Wydział
Informatyki

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. *edge based)* 11
1.1.2 Metody oparte na kolorach (ang. *color based)* 13
1.1.3 Metody oparte na teksturach (ang. *texture based)* 14
1.1.4 Klasyfikatory 14
1.1.5 Metody głębokiego uczenia (ang. *Deep learning*) 19
1.2 Przegląd istniejących metod segmentacji tablic rejestracyjnych 20
1.2.1 Binaryzacja 21
1.2.2 Metoda rzutów jasności 23
1.2.3 Metoda elementów połączonych (ang. *Connected components*) 23
1.3 Przegląd istniejących metod rozpoznawania tablic rejestracyjnych 23
1.3.1 OCR 23
1.3.2 Metody oparte na wzorcach 23

2 Zebranie materiału uczącego 25

3 Opracowany algorytm 29

3.1 Proces uczenia klasyfikatora 29
3.1.1 Przygotowanie danych uczących 29
3.1.2 Uczenie klasyfikatora 32
3.2 Schemat algorytmu 34
3.2.1 Algorytm detekcji 35
3.2.2 Algorytm segmentacji i rozpoznawania znaków 36
3.3 Biblioteki użyte w programie 38
3.3.1 OpenCV 38
3.3.2 Tesseract OCR 39
3.3.3 Numpy 39
3.3.4 Sklearn 39
3.3.5 Pickle 39

3.3.6 Numba 39

4 Wyniki badań 41
4.0.1 Wyniki detekcji tablic rejestracyjnych 41
4.0.2 Wyniki rozpoznawaniu znaków 41

Podsumowanie 42

Spis literatury 45
Książki 45
Artykuły 45
Źródła internetowe i inne 47

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 lub pora 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

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

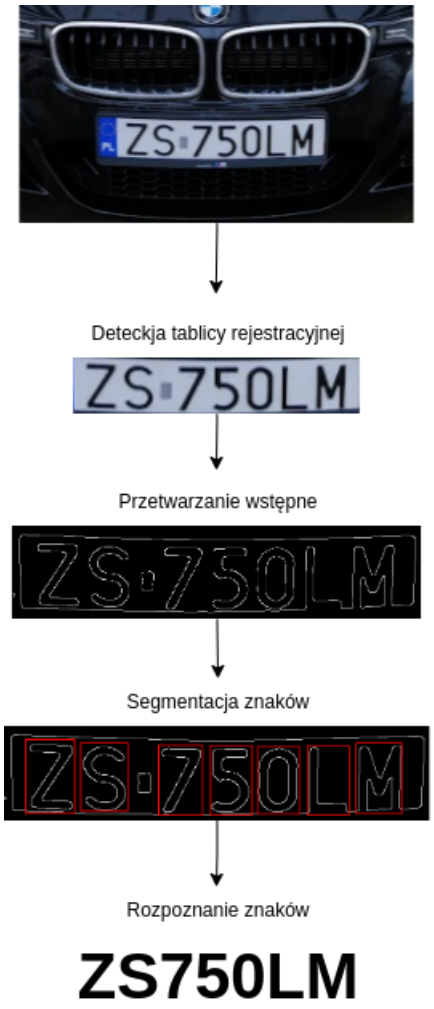
Na przestrzeni ostatnich lat stosowanie Systemów Automatycznego Rozpoznawania Tablic Rejestracyjnych (ARTR) (ang. *Automatic Licence Plate Recognition* — ALPR) stało się znacznie bardziej powszechne. W większości dużych miast istnieją parkingi, gdzie po umieszczeniu opłaty za postój, przy zbliżeniu się 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 tańszą 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 polegająca na wykryciu i odczytaniu znaków z tablicy rejestracyjnych na podstawie zarejestrowanego obrazu. Do tego celu wykorzystywany jest aparat o wysokiej rozdzielczości 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 średnioprzepustowego) w celu redukcji szumu. W procesie tym można wyróżnić trzy etapy [0]:

- **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 powstały 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 odczycie numerów rejestracyjnych pojazdu i potwierdzeniu opłaty za postój w zewnętrznej bazie danych. Takie systemy pracują z reguły 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 słowikiem języka polskiego, definicja tablicy rejestracyjnej brzmi następująco:

> **Definicja 1.1.1 — Tablica rejestracyjna.** Płytka zawierająca 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" [0]. W oparciu o powyższe cechy zaprezentowano wiele algorytmów do rozwiązania 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 wyzywającym zadaniem ze względu na poniższe czynniki:

- zajmowanie niewielkiego obszaru na zdjęciu przez tablicę rejestracyjną,
- istnienie ogromnej ilości 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.[0]. 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 kontraście 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.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1.1}$$

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

Często wykorzystywaną metodą do wykrywania krawędzi obiektów w obrazach jest *Binary Image Processing* [0]. 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.

