

Szkoła Główna Gospodarstwa Wiejskiego w Warszawie
Wydział Zastosowań Informatyki i Matematyki

Sylwester Turski
167497

Inteligentny algorytm rozpoznawania
wolnych miejsc parkingowych
na podstawie obrazów z kamery cyfrowej

Intelligent algorithm to recognize free parking places based on
images from a digital camera

Praca dyplomowa inżynierska
na kierunku informatyka

Praca wykonana pod kierunkiem
Dra Pawła Hosera
Katedra Zastosowań Informatyki

Warszawa, 2017 rok

Oświadczenie promotora pracy

Oświadczam, że niniejsza praca*/wskazane przez autora rozdziały pracy dyplomowej przygotowanej zespołowo* została/zostały* przygotowana pod moim kierunkiem i stwierdzam, że spełnia*/spełniają* warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data Podpis promotora pracy

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 90 poz. 631 z późn zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data Podpis autora pracy

Streszczenie

Inteligentny algorytm rozpoznawania wolnych miejsc parkingowych na podstawie obrazów z kamery cyfrowej

Praca przedstawia wykonanie i przetestowanie algorytmu do rozpoznawania wolnych miejsc parkingowych, których lokalizacja jest oznaczone na obrazie. Również znajduje się opis tworzenia makiety do pozyskiwania zdjęć testowych. Na początku omówione są zagadnienia później używane podczas opisu algorytmu. Następny rozdział opisuje jest proces tworzenia algorytmu. Na koniec przedstawione są wyniki klasyfikacji i wnioski.

Słowa kluczowe – SVM, OpenCV, Rozpoznawanie obrazów, HSV, Canny

Summary

Intelligent algorithm to recognize free parking places based on images from a digital camera

Summary - 1000 words max (12)

Keywords – SVM, OpenCV, Pattern recognition, HSV, Canny

Spis Treści

1. Wstęp	8
2. Przegląd piśmiennictwa	9
2.1. Rozpoznawanie obrazów	9
2.2. Modele kolorów	10
2.3. Wykrywanie krawędzi	14
3. Projekt rozwiązania	18
3.1. Biblioteki i technologie użyte podczas pisania algorytmu	18
3.2. Makietą parkingu	19
3.3. Oznaczanie konturów dla miejsc parkingowych	22
3.4. Badane cechy obrazu	24
3.5. Klasyfikator	35
3.6. Walidacja jakości klasyfikacji	38
4. Podsumowanie	42
5. Bibliografia	44
6. Spis obrazków	46
7. Spis listingów	48

1. Wstęp

Wraz z rozwojem informatyki na przestrzeni lat rozwijały się różne koncepcje inteligentnego miasta (ang. smart city) i nie tylko miasta. Wiele pomysłów początkowo pojawiało się w literaturze z gatunku fantastyki naukowej, jednak rozwój informatyki pozwolił, aby zacząć myśleć o tych pomysłach na poważnie w przestrzeni kilkunastu lat. Część pomysłów z tych pomysłów jest nawet zrealizowana. Na przykład firma Rolls-Royce zapowiedziała pracę nad autonomicznymi zdalnie sterowanymi statkami [1], zapewniają, że do roku 2020 wprowadzą pierwsze komercyjne okręty, które będą całkowicie pozbawione załogi i sterowane będą przez operatora przebywającego na lądzie. Firma Tesla Motors wprowadziła do swoich samochodów funkcjonalność autopilota [2], dzięki której samochód sam jeździ po drogach bez ingerencji kierowcy. Od lat w miastach występują inteligentne sygnalizacje świetlne, które rozładowuje korki w miastach.

Od dawna brakuje systemu, który pozwoliłby kierowcom na wyszukanie miejsca wolnego miejsca parkingowego w centrum miasta. Dzięki temu kierowcy nie traciliby czasu na znalezienie wolnego miejsca parkingowego, jednocześnie przyniosłoby to pozytywne skutki dla powietrza, które w miastach nie jest najlepsze poprzez zmniejszenie emisji spalin, które samochód wydziela, podczas gdy kierowca szuka miejsca parkingowego. Wiele parkingów centrach handlowych posiada już zintegrowany system, który zlicza ilość wolnych miejsc i pokazuje kierowcy gdzie są wolne miejsca parkingowe. Jednak takie podejście wymaga instalowania specjalnych czujników dla pojedynczego miejsca. Zastosowanie takiego rozwiązania w skali miasta może być kosztowne i wiązać się z budową nowej infrastruktury. Mimo iż istnieje aktualnie infrastruktura, która mogłaby nadaje się do śledzenia miejsc parkingowych, mowa tu o monitoringu miejskim, który dzięki jest gęsto rozstawiony w centrum miast. Dzięki analizie obrazów z monitoringu miejskiego można by wykrywać czy miejsca parkingowe są wolne, a dzięki połączeniu z nawigacją samochodową, można kierować kierowców bezpośrednio do wolnych miejsc parkingowych.

Ta praca dyplomowa ma sprawdzić czy jest możliwe wykrywanie wolnych miejsc parkingowych na podstawie obrazu z kamery cyfrowej.

Celem pracy jest stworzenie algorytmu, który będzie w stanie sklasyfikować miejsce parkingowe zaznaczone na zdjęciu, jako wolne lub zajęte. Do algorytmu będzie dostarczany również zbiór uczący, składający się ze zdjęć parkingu wraz z oznaczeniami konturu gdzie znajdują się miejsca parkingowe i czy są one zajęte czy wolne.

Z racji problemów z uzyskaniem zdjęć z monitoringu parkingu, jako substytut obrazów testowych zdjęcia zostaną zrobione na specjalnie przygotowanej makiecie. Stworzony zostanie też program ułatwiający oznaczanie konturów miejsc parkingowych i ich stanu. Przygotowanie programu testującego skuteczność algorytmu.

2. Przegląd piśmiennictwa

W tym rozdziale zostaną przedstawione podstawy rozpoznawania obrazów, wykorzystywane modele kolorów i sposoby wykrywania krawędzi.

2.1. Rozpoznawanie obrazów

Rozpoznawanie obrazów lub bardziej intuicyjne pojęcie rozpoznawanie wzorców (ang. pattern recognition) jest stosunkowo nową dziedziną z pogranicza informatyki, statystyki i matematyki. Zajmuje się ona rozpoznawaniem wzorców i regularności w danych. W książce pt. „Rozpoznawanie obrazów” Ryszard Tadeusiewicz i Mariusz Flasiński opisują rozpoznawanie obrazów jako złożone z trzech odwzorowań [3]

$$A = F \cdot C \cdot B$$

Pierwsze z nich oznaczone literą B jest recepcją, to jest zamiana obrazu na szereg cech reprezentowany wektor n-elementowy. Na przykład w algorytmie przedstawionym w tej pracy recepcją jest przetwarzanie obrazu za pomocą algorytmów do wykrywania krawędzi, zamiana przestrzeni kolorów i wyliczanie cech z tych obrazów.

Kolejne odwzorowanie oznaczone literą C jest klasyfikacja. Oznacza to, że obraz reprezentowany szeregiem cech musi zostać zaklasyfikowany do jednej z klas (lub do grupy klas, jeżeli klasyfikacja jest bardziej skomplikowana). W omawianym algorytmie za odwzorowanie klasyfikacji służy maszyna wektorów nośnych, jest ona w stanie na podstawie zbioru uczącego wywnioskować, do której klasy mogą należeć nowe obserwacje.

Kolejnym krokiem jest odwzorowanie F, jest to proces podejmowania decyzji. Na podstawie klasyfikacji odwzorowanie to podejmując decyzję, co zrobić dalej. Omawiany algorytm nie posiada odwzorowania F, jednak takim odwzorowaniem mogłoby być podejmowanie decyzji przez zewnętrzny system odpowiedzialny za nawigację kierowców do pobliskich wolnych miejsc parkingowych.

2.2. Modele kolorów

W programowaniu obrazy rastrowe są przechowywane jako dwuwymiarowa macierz pikseli. Słowo piksel pochodzi z angielskiego od złączenia słów picture i element. Piksele są kolorowymi kropkami, z których są zbudowane obrazy. Istnieje wiele sposobów na przedstawienie koloru w programowaniu. Ten rozdział został napisany z wykorzystaniem [4] [5].

Najpopularniejszym, powszechnie używanym modelem kolorów jest model RGB. Został on oparty o barwy podstawowe, czyli barwy, z których można uzyskać inne kolory poprzez ich mieszanie. Model RGB został oparty o 3 barwy podstawowe czerwony (ang. Red), zielony (ang. Green) i niebieski (ang. Blue). Model RGB jest modelem addytywnym, oznacza to, że opiera się na łączeniu światła emitowanego i kolor biały uzyskuje się przez dodanie wszystkich kolorów. Model ten nadaje się idealnie do wyświetlania obrazu na monitorach czy przechwytywania zdjęć przez matrycę aparatu, jednak przestrzeń kolorów RGB kiepsko opisuje to, w jaki sposób człowiek postrzega kolory.

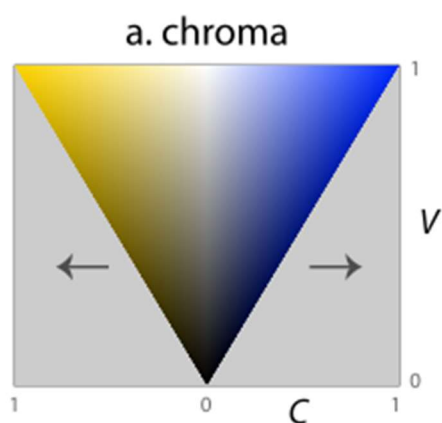
Często używanymi modelami kolorów wykorzystywanym przy rozpoznawaniu obrazów są modele HSL i HSV, te dwa kolory opierają się na innym podejściu do reprezentacji koloru niż podejście addytywne. Pierwszą składową obu modeli kolorów jest odcień (ang. Hue), który reprezentuje barwę koloru w wartościach od 0° - 360° . Kąt 0° oznacza kolor czerwony, który przechodzi w kolor zielony na 120° , niebieski kolor na 240° i z powrotem ma czerwony przy 360° . Kolejną składową obu modeli jest składowa nasycenia (ang. Saturation), oznacza to jak bardzo kolor jest nasycony, przyjmuje wartości od 0% do 100%, przy czym 0% oznacza kolor całkowicie nienasycony, czyli czarnobiały, natomiast im większe nasycenie tym barwa jest odbierana jako bardziej „żywa”, należy też wspomnieć, że saturacja jest mylona z chrominancją. Nasycenie 100% koloru oznacza maksymalną chrominancję przy danej jaskrawości/jasności. Ponieważ nie wszystkie kombinacje chrominancji i jasności/jaskrawości nie oznaczają żadnego koloru stosuje się przekształcenie chrominancji w saturację (Rysunek 5). W modelu barw HSL trzecim parametrem opisującym kolor jest jasność. Jasność przyjmuje wartości od 0% - kolor czarny do 100% kolor biały, wartości o największym nasyceniu mają jasność 50%. Natomiast w modelu HSV trzecim parametrem opisującym przestrzeń jest jaskrawość. Jaskrawość

przyjmuje wartości od 0% - czarny kolor do 100% - biały kolor i kolory o chrominancji 100%.

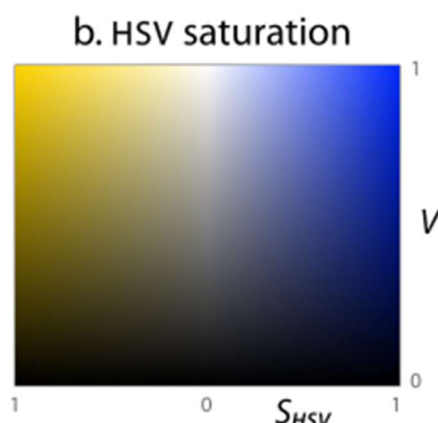
Tabela 1 Przestrzenie kolorów HSV i HSL przed i po przekształceniu

HSV	HSL
Rysunek 1 Stożek HSV [6]	Rysunek 2 Podwójny stożek HSL [7]
Rysunek 3 Cylinder HSV [8]	Rysunek 4 Cylinder HSL [9]

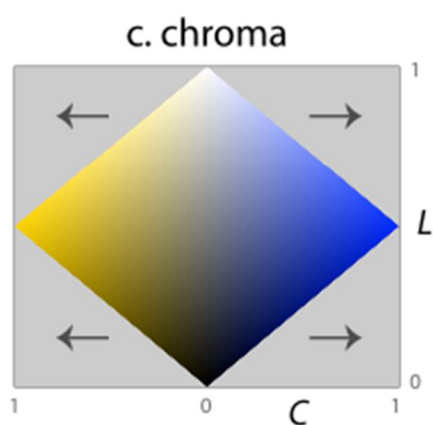
$$H = 50^\circ / 230^\circ$$



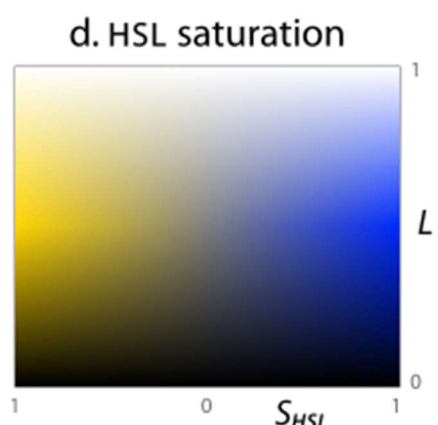
$$V = \max(R, G, B)$$



$$S_{HSV} = C/V$$



$$L = \frac{1}{2}\max(R, G, B) + \frac{1}{2}\min(R, G, B)$$



$$S_{HSL} = \begin{cases} C/2L & \text{if } L \leq 1/2 \\ C/2-2L & \text{if } L > 1/2 \end{cases}$$

Rysunek 5 Pochodzenie nasycenia koloru z chrominancji i jasności/jaskrawości w modelach HSL i HSV [10]

Poniżej przedstawione są wzory przejścia z modelu kolorów RGB na HSV i HSL i z powrotem.

