znaków, dodaje się nowe wzorce. To z kolei zwiększa złożoność obliczeniową systemu i zapotrzebowanie na pamięć operacyjną [37].

2. Zebranie materiału uczącego

Niewątpliwą przyczyną rozwoju dziedziny uczenia maszynowego na przestrzeni ostatnich lat jest wzrost wydajności komputerów będących w stanie zbierać i przetwarzać ogromne ilości danych. Uczenie maszynowe jest obszarem sztucznej inteligencji poświęconej algorytmom, które poprawiają się automatycznie poprzez doświadczenie [2]. Tworząc system automatycznego rozpoznawania tablic rejestracyjnych niezbędne jest przygotowanie odpowiedniego zbioru danych. Jedną z kluczowych kwestii przed rozpoczęciem kolekcjonowania informacji, jest ustalenie wielkość zbioru. Najbardziej powszechnie stosowanym podejściem jest *zasada 10 razy*. W tym kontekście oznacza ona, że aby ilość danych była wystarczająca, zbiór danych wejściowych powinien być 10 razy większy od liczby parametrów w opracowywanym modelu. Dla modelu o 1000 parametrach, zbiór powinien zawierać 10 tysięcy próbek. Zwykle jest to jeden z bardziej czasochłonnych etapów podczas tworzenia modeli uczenia maszynowego.

Do przygotowania danych uczących wykorzystano nagrania z rejestratorów wideo zamontowanych w samochodzie. Zgromadzony materiał został zarejestrowany za pomocą wielu rejestratorów, stąd rozdzielcość oraz liczba klatek na sekundę różni się pomiędzy plikami wideo. Wykorzystane obrazy zostały zebrane samodzielnie przez autora. Pozyskane zdjęcia pochodzą zarówno z przejazdów nocnych jak i za dnia. Udało się również zebrać materiał ze zróżnicowanymi warunkami atmosferycznymi (tj. jazda w deszczu lub jazda w bezchmurny dzień). Rozdzielcość obrazu zawierała się w zakresie 1920×1080 do 3840×2160 pikseli. Liczba klatek na sekundę dla części kamer wynosiła 60 klatek na sekundę, a dla pozostały części 30 klatek na sekundę. Algorytm przygotowujący dane uczące potrzebował na wejściu wyekstrahowane cechy z konkretnych obiektów. Aby to osiągnąć, należało przygotować zdjęcia, na których następnie zaznaczone zostaną obiekty.

W pierwszej kolejności przygotowano zdjęcia na podstawie plików wideo. Każdy plik wideo miał 60 sekund długości. Podzielono go w ten sposób, aby z każdej sekundy nagrania powstały 3 zdjęcia. Dla plików z 60 klatkami na sekundę, próbkiwowano co 20 klatek, natomiast dla pozostałych co 10. Przykładowe obrazy pochodzące z nagrania z rejestratora przedstawia Rysunek 2.1. W ten sposób pozyskano 10985 zdjęć. Jak można zauważyć, większość rejestratorów dodaje metadane takie jak prędkość pojazdu, data i godzina. Dla algorytmu rozpoznawania tablic rejestracyjnych może to stanowić trudność, ponieważ wyświetlany tekst może być klasyfikowany jako tablica.



Rysunek 2.1: Pozyskane zdjęcia z wideorejestratora (źródło: opracowanie własne).

Kolejnym krokiem niezbędnym przed rozpoczęciem procesu uczenia było nadanie etykiet obiektom na wyeksportowanych zdjęciach. W tym celu opracowano skrypt służący do ręcznego oznaczania tablic rejestracyjnych. Skrypt powstał na podstawie narzędzia [48]. Jego fragment w postaci pseudokodu przedstawiono na poniższym schemacie.

Algorytm 3 Procedura zapisująca współrzędne pozytywnych próbek.

```

1: procedure OBJECTMARKERONCLICK((event, x, y))
2:   if event == kliknięcie lewego przycisku myszy then
3:     if Liczba kliknięć jest nieparzysta then
4:       Przypisz współrzędne (x, y) kliknięcia jako początek obszaru próbki.
5:     else
6:       Przypisz współrzędne (x, y) kliknięcia jako koniec obszaru próbki.
7:       ZwiększMiejsce liczbę obiektów w obrazie.
8:       Przeskaluj współrzędne do stałej wartości.
9:       Dodaj do listy pozytywów zaznaczony fragment.
10:    end if
11:   end if
12: end procedure
```

Wszystkie zdjęcia wyświetlane po kolejno na ekranie. Zdarzenie kliknięcia myszką na obrazie przechwytywano w programie. Pierwsze kliknięcie oznaczało współrzędne początku obszaru pozytywnej próbki. Drugie kliknięcie oznaczało współrzędne końca rozpatrywanego obszaru. Jak wspomniano wcześniej, obrazy posiadały różne rozdzielczości, często przewyższające rozdzielcość monitora. Aby zdjęcia wyświetlały się poprawnie, skalowano je do rozdzielcości ekranu, na którym przeprowadzono operację nadawania

etykiet. Pozyskane koordynaty zapisywano do globalnej tablicy. Na koniec procesu dane zapisano do pliku tekstowego, z którego odczytywano dane podczas etapu uczenia. Dane w pliku zapisano w formacie *ścieżka do pliku, liczba obiektów w pliku, kolejno współrzędne każdego z prostokątów*. W Tabeli 2.1 przedstawiono cechy charakterystyczne przygotowanego zbioru.

Tabela 2.1: Parametry opracowanego zbioru.

Parametr	Wartość
Liczba zdjęć	10985
Format zdjęć	JPG
Liczba zdjęć zawierających tablice	5248
Ogółem liczba tablic	10301
Zakres liczby tablic na jednym zdjęciu	0–6
Średnia wysokość próbki	38px
Średnia szerokość próbki	91px
Maksymalne rozmiary próbki	378×136px
Minimalne rozmiary próbki	15×11px
Rozdzielcości zdjęć	1920×1080, 2560×1440, 3840×2160
Liczba klatek na sekundę	30 kl/s, 60 kl/s

3. Opracowany algorytm

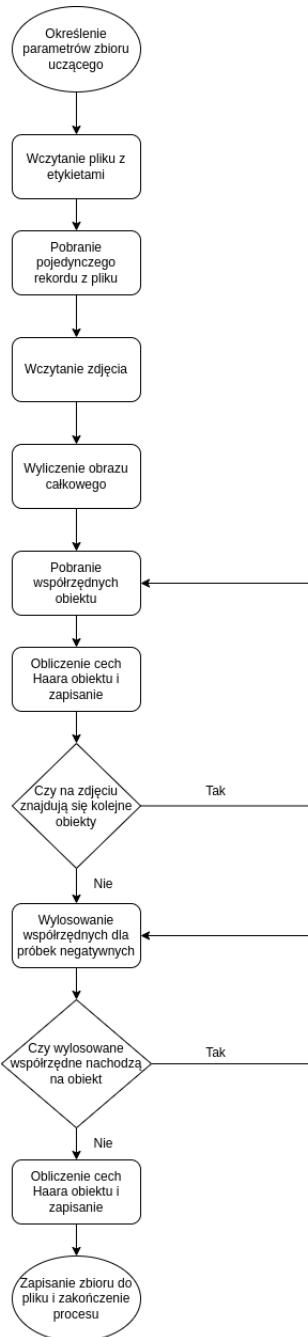
W przedstawionej pracy zrealizowano program rozpoznający tablice rejestracyjne na sekwencjach wideo pochodzących z kamery samochodowej. Do detekcji użyto skanowania obrazu oknem przesuwnym i klasyfikowanie na podstawie cech Haara za pomocą algorytmu RealBoost. Jako słaby klasyfikator wykorzystano algorytm koszykowania wartości funkcji logit. Rozpoznane fragmenty obrazu zawierające tablice poddano operacjom morfologicznym. Z przetworzonych obrazów segmentowano znaki tablicy rejestracyjnej. Do rozpoznania znaków użyto biblioteki Tesseract opartej o rekurencyjne sieci neuronowe LSTM.