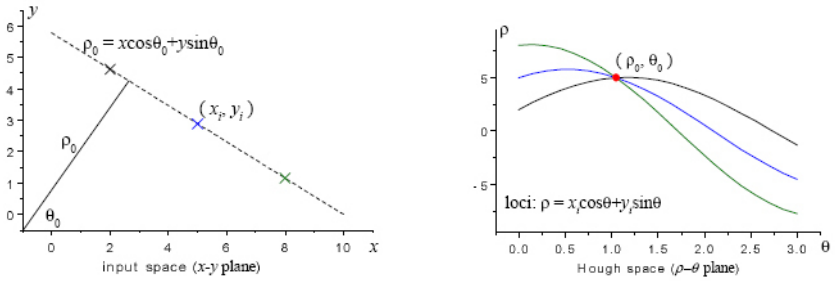
Inną stosowaną metodą do wykrywania linii na obrazach binarnych jest transformata Hougha [0]. Motywacją do jej opracowania była metoda siłowa (ang. *brute force*), która jest jednak znacznie bardziej zasobożerna. Złożoność algorytmu siłowego wynosi $O(n^3)$, gdzie n oznacza liczbę niezbędnych operacji 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)

$$x \cos \alpha + y \sin \alpha = \rho. \tag{1.2}$$

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 [0].

Zasadę transformacji ilustruje Rysunek 1.2.



Rysunek 1.2: Transformata Hougha (źródło: [0]).

Innym spotykanym podejściem [0] 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 dokładności, szczególnie dla pojazdów znajdujących się w ruchu. Metode oparte na krawędziach są stosowane w wielu rozwiązaniach 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* [0] i logika rozmyta [0].

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 wysoką częstość zmiany kolorów w obszarze występowania tablic rejestracyjnych. W [0] 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%.

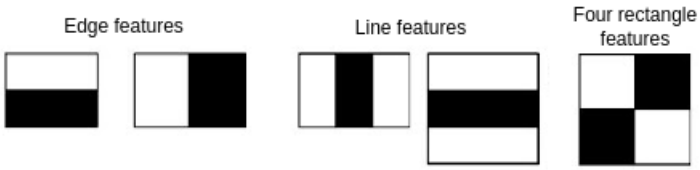
W analizie tekstur, często stosuje się filtr Gabora. Jest to filtr liniowy, pozwalający na przefiltrowanie obrazu z precyzyjnie dobranym zakresem częstotliwości. Reprezentacje częstotliwości i orientacji filtrów Gabora są uważane przez wielu współczesnych naukowców zajmujących się widzeniem komputerowym za podobne do tych z ludzkiego układu wzrokowego [0]. W [0] 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 algorytmem AdaBoost (ang. *Adaptative Boosting — AdaBoost*) do wyuczenia kaskady klasyfikatorów [0]. Podejście takie zostało zaproponowane po raz pierwszy w [0]. Algorytm Violi-Jonesa został zaprezentowany w 2001. Wykorzystuje on algorytm AdaBoost, który został zaproponowany w pracy [0]. 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” [0].



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:

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

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 [0, 0]:

$$s(j, k) = s(j, k-1) + f(j, k), \tag{1.4}$$

$$ii(j, k) = ii(j, k-1) + s(j, k), \tag{1.5}$$

gdzie $s(j, k)$ to skumulowana suma w wierszu, $f(j, k)$ to wartość obrazu w punkcie (j, k) [0]. 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).

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

Na podstawie powyższego wzoru można zauważyć, że wystarczą operacje tylko na 4 punktach obrazu całkowego [0]. 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) ilości 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 [0]. 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

procedure DiscreteAdaBoost((x_i, y_i)) ▷ $i = 1, \ldots, m, y_i \in \{-1, +1\}$
  Rozpocznij od jednostajnego rozkładu wag: $w_i := \frac{1}{m}, i = 1, \ldots, m.$
  for $t := 1, \ldots, T$ **do**
    Naucz klasyfikator $f_t(x) \in \{-1, +1\}$ na danych uczących używając wag w_i.
    Oblicz błąd uczący:
      $\epsilon_t := \sum_{i=1}^{m} w_i [f(x_i) \neq y_i]$.
    Oblicz wagę klasyfikatora:
      $\alpha_t := \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$.
    Aktualizuj wagi przykładów wg:
      $Z_t := \sum_{i=1}^{m} w_i \exp(-\alpha_t f_t(x_i) y_i)$,
      $w_i := \frac{w_i \exp(-\alpha_t f_t(x_i) y_i)}{Z_t}$.
  end for
  Zwróć zbiorowy klasyfikator $F(x) := \sum_{t=1}^{T} \alpha_t f_t(x)$.
end procedure