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

$$Hue = \begin{cases} 0^\circ, & \Delta = 0 \\ 60^\circ \times \frac{G' - B'}{\Delta} \bmod 6, & C_{max} = R' \\ 60^\circ \times \frac{B' - R'}{\Delta} + 2, & C_{max} = G' \\ 60^\circ \times \frac{R' - G'}{\Delta} + 4, & C_{max} = B' \end{cases}$$

$$Saturation = \begin{cases} 0, & C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & C_{max} \neq 0 \end{cases}$$

$$Value = C_{max}$$

Wzór 1 Przekształcenie z RGB do HSV

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

$$Hue = \begin{cases} 0^\circ, & \Delta = 0 \\ 60^\circ \times \frac{G' - B'}{\Delta} \bmod 6, & C_{max} = R' \\ 60^\circ \times \frac{B' - R'}{\Delta} + 2, & C_{max} = G' \\ 60^\circ \times \frac{R' - G'}{\Delta} + 4, & C_{max} = B' \end{cases}$$

$$Saturation = \begin{cases} 0, & \Delta = 0 \\ \frac{\Delta}{1 - |2 \times Lightnes - 1|}, & \Delta \neq 0 \end{cases}$$

$$Lightnes = \frac{C_{max} + C_{min}}{2}$$

Wzór 2 Przekształcenie z RGB do HSV

$$C = Value \times Saturation$$

$$X = C \times \left(1 - \left\lfloor \left(\frac{Hue}{60}\right)^\circ \bmod 2 - 1 \right\rfloor\right)$$

$$m = Value - C$$

$$(R', G', B') = \begin{cases} (C, X, 0), & 0^\circ \leq Hue < 60^\circ \\ (X, C, 0), & 60^\circ \leq Hue < 120^\circ \\ (0, C, X), & 120^\circ \leq Hue < 180^\circ \\ (0, X, C), & 180^\circ \leq Hue < 240^\circ \\ (X, 0, C), & 240^\circ \leq Hue < 300^\circ \\ (C, 0, X), & 300^\circ \leq Hue < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

Wzór 3 Przekształcenie HSV na RGB

$$C = (1 - |2 \times Lightness - 1|) \times Saturation$$

$$X = C \times \left(1 - \left|\left(\frac{Hue}{60}\right) \bmod 2 - 1\right|\right)$$

$$m = Lightness - \frac{C}{2}$$

$$(R', G', B') = \begin{cases} (C, X, 0), & 0^\circ \leq Hue < 60^\circ \\ (X, C, 0), & 60^\circ \leq Hue < 120^\circ \\ (0, C, X), & 120^\circ \leq Hue < 180^\circ \\ (0, X, C), & 180^\circ \leq Hue < 240^\circ \\ (X, 0, C), & 240^\circ \leq Hue < 300^\circ \\ (C, 0, X), & 300^\circ \leq Hue < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

Wzór 4 Przekształcenie HSL na RGB

2.3. Wykrywanie krawędzi

Wykrywanie krawędzi jest jednym z elementów rozpoznawania obrazów. Ponieważ informację o kształcie obiektu zamknięte są w krawędziach zapisanych na obrazie, dlatego powstało wiele technik wykrywania krawędzi. Krawędzie na obrazie definiuje się jako skoki lub bardziej dokładnie nieciągłości w luminacji obrazu. Wykrywanie krawędzi dzieli się na dwie kategorie metod. Metody z pierwszej kategorii opierają się na badaniu pierwszej pochodnej i szukania lokalnych maksimów i minimów. Metody z drugiej kategorii opierają się na badaniu drugiej pochodnej i szukania miejsc, w których przechodzi przez zero. Następnie, aby odjąć szum i mało znaczące krawędzie stosuje się progowanie wartości. Ten rozdział został napisany z wykorzystaniem książki [11].

Z racji, iż obrazy nie są funkcją ciągłą, tylko są zdyskretyzowane, do szukania krawędzi używa się splotów z różnymi maskami. Wzór 5 przedstawia sposób obliczania splotu na

obrazach, zmienna src – obraz wejściowy, dst obrazy wyjściowy, $mask$ to funkcja, którą splatamy obraz w postaci dwuwymiarowej tablicy, K to suma wartości z maski, lub 1 gdy ta suma jest równa 0. W poniższych wzorach w tym rozdziale znak $*$ będzie oznaczał splot.

$$dst[m, n] = (\sum_a \sum_b mask[a, b] * src[m - a, n - b]) / K$$

Wzór 5 Splot

Przykładami operatorów detekcji krawędzi działającymi na pierwszej pochodnej są operatory Prewitt'a (Wzór 6) i Sobla (Wzór 7). Na początku wykrywane są krawędzie horyzontalnie i wertykalnie, dopiero na koniec te wartości są przeliczane do jednej macierzy. Wzór 8 przedstawia operator Laplac'a, która działa w oparciu o drugą pochodną. Wykrywa od razu krawędzie we wszystkich kierunkach.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Wzór 6 Wykrywanie krawędzi przy pomocy Prewitt'a

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Wzór 7 Wykrywanie krawędzi przy pomocy Sobla

$$G = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} * A$$

Wzór 8 Wykrywanie krawędzi Laplace

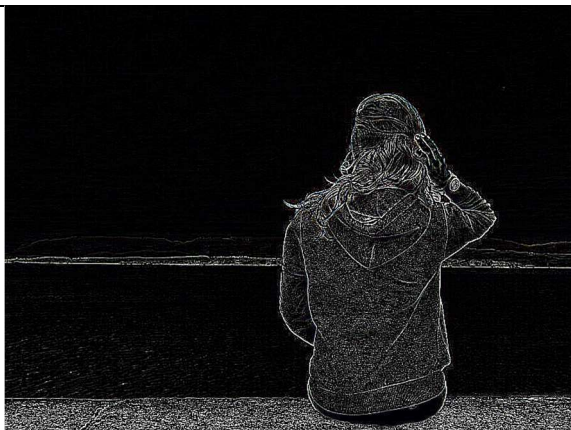
Jak widać w poniższej tabeli różnica pomiędzy wykrywaniem krawędzi operatorem Prewitt'a a operatorem Sobla są praktycznie nie widoczne na tych obrazkach, w praktyce różnią się nieznacznie. Wykrywanie krawędzi operatorem Laplac'a znacząco różni się od pozostałych dwóch metod, wykryte krawędzie są znacznie cieńsze.



Rysunek 6 Zdjęcie oryginalne przed wykrywaniem krawędzi [12]



Rysunek 7 Zdjęcie po wykrywaniu krawędzi operatorem Prewitt'a



Rysunek 8 Zdjęcie po wykryciu krawędzi operatorem Laplac'a



Rysunek 9 Zdjęcie po wykrywaniu krawędzi operatorem Sobela




Algorytm wykrywania krawędzi Canny'ego powstał o 3 założenia:

- Algorytm ma wykryć tak dużo krawędzi jak to tylko możliwe
- Oznaczone krawędź powinna znajdować się jak najbliżej krawędzi na obrazie
- Rzeczywista krawędź powinna być oznaczona tylko raz i jeśli tylko to możliwe nie powinien być wykrywany szum

Algorytm wykonuje się w 5 krokach:

1. Redukcja szumu poprzez zastosowanie filtra gausa
2. Szukanie natężenia gradientu obrazu przy użyciu np. operatora Sobla (Wzór 7) lub Prewitt'a (Wzór 6)
3. „Pocienianie” krawędzi, czyli usuwanie zbędnych pikseli tak, aby została z nich cienka linia
4. Progowanie krawędzi i histereza – polega na usunięciu krawędzi z nachyleniem poniżej dolnego progu i usunięciu krawędzi z nachyleniem poniżej górnego progu, które nie są połączone z krawędziami z nachyleniem powyżej górnego progu.

Tabela 2 Porównanie operatora Sobla, Laplac'a i algorytmu Canny'ego

 <p>Rysunek 10 Zdjęcie oryginalne przed wykrywaniem krawędzi [12]</p>	 <p>Rysunek 11 Zdjęcie po wykryciu krawędzi operatorem Laplac'a</p>
 <p>Rysunek 12 Zdjęcie po wykrywaniu krawędzi operatorem Sobla</p>	 <p>Rysunek 13 Zdjęcie po wykryciu krawędzi algorytmem Canny'ego</p>

Jak widać w Tabeli 2 wykrywanie krawędzi algorytmem Canny'ego daje lepsze wyniki w porównaniu do operatora Sobla i Laplac'a. Na obrazie nie ma szumu, nieistotne krawędzie zostały pominięte, a krawędzie są szerokości jednego piksela.

3. Projekt rozwiązania

W tym rozdziale zostanie przedstawione rozwiązanie, które zostało stworzone na potrzeby tej pracy. Na początku zostaną omówione biblioteki użyte w projekcie, następnie zostanie omówiony sposób realizacji makiety parkingu. Później zostanie omówiony projekt służący do przygotowania danych testowych. Następnie zostaną omówione cechy, które będą obliczane z miejsc parkingowych. Pod koniec zostanie omówiony klasyfikator użyty do rozpoznawania przynależności miejsc parkingowych do odpowiedniej klasy, a na koniec zostanie omówiony projekt, który testuje skuteczność całego algorytmu.

3.1. Biblioteki i technologie użyte podczas pisania algorytmu

OpenCV (Open Source Computer Vision) – Popularna biblioteka funkcji do rozpoznawania obrazów w czasie rzeczywistym. Posiada między innymi funkcje do manipulacji obrazami, wykrywania cech obrazu i uczenia maszynowego. W 1999 roku projekt OpenCV został zainicjalizowany przez firmę Intel. Biblioteka jest napisana w języku C++. Jako wrapper¹ dla platformy .NET wybrałem bibliotekę OpenCvSharp. API tej biblioteki jest bardzo zbliżone do oryginalnego API w języku C++. Wiele klas ma zaimplementowany interfejs *IDisposable*, dzięki czemu nie trzeba się martwić zwalnianiem pamięci obiektów z poza platformy .NET. Biblioteka dodatkowo posiada API, które pozwolą na wywołanie łańcuchowe metod².

WPF (Windows Presentation Foundation) – Framework do tworzenia interfejsu użytkownika (ang. user interface) dla platformy .NET, stworzony przez firmę Microsoft. WPF kładzie nacisk na grafikę wektorową, dzięki której większość kontrolek można skalować bez utraty jakości, czy pikselizacji. Wybór na tą technologię zapadł, ponieważ można w łatwy sposób projektować wygląd aplikacji, który jest zarazem elastyczny i dopasowuje się do wielkości okna. WPF pozwolą na tworzenie aplikacji przy użyciu wzorca

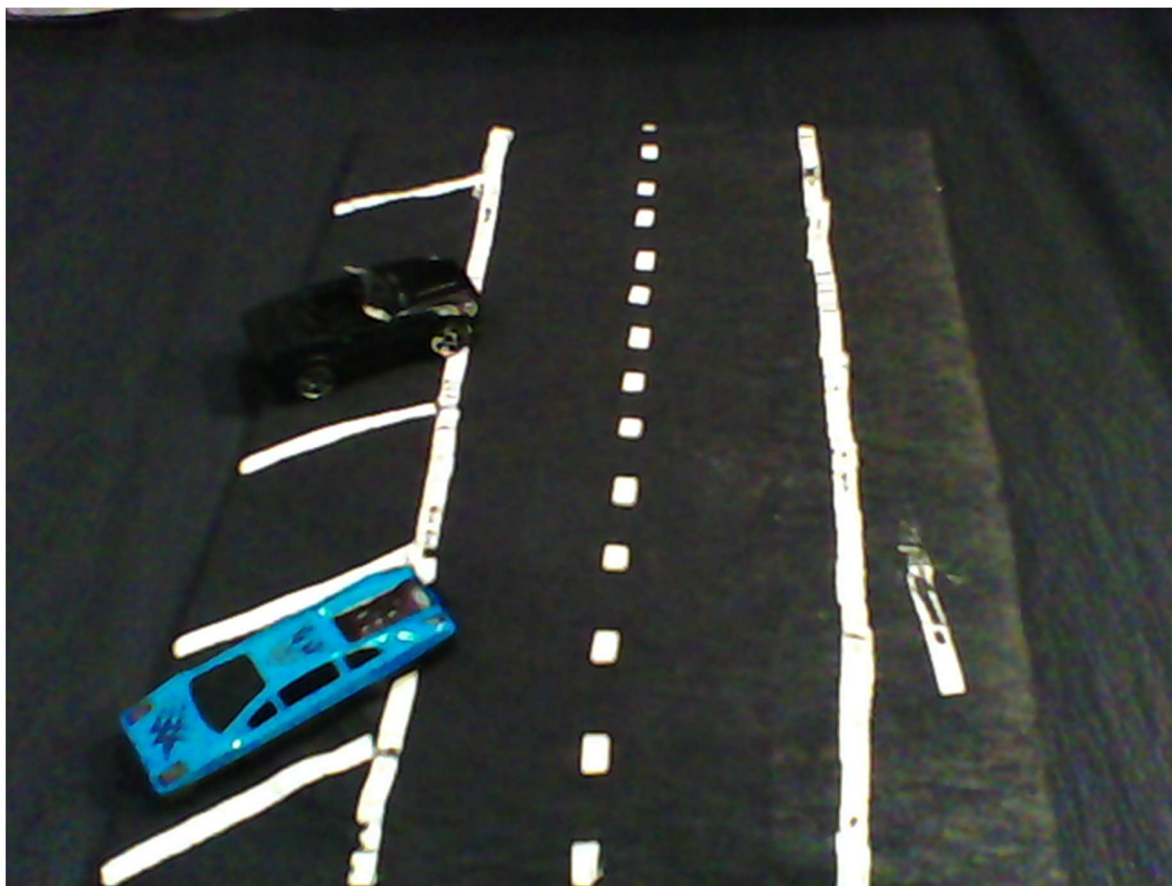
¹ Wrapper – biblioteka programistyczna, która opakowuje wywołania natywnych metod biblioteki w sposób ujednolicony z językiem, dla którego została przygotowana. Zazwyczaj posiada konwersję z popularnych typów w języku na typy, które obsługuje biblioteka

² Wołanie łańcuchowe (ang. method chaining) – polega na łączeniu wywołań metod na obiekcie. Każda metoda zwraca obiekt, który pozwala na wywołanie kolejnych metod w pojedynczej instrukcji (np. `person.SetName("Jan Kowalski").SetAge(35);`). Najczęściej osiąga się to poprzez zwracanie przez metody obiektu w kontekście, którego zostały wykonane (np. `return this;`). Drugim stosowanym podejściem jest zwracanie kopii obiektu w kontekście, którego została wywołana metoda, z zastosowanymi zmianami. Drugie podejście stosowane jest wtedy, kiedy zależy nam na zachowaniu obiektu oryginalnego lub obiekt jest niezmienny.