3.1 Proces uczenia klasyfikatora

Proces uczenia klasyfikatora jest kluczowym etapem budowy modelu uczenia maszynowego. Jeżeli na tym etapie dojdzie do błędu, będzie to rzutować na działanie całej aplikacji. Częstym zjawiskiem w uczeniu maszynowym jest przeuczenie (ang. *overfitting*). Polega ono na wykrywaniu pozornych prawidłowości w dużej ilości danych, gdzie prawdziwe prawidłowości są prostsze lub słabsze, lub są maskowane przez błędy, lub są całkowicie nieistniejące [58]. Innym niepożądany przypadkiem błędного wyuczenia jest niedouczenie (ang. *underfitting*). Wynika ono najczęściej z zastosowania zbyt uproszczonego modelu lub niewystarczającej liczby próbek uczących [58]. Mając powyższe na uwadze, należy przyłożyć staranną uwagę do przygotowania odpowiedniego zbioru uczącego. Nieprawidłowości na tym etapie będą bardzo trudno do wyeliminowania w późniejszych częściach systemu.

3.1.1 Przygotowanie danych uczących

W rozdziale 2 przedstawiono opracowany zbiór zdjęć z kamery samochodowej zawierający poruszające się pojazdy wraz z ich tablicami rejestracyjnymi. Klasyfikatory działają jednak w inny sposób niż ludzki mózg i wymagają innych wartości, na których będą mogły podejmować decyzje. W niniejszej pracy zdecydowano się wykorzystać cechy Haara jako cechy reprezentujące poszukiwane obiekty. W celu wyuczenia klasyfikatora, należało opracowany wcześniej zbiór przetworzyć, aby każda negatywna i pozytywna próbka miała swoją reprezentację liczbową. Na Rysunku 3.1 przedstawiono schemat blokowy procesu przygotowania danych uczących dla klasyfikatora.



Rysunek 3.1: Schemat blokowy przygotowania danych uczących dla klasyfikatora (źródło: opracowanie własne).

Pierwszą czynnością jest określenie parametrów zbioru uczącego. Przed rozpoczęciem uczenia należy określić, ile cech ma być liczonych z jednej próbki. Każdy szablon może być odpowiednią ilością razy skalowany wzdłuż każdego z kierunków. Parametr ten oznacza się literą s . Parametr $p = 1, 2, \dots$ przekłada się na rozmiary regularnej siatki $(2p-1) \times (2p-1)$ punktów zaczepienia cech [50]. Na podstawie powyższych parametrów określa się liczbę cech niezbędnych do wyliczenia w trakcie przetwarzania jednego okna

wg wzoru (3.1)

$$n(s, p) = 6s^2(2p - 1)^2. \quad (3.1)$$

Przykładowo, dla $s = 3$ i $p = 4$ liczba niezbędnych do wyliczenia cech jest równa 2205. Dla $s = p = 5$ liczba ta rośnie do 10125. Im większa jest liczba cech tym dokładność detekcji powinna rosnąć. Jednak wraz ze wzrostem liczby cech, rośnie również czas wykonania skryptu ze względu na większą liczbę niezbędnych do wykonania obliczeń. Innym wejściowym parametrem jest ustalenie stosunku próbek pozytywnych do próbek negatywnych.

Po ustaleniu wejściowych parametrów, wczytywany jest plik tekstowy ze współrzędnymi tablic dla konkretnych plików. Dla każdego rekordu w zbiorze, wczytywane jest odpowiednie zdjęcie. Następnie algorytm iteruje po wszystkich pozytywnych próbkach znajdujących się w rozpatrywanym pliku. Dla każdej próbki wyliczane są cechy Haara za pomocą wygenerowanych wcześniej szablonów.

W opisywanym programie użyto 5 szablonów cech Haara. Szablony są skalowane odpowiednim skokiem zależnym od s . Poniżej zamieszczono kod procedury generującej współrzędne szablonu dla każdej cechy 3.1.

```

1 def haar_coords(s, p, indexes):
2     coords = []
3     f_jump = (FEATURE_MAX - FEATURE_MIN) / (s - 1)
4     for t, s_j, s_k, p_j, p_k in indexes:
5         f_h = FEATURE_MIN + s_j * f_jump
6         f_w = FEATURE_MIN + s_k * f_jump
7         p_jump_h = (1.0 - f_h) / (2 * p - 2)
8         p_jump_w = (1.0 - f_w) / (2 * p - 2)
9         pos_j = 0.5 + p_j * p_jump_h - 0.5 * f_h
10        pos_k = 0.5 + p_k * p_jump_w - 0.5 * f_w
11        single_coords = [np.array([pos_j, pos_k, f_h,
12                                     f_w])] # whole rectangle for single feature
13        for white in HAAR_TEMPLATES[t]:
14            white_coords = np.array([pos_j, pos_k, 0.0,
15                                      0.0]) + white * np.array([f_h, f_w, f_h,
16                                      f_w])
17            single_coords.append(white_coords)
18        coords.append(np.array(single_coords))
19    return np.array(coords, dtype=object)

```

Algorytm 3.1: Procedura generująca szablony cech Haara dla konkretnych współrzędnych.

Na Rysunku 3.2 pokazano przykładowe szablony podczas procesu uczenia.



Rysunek 3.2: Ekstrakcja cech za pomocą obrazów całkowych (źródło: opracowanie własne).

Po przeprocesowaniu wszystkich pozytywnych próbek, algorytm generuje negatywy. Negatywne próbki są generowane losowo. Losowana jest szerokość okna jako wartość w przedziale od 1 do 10 procent szerokości całego obrazu. Wysokość okna jest zależna od szerokości. Przyjęto, że wysokość okna powinna być równa $\frac{1}{3}$, $\frac{1}{4}$ lub $\frac{1}{5}$ szerokości. Następnie dane są zapisywane do pliku. W ten sposób przygotowane dane w następnym etapie posłużą do wyuczenia klasyfikatora.

3.1.2 Uczenie klasyfikatora

W niniejszej pracy zaimplementowano i wykorzystano algorytm wzmacniający RealBoost oparty o koszykowanie wartości funkcji logit. Zasadę działania algorytmu RealBoost opisano szerzej w rozdziale 1.1.4. Na wejściu metody uczącej, funkcja przyjmuje cechy uczące oraz ich etykiety. Dla każdej cechy obliczane są wartości maksymalne i minimalne. Ustalone są one poprzez posortowanie wartości konkretnej cechy ze wszystkich próbek ze zbioru uczącego. Algorytm dodaje odpowiedni margines, aby odrzucić skrajne wyniki. Współczynnik ten ustalono na poziomie 0.05. Oznacza to, że dla zbioru o wielkości 100 próbek, odrzucone zostanie 5 pierwszych i 5 ostatnich wartości cech do wyznaczenia wartości brzegowych. Następnie przypisywane są indeksy koszyków do poszczególnych cech. Kolejnym krokiem jest przygotowanie indeksów, które przechowują informację o tym czy cecha dla konkretnego koszyka powiązana jest z próbką pozytywną czy negatywną. Po przygotowaniu powyższych danych, algorytm przechodzi do wybrania najważniejszych cech z punktu widzenia klasyfikatora. Obliczane jest to poprzez iterację po wszystkich cechach i określeniu cechy dla każdego klasyfikatora o najmniejszym błędzie. Wyznaczane jest to za pomocą funkcji logit, która jest wyznaczana dla każdego z koszyków z osobna. Na koniec dochodzi do reważenia wag. Czynność jest powtarza dla każdego słabego klasyfikatora. Algorytm 3.2 ukazuje fragment metody uczącej odpowiedzialny za wyliczenie odpowiedzi dla każdego ze słabych klasyfikatorów i wyznaczenie indeksów najlepszych cech.