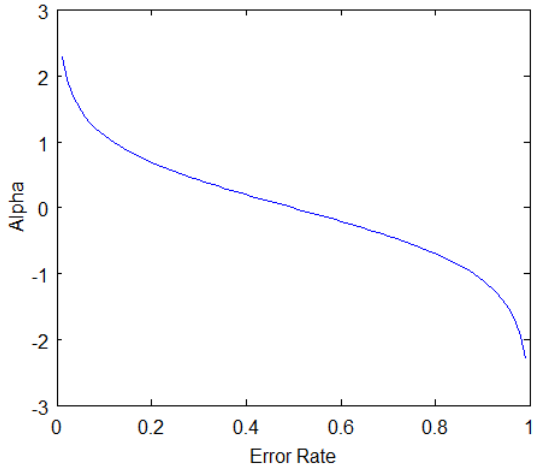
Algorytm przyjmuje na wejściu zbiór uczący wraz z etykietami opisującymi poszczególne elementy zbioru. Do każdej pary uczącej (x_i, y_i) przypisywana jest waga równa w_i. Na początku uczenia, wszystkie wagi mają taką samą wartość, a ich suma jest równa 1. Na koniec każdej pętli uczącej dochodzi do reważenia wag przykładów, a wagi błędnie sklasyfikowanych próbek zostają zwiększone. Wagi ustalane są na podstawie błędu klasyfikacji ϵ_t. Dzięki temu, w następnej iteracji uczenia kolejny klasyfikator będzie mógł lepiej rozpoznawać niepoprawnie oznaczone jednostki treningowe w poprzednim kroku. W momencie, gdy każdy klasyfikator zostanie wyuczony, AdaBoost przypisuje do nich wagi na podstawie dokładności każdego z nich. Ostateczną odpowiedź klasyfikatora

można opisać wzorem (1.7).

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

Finalny klasyfikator zawiera T słabych klasyfikatorów. Każdy ze słabych klasyfikatorów zwraca jedną z dwóch odpowiedzi $\{-1, 1\}$. Waga klasyfikatora, wyliczona przez AdaBoost, została określona symbolem α_t. 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_alphacurve.png`).

Z Rysunku 1.5 można odczytać, że waga rośnie wykładniczo 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 próbkę 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 występuję w połączeniu z popularnymi wariantami słabych klasyfikatorów [0]:

- 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 [0]. Główną różnicą w stosunku do AdaBoosta, jest założenie, że słabe klasyfikatory są rzeczywistoliczbowe, a nie binarne [0]. Na poniższych schemacie przedstawiono pseudokod algorytmu uczącego RealBoost.

Algorytm 2 Algorytm uczący klasyfikatora RealBoost

procedure RealAdaBoost((x_i, y_i)) ▷ $i = 1, \ldots, m, y_i \in \{-1, +1\}$
  Rozpocznij od jednostajnego rozkładu wag: $w_i := \frac{1}{m}, i = 1, \ldots, m$.
  for $t := 1, \ldots, T$ **do**
    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:
    $$f_t(X) = \frac{1}{2} \ln \frac{\hat{P_w}(y = 1|x)}{\hat{P_w}(y = -1|x)}.$$
    Aktualizuj wagi przykładów wg:
    $$Z_t := \sum_{i=1}^{m} w_i \exp(-f_t(x_i)y_i),$$
    $$w_i := \frac{w_i \exp(-f_t(x_i)y_i)}{Z_t}.$$
  end for
  Zwróć zbiorowy klasyfikator $F(x) := \sum_{t=1}^{T} f_t(x)$.
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 [0]. Odpowiedź słabego klasyfikatora jest zwykle ustalana jako przybliżenie połowy przekształcenia logit:

$$f_t(X) = \frac{1}{2} \ln \frac{\hat{P_w}(y = 1|x)}{\hat{P_w}(y = -1|x)}, \tag{1.8}$$

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).

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

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

Słabe klasyfikatory stosowane w algorytmach boostingowych mogą być realizowane poprzez koszykowanie wartości funkcji logit (ang. *response binning*). Ten typ klasyfikatora został zaprezentowany w [0]. Przedstawiony sposób działania polega na przybliżaniu

rozkładów warunkowych przez funkcje kawałkami stałe [0]. Przed uczeniem algorytmu wyznaczana jest liczba koszy o równej szerokości oznaczana literą B. Na podstawie ustalonych przedziałów $[a_1, a_2]$, każdej z cech przypisywany jest odpowiedni indeks kosza. Indeks kosza $\beta(x) \in \{1, \ldots, B\}$, do którego należy x obliczany jest na podstawie wzoru (1.10).

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

Odpowiedź słabego klasyfikatora dla 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.

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

Autorzy algorytmu zalecają stosowanie dużej ilości koszy dla gładkiego histogramu rozkładu cech. Jeżeli histogram zawiera wiele maksimów lokalnych, należy zmniejszyć liczbę koszyków. W przeciwnym razie zbyt duża rozdzielczość może wprowadzić szum do klasyfikatora, natomiast zbyt niska liczba koszy może pozbawić klasyfikatora istotnych cech [0].

