Monika Bartczak[1], **Marcin Kacprowicz**[1,2]
Zakład Informatyki PWSZ we Włocławku[1]
Instytut Informatyki Politechnika Łódzka[2]

DETEKCJA ZNAKÓW DROGOWYCH NA OBRAZACH

Detection of traffic signs in images

Streszczenie:

Niniejsza praca dotyczy detekcji obiektów na fotografiach. Wykrywanymi obiektami są znaki drogowe. W rezultacie zostaje przedstawionych kilka rozwiązań, pozwalających na ich poprawną identyfikację. W artykule została przedstawiona interpretacja otrzymanych wyników, skuteczności działania metod. Praca podkreśla istotność skrupulatnie dobranych, wyregulowanych parametrów cech obiektów, by zwiększyć skuteczność działania algorytmu.

Słowa kluczowe: detekcja znaków drogowych na obrazach, identyfikacja znaków drogowych, rozpoznawanie obiektów na obrazach, znalezienie znaków drogowych na fotografiach

Abstract:

This work concerns the detection of objects in photographs. The detected objects are road signs. As a result, several solutions are presented that allow their correct identification. The article presents the interpretation of the results obtained, the effectiveness of the methods. The work emphasizes the importance of meticulously selected, adjusted parameters of object features to increase the efficiency of the algorithm.

Keywords: detection of road signs on images, identification of road signs, recognition of objects on images, finding road signs in photographs

1. Wprowadzenie

Rozpoznawanie obrazów to proces analizowania i przetwarzania obrazu cyfrowego, w celu identyfikacji interesującego nas obiektu. Najogólniej mówiąc przetwarzanie obrazu pozwala nam stwierdzić czy na danej fotografii, bądź rysunku znajduje się szukany przez nas obiekt. Wykryte i niewykryte obiekty podlegają odpowiedniej klasyfikacji.[1]

---

[1] Onaszkiewicz P. Rozpoznawanie znaków drogowych w sekwencjach wideo. Praca Dyplomowa Inżynierska. Politechnika Warszawska. Wydział Elektroniki i Technik Informacyjnych. Instytut Informatyki. 2014, s. 3.

W dalszej części artykułu przedstawiono oraz porównano zastosowane metody. Omówiono w sposób szczegółowy dostosowywanie parametrów. Zademonstrowano technologie oraz narzędzia wykorzystane, w celu opracowania tego typu projektu.

Badania do publikacji naukowej zostały poprowadzone w oparciu o bazę zdjęć [więcej informacji na ten temat można znaleźć w [1]]. Artykuł nawiązuje do składowych bazy. Dlatego też warto zapoznać się z tym zbiorem danych przed przeczytaniem publikacji.

2. Porównanie metod i dostosowywanie parametrów

W tej części materiału wydawniczego zostały zademonstrowane obmyślone rozwiązania wraz z ich rezultatami. Wszystkie metody były weryfikowane dla 383 fotografii testowych oraz 161 obrazów treningowych.

2.1. Filtr kolorów

Pierwsze rozwiązanie opiera się na dobraniu odpowiednich parametrów zakresu barwy. Dla dolnego zakresu kolorów w modelu RGB zostały przyjęte w kolejności: [18, 5, 10]. Zaś dla górnego zakresu kolorów [27, 28, 19]. Filtr ten obejmuję barwę niebieską, która widnieje na znakach drogowych pionowych – nakazu. Metoda ta nie przyniosła oczekiwanych rezultatów, lecz umożliwiła zdobycie wiedzy z zakresu wzrostu, bądź spadku skuteczności w zależności od doboru parametrów koloru. Najlepsze, dające najbardziej skuteczne wyniki parametry zostały opisane powyżej. Parametry zostały dobrane w skutek obserwacji zdjęć z zaznaczonymi znakami wykrytymi przez metodę. Precyzja dla tej metody wynosi zaledwie 0,53 %, a czułość 0,71 %.