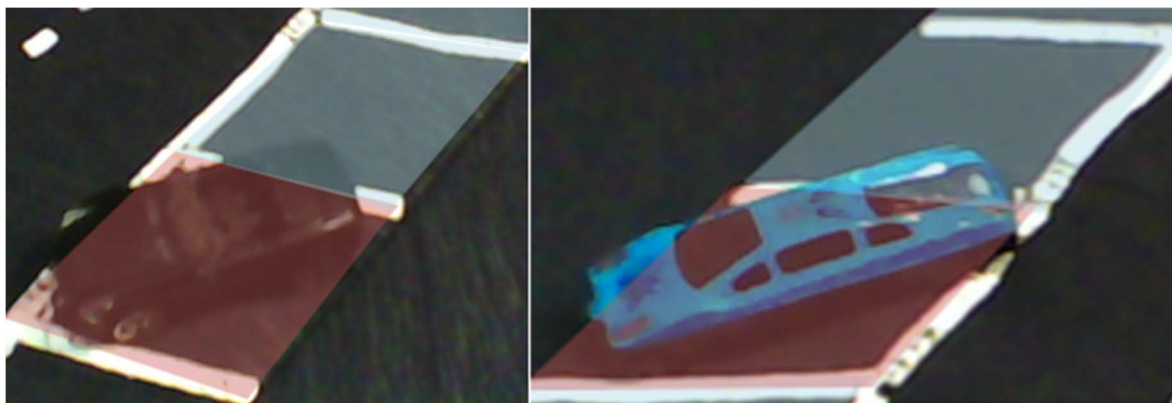
MVVM, który pozwala na separację kodu odpowiedzialnego za widok od logiki aplikacji. Nazwa MVVM pochodzi o inicjałów Model, Widok (ang. View), Model Widoku (ang. View Model). Jako model najczęściej określa się warstwę biznesową aplikacji lub warstwę dostępu do danych. Model widoku jest odpowiedzialny za komunikację widoku z modelem, poprzez przygotowanie danych z modelu dla widoku oraz udostępnia komendy, które służą do interakcji przez użytkownika. Widok jest to część aplikacji, która jest odpowiedzialna za prezentację wizualną aplikacji, wyświetlanie danych i obsługę interakcji użytkownika za pomocą zdefiniowanych komend zdefiniowanych w modelu widoku.

3.2. Makieta parkingu

Pierwsza wersja makiety parkingu była zrobiona z czarnej bibuły zawiniętej i oklejonej na sztywnej podkładce. Pasy miejsc parkingowych zostały narysowane korektorem w taśmie. Faktura bibuły miała odwzorowywać fakturę asfaltu. Jako modelu samochodów zostały użyte popularne zabawkowe resoraki w kilku kolorach, i jeden resorak w kolorze czarnym, który z założenia miał zlewać się z kolorem podłoża i stanowić przypadek pesymistyczny. Pierwszą makietę przedstawia Rysunek 14. Zdjęcia robione były przy użyciu kamery internetowej w rozdzielczości 640x480, model ustawiany był pod różnymi obrotami względem kamery. Zdjęcia były tak dobierane, aby były przypadki optymistyczne, gdzie samochód pozostaje w swoim miejscu parkingowym i takie gdzie kawałek samochodu zasłania kawałek miejsca parkingowego obok (Rysunek 15).



Rysunek 14 Zdjęcie pierwszej makiety

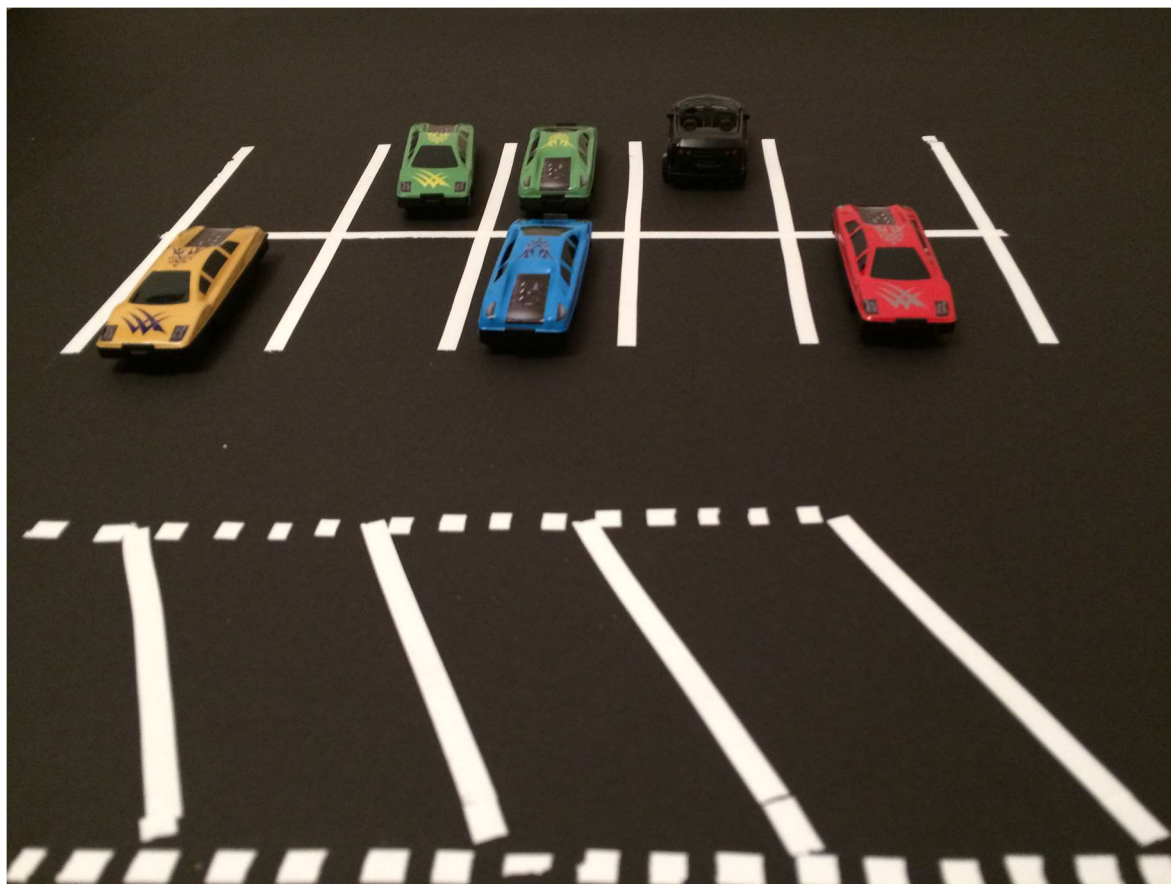


Rysunek 15 Przypadki pesymistyczne gdzie samochody wystają po za swój obszar

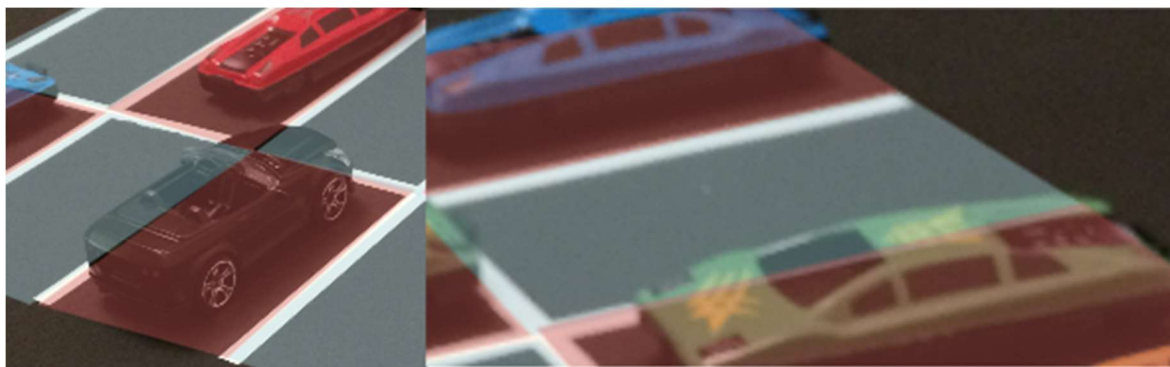
Takie podejście powodowało problemy. Faktura bibuły była wyraźnie widoczna ze względu na bliską odległość kamery od modelu, pomięta faktura bibuły utrudniała również rysowanie pasów korektorem w taśmie. Słabe oświetlenie sceny powodowało, iż samochody o kolorze czarnym były słabo widoczne dla ludzkiego oka na obrazach. Doświetlanie sceny powodowało, że samochody o kolorze czarnym zaczynały być widoczne, a algorytm wykrywania krawędzi zaczął wykrywać krawędzie na samochodzie, jednakże algorytm

wykrywania krawędzi znacząco więcej wykrywał pomarszczoną fakturę bibuły, a samochody kolorowe zaczynały być prześwietlone. Kamera przy słabej jakości oświetlenia zaczęła generować szum cyfrowy. Całość tych niedogodności wymagała bardzo precyzyjnego ustawienia oświetlenia sceny.

Druga wersja makiety powstała na brystolu o jednolitym ciemnoszarym kolorze. Z narysowanymi pasami korektorem w taśmie, zestaw samochodów resorków pozostał ten sam. Makietę przedstawia Rysunek 16. Zdjęcia zostały wykonane aparatem telefonu komórkowego o rozdzielczości 3264x2448. Zdjęcia zostały zrobione pod różnymi kontami względem modelu, w przeciwieństwie do poprzedniego zdjęcia mają również różne nachylenie względem modelu. Są zdjęcia robione z góry, gdzie samochody idealnie się wpasowują w miejsca parkingowe, jak i zdjęcia robione pod niskim kontem, gdzie samochody zasłaniają oboczne miejsca parkingowe (Rysunek 17).



Rysunek 16 Prezentacja drugiej makiety



Rysunek 17 Przypadki pesymistyczne na drugiej makiecie gdzie samochody wystają po za swoje miejsce

Drugie drugi model poprawiał znacząco jakość danych. Jednolita faktura brystolu eliminowała problem wykrywanych krawędzi. Dzięki sztywności brystolu nie potrzebna dodatkowych elementów usztywniających model, również rysowanie pasów korektorem w taśmie nie powodowało problemów. Dzięki zastosowaniu kamery o lepszych parametrach i lepszego źródła oświetlenia pomieszczenia, scena nie wymagała dodatkowego doświetlenia, a samochody o kolorze czarnym są wyraźnie widoczne na zdjęciach. Nie występuje również problem szumu cyfrowego.

3.3. Oznaczanie konturów dla miejsc parkingowych

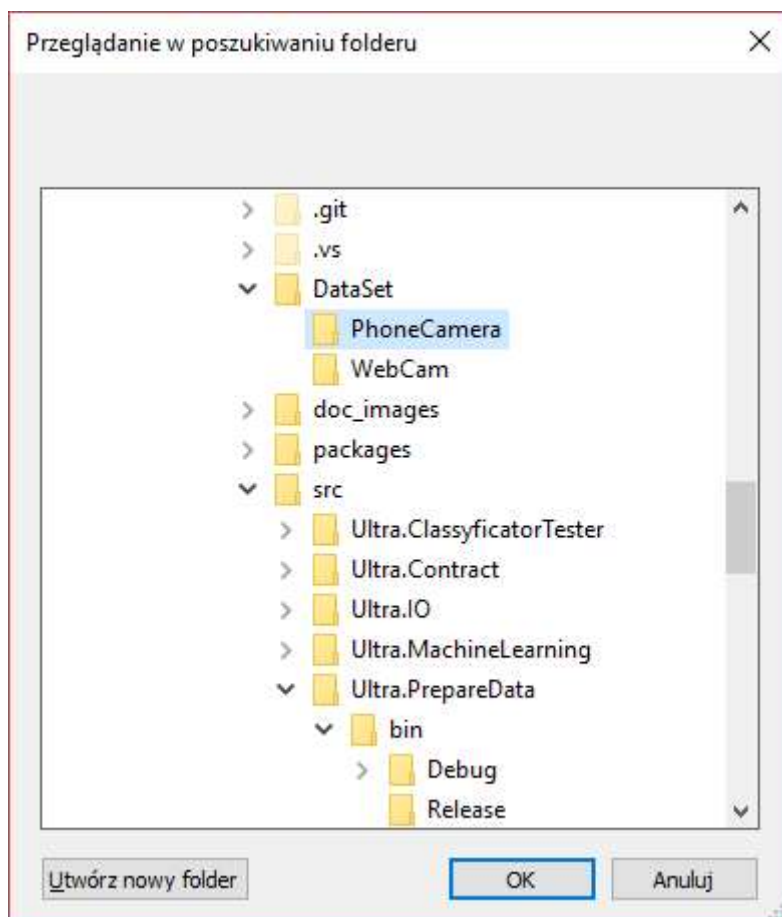
Do nauczania algorytmu rozpoznawania zdjęć trzeba dostarczyć zdjęcia w raz z oznaczonymi miejscami parkingowymi. Przyjęta konwencja zakłada że zdjęcia z rozszerzeniami *.png*, *.jpg*, *.jpeg*, *.bmp* znajdują się w jednym folderze, a oznaczenia miejsc parkingowych do danego zdjęcia znajdują się w pliku o nazwie identycznej jak zdjęcie i rozszerzeniem *.json*. Pliki z oznaczeniami zawiera listę obiektów klasy *ParkingSlot* przedstawioną w Listing 1 zakodowane w formacie *JSON*.

```
public class ParkingSlot
{
    public Contour Contour { get; set; }
    public bool IsOccupied { get; set; }
}

public class Contour : List<Contour.Point>
{
    public struct Point
    {
        public double X { get; set; }
        public double Y { get; set; }
    }
}
```

Listing 1 Klasa *Parking slot* i *Contour*

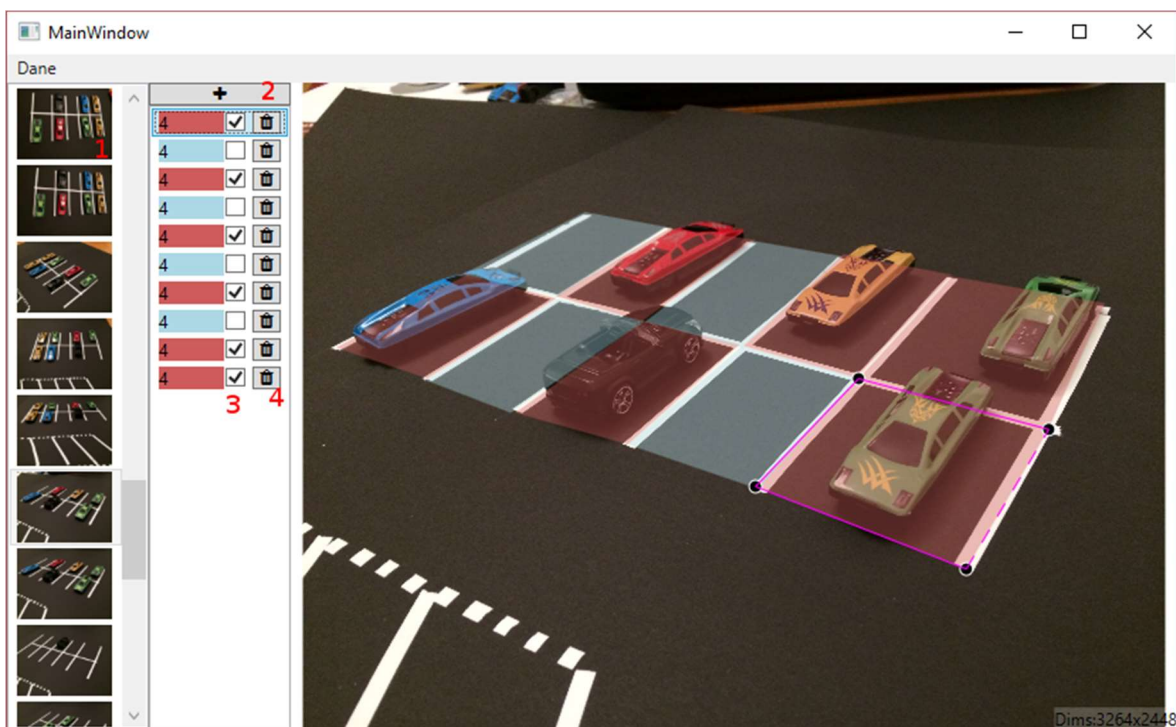
Do przygotowania danych uczących dla algorytmu powstał oddzielny program. Po otwarciu programu do przygotowania danych pojawi się okienko (Rysunek 18) z wyborem folderu gdzie przygotowane są zdjęcia testowe, można również użyć zbiorów zdjęć dołączonych razem z pracą. Jeśli w zbiorze danych były już zapisane kontury to program wczyta je automatycznie.