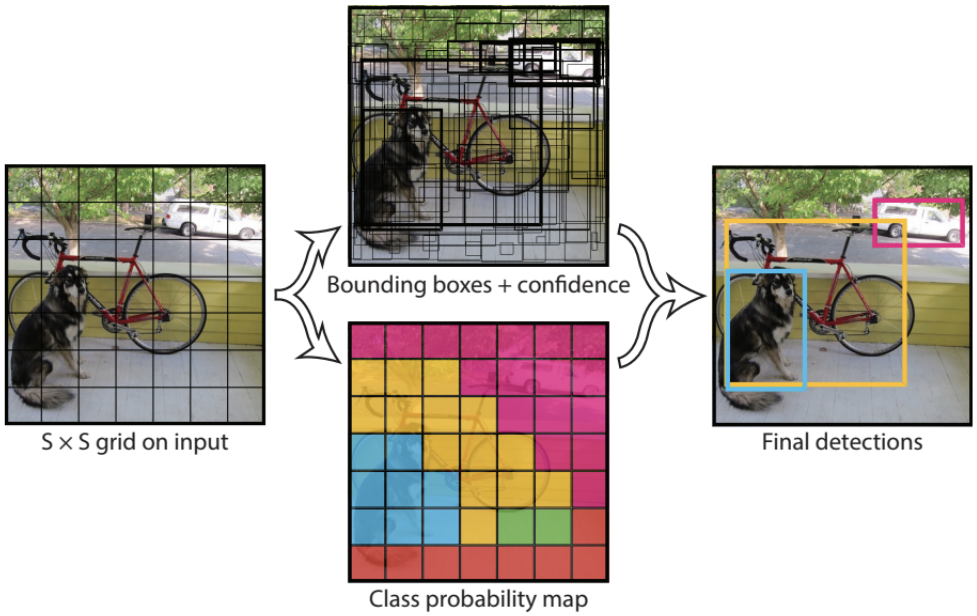
We wspomnianym wcześniej algorytmie Violi-Jonesa wykorzystano boosting techniką AdaBoost. Autorzy zaproponowali użycie kaskady klasyfikatorów. Takie podejście bazuje na obserwacji, że okna pozytywne stanowią średnio 0.01% wszystkich okien. Rozpatrywane okno w pierwszej kolejności badane jest przez słabsze 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 przerywany. 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) [0]. Składają się one z jednej lub wielu warstw splotowych (typowych dla kroku próbkowania, określającego subwzorce), a następnie przez jedną lub w pełni

połączone warstwy tak jak w klasycznej wielowarstwowej sieci. Sieci splotowe są łatwe do uczenia, gdyż zawierają mniej parametrów (wykorzystując te same wagi) niż typowe sieci neuronowe z dokładnością do ilości warstw splotowych i ich rozmiaru. Ten rodzaj sieci neuronowych jest predestynowany do obliczeń na strukturach 2D (tjòbrazy) [0].

Jednym z najnowocześniejszych systemów detekcji w czasie rzeczywistych jest algorytm YOLO (ang. *You only look once*) [0]. 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 wpierw segmentowane. Jest to szczególny przypadek optycznego rozpoznawania znaków [0]. 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 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. Aby zwiększyć w jak największym stopniu prawdopodobieństwo prawidłowego odczytania znaków, używa się szereg czynności na obrazie wejściowym. Na przykład dla obróconych obiektów stosuje się transformatę biliniową, zwaną również metodą Tustina [0].

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, wyznaczenie progu binaryzacji musi zostać wykonane właściwie, w celu uniknięcia połączenia znaków lub scalenia ich z obramowaniem tablicy rejestracyjnej [0]. 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 [0]. 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 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, wybranie 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 znanym algorytmem stosowanym do wyznaczania progu binaryzacji jest algorytm Otsu [0]. Metoda Otsu jest techniką opartą na wariancji w celu znalezienia wartości progowej, przy której ważona wariancja między pikselami pierwszego planu i tła jest najmniejsza [0]. 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 wewnątrz klasy, zdefiniowaną jako ważona suma wariancji dwóch klas (tła i pierwszego planu). Wzór na znalezienie wariancji wewnątrz klasowej przy dowolnym progu t jest określony wzorem (1.12)

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

gdzie $\omega_{bg}(t)$ i $\omega_{fg}(t)$ reprezentują prawdopodobieństwo 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).

$$\sigma^2_{bg|fg}(t) = \frac{\sum(x_i - \hat{x})^2}{N-1} \tag{1.13}$$

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) [0]. Algorytm szczególnie dobrze sprawdza się w przypadkach, gdy liczby pikseli tła i obiektów pierwszego planu są zbliżone [0].