<u>Implementacja – etapy nałożenia filtru:</u>

I. W pierwszym kroku konwertujemy przestrzeń kolorów obrazu do formatu BGR. Biblioteka OpenCV potrzebuje takiego modelu barw dla dalszej wykorzystanej metody.

```
Python: hsv = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
```

II. W kolejnym kroku tworzymy maskę. Maska posiada górny i dolny zakres barw.

```
Python: lower_treshol = np.array([18, 5, 10]) #dolny zakres kolorów

upper__treshold = np.array([27, 28, 19]) #górny zakres kolorów
```

```
mask = cv2.inRange(hsv, lower_treshold, upper__
treshold) #tworzenie maski
```

III. Następnie nakładamy na wejściową (oryginalną) fotografię maskę i negujemy piksele, które nie zaliczają się do niej (tworzymy tło). W rezultacie pozostają nam na obrazie kolory o zakresie zdefiniowanym powyżej, bądź czarny kolor (tło).
W celu wykonania w/w czynności posługujemy się metodą: bitwise_and. Funkcja ta oblicza bity dla każdego elementu dwóch tablic lub tablicy i skalaru.

```
Python: cv2.bitwise_and(src1,src2[,dst[,mask]])
→ dst
```

Przyjmuję parametry tj.:
- src1 – pierwszą tablicę wejściową lub skalar,
- src2 – drugą tablicę wejściową lub skalar,
- src – pojedynczą tablicę wejściową,
- dst – tablicę wyjściową, która ma ten sam rozmiar i typ, co tablice wejściowe,
- maska – opcjonalna operacja maski, 8-bitowa tablica pojedynczych kanałów, która określa elementy tablicy wejściowej podlegające zmianie.

Przekazano parametry ukazane poniżej:

```
Python: res = cv2.bitwise_and(image, image,
mask=mask)
```

gdzie image to wczytany plik graficzny.[2]

Wnioski z zastosowanego filtru kolorów są następujące: wszystkie poszczególne znaki drogowe posiadają barwę, która mieści się w stosunkowo wąskim zakresie kolorów umożliwiającą skuteczne ich oddzielenie od tła. Zastosowanie zbyt szerokiego, bądź zbyt wąskiego zakresu koloru (niebieskiego) skutkuje mniejszą skutecznością.

2.2. Filtr kształtów

Druga metoda brała pod uwagę kształt obiektów. Na zdjęciach wyszukiwana była elipsa. Aby wyrysować elipsę skorzystałam z biblioteki OpenCV z metody HoughCircles. HoughCircles wykorzystując transformację Hough,

[2] Źródło: http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html (stan na dzień: 03.07.2017 r.).

znajduję koła w odcieniach szarości. Dla funkcji przekazywane były następujące parametry:

```
Python: cv2.HoughCircles(image, method, dp,
minDist[, circles[, param1[, param2[, minRadius[,
maxRadius]]]]])
```

- image – 8-bitowy, jednokanałowy, w odcieniach szarości obraz wejściowy.
- circles – wektor wyjściowy odnalezionych okręgów. Każdy wektor jest zakodowany, jako trójwymiarowy wektor zmiennoprzecinkowy (x, y, promień).
- method – użyta metoda do detekcji. Obecnie jedynym zastosowanym sposobem jest CV_HOUGH_GRADIENT.
- dp – współczynnik odwrotności rozdzielczości akumulatora do rozdzielczości obrazu. Np., jeśli dp=1, akumulator ma tę samą rozdzielczość co obraz wejściowy.
- minDist – Minimalna odległość między środkami wykrytych okręgów.
- Param1 – pierwszy parametr specyficzny dla metody. W przypadku CV_HOUGH_GRADIENT jest to wyższy próg obu przejść do detektora krawędzi.
- Param[2] – Drugi parametr specyficzny dla metody. W przypadku CV_HOUGH_GRADIENT jest to próg akumulatora dla centrów okręgu na etapie detekcji. Im mniejsze, tym więcej może wykryć fałszywych okręgów. Okręgi, odpowiadające większym rozmiarom akumulatora, zostaną najpierw zwrócone.
- MinRadius – minimalny promień okręgu.
- MaxRadius – maksymalny promień okręgu.[3]