Rysunek 18 Wybieranie zbioru danych

Po wybraniu zbioru testowego ukaże nam się okno do oznaczania miejsc parkingowych. (Rysunek 19). Po lewej stronie okna jest lista do wybierania zdjęcia (oznaczone czerwoną jedynką). Możemy zmieniać aktywne zdjęcie myszką i skrótem klawiaturowym; **TAB**, aby zmienić zdjęcie na następne i **CTRL + TAB**, aby zmienić zdjęcie na poprzednie. Po prawej stronie od listy zdjęć znajduje się lista konturów dla obecnie zaznaczonego zdjęcia. Nowy kontur można dodać przy użyciu przycisku oznaczonym czerwonym numerem 2 lub skrótu klawiaturowego **A**. Punkty konturu dodaje poprzez klikanie lewym przyciskiem na obraz, punkty możemy przeciągać metodą przeciągnij i upuść (ang. drag and drop). Linia przerywaną zaznaczona jest krawędź pomiędzy pierwszym a ostatnim punktem konturu, po dodaniu nowego punktu ta krawędź zostanie zastąpiona nowymi dwiema. Aby usunąć punkt

wystarczy kliknąć go prawym przyciskiem myszy. Aby usunąć kontur wystarczy kliknąć ikonkę kosza przy odpowiednim konturze. Skrót klawiaturowy **D** usuwa obecnie zaznaczony kontur. Do oznaczania miejsca parkingowego jako wolne lub zajęte służy pole do zaznaczenia (ang. checkbox) oznaczony czerwonym numerem 3. Alternatywnie można użyć skrótów klawiaturowych **Q** do oznaczenia obecnie zaznaczonego miejsca jako wolne i **W** do miejsca zajętego. Aby zapisać wyniki oznaczania miejsc parkingowych należy wybrać z paska menu **Dane** następnie **Zapisz**, lub opcjonalnie można użyć popularnego skrótu klawiaturowego **CTRL + S**.



Rysunek 19 Okno do oznaczania konturów

3.4. Badane cechy obrazu

Łatwo można zauważyć na zdjęciach testowych, że miejsca parkingowe z samochodami zazwyczaj odróżniają się kolorem od zdjęć pustych. Bazując na tej obserwacji pierwszą badaną cechą jest badanie współczynnika liczności pikseli z chrominancją >100 (w skali $[0-255]$) do całości obszaru miejsca parkingowego. Listing 2 przedstawia sposób obliczania liczności pikseli z chrominancją większą niż 100. Jako argumenty funkcji przyjmowane są kontur miejsca parkingowego w postaci listy punktów definiującej wierzchołki wieloboku, i obiekt **Mat** reprezentujący zawierające miejsce parkingowe. Na początku wyliczamy region zainteresowania (ROI) obramowujący kontur, będzie później używany do wycięcia kawałka

obszaru, w którym znajduje się miejsce parkingowe. Następnie kontur jest przekształcany w maskę w postaci macierzy, po czym maska jest przycinana do obszaru zainteresowania i zamiany schematu kolorów w skali szarości. Następnie z obrazu wejściowego wycinamy obszar zainteresowania, później wycinamy część obrazu w obszarze zainteresowania, następnie zamieniamy schemat kolorów na HSV, następnie z obrazu w przestrzeni HSV wyliczana jest warstwa chrominancji, co dokładnie przedstawia (Listing 3), następnie stosujemy maskę na warstwie chrominancji, następnie stosujemy progowanie a na koniec zliczamy licznosc pikseli, które na warstwie nasycenia mają wartość nie zerową. Listing 4 przedstawia obliczanie liczności pikseli maski, która jest obliczana w analogiczny sposób do liczności pikseli nasyconych. Obliczanie proporcji pikseli nasyconych do całości obszaru przedstawia Listing 5

```
public static int CountChromatedPixels(Contour contour, Mat src)
{
    var rect = GetContourRect(contour, src.Height, src.Width);

    var mask = GetMask(contour, src.Size(), color: Scalar.White,
background: Scalar.Black)
        .Clone(rect)
        .CvtColor(ColorConversionCodes.BGR2GRAY);

    return src
        .Clone(rect)
        .CvtColor(ColorConversionCodes.BGR2HSV)
        .GetChromaLayer()
        .BitwiseAnd(mask)
        .Threshold(100, 255, ThresholdTypes.Binary)
        .CountNonZero();
}
```

Listing 2 Obliczanie liczności nasyconych pikseli

```
public static Mat GetChromaLayer(this Mat src)
{
    var mats = src.Split();
    return mats[1].Mul(mats[2], 1.0/255);
}
```

Listing 3 Skalowanie nasycenia z jasnością koloru

```
public static int CountMaskArea(Contour contour, Mat src)
{
    var rect = GetContourRect(contour, src.Height, src.Width);

    return GetMask(contour, src.Size(), color: Scalar.White, background:
Scalar.Black)
        .Clone(rect)
        .CountNonZero();
}
```

```

        .CvtColor(ColorConversionCodes.BGR2GRAY)
        .CountNonZero();
    }

```

Listing 4 Obliczanie liczności pikseli w masce

```

public float ChromatedPixelsRatio => (float) ChromatedPixels/MaskPixels;

```

Listing 5 Obliczanie współczynnika nasyconych pikseli względem obszaru

```

C:\Users\Sylwekqaz\Documents\Visual Studio 2015\Projects\Inz\src\Ultra.ClassificatorTester\bin\Debug\Ultra.ClassificatorTester.exe
Odświeżyć cache? Y/N: Y
[ ] 48 of 48 Liczba obserwacji: 351
Crossvalidacja 1000 iteracji , podział zbioru 70%-30%
+-----+-----+
| Predicted\Actual | True | False |
+-----+-----+
| True | 38914 | 2576 |
+-----+-----+
| False | 26177 | 37333 |
+-----+-----+

Sensitivity TPR: 0,597839947150912
Sensitivity TNR: 0,93545315592974
Accuracy ACC: 0,726161904761905

Walidacja N-1
+-----+-----+
| Predicted\Actual | True | False |
+-----+-----+
| True | 131 | 9 |
+-----+-----+
| False | 87 | 124 |
+-----+-----+

Sensitivity TPR: 0,600917431192661
Sensitivity TNR: 0,932330827067669
Accuracy ACC: 0,726495726495726

```



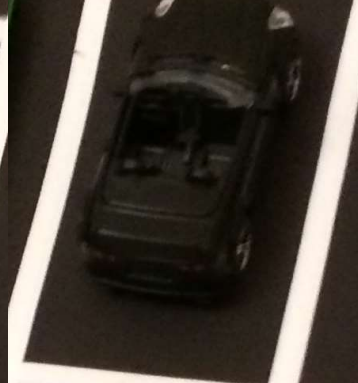
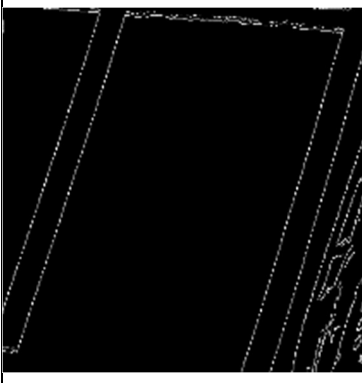
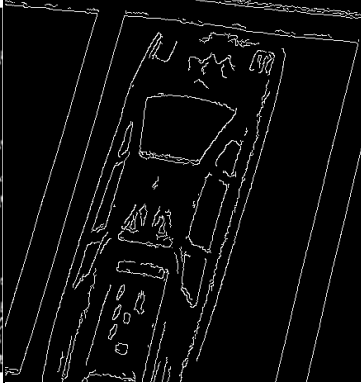
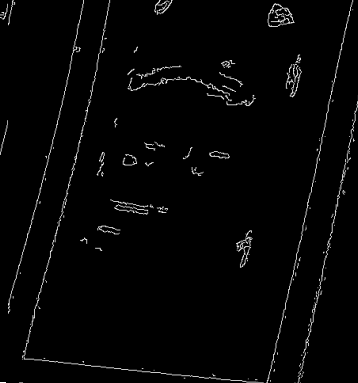
Rysunek 20 Wyniki testów klasyfikacji z cechą współczynnika pikseli z chrominancją >100

Rysunek 20 przedstawia wyniki walidacji dla klasyfikacji z użyciem wyżej opisanej cechy.

Cecha nie jest skuteczna przy wykrywaniu samochodów o kolorze karoserii, który posiada niskie nasycenie koloru (np. białym, czarnym, szarym) dając błędnie klasyfikując miejsca zajęte jako puste (błąd II typu). Szum cyfrowy generowany przez kamerę przy słabym oświetleniu może powodować, że puste miejsce może zostać zakwalifikowane jako miejsce zajęte (błąd I typu).

Tabela 3 przedstawia porównanie zdjęć, na których zastawano wykrywanie krawędzi metodą *Canny'ego*, można zauważyć, że miejsca parkingowe, na których znajdują się samochody dają widocznie więcej krawędzi od pustego miejsca parkingowego.

Tabela 3 Porównanie zdjęć po zastosowaniu algorytmu wykrywania krawędzi

	Puste miejsce	Kolorowy samochód	Czarny samochód
Zdjęcie oryginalne			
Po wykryciu krawędzi			

Na podstawie obserwacji powstała koncepcja obliczania kolejnej cechy. Wyliczany jest stosunek pikseli z krawędziami do całości obszaru. Listing 6 prezentuje algorytm obliczający tą cechę. Jako wejście przyjmowany jest kontur, czyli lista punktów reprezentujących wierzchołki wieloboku, w którym znajduje się miejsce parkingowe i obiekt Mat, który reprezentuje zdjęcie parkingu. Na początku wyliczany jest z konturu obszar zainteresowania (ROI). Następnie wyliczana jest maska z konturu i przycinana do obszaru zainteresowania. Następnie przycinamy obraz wejściowy do obszaru zainteresowania, następnie wykrywamy krawędzie bezparametrową metodą *Canny'ego*, następnie na zdjęciu stosowana jest maska i zliczane są piksele o wartości większej niż zero. Następnie obliczana jest liczność pikseli w masce, które zostało opisane przy obliczaniu poprzedniej cechy Listing 4. Obliczanie samego stosunku pikseli z krawędziami przedstawia Listing 7. Do wykrywania krawędzi używana jest bezparametrowa implementacja metody *Canny'ego*, która automatycznie dobiera progi, na podstawie średniej jasności koloru (Listing 8), metoda została opisana w [13].

```
public static int CountEdgePixels(Contour contour, Mat src)
```

```

{
    var rect = GetContourRect(contour, src.Height,src.Width);

    var mask = GetMask(contour, src.Size(), color: Scalar.White,
background: Scalar.Black)
        .Clone(rect)
        .CvtColor(ColorConversionCodes.BGR2GRAY);

    return src
        .Clone(rect)
        .DetectEdges()
        .BitwiseAnd(mask)
        .CountNonZero();
}

```

Listing 6 Obliczanie liczności pikseli z krawędziami

```

public float EdgePixelsRatio => (float) EdgePixels/MaskPixels;

```

Listing 7 Obliczanie stosunku pikseli z krawędziami do całego obszaru

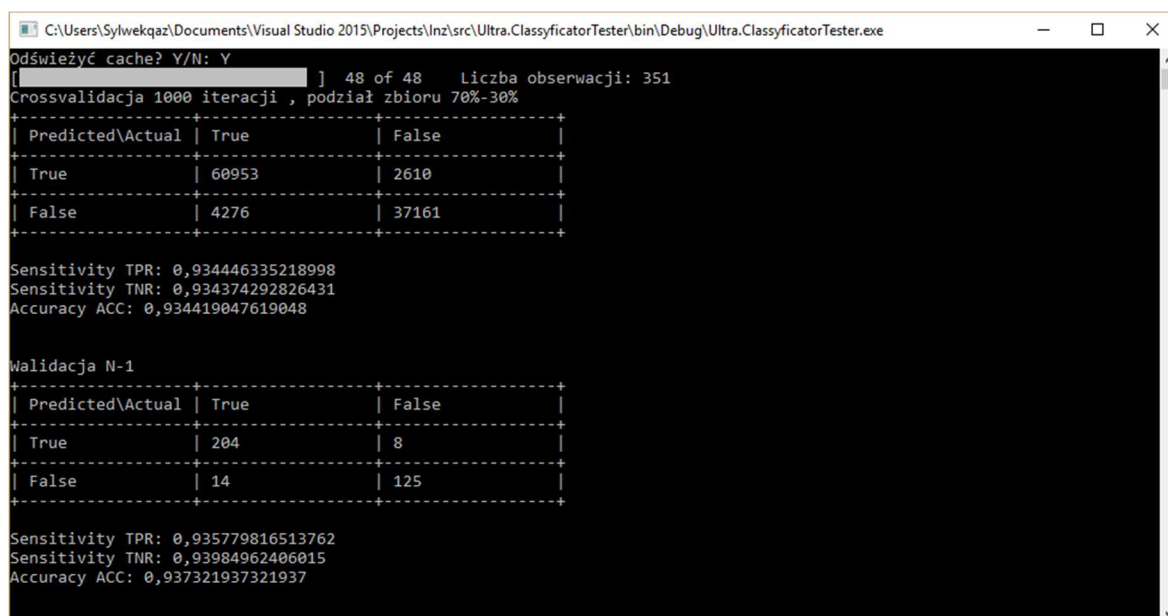
```

public static Mat DetectEdges(this Mat src, double sigma = 0.33)
{
    var graySrc = src.CvtColor(ColorConversionCodes.BGR2GRAY);
    Cv2.MeanStdDev(graySrc, out var meanScalar, out var stddevScalar);
    var mean = meanScalar[0];
    var lower = (int) Math.Max(0, (1.0 - sigma) * mean);
    var upper = (int) Math.Min(255, (1.0 + sigma) * mean);
    return graySrc.Canny(lower, upper);
}

```

Listing 8 Bezparametrowa detekcja krawędzi Canny'ego

Rysunek 21 Pokazuje wyniki sprawdzianu klasyfikacji.