1.2.2 Metoda rzutów jasności

todo

1.2.3 Metoda elementów połączonych (ang. *Connected components*)

todo

1.3 Przegląd istniejących metod rozpoznawania tablic rejestracyjnych

1.3.1 OCR

tesseract korzysta z sieci lstm wiec warto by to opisać i jakieś inne sieci

1.3.2 Metody oparte na wzorcach

@TODO - TEMPLATE AND PATTERN MATCHING TECHNIQUES

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 [0]. 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 powszechnym 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 rozdzielczość oraz liczba klatek na sekundę różni się pomiędzy plikami wideo. Pozyskane zdjęcia pochodzą zarówna 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). Rozdzielczość obrazu zawierała się w zakresie 1920x1080 do 3840x2160 pikseli. Liczba klatek na sekundę dla części kamer wynosiła 60 klatek na sekundę, a dla pozostałej 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 sekundy, próbkowano co 20 klatek, natomiast dla pozostałych co 10. Przykładowy 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 [0]. Jego fragment przedstawiono na algorytmie 2.1.

```
import cv2

def obj_marker(event, x, y):
    global click_count
    global debug
    global obj_list
    global obj_count
    global frameName
    global x1
    global y1
    global w
    global h
    global frame
    global frameResized
    if event == cv2.EVENT_LBUTTONDOWN:
        click_count += 1
        if click_count % 2 == 1:
            x1 = x
            y1 = y
        else:
            orgShape = frame.shape
            ratioH = orgShape[0] / 1080
```

```
23                ratioW = orgShape[1] / 1920
24
25                w = abs(x1 - x)
26                h = abs(y1 - y)
27                obj_count += 1
28                if x1 > x:
29                    x1 = x
30                if y1 > y:
31                    y1 = y
32                obj_list.append('%d %d %d %d ' % (x1 *
                      ratioW, y1 * ratioH, w * ratioW, h *
                      ratioH))
33                cv2.rectangle(frameResized, (x1, y1), (x1 +
                      w, y1 + h), (0, 255, 0), 1)
34                cv2.imshow(frameName, frameResized)
```

Algorytm 2.1: Funkcja do oznaczania fragmentów obrazu zawierających poszukiwany obiekt

Jak wspomniano wcześniej, obrazy posiadały różne rozdzielczości, często przewyższające rozdzielczość monitora. Aby zdjęcia wyświetlały się poprawnie, skalowano je do rozdzielczości ekranu, na którym przeprowadzono operację nadawania etykiet. Następnie współrzędne wybranego punktu konwertowano do współrzędnych w obrazie oryginalnym. Pozyskane koordynaty zapisywano do globalnej zmiennej *obj_list*. 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	378x136px
Minimalne rozmiary próbki	15x11px
Rozdzielczości zdjęć	1920x1080, 2560x1440, 3840x2160
Liczba klatek na sekundę	30 kl/s, 60 kl/s

3. Opracowany algorytm

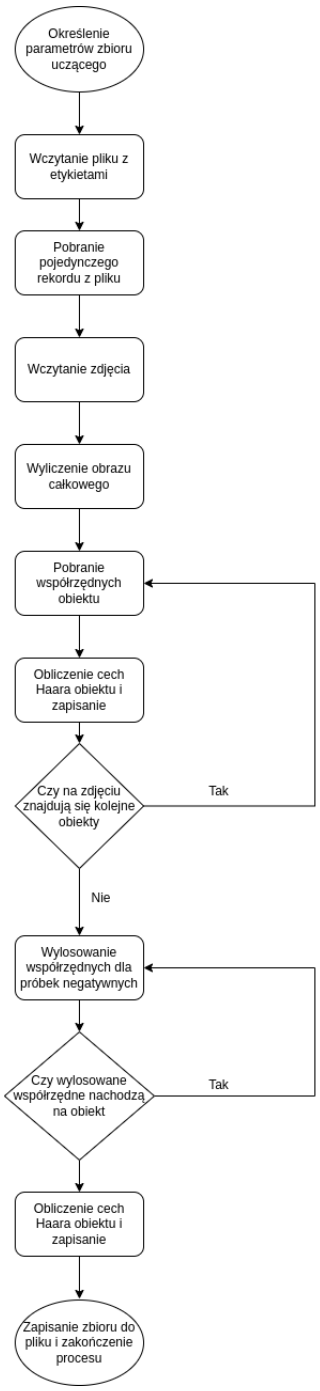
W przedstawionej pracy zrealizowano program rozpoznający tablice rejestracyjne na sekwencjach wideo pochodzących z kamery samochodowej. Do detekcji użyto skanowania obrazu oknem przesuwnym i klasyfikowanie na podstawie cech za pomocą algorytmu RealBoost opartego o koszyki. Do ekstrakcji cech zastosowano obrazy całkowe wykorzystujące cechy Haara. 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 [0]. Innym niepożądanym przypadkiem błędnego wyuczenia jest niedouczenie (ang. *underfitting*). Wynika ono najczęściej z zastosowania zbyt uproszczonego modelu lub niewystarczającej liczby próbek uczących [0]. 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 ilość razy skalowany wzdłuż każdego z kierunków. Parametr ten oznacza się literą s. Parametrem p oznacza rozmiary regularnej siatki $((2p-1)x(2p-1))$ punktów zaczepienia cech [0]. Na podstawie powyższych parametrów określa się liczbę cech

niezbędnych do wyliczenia w trakcie przetwarzania jednego okna wg wzoru (3.1).