Przekazano następujące parametry:

```
cv2.HoughCircles(gray,cv2.HOUGH_GRADIENT,1,10,
param1=58,param2=29,minRadius=14,maxRadius=70)
```

Tak opracowany algorytm posiadał 2,3 % precyzji i 2,96 % czułości. Dla przedstawionego przykładu można zauważyć, że wykrywanie kształtu obiektu daję lepsze rezultaty niż odpowiednie wskazanie barwy obiektu na fotografii.

[3] Źródło: http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles (stan na dzień: 20.06.2017 r.) (tłumaczenie własne).

Wnioski Gwałtowne zmniejszenie parametru minDist powoduje wykrycie sąsiednich okręgów, a nie prawdziwego. Gwałtowne zwiększenie parametry minDist może spowodować pomijanie kół na obrazie.

2.3. Filtr kształtu i koloru

Trzecia metoda jest połączeniem wcześniejszych dwóch rozwiązań. Uzyskała ona największą precyzję i czułość, w kolejności: 3,91 % i 5,23 %.

Wnioski wykrywanie obiektów w oparciu o większą ilość cech daję lepszą skuteczność.

Poprzez stosowanie wyżej wspomnianych metod nie otrzymano oczekiwanych wyników. Metody nie są właściwe do tak złożonego zbioru danych. Dlatego też został wykorzystany i przedstawiony klasyfikator Haara.

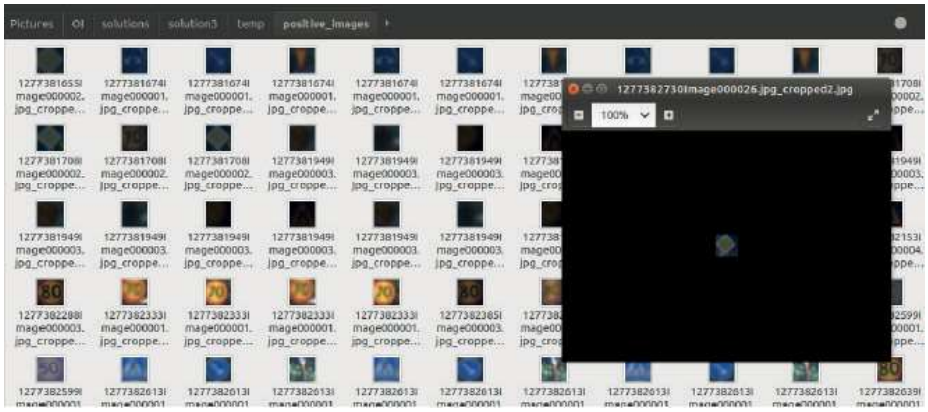
2.4. Kaskadowe klasyfikatory Haara

Zbudowanie klasyfikatora wymagało przygotowania bazy zdjęć, w skład, której wchodzą obrazy przedstawiające tylko interesujące nas obiekty, tzw. pozytywne wzorce oraz zdjęcia niezawierające instancji rozpoznawanych klas obiektów, tzw. negatywne wzorce lub tło obiektu[4]. Pozytywne wzorce zostały wygenerowane poprzez metodę zawartą w pliku „cut.py". Algorytm ten korzystając z informacji o współrzędnych znaków drogowych znajdujących się w folderze annotations, wycina wyszukiwane obiekty. Znaki drogowe pionowe zapisywane są w folderze o nazwie „positive_images". Informacje na temat pozytywnych wzorców wprowadzone są do pliku o nazwie „info. dat" (Rysunek 3). Plik ten zawiera następujące wiadomości: ścieżkę do obrazu, ilość obiektów na zdjęciu, współrzędne punktu początkowego obrazka (w analizowanym przypadku jest to lewy górny róg kwadratu) oraz rozmiar fotografii w pikselach. Ilość obiektów na zdjęciu, współrzędne punktu początkowego obrazu oraz rozmiar fotografii są identyczne dla każdego zdjęcia w pozytywnym wzorcu. Wynika to z tego, że w folderze „positive_images" (Rysunek 1) znajdują się zapisane pojedynczo obrane obiekty, dostosowane do rozmiaru 30 x 30 pikseli. Metoda ta tworzy również negatywne wzorce poprzez zamazywanie obiektów, które chcemy wykryć. Wcześniej jednak obraz jest konwertowany do odcieni szarości. Zdjęcia z wyciętymi znakami drogowymi zapisywane są do folderu o nazwie „negative_images" (Rysunek 2).