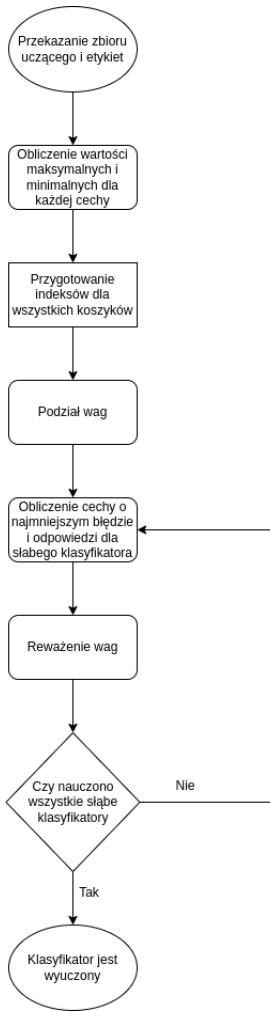
```

1 | w = np.ones(m) / m
2 | for t in range(self.T_):
3 |     j_best = None
4 |     logits_best = None
5 |     err_exp_best = np.inf
6 |     for j in range(n):
7 |         logits = np.zeros(self.B_)
8 |         for b in range(self.B_):
9 |             W_positive = w[indexer_positive[j, b]].sum()
10 |            W_negative = w[indexer_negative[j, b]].sum()
11 |            logits[b] = self.logit(W_positive,
12 |                                      W_negative)
12 |            err_exp = np.sum(w * np.exp(-yy *
13 |                                      logits[X_binned[:, j]]))
13 |            if err_exp < err_exp_best:
14 |                err_exp_best = err_exp
15 |                logits_best = logits
16 |                j_best = j
17 |            self.feature_indexes_[t] = j_best
18 |            self.logits_[t] = logits_best
19 |            w = w * np.exp(-yy * logits_best[X_binned[:, j_best]])
20 |            w /= err_exp_best

```

Algorytm 3.2: Procedura ucząca klasyfikator RealBoostBins.

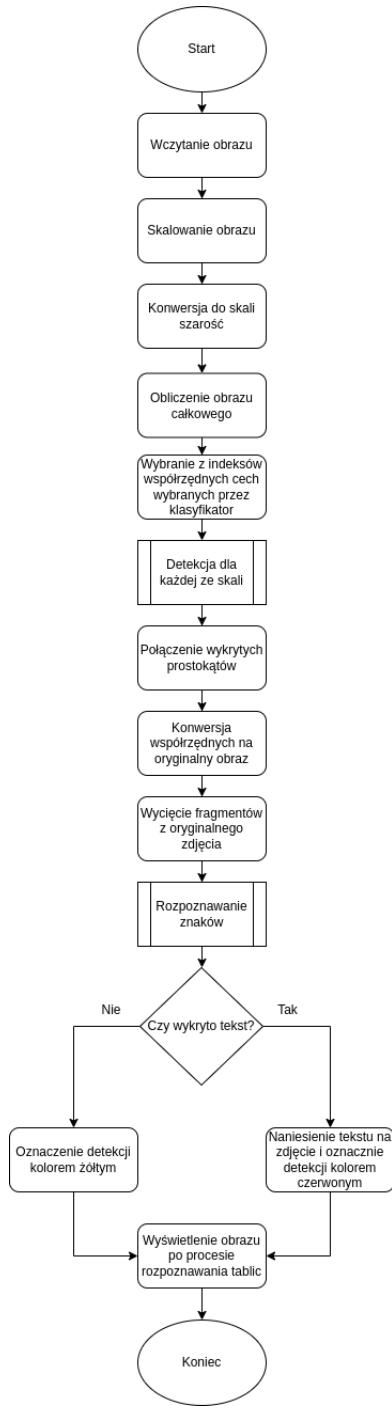
Na Rysunku 3.3 przedstawiono schemat blokowy uczenia klasyfikatora.



Rysunek 3.3: Schemat blokowy funkcji uczącej klasyfikatora RealBoostBins (źródło: opracowanie własne).

3.2 Schemat algorytmu

Główna część programu można podzielić na dwa osobne moduły. Pierwszy z nich odpowiedzialny jest za detekcję tablic na obrazie wejściowym. Drugi moduł odpowiada za segmentację i rozpoznawanie znaków. Przed rozpoczęciem detekcji, obraz jest skalowany. W opracowanym algorytmie, ustalono wysokość na poziomie 480px. Szerokość obrazu była proporcjonalnie skalowana. Następnie obraz konwertowano do skali szarości. Przed rozpoczęciem detekcji oknem przesuwnym, obliczano obraz całkowy. Szablony cech Haara ograniczono tylko do tych odpowiadających cechom wybranym przez słabe klasyfikatory na etapie uczenia klasyfikatora. Na Rysunku 3.4 przedstawiono schemat blokowy algorytmu rozpoznawania tablic rejestracyjnych.



Rysunek 3.4: Schemat blokowy algorytmu rozpoznawania tablic rejestracyjnych (źródło: opracowanie własne).

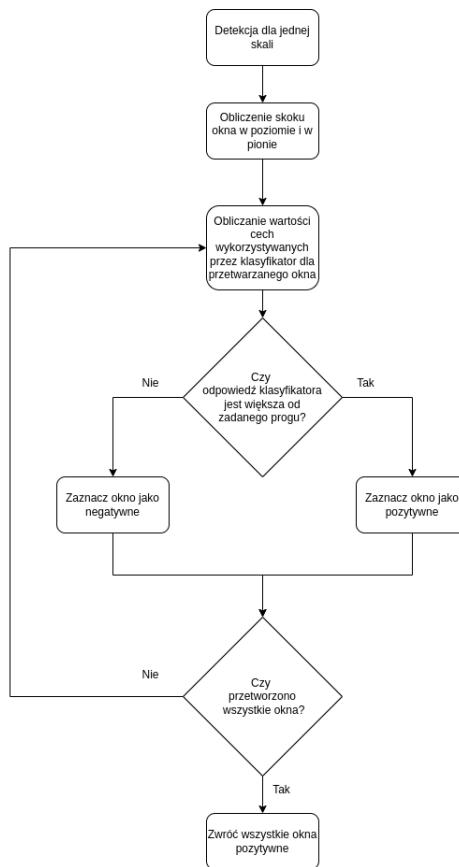
3.2.1 Algorytm detekcji

Algorytm detekcji odpowiada za zlokalizowanie obszarów potencjalnie zawierających tablice rejestracyjne. Mając na uwadze fakt, że tablice mogą mieć różne rozmiary na zdjęciu, dla procedury skanującej oknem przesuwnym należało zastosować kilka skal

okna. Okno przesuwano w obrazie odpowiednim skokiem w poziomie oraz w pionie. Poziomy skok d_w obliczano na podstawie wzoru (3.2)

$$d_w = \text{round}(w * \lambda), \quad (3.2)$$

gdzie w jest szerokością okna, a λ oznacza współczynnik przemieszczania okna. Skok pionowy wyznaczano w analogiczny sposób. Dla każdego okna obliczano cechy Haara dla indeksów wyznaczonych podczas procesu uczenia klasyfikatora. Następnie obliczano odpowiedź klasyfikatora. Jeśli była ona większa od podanego progu, okno oznaczano jako pozytywne. Na Rysunku 3.5 przedstawiono schemat blokowy algorytmu detekcji tablic rejestracyjnych.



Rysunek 3.5: Schemat blokowy algorytmu detekcji tablic rejestracyjnych (źródło: opracowanie własne).

Po wykonaniu procedury skanowania oknem przesuwnym, rozpoczęto przetwarzanie okien oznaczonych jako pozytywne. Z uwagi na fakt, że w otoczeniu tablicy rejestracyjnej wiele okien mogło zostać oznaczonych jako pozytywne, należało te okna połączyć. Dzięki temu moduł rozpoznawania znaków otrzyma znacznie mniejszą ilość okien. W kolejnym kroku współrzędne połączonych okien konwertowano do współrzędnych w oryginalnym obrazie. Taka operacja ma na celu przekazanie do modułu OCR fragmentu obrazu w jakości równej obrazowi przed procedurą skalowania. Rysunek 3.6 przedstawia porównanie procedury detekcji bez oraz z zastosowaniem techniką łączenia okien. Do połączenia okien zastosowano algorytm *non-max suppression*.



Rysunek 3.6: Porównanie detekcji tablicy rejestracyjnej bez i z zastosowaniem algorytmu łączenia okien (źródło: opracowanie własne).