$$n(s, p) = 6s^2(2p - 1)^2 \tag{3.1}$$

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 też 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.

```
def haar_coords(s, p, indexes):
    coords = []
    f_jump = (FEATURE_MAX - FEATURE_MIN) / (s - 1)
    for t, s_j, s_k, p_j, p_k in indexes:
        f_h = FEATURE_MIN + s_j * f_jump
        f_w = FEATURE_MIN + s_k * f_jump
        p_jump_h = (1.0 - f_h) / (2 * p - 2)
        p_jump_w = (1.0 - f_w) / (2 * p - 2)
        pos_j = 0.5 + p_j * p_jump_h - 0.5 * f_h
        pos_k = 0.5 + p_k * p_jump_w - 0.5 * f_w
        single_coords = [np.array([pos_j, pos_k, f_h,
            f_w])] # whole rectangle for single feature
        for white in HAAR_TEMPLATES[t]:
            white_coords = np.array([pos_j, pos_k, 0.0,
                0.0]) + white * np.array([f_h, f_w, f_h,
                f_w])
            single_coords.append(white_coords)
        coords.append(np.array(single_coords))
    return np.array(coords, dtype=object)
```

Algorytm 3.1: Procedura generujące szablony cech Haara dla konkretnych wpsółrzędnych

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

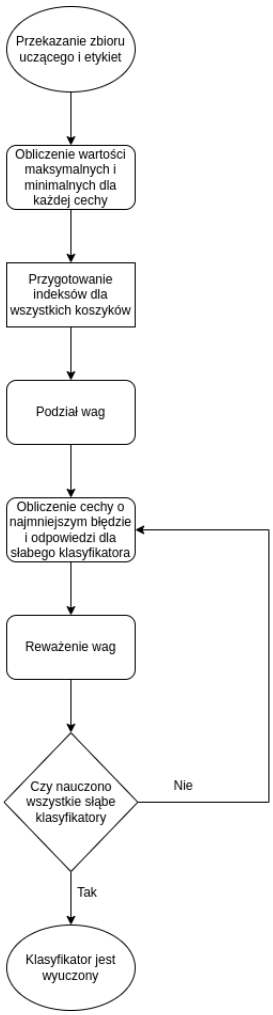
W niniejszej pracy zaimplementowano i wykorzystano algorytm wzmacniający RealBoost oparty o koszyki (ang. *bins*). Takie podejście zostało zaprezentowane po raz pierwszy w [0]. Na wejściu metody uczącej funkcja przyjmuje cechy uczące oraz ich etykiety. Dla każdej cechy obliczane są wartości maksymalne i minimalne. Ustalane 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 to 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.

```
w = np.ones(m) / m
for t in range(self.T_):
    j_best = None
    logits_best = None
    err_exp_best = np.inf
    for j in range(n):
```

```
        logits = np.zeros(self.B_)
        for b in range(self.B_):
            W_positive = w[indexer_positive[j, b]].sum()
            W_negative = w[indexer_negative[j, b]].sum()
            logits[b] = self.logit(W_positive,
                W_negative)
        err_exp = np.sum(w * np.exp(-yy *
            logits[X_binned[:, j]]))
        if err_exp < err_exp_best:
            err_exp_best = err_exp
            logits_best = logits
            j_best = j
    self.feature_indexes_[t] = j_best
    self.logits_[t] = logits_best
    w = w * np.exp(-yy * logits_best[X_binned[:,
        j_best]])
    w /= err_exp_best
```