---

[4] Źródło: Wilkowski P., Wykorzystanie algorytmu detekcji i lokalizacji w zadaniu chwytania. Praca magisterska, Politechnika Warszawska. Warszawa 2009, s. 27. Dostęp pod adresem: http://robotyka.ia.pw.edu.pl/papers/pwilkowski-msc-09-twiki.pdf.

Rysunek 1. Ukazanie zawartości folderu „positive_images”



Źródło: Opracowanie własne.

Rysunek 2. Ukazanie zawartości folderu „negative_images”

Źródło: Opracowanie własne.

Rysunek 3. Zaprezentowanie zawartości pliku o nazwie „info.dat”

```
./positive_images/1277388138Image000098.jpg_cropped1.jpg 1 0 0 30 30
./positive_images/1277388138Image000098.jpg_cropped2.jpg 1 0 0 30 30
./positive_images/1277388138Image000098.jpg_cropped3.jpg 1 0 0 30 30
./positive_images/1277388138Image000098.jpg_cropped4.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped1.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped2.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped3.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped4.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped5.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped6.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped7.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped8.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped9.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped10.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped11.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped12.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped13.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped14.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped15.jpg 1 0 0 30 30
./positive_images/1277391703Image000053.jpg_cropped16.jpg 1 0 0 30 30
./positive_images/1277388297Image000027.jpg_cropped.jpg 1 0 0 30 30
./positive_images/1277386257Image000004.jpg_cropped1.jpg 1 0 0 30 30
./positive_images/1277386257Image000004.jpg_cropped2.jpg 1 0 0 30 30
./positive_images/1277386257Image000004.jpg_cropped3.jpg 1 0 0 30 30
./positive_images/1277386257Image000004.jpg_cropped4.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped1.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped2.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped3.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped4.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped5.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped6.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped7.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped8.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped9.jpg 1 0 0 30 30
./positive_images/1277391651Image000113.jpg_cropped10.jpg 1 0 0 30 30
```

Źródło: Opracowanie własne.

Wszystkie ścieżki z tłem obiektów znajdują się w pliku „big.txt”. Pik został utworzony poprzez polecenie:

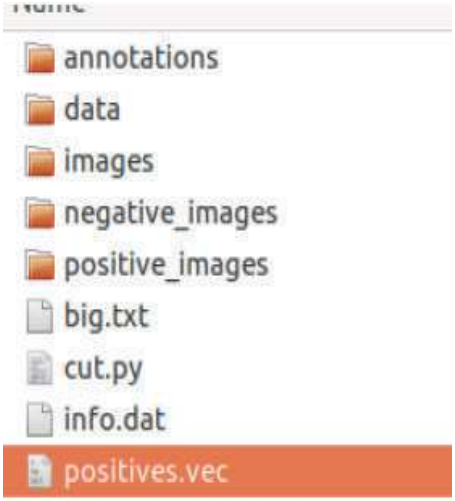
```
find./negative_images -name ‚*.jpg’ >big.txt
```

Następnym krokiem jest stworzenie wynikowego wektora łączącego wszystkie pozytywne wzorce. Aby to zrealizować zostało wykorzystane polecenie opencv_createsamples. W oknie terminala zostało wprowadzone:

```
opencv_createsamples -info info.dat -num 850
-w 30 -h 30 -vec positives.vec
```

W rezultacie otrzymujemy plik wektorowy. W komendzie podajemy następujące parametry: miejsce, gdzie znajduje się plik informacyjny, ilość zdjęć, którą powinien posiadać plik, jakie wymiary obrazów mają znajdować się w pliku wektora oraz gdzie wydrukować wyniki.