3.2.2 Algorytm segmentacji i rozpoznawania znaków

Moduł odpowiedzialny za rozpoznawanie znaków otrzymuje na wejściu wycięty fragment obrazu w oryginalnej jakości. Następnie przed rozpoczęciem segmentacji znaków, obraz jest skalowany do wysokości 300px. W kolejnym kroku, obraz jest konwertowany do modelu barw w skali szarości. W celu segmentacji znaków, zaproponowano podejście łączące dwie techniki. Zdecydowano się na takie rozwiązanie, ponieważ zauważono, że wzajemnie się one uzupełniają i ich połączenie daje lepsze wyniki. Pierwsza z technik polegała na binaryzacji obrazu z wykorzystaniem metody Otsu. Na tak przygotowanym obrazie wykryto kontury korzystając z metody z biblioteki OpenCV *findContours*. Druga technika wykorzystywała maskę binarną w przestrzeni barw HSV. Następnie przetworzony obraz poddawano operacjom morfologicznym. Wynikowy obraz przekazano do metody wykrywającej kontury.

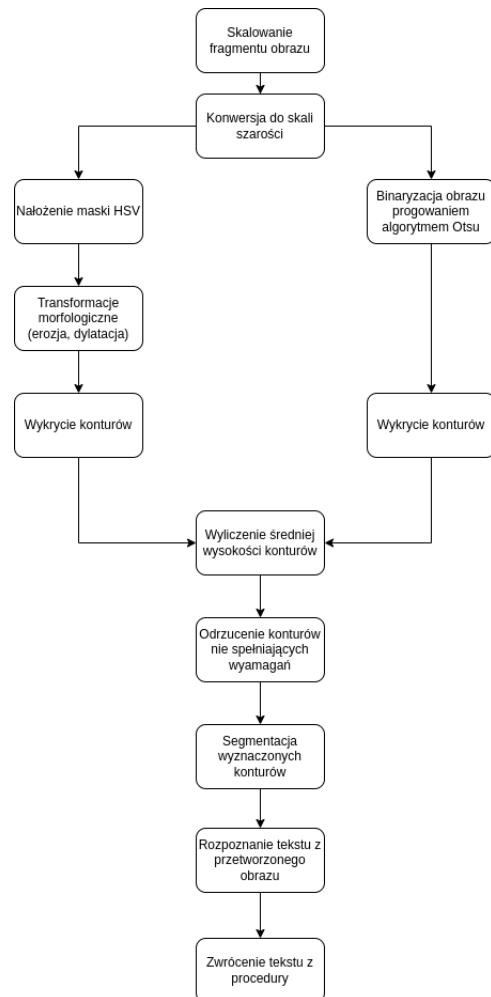
Metoda *findContours* przyjmuje parametr *mode*, który odpowiada za sposób selekcji konturów. Procedura może zwracać np. tylko kontury zewnętrzne lub wewnętrzne lub zwracać strukturę hierarchiczną [54]. W opisywanym rozwiązaniu wybrano opcję *RETR_TREE*, która pobiera wszystkie kontury i rekonstruuje pełną hierarchię zagnieżdzonych konturów. W *findContours* możliwa jest również do wybrania metoda aproksymacji konturów. Dla wartości *CHAIN_APPROX_NONE* zbierane są wszystkie wartości konturów. Dla wartości *CHAIN_APPROX_SIMPLE* algorytm kompresuje poziomie, pionowe oraz diagonalne segmenty i pozostawia tylko punkty końcowe. Zdecydowano się na drugą metodę aproksymacji.

Dla obu podejść zbiory wykrytych konturów połączono w całość. Wykryte kontury należało przefiltrować, w celu wybrania tylko tych odpowiadającym znakom na tablicy rejestracyjnej. Opracowano procedurą z następującymi krokami:

1. pobranie prostokąta otaczającego dany kontur,
2. odrzucenie konturu, jeśli wysokość prostokąta jest większa od wysokości obrazu pomniejszonej o stałą wartość
3. obliczenie powierzchni prostokąta,
4. odrzucenie konturu, jeśli powierzchnia jest mniejsza od stałej wartości,
5. odrzucenie konturu, jeśli stosunek boków prostokąta jest spoza przyjętego zakresu.

Na podstawie powyższej procedury odrzucono część wykrytych wcześniej konturów. Korzystając z założenia, że wszystkie znaki na tablicy rejestracyjnej są takiej samej wysokości, odrzucono z powstałego zbioru kontury, których wysokość najbardziej odbiegała od średniej wysokości całego zbioru. Ostateczny zbiór konturów posłużył do wycięcia znaków ze zbinaryzowanego obrazu. W ten sposób przeprowadzono segmentację znaków. Nowy

obraz, z oddzielonymi znakami od tła, przekazano do biblioteki Tesseract. Skorzystano z metody *image_to_string*, która zwraca tekst na podstawie obrazu wejściowego. Na Rysunku 3.7 przedstawiono schemat blokowy algorytmu rozpoznawania znaków.



Rysunek 3.7: Schemat blokowy algorytmu rozpoznawania tablic (źródło: opracowanie własne).

Na Rysunku 3.8 przedstawiono przykładowy proces segmentacji znaków. Pierwszy obraz jest obrazem wejściowym, który trafił do algorytmu z detektora tablic rejestracyjnych. Ostatni obraz jest wynikiem segmentacji. Obraz środkowy jest obrazem wejściowym, na który nałożono wykryte pozycje znaków (kolor niebieski). Kolorem różowym oznaczono kontury, które zostały wykryte, ale zostały odrzucone z powodu braku spełnienia kryteriów geometrycznych. Założono, że kontur pojedynczego znaku musi być prostokątem o dłuższych bokach zorientowanych w pionie. Kolorem żółtym oznaczono kontury, które zostały odrzucone, z powodu ich zbyt dużego odchylenia od średniej wysokości konturów.



Rysunek 3.8: Proces segmentacji znaków (źródło: opracowanie własne).

3.3 Biblioteki użyte w programie

Środowisko Python jest niezwykle popularne m.in. ze względu na mnogość dostępnych gotowych bibliotek. Jednym z celów pracy było zaimplementowanie algorytmu uczącego i klasyfikującego. Cel ten udało się zrealizować, natomiast nie byłoby to możliwe bez użycia gotowych rozwiązań do elementarnych operacji t.j. listowanie plików w katalogach, odczyt i zapis zdjęć, pobieranie pojedynczych klatek z filmu wideo czy operacje na tablicach. Poniżej wymieniono i opisano najważniejsze oraz najczęściej używane biblioteki w opisywanej pracy.

3.3.1 OpenCV

OpenCV jest biblioteką o otwartym źródle (ang. *open source*) [52, 53]. Do jej głównych zastosowań należą przetwarzanie obrazów oraz uczenie maszynowe. W przedstawionym programie wykorzystano najnowszą dostępną wersję na moment pisania pracy 4.6.0. Została zaprojektowana, aby zapewnić ustandaryzowaną infrastrukturę dla aplikacji widzenia komputerowego. Oprogramowanie dystrybuowane jest na licencji BSD. Pierwsza wersja została opracowana w roku 1999. Oryginalnie powstała w języku C++, natomiast istnieją biblioteki pozwalające używać jej w innych językach programowania. Bibliotekę można podzielić na kilka głównych modułów:

- przetwarzanie obrazów - moduł zawiera zestaw metod do przeprowadzania takich operacji jak filtrowanie, przekształcenia geometryczne, zmiana przestrzeni kolorów, histogramy,
- przetwarzanie wideo - zestaw metod, które pozwalają na m. in. usuwanie tła, śledzenie obiektów, wykrywanie ruchu,
- operacje wejścia/wyjścia na wideo - wyciąganie poszczególnych klatek z wideo, kodowanie, zapis i odczyt wideo,
- HighGUI - moduł służący do wizualizacji wyników, wyświetlania okien, zaznaczania ROI (ang. *Region of interest*).