Algorytm 3.2: Procedura ucząca klasyfikator RealBoostBins

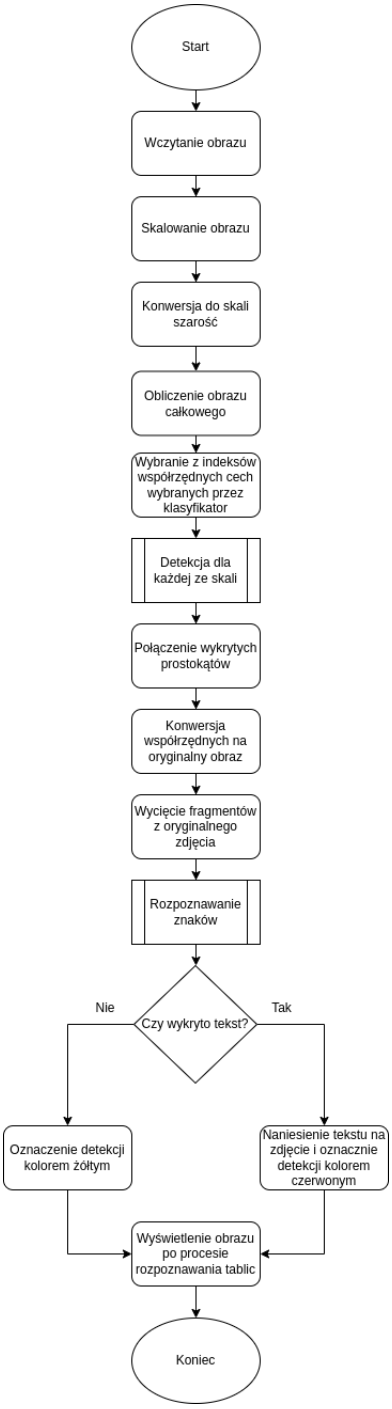
Na Rysunku 3.3 przedstawiono schemat blokowy uczenia klasyfikatora.



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

3.2 Schemat algorytmu

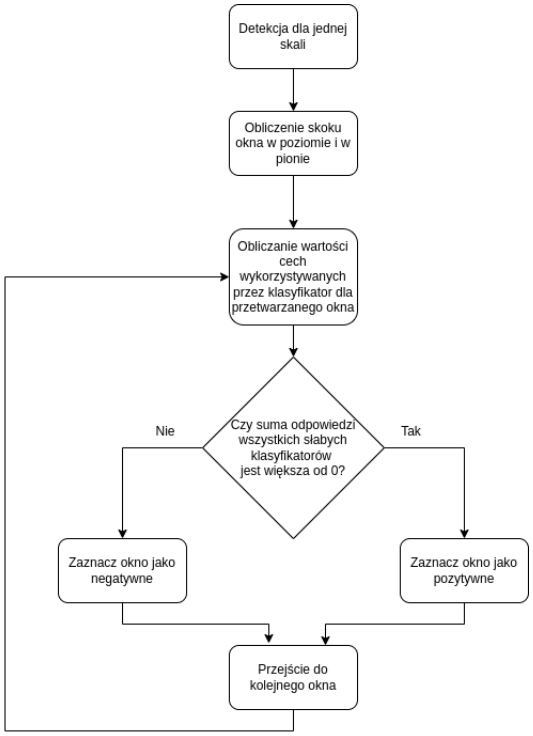
@todo - opis schematu blokowego Na Rysunku 3.4 przedstawiono schemat blokowy algorytmu rozpoznawania tablic rejestracyjnych.



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

3.2.1 Algorytm detekcji

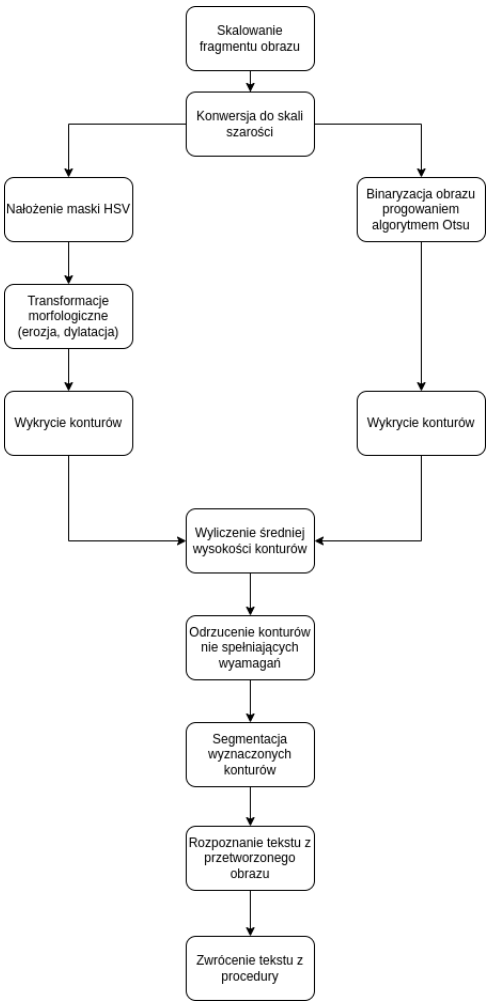
@todo - opis schematu blokowego Na Rysunku 3.5 przedstawiono schemat blokowy algorytmu detekcji tablic rejestracyjnych.



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

3.2.2 Algorytm segmentacji i rozpoznawania znaków

@todo - opis schematu blokowego Na Rysunku 3.6 przedstawiono schemat blokowy algorytmu rozpoznawania znaków.



Rysunek 3.6: Schemat blokowy algorytmu rozpoznawania tablic (źródło: opracowanie własne).