Rysunek 4. Ukazanie wygenerowanego pliku „positives.vec”



Źródło: Opracowanie własne.

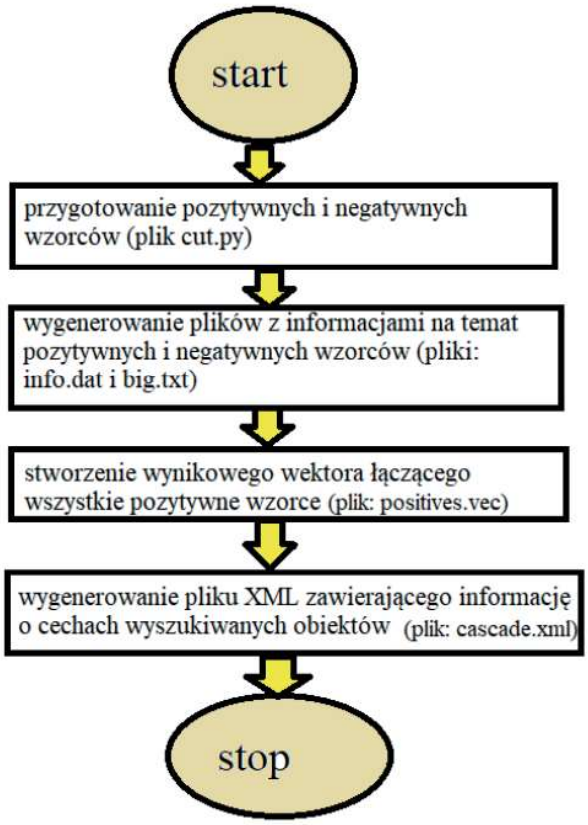
Ostatnim, lecz nie mniej ważnym punktem jest trenowanie kaskadowe. Wykorzystując OpenCV polecenie opencv_traincascade (informację o negatywnych i pozytywnych wzorcach) jesteśmy w stanie wygenerować plik o rozszerzeniu .xml, który reprezentuje cechy wyszukiwanych obiektów. Plik ten jest kluczem do wykrywania obiektów na obrazach.

Trenowanie kaskadowe uruchamiamy poniższym poleceniem:

```
opencv_traincascade -data data -vec positives.vec
-bg big.txt -numPos 800 -numNeg 400 -numStages 10
-w 30 -h 30
```

Za pomocą tej komendy informujemy, gdzie chcemy przenieść dane, jaka jest ścieżka do pliku wektorowego, gdzie znajduje się plik tła, ile jest pozycji pozytywnych i negatywnych obrazów, jaką liczbę kroków chcemy aby wykonał algorytm oraz jaka jest wysokość i szerokość obrazów. Zostało użyte znacznie mniej numPos niż mamy, ma to zapewnić przestrzeń dla kroków, która się dodaje do tego. Stosunek ilości obrazów pozytywnych do negatywnych powinien wynosić 2:1

Rysunek 5. Przedstawienie kroków zbudowania klasyfikatora Haar Cascade



Źródło: Opracowanie własne.

3. Opis wykorzystanych narzędzi i technologii

W celu realizacji postawionego celu logika programu została zaimplementowana w języku programowania python. Skorzystano również z biblioteki OpenCV.