Rysunek 21 Wyniki walidacji dla przy użyciu cech stosunku pikseli nasyconych i stosunku pikseli z krawędziami

Koncepcja na kolejne cztery cechy powstała podczas porównania histogramów jasności i saturacji zdjęć miejsc parkingowych. Tabela 4 przedstawia histogramy dla pustego miejsca parkingowego i zajętego przez samochód o kolorze żółtym i samochód o kolorze czarnym. Na histogramach zaznaczono czerwonym kolorem wartość średniej jasności. Łatwo zauważyć, że histogram niezajętego miejsca parkingowego ma bardzo małe odchylenie standardowe. Tabela 5 Przedstawia porównanie histogramów nasycenia dla zdjęć miejsc parkingowych zajętych przez czarny samochód, kolorowy samochód i pustego miejsca parkingowego. Kolorem czerwonym na histogramie zaznaczono średnią wartość nasycenia. Można zauważyć, że wartość odchylenia standardowego i średniego nasycenia dla miejsca parkingowego zauważalnie różni się od miejsc zajętych przez samochody.

Tabela 4 Porównanie histogramów jasności miejsc parkingowych


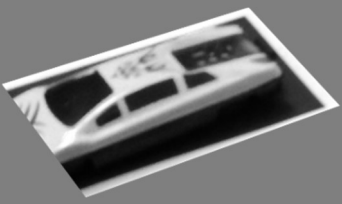
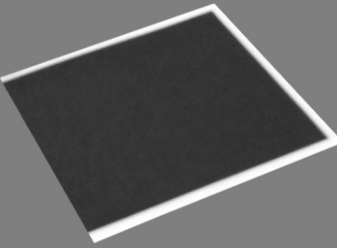
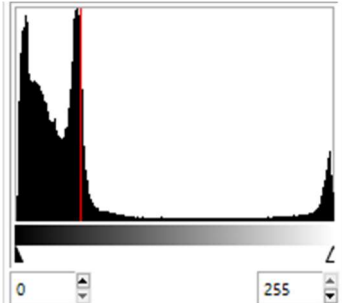
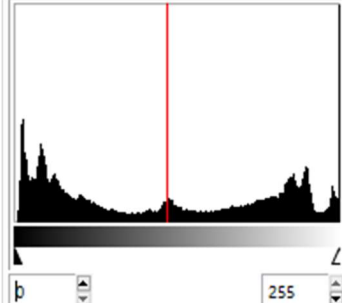
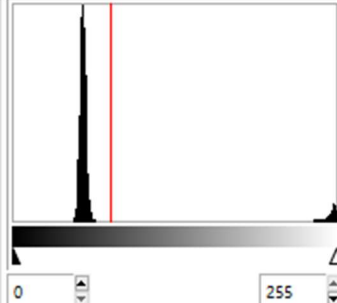
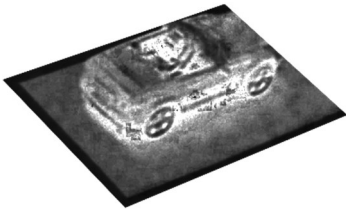
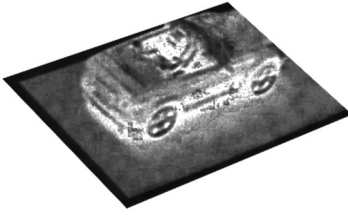
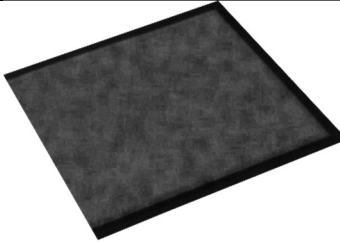
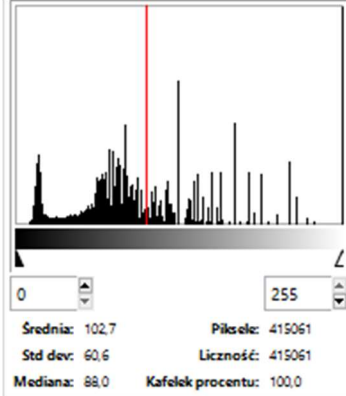
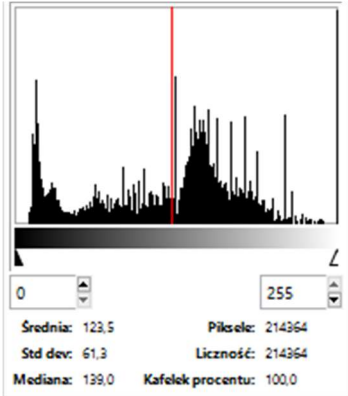
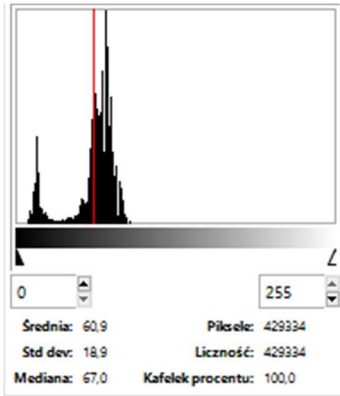
	Czarny samochód	Żółty samochód	Puste miejsce
Jasność z HSV			
Histogram jasności	 <p>Średnia: 52,2 Piksle: 413790 Std dev: 64,9 Liczność: 413790 Mediana: 34,0 Kafelek procentu: 100,0</p>	 <p>Średnia: 119,0 Piksle: 213453 Std dev: 87,8 Liczność: 213453 Mediana: 116,0 Kafelek procentu: 100,0</p>	 <p>Średnia: 77,3 Piksle: 427637 Std dev: 59,1 Liczność: 427637 Mediana: 55,0 Kafelek procentu: 100,0</p>

Tabela 5 Porównanie histogramów nasycenia dla miejsc parkingowych

	Czarny samochód	Żółty samochód	Puste miejsce
Nasycenie			
Histogram nasycenia	 Średnia: 102,7 Píksele: 415061 Std dev: 60,6 Licznořć: 415061 Mediana: 88,0 Kafelek procentu: 100,0	 Średnia: 123,5 Píksele: 214364 Std dev: 61,3 Licznořć: 214364 Mediana: 139,0 Kafelek procentu: 100,0	 Średnia: 60,9 Píksele: 429334 Std dev: 18,9 Licznořć: 429334 Mediana: 67,0 Kafelek procentu: 100,0

Obliczanie tych cech przedstawia Listing 9. Funkcja na wejřcie przyjmuje kontur reprezentowany poprzez listę punktów będućych wierzchołkami wieloboku, i obiekt *Mat* reprezentujący zdjęcie parkingu. Na poczátku z konturu wyznaczany jest obszar zainteresowania (ROI), następnie kontur jest przekształćany na maskę i przycinany do obszaru zainteresowania. Następnie obraz wejřciowy jest przycinany do obszaru zainteresowań, później zamieniany jest schemat kolorów z RGB na HSV, i przy użyciu funkcji *Split* obraz zostaje rozłożony na warstwy. Następnie kolejno warstwy nasycenia i jasnořci sę przekazana do lokalnej funkcji *LocalMeanStdDev* gdzie wyliczana jest řrednia i odchylenie standardowe z pojedynczej warstwy. Przed zwróceniem wartořci sę skalowane z przedziału [0-255] do przedziału [0-1].

```
public static ((float mean, float stddev) saturation, (float mean, float
stddev) value) GetHSVColorStats(
    Contour contour, Mat src)
{
    var rect = GetContourRect(contour, src.Height, src.Width);

    var mask = GetMask(contour, src.Size(), color: Scalar.White,
background: Scalar.Black)
        .Clone(rect)
```

```

        .CvtColor(ColorConversionCodes.BGR2GRAY);

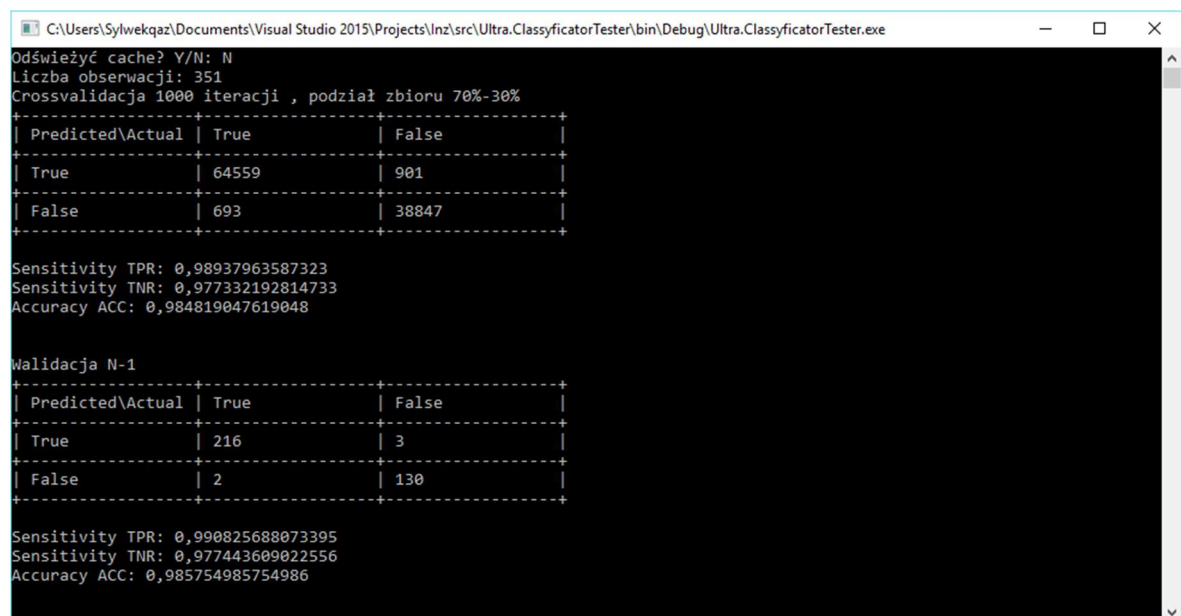
var layers = src.Clone(rect)
    .CvtColor(ColorConversionCodes.BGR2HSV)
    .Split();

(float mean, float stddev) LocalMeanStdDev(Mat area)
{
    Cv2.MeanStdDev(area, out var scalarMean, out var scalarStddev,
mask);
    var mean = (float) (scalarMean[0] / 255);
    var stddev = (float) (scalarStddev[0] / 255);
    return (mean, stddev);
}

return (LocalMeanStdDev(layers[1])/*saturation layer*/,
LocalMeanStdDev(layers[2])/* value layer*/);
}

```

Listing 9 Obliczanie statystyk średniej i odchylenia standardowego dla saturacji i wartości

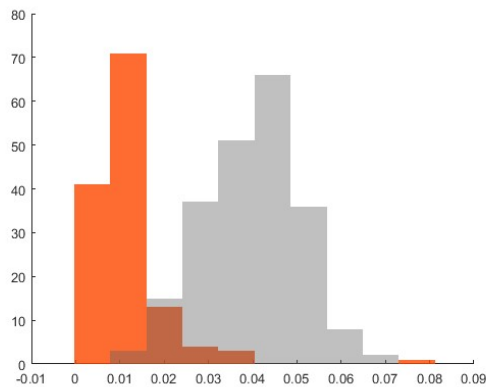


Rysunek 22 Wyniki sprawdzianu klasyfikacji dla cech: współczynnik krawędzi, współczynnik saturacji, średnia jasność, odchylenie jasności, średnie nasycenie, odchylenie standardowe nasycenia

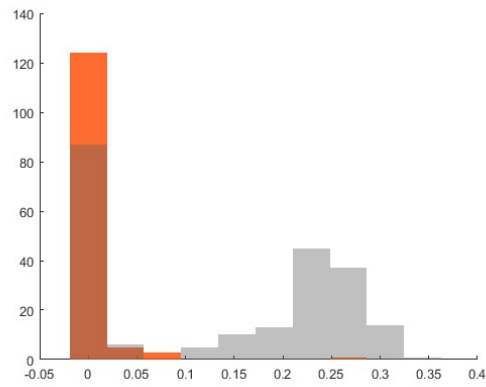
Rysunek 22 przedstawia wyniki sprawdzianu klasyfikacji z użyciem poprzednio omawianych cech. Dzięki ich wprowadzeniu jest kilkuprocentowy zysk w rozpoznawaniu miejsc parkingowych.

Poniżej rysunki przedstawiają histogramy porównujące dwie klasy dla 6 cech, kolorem pomarańczowym przedstawiono klasę miejsc wolnych, a szarym klasę miejsc zajętych. Tabela 6 przedstawia wykresy punktowe przedstawiające rzuty 6cio wymiarowej przestrzeni

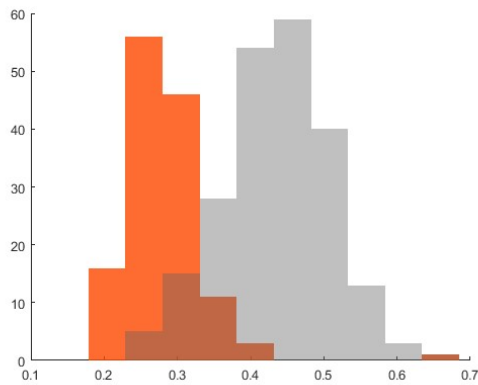
parametrów. Punkty o kolorze pomarańczowym prezentują wolne miejsca parkingowe, zajęte miejsca parkingowe zaznaczone są kolorem szarym.