Na Rysunku 3.7 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 nie zostały spełniły kryteriów geometrycznych. Założono, że kontur musi być prostokątem o dłuższych bokach zorientowanych w pionie. Dodatkowo ustalono minimalną wartość powierzchni prostokąta. Kolorem żółtym oznaczono kontury, które zostały odrzucone, z powodu ich zbyt dużego odchylenia od średniej wysokości konturów.

Rysunek 3.7: 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 bilbiotek. 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*) [0, 0] . 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 [0]
- *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* [0]. 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* [0]. Autorzy deklarują, że najnowsza wersja 5, wydana w listopadzie 2021, wspiera ponad 100 języków. Biblioteka wykorzystuje rekurencyjne sieci neuronowe LSTM (ang. *Long Short-Temp Memory*) [0].

3.3.3 Numpy

TODO

3.3.4 Sklearn

TODO - do zastanowienia czy sklearn jest istotny w niniejszej pracy

3.3.5 Pickle

TODO

3.3.6 Numba

Biblioteka od przyspieszania kodu (jit)

4. Wyniki badań

4.0.1 Wyniki detekcji tablic rejestracyjnych

TODO - przykład przedstawienia wyników detekcji
opis przeprowadzony testów
zrzuty z nagrań
czasy
wyraportowanie błędów

Tabela 4.1: Parametry opracowanego zbioru

Liczba cech	Liczba słabych klasyfikatorów	Liczba koszyków	Dokładność klasyfikatora	Dokładność klasyfikatora dla próbek pozytywnych	Dokładność klasyfikatora dla próbek negatywnych
2205	64	8	99,8%	82,2%	99,9%

można pokazać zdjęcia nocne
duże tablice, małe tablice
różny threshold, różne wyniki
jako pozytywy wykrywane sa oznaczenia samochodów (modelu)
reklamy i banery
czas wykonywania
krzywe roc

4.0.2 Wyniki rozpoznawaniu znaków

TODO - przedstawienie rozpoznawania na podstawie zdjęć
przykłady:
mylony znak “(” z “J”
mylone S z 5

Podsumowanie

Podsumowanie pracy powinno na maksymalnie dwóch stronach przedstawić główne wyniki pracy dyplomowej. Struktura zakończenia to:

1. Przypomnienie celu i hipotez
2. Co w pracy wykonano by cel osiągnąć (analiza, projekt, oprogramowanie, badania eksperymentalne)
3. Omówienie głównych wyników pracy
4. Jak wyniki wzbogacają dziedzinę
5. Zamknięcie np. poprzez wskazanie dalszych kierunków badań.

Spis literatury

Książki

[0] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

Artykuły

[0] 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.

[0] 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.

[0] 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.

[0] C. K. Chow i T. Kaneko. "Boundary Detection of Radiographic Images by a Threshold Method". W: *IFIP Congress*. 1971.

[0] 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. URL: https://doi.org/10.1145/800031.808600.

[0] 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.

[0] Tran Duc Duan i in. "Building an Automatic Vehicle License-Plate Recognition System". W:

[0] Yoav Freund i Robert E. Schapire. "Experiments with a New Boosting Algorithm". W: *ICML*. 1996.

[0] 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.

[0] 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.

[0] 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.

[0] 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. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-8-9-503.

[0] 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.

[0] 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.

[0] 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.

[0] “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.

[0] 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.

[0] 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.

[0] 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. URL: https://doi.org/10.1023/A:1007614523901.

[0] 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.

[0] 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.

[0] 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.

[0] Fayez Tarsha-Kurdi. “HOUGH-TRANSFORM AND EXTENDED RANSAC ALGORITHMS FOR AUTOMATIC DETECTION OF 3D BUILDING ROOF PLANES FROM LIDAR DATA”. W: (wrz. 2007).

[0] 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.

[0] Feng Wang i in. “Fuzzy-based algorithm for color recognition of license plates”. W: *Pattern Recognit. Lett.* 29 (2008), s. 1007–1020.

[0] 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.

[0] 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.

[0] 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.

Źródła internetowe i inne

[0] *A Python wrapper for Google Tesseract*. URL: https://github.com/madmaze/pytesseract.

[0] *A small python tool for creating positive text file for OpenCV haar training*. URL: https://github.com/dhruvvyas90/haar-object-marker.

[0] Adrian Horzyk. *Uczenie głębokich sieci neuronowych*. URL: https://home.agh.edu.pl/~horzyk/lectures/miw/MIW-UczenieG%c5%82%c4%99bokichSieciNeuronowych.pdf.

[0] 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.

[0] *OpenCV documentation*. URL: https://docs.opencv.org/4.x/.

[0] *OpenCV overview*. URL: https://opencv.org/about/.

[0] *Otsu's method for image thresholding explained and implemented*. URL: https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/.

[0] *Tesseract Open Source OCR Engine*. URL: https://github.com/tesseract-ocr/tesseract.

[0] *Wprowadzenie do uczenia maszynowego*. URL: https://kcir.pwr.edu.pl/~witold/ai/ml_intro_s.pdf.