Biblioteka OpenCV dostarcza trzech narzędzi (opencv-createsamples,

opencv_traincascade, opencv-performance) wspomagających proces uczenia kaskady klasyfikatorów cech haaro-podobnych. Ja wykorzystałam dwóch z nich: opencv_createsamples oraz opencv_traincascade.

opencv_createsamples zostało użyte do budowania zbiorów uczących. Aby tego dokonać przygotowywane są zbiory danych treningowych zawierające pozytywne próbki testowe. Wygenerowany plik ma rozszerzenie .vec, jest to format binarny zawierający obrazy.

opencv_traincascade przechowuje przeszkolony klasyfikator w różnych formatach plików.[5]

5. Klasyfikatory kaskadowe

„Opis kaskady klasyfikatorów cech haaro-podobnych. Metoda ta wykorzystuje uczenie maszynowe do rozwiązania zadania klasyfikacji obiektów. Jej zaletą jest wysoka szybkość detekcji wynikowego klasyfikatora oraz wysoka poprawność jego działania

Kaskada klasyfikatorów cech haaro-podobnych wykorzystuje algorytm uczenia maszynowego AdaBoost (ang. Adaptive Boosting), którego zadaniem jest wybór słabych klasyfikatorów bazujących na cechach haaro-podobnych.

Adabost. Rozwiązanie tego problemu przez zastosowanie technik maszynowego uczenia polega na zebraniu jak największego zbioru danych uczących, składającego się ze zdjęć obiektu oraz bez obiektu, którym przyporządkowywane są etykiety (w tym konkretnym przykładzie binarne etykiety: obiekt, nie-obiekt). Tak przygotowane dane stanowią wejście algorytmu uczącego (uczenie z nadzorem), który w sposób automatyczny tworzy regułę klasyfikującą zwaną klasyfikatorem. Mając nowy zbiór zdjęć, tym razem bez etykiet, klasyfikator ma za zadanie określić czy dane zdjęcie zawiera obiekt, czy też nie. Nadrzędnym celem jest wygenerowanie klasyfikatora, który będzie poprawnie klasyfikował dane niewykorzystywane w procesie uczenia.

Boosting jest meta-algorytmem opisującym sposób konstrukcji wysoce poprawnego klasyfikatora spośród słabych klasyfikatorów. Mając zdefiniowany algorytm uczenia prostych reguł klasyfikujących, algorytm boosting'u wywołuje go w serii rund, za każdym razem na innym rozkładzie wag danych uczących. W każdej rundzie wybierany jest jeden słaby klasyfikator, który lepiej klasyfikuje wcześniej błędnie sklasyfikowane dane"[6].

---

[5] Źródło: http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html (stan na dzień: 03.09.2017 r.)

[6] Źródło: Wilkowski P., Wykorzystanie algorytmu detekcji i lokalizacji w zadaniu chwytania. Praca magisterska, Politechnika Warszawska. Warszawa 2009, s. 27. Dostęp pod adresem: http://robotyka.ia.pw.edu.pl/papers/pwilkowski-msc-09-twiki.pdf

6. Korekcja parametrów atrybutów obiektów

Tabela 1. Porównanie skuteczności algorytmu dla różnych danych wejściowych (rozmiar obrazu wejściowego oraz ilość kroków)

Rozmiar obrazu wejściowego	Ilość kroków trenowania kaskadowego	Precyzja	Czułość
minSize -2 maxSize -5	5	1,22%	35,05%
minSize -2 maxSize -5	10	2,58%	28,48%
minSize -2 maxSize -5	17	38,09%	26,88%
minSize -4 maxSize -4	15	12.06%	19,76%
minSize -4 maxSize -4	17	37,49%	10,23%
minSize -2 maxSize -4	17	28,11%	29,74%
minSize -4 maxSize -3	17	21,06%	14,21%

Źródło: Opracowanie własne.

W tabeli za pomocą barwy żółtej zostały zaznaczone najkorzystniejsze, najskuteczniejsze wyniki.

minSize oznacza minimalny rozmiar obiektu. Obiekty mniejsze niż te są ignorowane.