Poniżej wymieniono użyte w stworzonym programie funkcje wraz z krótkim opisem ich zastosowania:

- *videoCapture* - przechwytywanie obrazów z pliku wideo,
- *cvtColor* - zmiana przestrzeni barw w obrazie,
- *imread* - wczytanie obrazu z pliku,
- *imshow* - wyświetlenie obrazu,
- *imwrite* - zapis obrazu do pliku,
- *rectangle* - rysowanie prostokąta na obrazie,
- *putText* - umiejscowienie tekstu na obrazie,
- *addWeighted* - połączenie dwóch obrazów poprzez nałożenie ich na siebie,
- *thresholding* - binaryzacja obrazu,
- *GaussianBlur* - wygładzanie obrazu za pomocą funkcji Gaussa,
- *Canny* - wykrywanie krawędzi za pomocą algorytmu Johna F. Canny'ego [7],
- *findContours* - wykrywanie konturów obiektów (punktów o tym samym kolorze lub intensywności łączących się w krzywe),
- *resize* - zmiana wielkości obrazu,
- *imdecode* - odczytywanie zdjęcia z bufora.

3.3.2 Tesseract OCR

Biblioteka Tesseract jest pakietem składającym się z programu lini poleceń *tesseract* oraz silnika OCR *libtesseract* [56]. Tesseract powstał między rokiem 1985, a 1994 na potrzeby firmy Hewlett-Packard. W roku 2005 firma upubliczniła bibliotekę jako rozwiązanie *open-source*. Od początku 2006 do listopada 2018 za rozwój Tesseract odpowiedzialna była firma Google. Obecnie głównym programistą projektu jest Ray Smith. Silnik programu oparty jest o język programowania C++. W celu wykorzystania jej w opisywanym programie, użyto nakładki *pytesseract* [47]. Autorzy deklarują, że najnowsza wersja 5, wydana w listopadzie 2021, wspiera ponad 100 języków. Biblioteka wykorzystuje rekurencyjne sieci neuronowe LSTM (ang. *Long Short-Term Memory*) [15].

3.3.3 NumPy

Biblioteka NumPy jest jednym z fundamentalnych pakietów dla obliczeń naukowych w środowisku Python [57]. Głównym elementem biblioteki jest obiekt *ndarray*. Obiekt ten pozwala na tworzenie wielowymiarowych tablic o ściśle określonym typie danych. Oprócz tego, biblioteka zapewnia wiele operacji matematycznych do wykonania na tablicach. Operacje wykonywane za pomocą biblioteki NumPy są znacznie bardziej wydajne od standardowych metod języka Python. Dzieje się tak, ponieważ biblioteka posiada zaimplementowane metody w języku C. W opisywanym programie, dzięki zastosowaniu NumPy m.in. do obliczenia obrazu całkowego za pomocą metody *cumsum*, osiągnięto znacznie większą wydajność w tym zakresie.

3.3.4 Pickle

Moduł Pickle wchodzi w skład podstawowego pakietu Python. Jego zastosowaniem jest serializacja danych. Dzięki temu, możliwe jest zapisanie dowolnego obiektu i jego późniejsze odczytanie. W trakcie opracowywania niniejszego oprogramowania, operacja ta była niezwykle przydatna. Dzięki temu rozwiążaniu, możliwe było np. wczytanie z pliku nauczonego klasyfikatora. Biblioteka jest niezwykle prosta w użyciu. Korzystając

z bibliotek do serializacji, należy mieć na uwadze fakt, że wczytywanie zserializowanych obiektów może prowadzić do naruszenia zasad bezpieczeństwa programu. Z tego powodu, należy wczytywać tylko takie obiekty, których pochodzenia użytkownik jest pewien.

3.3.5 Numba

Numba jest biblioteką służącą do poprawiania wydajności programów napisanych w języku Python. Pakiet kompiluje kod w trakcie trwania programu (ang. *just in time* — JIT), w celu zwiększenia jego wydajności. Operacja ta jest działa najlepiej dla kodu opartego o tablice NumPy, pętle i funkcje. W celu komplikacji, fragmenty kodu oznacza się specjalnymi adnotacjami. Po natrafieniu na odpowiednią komendę, biblioteka kompiluje w trakcie trwania programu kod do postaci kodu maszynowego, dzięki czemu wydajność skompilowanego fragmentu jest porównywalna do wykonywania kodu maszynowego [46].

3.3.6 Joblib

Biblioteka Joblib zapewnia zestaw narzędzi do zarządzaniem przepływem (ang. *flow*) kodu w programach napisanych w języku w Python. W opisywanym rozwiążaniu wykorzystano moduł Parallel do równoległego wykonywania kodu. Poza tym, Joblib oferuje cache'owanie danych oraz ich leniwe ładowanie (ang. *lazy loading*). Jednymi z istotnych zalet jest brak zależności do innych niestandardowych pakietów oraz wysoka wydajność na dużych zbiorach danych.

4. Wyniki badań

Do przeprowadzenia testów aplikacji wykorzystano komputer przenośny Dell Vostro 5515. Komputer wyposażony został w 32GB pamięci RAM oraz procesor AMD Ryzen 5700u. Bazowe taktowanie procesora było równe 1.8GHz, natomiast w trybie boost taktowanie rosło do 4.3GHz. Procesor wyposażono w 8 rdzeni i 16 wątków. Program testowano na systemie operacyjnym Ubuntu 20.04.5 LTS.

4.1 Wyniki detekcji tablic rejestracyjnych

W celu oceny dokładności klasyfikatora służącego do detekcji tablic rejestracyjnych, przygotowano zbiór testowy. W zbiorze testowym znajdowało się 100 próbek pozytywnych oraz 10000 próbek negatywnych. Wyniki dla poszczególnych klasyfikatorów przedstawia Tabela 4.1.

Tabela 4.1: Dokładność klasyfikatorów w zadaniu klasyfikacji obrazów z tablicami rejestracyjnymi.

Liczba cech	Liczba słabych klasyfikatorów	Liczba koszyków	Dokładność klasyfikatora	Dokładność klasyfikatora dla próbek pozytywnych	Dokładność klasyfikatora dla próbek negatywnych
2205	32	8	99.70%	75.15%	99.94%
2205	64	8	99.77%	82.24%	99.94%
2205	256	8	99.87%	91.12%	99.96%

Z racji ograniczeń sprzętowych, nie udało się nauczyć klasyfikatorów dla okien z większą ilością cech.

Poza zbiorem testowym, algorytm testowany na rzeczywistym nagraniu wideo pochodzący z kamery samochodowej. Poniżej przedstawiono wyniki dla tego rodzaju obrazów wejściowych. Na wynikowych obrazach kolorem żółtym oznaczano okna, które klasyfikator oznaczył jako pozytywne dla progu t . Kolorem czerwonym oznaczono okna, które zostały oznaczone jako pozytywne oraz udało się odczytać z nich tekst.

Na Rysunku 4.1 przedstawiono wyniki detekcji dla progu $t = 1.5$. Dla takiego obrazu, wykryto dwie tablice, z czego jedna została rozpoznana. Po lewej stronie obrazu znajdują się jeszcze dwie kolejne tablice, jednak ich rozmiar jest znacznie mniejszy od wielkości okna przesuwnego. Na obrazie można zauważać wiele fałszywych pozytywów. Biorąc pod uwagę cechy, którymi kieruje się klasyfikator, takie zaznaczenie nie jest bezpodstawne. Przykładowo, zaznaczony został podpis pod znakiem zakazu wjazdu dla

cięzarówek. Tablica ta ma podobny kształt do tablicy rejestracyjnej oraz zawiera czarny tekst na białym tle. Oprócz tego, po lewej stronie obrazu znajduje się duży szyld reklamowy zawieszony na ścianie budynku. Zawiera on biały prostokąt z tekstem, co również jest problematyczne dla klasyfikatora.



Rysunek 4.1: Wyniki detekcji tablic rejestracyjnych dla progu $t = 1.5$. Liczba cech $T = 64$ (źródło: opracowanie własne).

Innym przykładem błędnej detekcji jest Rysunek 4.2. Jako pozytyw oznaczono wyświetlacz autobusu, zawierający informację o numerze linii autobusowej.



Rysunek 4.2: Wyświetlacz na autobusie wykryty jako tablica rejestracyjna dla progu $t = 1.5$. Liczba cech $T = 64$ (źródło: opracowanie własne).

Kolejnym przykładem błędnej detekcji jest Rysunek 4.3. Jako pozytyw wykryto szyld banku. Podobnie jak w przypadku wyświetlacza na autobusie, okno zawiera kontrastowy tekst.