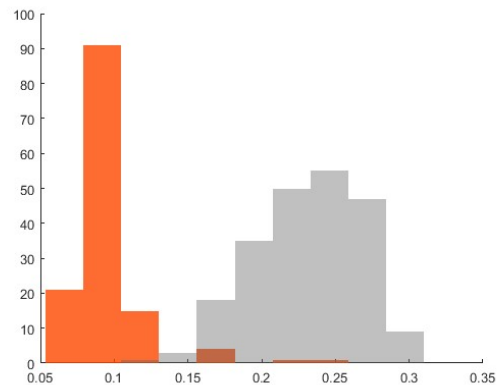
Rysunek 23 Histogram przedstawiający rozkład współczynnika krawędzi w dla danych testowych



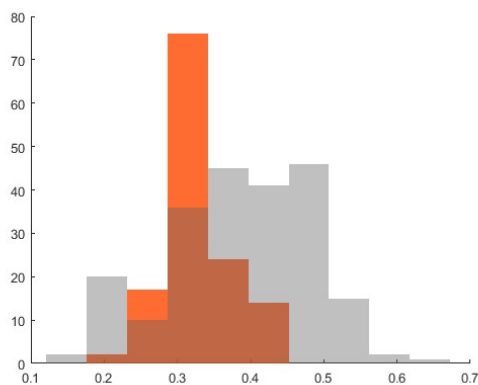
Rysunek 24 Histogram przedstawiający rozkład współczynnika chrominancji dla danych testowych



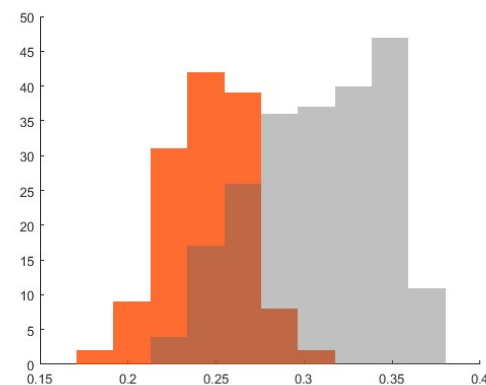
Rysunek 25 Histogram przedstawiający rozkład średniej saturacji dla danych testowych



Rysunek 26 Histogram przedstawiający rozkład odchylenia standardowego dla danych testowych

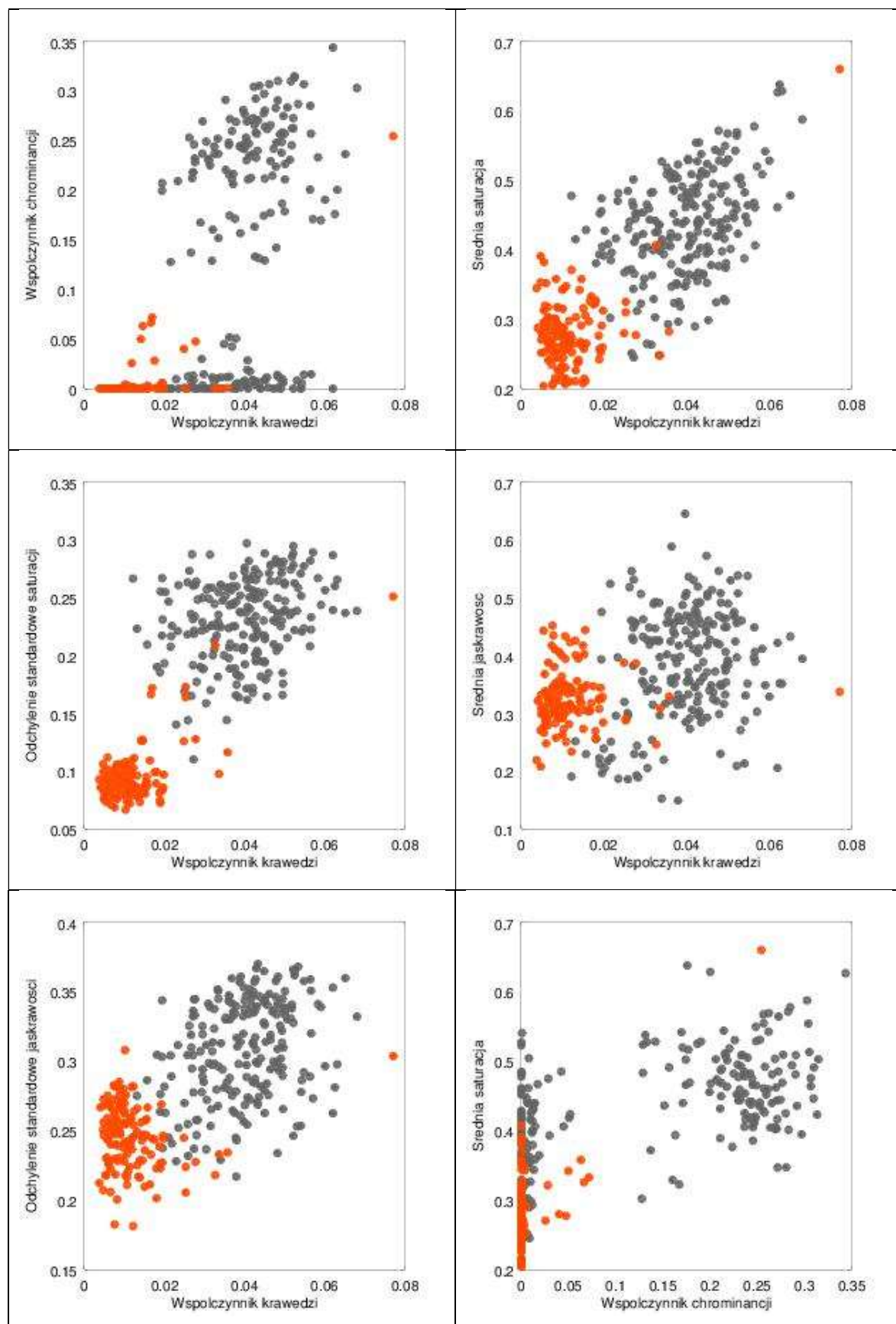


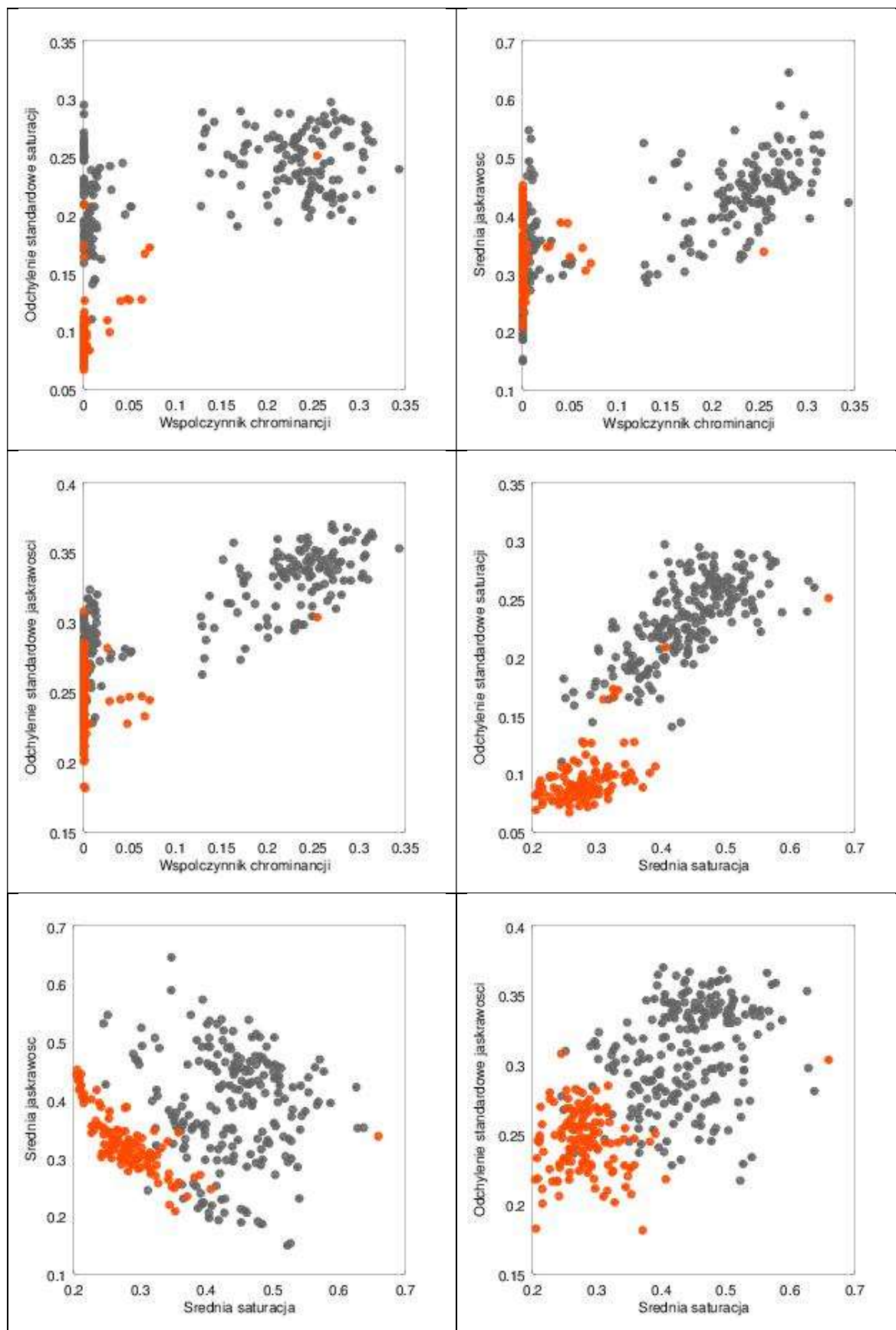
Rysunek 27 Histogram przedstawiający rozkład średniej jasności dla danych testowych

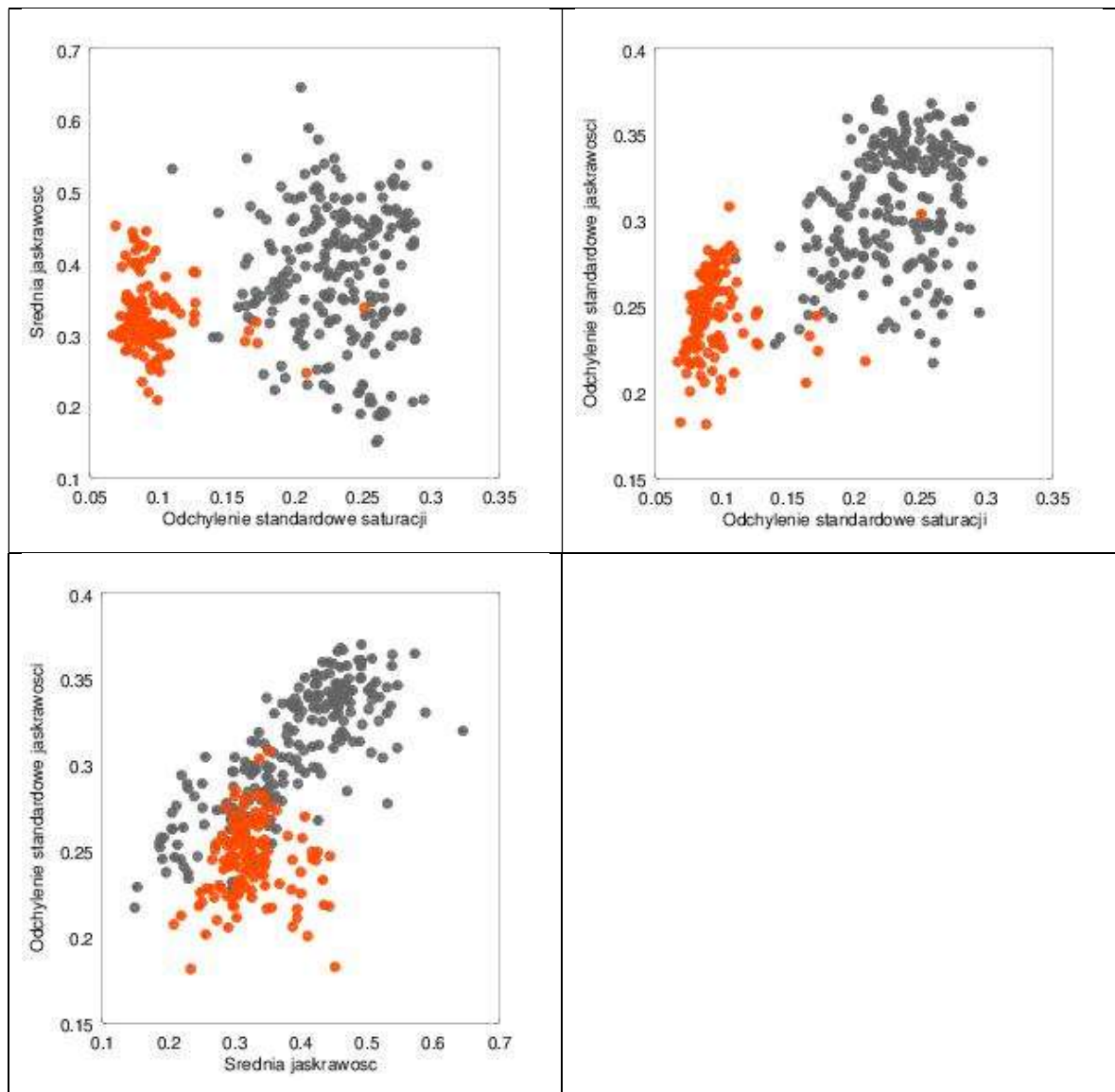


Rysunek 28 Histogram przedstawiający odchylenie standardowe jasności dla danych testowych