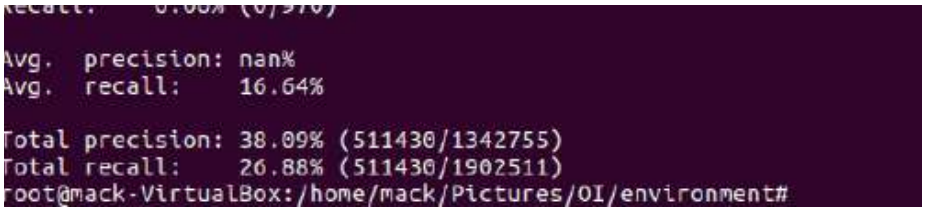
maxSize oznacza maksymalny rozmiar obiektu. Obiekty większe niż te są ignorowane.

Tabela 2. Porównanie wyników skuteczności algorytmu na zbiorze treningowym i testowym przy różnych parametrach wejściowych

Rodzaj zbioru\Cechy	Rozmiar obrazu wejściowego	Ilość kroków	Precyzja	Czułość
Zbiór testowy	minSize 4 maxSize 4	17	37,49%	10,32%
Zbiór treningowy	minSize 4 maxSize 4	17	24,29%	10,75%
Zbiór testowy	minSize 2 maxSize 5	17	38,09%	26,88%
Zbiór treningowy	minSize 2 maxSize 5	17	26,93%	23,34%

Źródło: Opracowanie własne.

Rysunek 6. Przedstawienie najefektywniejszego wyniku działania algorytmu



Źródło: Opracowanie własne.

Poprzez empiryczne testowanie metody można zaobserwować, że:

- stosowanie większej liczby kroków wykonania algorytmu w trenowaniu kaskadowym skutkuje zwiększoną liczbą skuteczności wykrywania znaków drogowych,
- zwiększenie ilości obrazów w zbiorze treningowym wpłynęło by na poprawienie wyniku skuteczności metody,
- im większe rozmiary obrazów w pliku wektora tym dłuższy czas trenowania,
- w „annotations” w niektórych plikach są powielone oznaczenia tego samego znaku. Skutkuje to mniejszą dokładnością, ponieważ w przypadku prawidłowego znalezienia znaku, algorytm przyjmie, że odnaleźliśmy dwa obiekty. Gdy nie znajdziemy w/w znaku drogowego metoda przyjmie, że nie odnaleźliśmy dwóch obiektów,

Rysunek 7. Ukazanie przypisanych oznaczonych współrzędnych, gdzie występują znaki (powielenie oznaczonych znaków na zdjęciach)

Źródło: Opracowanie własne.

- obrazy znajdujące się w negatywnym wzorcu mogą mieć różne rozmiary. Ale każdy obraz powinien być większy, ale niekoniecznie większy od rozmiaru okna szkoleniowego. Ponieważ te obrazy są używane do podpróbkowania negatywnego obrazu do rozmiaru szkolenia,
- algorytm uczący klasyfikatora wymaga dużej liczby danych treningowych. W przypadku braku wystarczającej liczby danych opencv-createsamples umożliwia wygenerowanie dowolnej liczby pozytywnych wzorców z jednego zdjęcia,
- większa ilość wyszukiwanych cech obiektu na obrazie, świadczy o większej precyzyjności i skuteczności znalezienia obiektu,
- im więcej danych w trenowaniu kaskadowym tym dokładniejszy wynik.

7. Wyniki

W tej części artykułu przedstawiono rezultaty działania algorytmu. Poniżej zamieszczone obrazy zostały zamieszczone ze zmniejszoną rozdzielczością, która uniemożliwia zauważenie szczegółów. Interpretacja wyników została przeprowadzona jednak w oparciu o obrazy w pełnej rozdzielczości.

Metoda 2.3 wyrysowywała elipsy na obrazach. Algorytm ten przyjmował współrzędne kwadratu.