Rysunek 4.3: Szyld reklamowy wykryty jako tablica rejestracyjna dla progu $t = 1.5$. Liczba cech $T = 64$ (źródło: opracowanie własne).

Z przedstawionymi błędami można walczyć za pomocą umiejętności dobierania progu decyzyjnego t . Dla progu $t = 2.5$ powyższe okna nie zostają uznane za pozytywne. Zwiększenie progu jednak objawia się również brakiem detekcji dla niektórych próbek prawdziwie pozytywnych. Rysunek 4.4 przedstawia problem braku detekcji tego samego samochodu pomiędzy klatkami. Takie zachowanie wynika z nieprzekroczenia progu dla środkowego zdjęcia.



Rysunek 4.4: Brak detekcji tego samochodu pomiędzy klatkami dla progu $t = 2.5$. Liczba cech $T = 64$ (źródło: opracowanie własne).

Warto również zaznaczyć fakt, że niektóre tablice są nieczytelne nawet dla ludzkiego oka. Ludzki umysł niejako się domyśla, że z przodu lub z tyłu samochodu znajduje się tablica rejestracyjna, nawet jeżeli na zdjęciu bardziej przypomina biały prostokąt. Opracowany klasyfikator nie ma takiej wiedzy. Z tego powodu, określenie granicy, dla której próbki powinny być klasyfikowane jako pozytywne, nie jest trywialnym zadaniem.

Należy podkreślić, że procent poprawnie sklasyfikowanych próbek przez klasyfikator nie zawsze powinien być jedyną miarą jakości systemu. Przykładowo, dla obrazu przedstawionego na Rysunku 4.1 procedura skanowania oknem przesuwnym, dla rozmiaru okna $45 \times 15\text{px}$, generuje 39188 okien. Jeżeli uznamy, że 20 okien zostało źle sklasyfikowanych, to poprawność klasyfikacji dalej jest bardzo wysoka i wynosi w tym przypadku 99.95%.

4.2 Wyniki rozpoznawaniu znaków

Jak wspomniano w poprzednim rozdziale, do rozpoznawania tekstu wykorzystano bibliotekę Tesseract. Zgodnie z [39], dokładność klasyfikacji biblioteki wynosi 70.2%. W rzeczywistości, jakość działania systemu w głównej mierze zależy od metod segmentacji znaków. Na Rysunku 4.5 zaprezentowano przykładowe wyniki segmentacji znaków z tablic rejestracyjnych.



Rysunek 4.5: Przykładowe wyniki segmentacji znaków z tablic rejestracyjnych (źródło: opracowanie własne).

Można przyjąć założenie, że im tablica jest większa, tym algorytm uzyskuje lepsze wyniki. Wynika to przede wszystkim z mniejszych zakłóceń w obrazie i lepiej widocznych przerwach pomiędzy znakami.

Na jakość segmentacji i następnie odczytu znaków wpływ ma również rotacja tablicy rejestracyjnej. Dodatkowo, w zależności od kąta, pod którym zrobiono zdjęcie, kontury znaków potrafią się łączyć ze sobą lub z ramką tablicy. Ten fakt zdecydowanie nie ułatwia późniejszej segmentacji. Śruby, które często stosowane są w celu zamocowania tablicy również negatywnie wpływają na segmentację znaków. Mogą one powodować łączenie znaków i w konsekwencji zmianę ich konturów.

Zauważono, że jednym z problematycznych dla biblioteki znaków jest litera „J”, która często mylona była ze znakiem „)”. Wykorzystując fakt, że tablice zawierają tylko znaki alfanumeryczne, można poprawić to programowo, poprzez zamianę tego znaku w rezultacie. Trudniejszym do rozwiązania problemem jest wzajemne mylenie litery „S” z cyfrą 5. Oba znaki mogą występować na tablicach rejestracyjnych. Patrząc na przykładowe zdjęcia z Rysunku 4.5, wyniki nie zawsze są oczywiste, nawet dla człowieka.

4.3 Porównanie wydajności czasowej

Dla systemów automatycznego rozpoznawania tablic rejestracyjnych wydajność jest jedną z kluczowych kwestii. Aby dane mogły być odpowiednio przetwarzane, system powinien na bieżąco procesować zdjęcia. Wydajność jest uzależniona od poniższych czynników:

- rozmiar okna przesuwnego,
- liczba skal dla okna przesuwnego,
- liczba pozytywnych okien wysłanych do modułu rozpoznawania znaków,
- liczba cech klasyfikatora.

W Tabeli 4.2 przedstawiono wyniki uzyskanych czasów dla opracowanego programu. W celu przyspieszenia obliczeń, zrównolegcono obliczenia cech dla procedury skanowania okiem przesuwnym. Poniższe dane uwzględniają zrównoleglenie, stąd czas detekcji nie jest równy iloczynowi liczby okien i czasu klasyfikacji jednego okna. Wszystkie obrazy

były skalowane na wejściu do rozmiaru 640×480px. Na podstawie zebranych danych,

Tabela 4.2: Wydajność czasowa systemu.

Liczba cech	Liczba unikalnych cech użytych przez klasyfikator	Liczba okien	Czas detekcji	Czas rozpoznawania jednej tablicy	Czas przetwarzania jednego okna
2205	32	39188	0.68s	0.11s	0.4ms
2205	64	39188	0.91s	0.12s	0.9ms
2205	256	39188	2.24s	0.11s	1.1ms

można zauważyć, że czas detekcji rośnie wraz ze wzrostem liczby używanych cech. Czas rozpoznawania dla jednej tablicy jest stały (w granicach błędu pomiarowego). Natomiast dla klasyfikatorów z większą liczbą cech, do modułu rozpoznawania znaków trafia mniej okien (z racji mniejszej ilości fałszywych pozytywów). Z racji jednak większej ilości czasu poświęcanej podczas detekcji, zysk ten jest pomijalny.

4.4 Propozycje udoskonalenia algorytmu

Praca ma charakter eksperymentalny. Opracowany program spełnia założenia detekcji tablic i rozpoznawania znaków na nich się znajdujących. Z racji ograniczonego czasu oraz jednoosobowego zespołu, istnieją pola do ulepszeń w opracowanym systemie.

Jednym z pomysłów na udoskonalenie modułu detekcji jest dołożenie do zbioru uczącego negatywnych okien, które obecnie klasyfikator wykrywa błędnie, jako pozytywne. Jest prawdopodobnym, że ponownie nauczony klasyfikator charakteryzowałby się wyższą dokładnością detekcji. Innym pomysłem, który mógłby poprawić wydajność oraz jakość detekcji jest użycie więcej niż jednego klasyfikatora do detekcji obiektów. W pierwszej kolejności jeden klasyfikator klasyfikowałby okna zawierające samochody. Tak działający klasyfikator działałby szybciej, ponieważ rozmiar okna w procedurze skanującej byłby większy. Wynika to z faktu, że samochód zajmuje większy obraz na zdjęciu niż jego tablica rejestracyjna. Następnie na wykrytych fragmentach należało by wykryć tablice rejestracyjne. W rezultacie liczba okien poddawanych procedurze predykcji mogłaby być mniejsza niż w oryginalnym rozwiązaniu. Aby zwiększyć wydajność rozwiązania, możliwe jest również zaimplementowanie programu w silnie typowanym, komplikowanym języku, np. C++.

W celu poprawienia jakości rozpoznawania znaków, możliwe jest nauczenie sieci neuronowej biblioteki Tesseract zbiorom znaków z tablic rejestracyjnych. Biblioteka domyślnie wyuczona jest na zbiorach znaków pochodzących z różnego rodzaju tekstów drukowanych, tj. książki lub dokumenty. Podczas przygotowywania zbioru uczącego, szczególną uwagę należało by przykuć do przygotowania znaków, które nie mają idealnych konturów. Ma to na celu zapewnienie przykładów uczących maksymalnie zbliżonych do próbek w realnym środowisku pracy.