Tabela 6 Wizualizacja danych testowych w postaci serii dwuwymiarowych wykresów

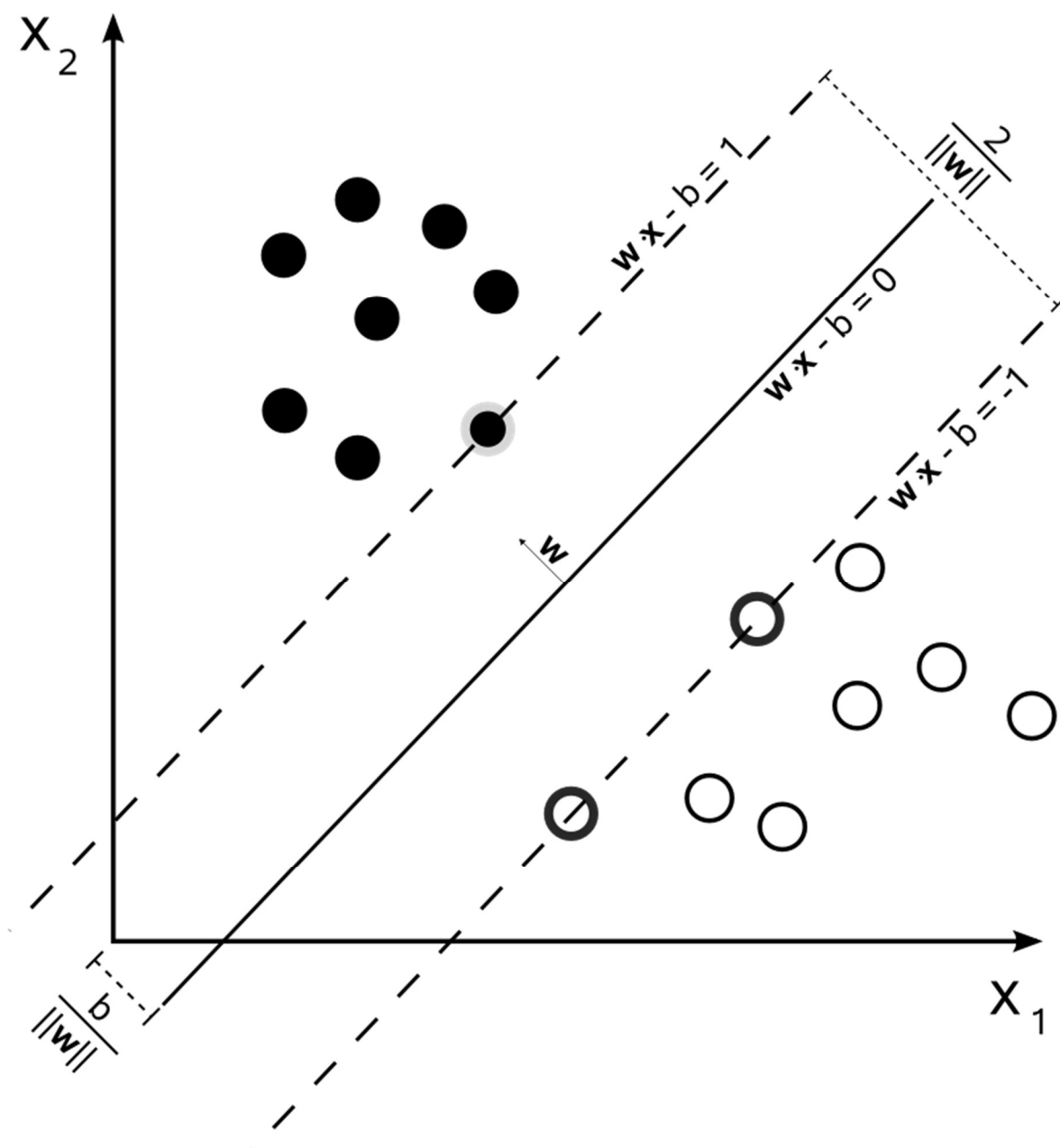




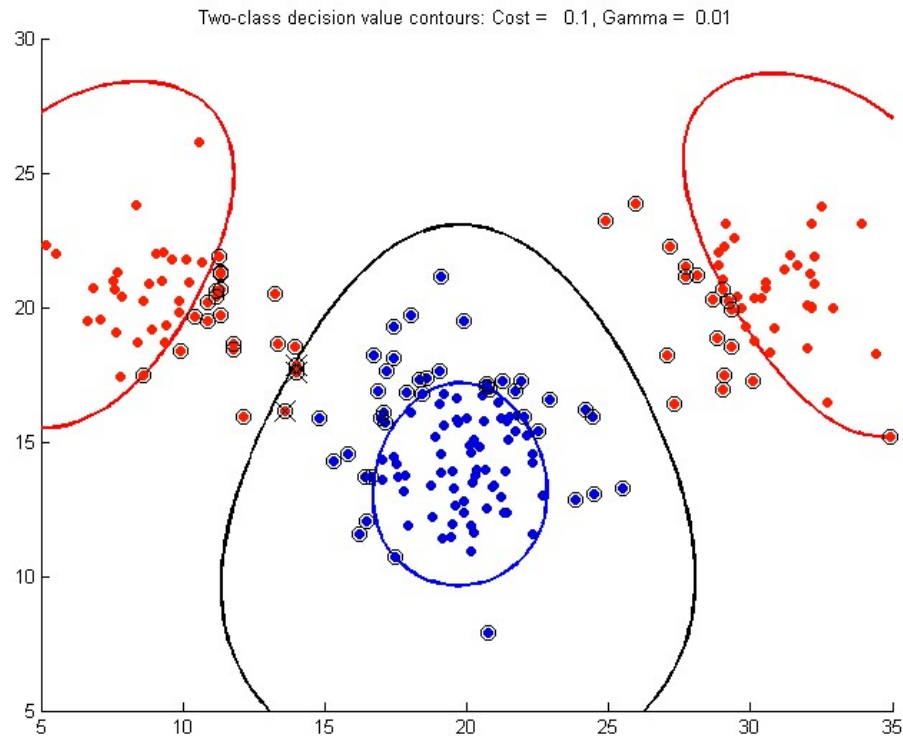


3.5. Klasyfikator

Jako klasyfikatora w algorytmie używam maszyny wektorów nośnych (ang. support vector machine, SVM). SVM próbuje dopasować hiperpłaszczyznę do danych testowych, tak, aby rozdzielić obie klasy od siebie. Hiperpłaszczyzna jest dobierana tak, aby margines (czyli odległość dowolnego punktu od hiperpłaszczyzny) był jak największy (Rysunek 29). SVM potrafi rozdzielać również dane, które nie da się rozdzielić w sposób liniowy, poprzez zastosowanie funkcji jądra, która dodaje dodatkowe wymiary w przestrzeni cech. Rysunek 30 przedstawia przykładowe rozdzielenie dwóch klas przy zastosowaniu gaussowskiej funkcji jądra.



Rysunek 29 SVM margines [14]



Rysunek 30 SMV z jądrem RBF [15]

W algorytmie używana jest gaussowska funkcja jądra. Dzięki podejściu nieliniowemu dostajemy kilkuprocentowy zysk w poprawności klasyfikacji (Rysunek 31).

Crossvalidacja 1000 iteracji , podział zbioru 70%-30%				Crossvalidacja 1000 iteracji , podział zbioru 70%-30%			
Predicted\Actual	True	False		Predicted\Actual	True	False	
True	62434	593		True	64559	901	
False	2649	39324		False	693	38847	
Sensitivity TPR: 0,959298127007053 Sensitivity TNR: 0,985144174161385 Accuracy ACC: 0,96912380952381				Sensitivity TPR: 0,98937963587323 Sensitivity TNR: 0,977332192814733 Accuracy ACC: 0,984819047619048			
Linear kernel				RBF kernel			
Walidacja N-1				Walidacja N-1			
Predicted\Actual	True	False		Predicted\Actual	True	False	
True	212	2		True	216	3	
False	6	131		False	2	130	
Sensitivity TPR: 0,972477064220184 Sensitivity TNR: 0,984962406015038 Accuracy ACC: 0,97720797207977				Sensitivity TPR: 0,990825688073395 Sensitivity TNR: 0,977443609022556 Accuracy ACC: 0,985754985754986			

Rysunek 31 Różnica poprawności klasyfikacji pomiędzy jądrem RBF i liniowym jądrem

Listing 10 przedstawia klasę opakowującą klasyfikator SVM. W funkcji *Create* tworzony jest nowy klasyfikator i ustawiane są wartości parametrów *Gamma* i *C*, które są używane w

optymalizacji wyznaczania hiperpłaszczyzn. Następnie klasyfikator jest uczony zbiorem test uczącym przekazanym, jako parametr wejściowy metody. Funkcja *Predict* służy do przewidywania klasy dla nowych obserwacji. Jako parametr wejściowy przyjmuje klasę przechowującą cechy miejsca parkingowego, następnie zamienia je na wewnętrzny obiekt używany przez OpenCV i wykonywana jest klasyfikacja. Wynik klasyfikacji jest konwertowana na wartość prawda/fałsz.

```
public class SVMClassifier : IClassifier
{
    private readonly SVM _svm;

    private SVMClassifier(SVM svm)
    {
        _svm = svm;
    }

    public bool Predict(ImageFeatures imageFeatures)
    {
        return
Convert.ToBoolean(_svm.Predict(imageFeatures.ToPredictionMat()));
    }

    public static SVMClassifier Create(List<ImageFeatures>
trainingData)
    {
        var svm = SVM.Create();
        svm.Type = SVM.Types.CSvc;
        svm.KernelType = SVM.KernelTypes.Rbf;
        svm.TermCriteria = TermCriteria.Both(maxCount: 1000,
epsilon: 0.000001);
        svm.Gamma = 100.0;
        svm.C = 1.0;

        svm.Train(trainingData.ToTrainingMat(),
SampleTypes.RowSample, trainingData.ToResponseMat());

        return new SVMClassifier(svm);
    }
}
```

Listing 10 Konfiguracja klasyfikatora SVM

3.6. Walidacja jakości klasyfikacji

W programie testującym poprawność dokładność klasyfikatora wykonuje dwa sprawdzenia.

Pierwszym sprawdzianem jest 1000-krotny sprawdzian krzyżowy. Polega on na 1000-krotnym podziale prób zbioru na dwa podzbiory – uczący i testowy w proporcjach ustalonych arbitralnie 70% zbioru uczący, 30% zbioru testowy. Zbiorem uczącym jest uczony klasyfikator, a wyniki są sprawdzane na zbiorze testowym i zapisywane do macierzy błędów. Powstałe macierze błędów są sumowane do jednej macierzy. Listing 11 przedstawia k-krotną walidację, jako parametry wejściowe przyjmuje zbiór prób, ilość iteracji i proporcję, w jakich ma podzielić zbiór na testowy i uczący. Do obliczania pojedynczej iteracji wykorzystywana jest funkcja *Validate*, którą przedstawia Listing 12.

```
public static ConfusionMatrix CrossValidation(List<ImageFeatures>
observations, int iterations, double splitRatio)
{
    var summaryConfusionMatrix = new ConfusionMatrix();
    for (int i = 0; i < iterations; i++)
    {
        var tuple = observations.Shuffle().Split(splitRatio);
        var iterationConfusionMatrix = Validate(tuple.Item1,
tuple.Item2);
        summaryConfusionMatrix += iterationConfusionMatrix;
    }
    return summaryConfusionMatrix;
}
```

Listing 11 K-krotna walidacja krzyżowa

```
public static ConfusionMatrix Validate(List<ImageFeatures> train,
List<ImageFeatures> validation)
{
    var svmClassifier = SVMClassifier.Create(train);

    var confusionMatrix = new ConfusionMatrix();
    foreach (var validationObservation in validation)
    {
        var predict = svmClassifier.Predict(validationObservation);
        confusionMatrix.AddVote(actual:
validationObservation.IsOccupied, predicted: predict);
    }
    return confusionMatrix;
}
```

Listing 12 Walidacja klasyfikatora przy użyciu dowolnego zbioru testowego i uczącego

Drugim sprawdzianem jest sprawdzian *Leave-one-out*, który jest odmianą k-krotnej walidacji krzyżowej. Różnicą jest to, że nie dzieli on prób według proporcji, ale oddziela jedną próbę od pozostałych; pojedyncza próba służy jako zbiór testowy, a pozostałe próby jako zbiór uczący. Krotność walidacji jest równa ilości prób w początkowym zbiorze. Listing 13 przedstawia sposób obliczania tej walidacji, do obliczania pojedynczej iteracji używana jest wcześniej przedstawiona funkcja *Validate*, którą prezentuje Listing 12.

```

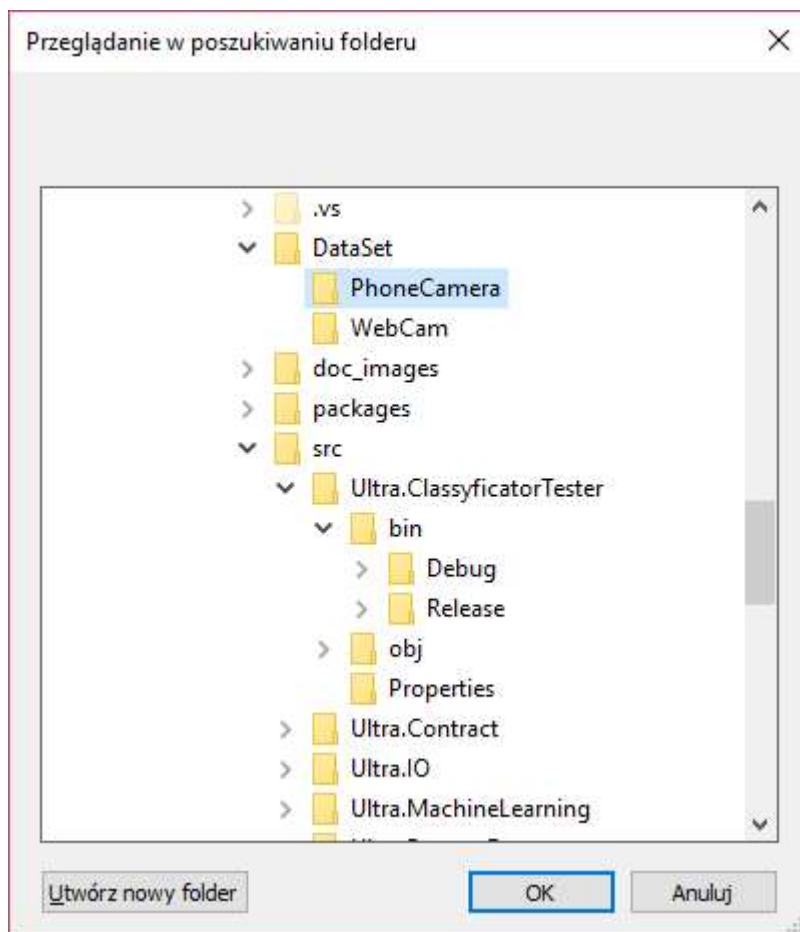
public static ConfusionMatrix LeaveOneOutValidation(List<ImageFeatures>
observations)
{
    var confumaMatrix = new ConfusionMatrix();
    for (var i = 0; i < observations.Count; i++)
    {
        var validation = new List<ImageFeatures> {observations[i]};
        var train = observations.WithoutElementAt(i);
        confumaMatrix += Validate(train, validation);
    }

    return confumaMatrix;
}

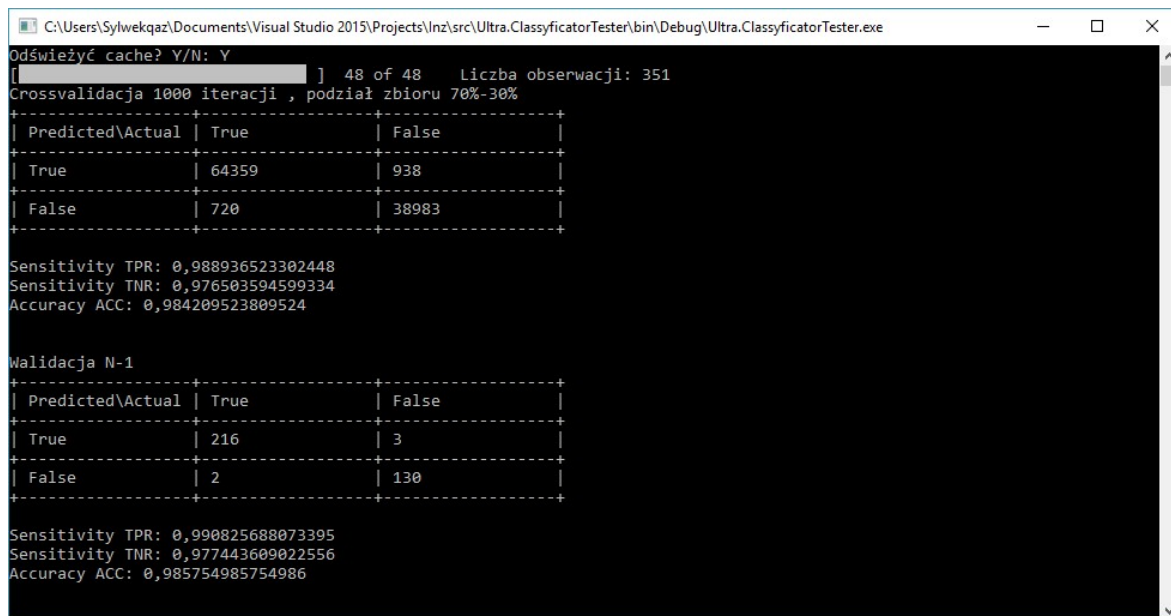
```