Rysunek 8. Prezentacja fotografii przed wykrywaniem znaków na obrazie (metoda 2.3)

Źródło: Opracowanie własne. Zdjęcie dostępne w zbiorze danych pod adresem url: http://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/.

Rysunek 9. Wyświetlenie obrazu wynikowego, po wykonaniu detekcji przy pomocy algorytmu (metoda 2.3)

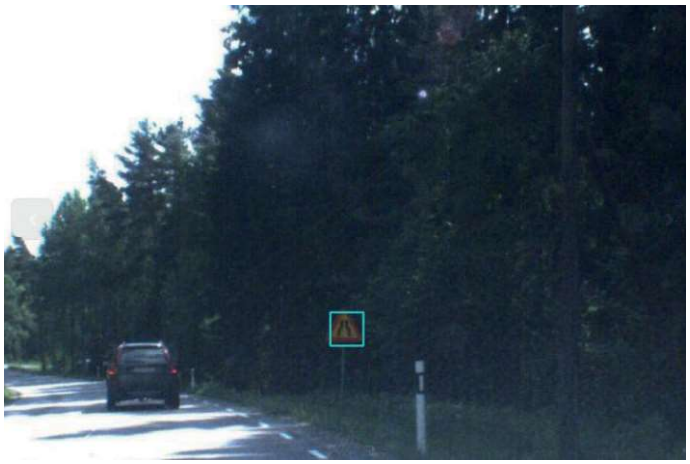
Źródło: Opracowanie własne. Zdjęcie dostępne w zbiorze danych pod adresem url: http://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/.

Metoda 2.4. przyjmowała współrzędne kwadratu i wyrysowywała kwadrat na obrazie.

Rysunek 10. Wyświetlenie obrazu wynikowego, po wykonaniu detekcji przy pomocy algorytmu (kaskadowe trenowanie) (metoda 2.4)

Źródło: Opracowanie własne. Zdjęcie dostępne w zbiorze danych pod adresem url: http://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/.

Rysunek 11. Wyświetlenie obrazu wynikowego, po wykonaniu detekcji przy pomocy algorytmu (kaskadowe trenowanie) (metoda 2.4).



Źródło: Opracowanie własne. Zdjęcie dostępne w zbiorze danych pod adresem url: http://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/.

6. Wnioski i perspektywy dalszych prac

Metody zostały poprawnie zaimplementowane i ocenione. Doświadczalnie wykazano, że najskuteczniejsza metoda wykorzystywała kaskadę klasyfikatorów. Stwierdzono w przyszłości do detekcji oraz lokalizacji wybranych obiektów skorzystano by z algorytmu wykorzystującego kaskadę klasyfikatorów bazujących na cechach haaro-podobnych. Algorytm ten jest na tyle uniwersalny, iż umożliwia pracę z różnymi klasami obiektów. Algorytm sam dobiera cechy, po których wyszukuje obiekty.

Aby zwiększyć skuteczność metody można:

- wygenerować większą ilość kroków;
- pracować na większym zbiorze treningowym;

Bibliografia:

Źródło bazy danych wykorzystanej na potrzebę pracy badawczej:

[1] http://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/

Źródła wykorzystane podczas tworzenia publikacji naukowej:

[2] http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles.

[3] Romadi M., Oulad Haj Thami R., Chiheb R., Romadi R., Detection and Recognition of Road Signs, ENSIAS – Med V Souissi University – Rabat-MOROCCO, 2013.

[4] http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html.

[5] Wilkowski P., Wykorzystanie algorytmu detekcji i lokalizacji w zadaniu chwytania. Praca magisterska, Politechnika Warszawska. Warszawa 2009, s. 27.

[6] http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html.

[7] Onaszkiewicz P. Rozpoznawanie znaków drogowych w sekwencjach wideo. Praca Dyplomowa Inżynierska. Politechnika Warszawska. Wydział Elektroniki i Technik Informacyjnych. Instytut Informatyki. 2014, s. 3.