Podsumowanie

Celem niniejszej pracy było opracowanie systemu rozpoznawania tablic rejestracyjnych na obrazach z kamery samochodowej z użyciem wybranych algorytmów z zakresu przetwarzania obrazów i uczenia maszynowego. W celu wykonania wspomnianego systemu, zebrano zbiór uczący składający się z 10985 zdjęć zawierających 10301 tablic rejestracyjnych. Opracowano system, w skład którego wchodzą moduł detekcji tablic oraz moduł rozpoznawania, znajdujących się na nich, znaków. Do klasyfikacji wykorzystano algorytm RealBoost ze słabymi klasyfikatorami realizowanymi poprzez koszykowanie wartości funkcji logit. Klasyfikator oparty został o cechy Haara. Do rozpoznawania znaków wykorzystano bibliotekę Tesseract opartą o rekurencyjne sieci neuronowe LSTM. Program zaimplementowano w środowisku Python. Testy przeprowadzono na surowych nagraniach z kilku popularnych kamer samochodowych ze średniej półki.

W stosunku do istniejących rozwiązań, opracowany system nie potrzebuje specjalnej aparatury do działania. W ramach testów wykazano, że nagrania pochodzące z wideorejestratorów kosztujących maksymalnie kilkaset złotych, są wystarczające. W przeprowadzonych eksperymentach udowodniono poprawność działania opisywanego rozwiązania. Przedstawiono słabości systemu, wynikające z niedoskonałości użytych algorytmów. Zrealizowany system cechuje się ogólną dokładnością równą 99.77% przy czym czułością 82%. Przetwarzanie jednej klatki obrazu z kamery trwa w przybliżeniu 1 sekundę. Stworzone oprogramowanie zostało przygotowane w ten sposób, że nie wymaga wielu modyfikacji, aby zwiększyć szybkość lub jakość detekcji. Można to zrealizować poprzez zmianę parametru liczby używanych cech do klasyfikacji.

Komercyjne rozwiązania zapewniają wyższą dokładność oraz są bardziej wydajne. Należy jednak mieć na uwadze fakt, że są one realizowane przez firmy mające odpowiednie zasoby. Poprawę systemu można by uzyskać przede wszystkim poprzez ulepszenie algorytmów, zarówno lokalizacji, jak i rozpoznawania znaków. Dodatkowo, stosując kamerę o wyższej rozdzielczości, możliwe jest poprawienie operacji rozpoznawania znaków. Do osiągnięcia wyższej wydajności systemu, możliwe jest użycie komputera o większej mocy obliczeniowej. Poniższa praca udowadnia słuszność stosowania cech Haara w zakresie lokalizacji tablic rejestracyjnych. Również algorytm boostingu zapewnił znaczne przyspieszenie klasyfikacji obiektów przy jednoczesnym zachowaniu odpowiedniego poziomu dokładności.

Opracowany system mógłby zostać wykorzystany np. do przetwarzania nagrani z wideorejestratorów znajdujących się w radiowozach. Rozpoznane tablice rejestracyjne mogłyby być sprawdzane w zewnętrznym systemie. Przykładowo, możliwe byłoby sprawdzenie czy pojazd posiada aktualny przegląd techniczny. Funkcjonariusze byliby na-

tychmiastowo powiadamiani o wykrytej nieprawidłowości. W konsekwencji doszłoby do zatrzymania potencjalnie niebezpiecznego pojazdu, co znacznie ograniczyłoby stwarzane bezpieczeństwo na drogach.

Spis literatury

Książki

- [1] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491962291.
- [2] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.
- [3] Maciej Szaleniec i Ryszard Tadeusiewicz. *LEKSYKON SIECI NEURONOWYCH [Lexicon on Neural Networks]*. Sty. 2015. ISBN: 978-83-63270-10-0.

Artykuły

- [4] C.N.E. Anagnostopoulos i in. “A License Plate-Recognition Algorithm for Intelligent Transportation System Applications”. W: *IEEE Transactions on Intelligent Transportation Systems* 7.3 (2006), s. 377–392. doi: [10.1109/TITS.2006.880641](https://doi.org/10.1109/TITS.2006.880641).
- [5] C. Busch i in. “Feature based recognition of traffic video streams for online route tracing”. W: *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No.98CH36151)*. T. 3. 1998, 1790–1794 vol.3. doi: [10.1109/VETEC.1998.686064](https://doi.org/10.1109/VETEC.1998.686064).
- [6] Hakan Caner, Hatice Geçim i Ali Alkar. “Efficient Embedded Neural-Network-Based License Plate Recognition System”. W: *Vehicular Technology, IEEE Transactions on* 57 (paź. 2008), s. 2675–2683. doi: [10.1109/TVT.2008.915524](https://doi.org/10.1109/TVT.2008.915524).
- [7] John Canny. “A Computational Approach to Edge Detection”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), s. 679–698. doi: [10.1109/TPAMI.1986.4767851](https://doi.org/10.1109/TPAMI.1986.4767851).
- [8] C. K. Chow i T. Kaneko. “Boundary Detection of Radiographic Images by a Threshold Method”. W: *IFIP Congress*. 1971.
- [9] Franklin C. Crow. “Summed-Area Tables for Texture Mapping”. W: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: Association for Computing Machinery, 1984, s. 207–212. ISBN: 0897911385. doi: [10.1145/800031.808600](https://doi.org/10.1145/800031.808600). URL: <https://doi.org/10.1145/800031.808600>.

- [10] Shan Du i in. "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review". W: *IEEE Transactions on Circuits and Systems for Video Technology* 23.2 (2013), s. 311–325. doi: [10.1109/TCSVT.2012.2203741](https://doi.org/10.1109/TCSVT.2012.2203741).
- [11] Tran Duc Duan i in. "Building an Automatic Vehicle License-Plate Recognition System". W:
- [12] Yoav Freund i Robert E. Schapire. "Experiments with a New Boosting Algorithm". W: *ICML*. 1996.
- [13] Gisu Heo i in. "Extraction of Car License Plate Regions Using Line Grouping and Edge Density Methods". W: *2007 International Symposium on Information Technology Convergence (ISITC 2007)*. 2007, s. 37–42. doi: [10.1109/ISITC.2007.79](https://doi.org/10.1109/ISITC.2007.79).
- [14] Sepp Hochreiter i Jürgen Schmidhuber. "Long Short-Term Memory". W: *Neural Computation* 9.8 (1997), s. 1735–1780. doi: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [15] Sepp Hochreiter i Jürgen Schmidhuber. "Long Short-term Memory". W: *Neural computation* 9 (grud. 1997), s. 1735–80. doi: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [16] Martin Jenckel, Saqib Syed Bukhari i Andreas Dengel. "Transcription Free LSTM OCR Model Evaluation". W: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2018, s. 122–126. doi: [10.1109/ICFHR-2018.2018.00030](https://doi.org/10.1109/ICFHR-2018.2018.00030).
- [17] Wenjing Jia i in. "Mean shift for accurate license plate localization". W: *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005*. 2005, s. 566–571. doi: [10.1109/ITSC.2005.1520110](https://doi.org/10.1109/ITSC.2005.1520110).
- [18] T. Kanungo i in. "An efficient k-means clustering algorithm: analysis and implementation". W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (2002), s. 881–892. doi: [10.1109/TPAMI.2002.1017616](https://doi.org/10.1109/TPAMI.2002.1017616).
- [19] Ningyang Li. "An Implementation of OCR System Based on Skeleton Matching". W: 1993.
- [20] Xin Lin i Kazunori Otobe. "Hough transform algorithm for real-time pattern recognition using an artificial retina camera". W: *Opt. Express* 8.9 (kw. 2001), s. 503–508. doi: [10.1364/OE.8.000503](https://doi.org/10.1364/OE.8.000503). URL: <http://opg.optica.org/oe/abstract.cfm?URI=oe-8-9-503>.
- [21] Adam Lipiński i Seweryn Lipiński. "Automatyczna ocena jakości oprysku na podstawie śladów kropel przy użyciu komputerowej analizy obrazu". W: *Inżynieria Rolnicza* 13 (sty. 2009), s. 163–168.
- [22] David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". W: *International Journal of Computer Vision* 60 (list. 2004), s. 91–. doi: [10.1023/B:VISI.0000029664.99615.94](https://doi.org/10.1023/B:VISI.0000029664.99615.94).
- [23] Marcin Namysł i Iuliu Konya. "Efficient, Lexicon-Free OCR using Deep Learning". W: wrz. 2019, s. 295–301. doi: [10.1109/ICDAR.2019.00055](https://doi.org/10.1109/ICDAR.2019.00055).