Listing 13 Walidacja Leave One Out

Po włączeniu programu do testowania klasyfikatora zostaniemy zapytani o wybranie folderu z przygotowanymi danymi testowymi (Rysunek 32). Program zapisuje obliczone cechy dla prób w folderze z danymi testowymi, tworząc mechanizm pamięci podręcznej (ang. cache) dla optymalizacji obliczeń wykonywanych przy kolejnych wywołaniach na tym samym zbiorze. Pamięć podręczna dla zbioru jest zapisywana w pliku *cachedFeatures.csv*. Po wybraniu folderu z danymi zostaniemy zapytani czy chcemy, aby program odświeżył pamięć podręczną zapisaną w pliku, co widać w pierwszej linijce na Rysunek 33. Jeżeli wybierzemy nie, program spróbuje odnaleźć plik z pamięcią podręczną, odczytać dane i przekaże je dalej do testerów klasyfikatorów. Jeżeli wybierzemy, aby program odświeżył cache, lub plik z pamięcią podręczną nie zostanie znaleziony lub jest uszkodzony, wtedy program wczyta kolejno zdjęcia wraz z oznaczeniami miejsc parkingowych i obliczy ich cechy. Następnie zapisze dane do pamięci podręcznej i zwróci próby do testerów klasyfikatorów. Z racji, iż przetwarzanie może być operacją czasochłonną program wyświetla pasek postępu widoczny w drugim wierszu programu na Rysunek 33. Następnie program wypisze ilość załadowanych obserwacji i obliczy skuteczność klasyfikatorów sposób opisany wcześniej w tym rozdziale.



Rysunek 32 Wybór folderu z danymi do programu walidującego klasyfikator



Rysunek 33 Okno programu do walidacji klasyfikatora

4. Podsumowanie

Celem pracy było stworzenie i przetestowanie algorytmu do klasyfikacji miejsc parkingowych jako wolne lub zajęte na podstawie przygotowanych zdjęć z makiety parkingu. Cele założone we wstępie pracy zostały osiągnięte, algorytm został opracowany, podczas testowania na zdjęciach wykonanych na makietach testowych osiągał wysoką skuteczność na poziomie 98% (Rysunek 34), również na zdjęciach z pierwszej makiety testowej, której zdjęcia nie były najlepsze. Warto zauważyć, że na pierwszej makiecie zostało wykonanych tylko 21 zdjęć, na których są 83 miejsca parkingowe, więc algorytm osiąga tak dobrą skuteczność na tak małej ilości danych. Niestety podczas porównywania wyników algorytmu na dwóch zbiorach na raz ta skuteczność spada. Na Rysunek 35 można zauważyć, że gdy nauczymy klasyfikator danymi z drugiej makiety, to algorytm całkowicie nie radzi sobie z rozpoznawaniem miejsc parkingowych (pierwsza macierz błędów na Rysunek 35), jeżeli jednak nauczymy klasyfikator danymi z pierwszej makiety i przetestujemy na danych z drugiej makiety, dostajemy 70% skuteczność rozpoznawania, co jest miernym wynikiem (druga macierz błędów na Rysunek 35). Jeżeli jednak oba zbiory danych połączymy i przetestujemy to w dotychczasowy sposób otrzymujemy wyniki na poziomie 98% (trzecia i czwarta macierz błędów na Rysunek 35), co może oznaczać, że dane testowe są mało różnorodne i dochodzi do przeuczenia się algorytmu.

Odświeżyć cache? Y/N: Y [] 21 of 21 Liczba obserwacji: 83 Crossvalidacja 1000 iteracji , podział zbioru 70%-30%		Odświeżyć cache? Y/N: Y [] 48 of 48 Liczba obserwacji: 351 Crossvalidacja 1000 iteracji , podział zbioru 70%-30%	
Predicted\Actual		Predicted\Actual	
True		True	
False		True	
True		False	
False		False	
Sensitivity TPR: 0,973964960360184		Sensitivity TPR: 0,989324608696988	
Sensitivity TNR: 0,994588378542921		Sensitivity TNR: 0,974935458806427	
Accuracy ACC: 0,98616		Accuracy ACC: 0,983857142857143	
Walidacja N-1		Walidacja N-1	
Predicted\Actual		Predicted\Actual	
True		True	
False		True	
True		False	
False		False	
Sensitivity TPR: 0,970588235294118		Sensitivity TPR: 0,990825688073395	
Sensitivity TNR: 1		Sensitivity TNR: 0,977443609022556	
Accuracy ACC: 0,987951807228916		Accuracy ACC: 0,985754985754986	

Rysunek 34 Porównanie wyników działania algorytmu na obu zbiorach testowych

```

Liczba obserwacji: 351
Liczba obserwacji: 83
Second set validated against first set
+-----+-----+
| Predicted\Actual | True       | False      |
+-----+-----+
| True             | 34         | 41         |
+-----+-----+
| False            | 0          | 8          |
+-----+-----+

Sensitivity TPR: 1
Sensitivity TNR: 0,163265306122449
Accuracy ACC: 0,506024096385542

First set validated against second set
+-----+-----+
| Predicted\Actual | True       | False      |
+-----+-----+
| True             | 215        | 101        |
+-----+-----+
| False            | 3          | 32         |
+-----+-----+

Sensitivity TPR: 0,986238532110092
Sensitivity TNR: 0,240601503750398
Accuracy ACC: 0,703703703703704

Crossvalidacja 1000 iteracji , podział zbioru 70%-30%
+-----+-----+
| Predicted\Actual | True       | False      |
+-----+-----+
| True             | 74613      | 1144       |
+-----+-----+
| False            | 1039       | 53204      |
+-----+-----+

Sensitivity TPR: 0,986266060381748
Sensitivity TNR: 0,978950467358504
Accuracy ACC: 0,983207692307692

Walidacja N-1
+-----+-----+
| Predicted\Actual | True       | False      |
+-----+-----+
| True             | 249        | 3          |
+-----+-----+
| False            | 3          | 179        |
+-----+-----+

Sensitivity TPR: 0,988095238095238
Sensitivity TNR: 0,983516483516483
Accuracy ACC: 0,986175115207373

```

Rysunek 35 Sprawdzenie skuteczności na obu zbiorach

5. Bibliografia

- [1] Rolls-Royce, „Rolls-Royce future shore control centre - YouTube,” [Online]. Available: <https://www.youtube.com/watch?v=vg0A9Ve7SxE>. [Data uzyskania dostępu: 27 Styczeń 2017].
- [2] Tesla Motors, „Autopilot | Tesla,” [Online]. Available: <https://www.tesla.com/autopilot>. [Data uzyskania dostępu: 27 Styczeń 2017].
- [3] R. Tadeusiewicz i M. Flasiński, Rozpoznawanie obrazów, Warszawa: Państwowe Wydawnictwo Naukowe, 1991.
- [4] H. Palus, „Representations of colour images in different colour spaces,” w *The Colour Image Processing Handbook*, SPRINGER-SCIENCE+BUSINESS MEDIA, B.V, 1998, pp. 67-90.
- [5] „HSL and HSV - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV#Saturation. [Data uzyskania dostępu: 27 Styczeń 2017].
- [6] SharkD, „HSV color solid cone chroma gray.png – Wikipedia, wolna encyklopedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Plik:HSV_color_solid_cone_chroma_gray.png. [Data uzyskania dostępu: 16 Styczeń 2017].
- [7] SharkD, „HSL color solid dblcone chroma gray.png – Wikipedia, wolna encyklopedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Plik:HSL_color_solid_dblcone_chroma_gray.png. [Data uzyskania dostępu: 16 Styczeń 2017].
- [8] SharkD, „HSV color solid cylinder alpha lowgamma.png – Wikipedia, wolna encyklopedia,” [Online]. Available:

- https://pl.wikipedia.org/wiki/Plik:HSV_color_solid_cylinder_alpha_lowgamma.png.
[Data uzyskania dostępu: 16 Styczeń 2017].
- [9] SharkD, „HSL color solid cylinder alpha lowgamma.png – Wikipedia, wolna encyklopedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Plik:HSL_color_solid_cylinder_alpha_lowgamma.png.
[Data uzyskania dostępu: 16 Styczeń 2017].
- [10] Jacobolus, „Hsl-hsv saturation-lightness slices.svg – Wikipedia, wolna encyklopedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Plik:Hsl-hsv_saturation-lightness_slices.svg. [Data uzyskania dostępu: 16 Styczeń 2017].
- [11] W. Malina i M. Smiatacz, Metody Cyfrowego Przetwarzania Obrazów, Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2005.
- [12] „Young woman sitting on ledge at looking at sea Photo - Visual Hunt,” [Online]. Available: <https://visualhunt.com/photo/1160/young-woman-sitting-on-ledge-at-looking-at-sea/>. [Data uzyskania dostępu: 25 Styczeń 2017].
- [13] R. Adrian , „Zero-parameter, automatic Canny edge detection with Python and OpenCV - PyImageSearch,” [Online]. Available: <http://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/>. [Data uzyskania dostępu: 25 Styczeń 2017].
- [14] P. Buch, „Svm max sep hyperplane with margin.png – Wikipedia, wolna encyklopedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Plik:Svm_max_sep_hyperplane_with_margin.png.
[Data uzyskania dostępu: 25 Styczeń 2017].
- [15] Donal, „C0p1g0p01.png - Eigenvector Documentation Wiki,” [Online]. Available: <http://wiki.eigenvector.com/index.php?title=File:C0p1g0p01.png>. [Data uzyskania dostępu: 25 Styczeń 2017].
- [16] D. Conway i J. M. White, Uczenie maszynowe dla programistów, Helion, 2015.

6. Spis obrazków

Rysunek 1 Stożek HSV [6]	11
Rysunek 2 Podwójny stożek HSL [7]	11
Rysunek 3 Cylinder HSV [8]	11
Rysunek 4 Cylinder HSL [9]	11
Rysunek 5 Pochodzenie nasycenia koloru z chrominancji i jasności/jaskrawości w modelach HSL i HSV [10]	12
Rysunek 6 Zdjęcie oryginalne przed wykrywaniem krawędzi [12]	16
Rysunek 7 Zdjęcie po wykrywaniu krawędzi operatorem Prewitt'a	16
Rysunek 8 Zdjęcie po wykryciu krawędzi operatorem Laplac'a	16
Rysunek 9 Zdjęcie po wykrywaniu krawędzi operatorem Sobela	16
Rysunek 10 Zdjęcie oryginalne przed wykrywaniem krawędzi [12]	17
Rysunek 11 Zdjęcie po wykryciu krawędzi operatorem Laplac'a	17
Rysunek 12 Zdjęcie po wykrywaniu krawędzi operatorem Sobla	17
Rysunek 13 Zdjęcie po wykryciu krawędzi algorytmem Canny'ego	17
Rysunek 14 Zdjęcie pierwszej makiety	20
Rysunek 15 Przypadki pesymistyczne gdzie samochody wystają po za swój obszar	20
Rysunek 16 Prezentacja drugiej makiety	21
Rysunek 17 Przypadki pesymistyczne na drugiej makiecie gdzie samochody wystają po za swoje miejsce	22
Rysunek 18 Wybieranie zbioru danych	23
Rysunek 19 Okno do oznaczania konturów	24
Rysunek 20 Wyniki testów klasyfikacji z cechą współczynnika pikseli z chrominancją >100	26
Rysunek 21 Wyniki walidacji dla przy użyciu cech stosunku pikseli nasyconych i stosunku pikseli z krawędziami	28
Rysunek 22 Wyniki sprawdzianu klasyfikacji dla cech: współczynnik krawędzi, współczynnik saturacji, średnia jasność, odchylenie jasności, średnie nasycenie, odchylenie standardowe nasycenia	31
Rysunek 23 Histogram przedstawiający rozkład współczynnika krawędzi w dla danych testowych	32
Rysunek 24 Histogram przedstawiający rozkład współczynnika chrominancji dla danych testowych	32

Rysunek 25 Histogram przedstawiający rozkład średniej saturacji dla danych testowych.	32
Rysunek 26 Histogram przedstawiający rozkład odchylenia standardowego dla danych testowych	32
Rysunek 27 Histogram przedstawiający rozkład średniej jaskrawości dla danych testowych	32
Rysunek 28 Histogram przedstawiający odchylenie standardowe jaskrawości dla danych testowych	32
Rysunek 29 SVM margins [14]	36
Rysunek 30 SMV z jądrem RBF [15].....	37
Rysunek 31 Różnica poprawności klasyfikacji pomiędzy jądrem RBF i liniowymmm jądrem	37
Rysunek 32 Wybór folderu z danymi do programu walidującego klasyfikator	41
Rysunek 33 Okno programu do walidacji klasyfikatora	41
Rysunek 34 Porównanie wyników działania algorytmu na obu zbiorach testowych.....	42
Rysunek 35 Sprawdzenie skuteczności na obu zbiorach.....	43

7. Spis listingów

Listing 1 Klasa Parking slot i Contour.....	22
Listing 2 Obliczanie liczności nasyconych pikseli	25
Listing 3 Skalowanie nasycenia z jasnością koloru	25
Listing 4 Obliczanie liczności pikseli w masce	26
Listing 5 Obliczanie współczynnika nasyconych pikseli względem obszaru.....	26
Listing 6 Obliczanie liczności pikseli z krawędziami.....	28
Listing 7 Obliczanie stosunku pikseli z krawędziami do całego obszaru.....	28
Listing 8 Bezparametrowa detekcja krawędzi Canny'ego.....	28
Listing 9 Obliczanie statystyk średniej i odchylenia standardowego dla saturacji i wartości	31
Listing 10 Konfiguracja klasyfikatora SVM.....	38
Listing 11 K-krotna walidacja krzyżowa	39
Listing 12 Walidacja klasyfikatora przy użyciu dowolnego zbioru testowego i uczącego .	39
Listing 13 Walidacja Leave One Out.....	40

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW
w tym w Archiwum Prac Dyplomowych SGGW

.....

(czytelny podpis autora pracy)