- [24] Bruno Olshausen i David Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. W: *Nature* 381 (lip. 1996), s. 607–9. doi: [10.1038/381607a0](https://doi.org/10.1038/381607a0).
- [25] Nobuyuki Otsu. “A Threshold Selection Method from Gray-Level Histograms”. W: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), s. 62–66. doi: [10.1109/TSMC.1979.4310076](https://doi.org/10.1109/TSMC.1979.4310076).
- [26] I. Paliy i in. “Approach to recognition of license plate numbers using neural networks”. W: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. T. 4. 2004, 2965–2970 vol.4. doi: [10.1109/IJCNN.2004.1381137](https://doi.org/10.1109/IJCNN.2004.1381137).
- [27] “Picture Thresholding Using an Iterative Selection Method”. W: *IEEE Transactions on Systems, Man, and Cybernetics* 8.8 (1978), s. 630–632. doi: [10.1109/TSMC.1978.4310039](https://doi.org/10.1109/TSMC.1978.4310039).
- [28] E. Półrolniczak. “Metoda aktywnych konturów w segmentacji znaków na tablicach rejestracyjnych”. W: *Pomiary Automatyka Kontrola* R. 58, nr 2.2 (2012), s. 176–179.
- [29] Tomasz Praczyk. “Użycie sieci probabilistycznej oraz metody najbliższego sąsiada do identyfikacji obcych radiostacji okrętowych”. W: *Logistyka* nr 3, CD (2011), s. 2235–2242. issn: 1231-5478.
- [30] C.A. Rahman, W. Badawy i A. Radmanesh. “A real time vehicle’s license plate recognition system”. W: *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003*. 2003, s. 163–166. doi: [10.1109/AVSS.2003.1217917](https://doi.org/10.1109/AVSS.2003.1217917).
- [31] B. Rasolzadeh, L. Petersson i N. Pettersson. “Response Binning: Improved Weak Classifiers for Boosting”. W: *2006 IEEE Intelligent Vehicles Symposium*. 2006, s. 344–349. doi: [10.1109/IVS.2006.1689652](https://doi.org/10.1109/IVS.2006.1689652).
- [32] Joseph Redmon i in. “You Only Look Once: Unified, Real-Time Object Detection”. W: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 779–788. doi: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91).
- [33] M. Sarfraz, M.J. Ahmed i S.A. Ghazi. “Saudi Arabian license plate recognition system”. W: *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*. 2003, s. 36–41. doi: [10.1109/GMAG.2003.1219663](https://doi.org/10.1109/GMAG.2003.1219663).
- [34] Robert E. Schapire i Yoram Singer. “Improved Boosting Algorithms Using Confidence-Rated Predictions”. W: *Mach. Learn.* 37.3 (grud. 1999), s. 297–336. issn: 0885-6125. doi: [10.1023/A:1007614523901](https://doi.org/10.1023/A:1007614523901). url: <https://doi.org/10.1023/A:1007614523901>.
- [35] Zied Selmi, Mohamed Ben Halima i Adel Alimi. “Deep Learning System for Automatic License Plate Detection and Recognition”. W: list. 2017, s. 1132–1138. doi: [10.1109/ICDAR.2017.187](https://doi.org/10.1109/ICDAR.2017.187).

- [36] M. Sezgin i Bulent Sankur. “Survey over image thresholding techniques and quantitative performance evaluation”. W: *Journal of Electronic Imaging* 13 (sty. 2004), s. 146–168. doi: [10.1117/1.1631315](https://doi.org/10.1117/1.1631315).
- [37] Jithmi Shashirangana i in. “Automated License Plate Recognition: A Survey on Methods and Techniques”. W: *IEEE Access* 9 (2021), s. 11203–11225. doi: [10.1109/ACCESS.2020.3047929](https://doi.org/10.1109/ACCESS.2020.3047929).
- [38] Donald F. Specht. “Probabilistic neural networks”. W: *Neural Networks* 3.1 (1990), s. 109–118. ISSN: 0893-6080. doi: [https://doi.org/10.1016/0893-6080\(90\)90049-Q](https://doi.org/10.1016/0893-6080(90)90049-Q). URL: <https://www.sciencedirect.com/science/article/pii/089360809090049Q>.
- [39] Dan Sporici, Elena Cușnir i Costin-Anton Boiangiu. “Improving the Accuracy of Tesseract 4.0 OCR Engine Using Convolution-Based Preprocessing”. W: *Symmetry* 12.5 (2020). ISSN: 2073-8994. doi: [10.3390/sym12050715](https://doi.org/10.3390/sym12050715). URL: <https://www.mdpi.com/2073-8994/12/5/715>.
- [40] Fayed Tarsha-Kurdi. “HOUGH-TRANSFORM AND EXTENDED RANSAC ALGORITHMS FOR AUTOMATIC DETECTION OF 3D BUILDING ROOF PLANES FROM LIDAR DATA”. W: (wrz. 2007).
- [41] P. Viola i M. Jones. “Rapid object detection using a boosted cascade of simple features”. W: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. T. 1. 2001, s. I–I. doi: [10.1109/CVPR.2001.990517](https://doi.org/10.1109/CVPR.2001.990517).
- [42] Feng Wang i in. “Fuzzy-based algorithm for color recognition of license plates”. W: *Pattern Recognit. Lett.* 29 (2008), s. 1007–1020.
- [43] Xiao-wei Xu i in. “A method of multi-view vehicle license plates location based on rectangle features”. W: *2006 8th international Conference on Signal Processing* 3 (2006), s. 1–4.
- [44] Lihong Zheng i Xiangjian He. “Character Segmentation for License Plate Recognition by K-Means Algorithm”. W: wrz. 2011, s. 444–453. ISBN: 978-3-642-24087-4. doi: [10.1007/978-3-642-24088-1_46](https://doi.org/10.1007/978-3-642-24088-1_46).
- [45] R. Zunino i S. Rovetta. “Vector quantization for license-plate location and image coding”. W: *IEEE Transactions on Industrial Electronics* 47.1 (2000), s. 159–167. doi: [10.1109/41.824138](https://doi.org/10.1109/41.824138).

Źródła internetowe i inne

- [46] A 5 minute guide to Numba. URL: <https://numba.readthedocs.io/en/stable/user/5minguide.html>.
- [47] A Python wrapper for Google Tesseract. URL: <https://github.com/madmaze/pytesseract>.
- [48] A small python tool for creating positive text file for OpenCV haar training. URL: <https://github.com/dhruvvyas90/haar-object-marker>.

- [49] Adrian Horzyk. *Uczenie głębkich sieci neuronowych*. URL: <https://home.agh.edu.pl/~horzyk/lectures/miw/MIW-UczenieG%C5%82%C4%99bokichSieciNeuronowych.pdf>.
- [50] Przemysław Klęsk. *Techniki szybkiej detekcji: ekstrakcja cech poprzez obrazy całkowe, boosting, kaskady klasyfikatorów*. URL: https://wikizmsi.zut.edu.pl/uploads/4/4a/Szybka_detekcja.pdf.
- [51] Agnieszka Ławrynowicz. *Rekurencyjne sieci neuronowe, LSTM i GRU*. URL: https://www.cs.put.poznan.pl/ala.wrynowicz/PJN_5.pdf.
- [52] *OpenCV documentation*. URL: <https://docs.opencv.org/4.x/>.
- [53] *OpenCV overview*. URL: <https://opencv.org/about/>.
- [54] *OpenCV RetrievalModes*. URL: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71.
- [55] *Otsu's method for image thresholding explained and implemented*. URL: <https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/>.
- [56] *Tesseract Open Source OCR Engine*. URL: <https://github.com/tesseract-ocr/tesseract>.
- [57] *What is NumPy?* URL: <https://numpy.org/doc/stable/user/whatisnumpy.html>.
- [58] *Wprowadzenie do uczenia maszynowego*. URL: https://kcir.pwr.edu.pl/~witold/ai/ml_intro_s.pdf.