



**POLITECHNIKA
GDAŃSKA**

WYDZIAŁ FIZYKI TECHNICZNEJ
I MATEMATYKI STOSOWANEJ

Imię i nazwisko studenta: Filip Cyglicki
Nr albumu: 169668
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Fizyka Techniczna
Specjalność: Informatyka stosowana

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Porównanie klasyfikatorów LBP i Haar

Tytuł pracy w języku angielskim: Comparison of LBP and Haar classifiers

Potwierdzenie przyjęcia pracy	
Opiekun pracy	Kierownik Katedry/Zakładu (pozostawić właściwe)
podpis	podpis
dr inż. Bartosz Reichel	

Data oddania pracy do dziekanatu:



**POLITECHNIKA
GDAŃSKA**

WYDZIAŁ FIZYKI TECHNICZNEJ
I MATEMATYKI STOSOWANEJ

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Porównanie klasyfikatorów LBP i Haar

Imię i nazwisko studenta: Filip Cyglicki

Data i miejsce urodzenia: 28.03.1996, Ława

Nr albumu: 169668

Wydział: Wydział Fizyki Technicznej i Matematyki Stosowanej

Kierunek: fizyka techniczna

Poziom kształcenia: pierwszy

Forma studiów: stacjonarne

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. 2018 poz. 1191 z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. 2018 poz. 1668 z późn. zm.),¹ a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia

.....
podpis studenta

Streszczenie

Praca powstała w celu porównania algorytmów HAAR'a i LBP (z ang. Local Binary Patterns) by sprawdzić w jaki sposób wpływa ilość pozytywnych i negatywnych próbek ich wysokość i szerokość czy ilość etapów mają wpływ na czas szkolenia oraz skuteczności jaką uzyskują wytrenowane klasyfikatory. Dodatkowym celem pracy jest zapoznanie się z narzędziami oferowanymi przez bibliotekę OpenCV. Efektem pracy jest powstanie programu komputerowego pozwalającego na masowe szkolenie klasyfikatorów z różną konfiguracją. Wyniki pracy zostały przedstawione w tabelach. W pracy przedstawiono w jaki sposób przebiega trening i użycie klasyfikatora jak również w jaki sposób zweryfikowano jego skuteczność.

Słowa kluczowe: sztuczna inteligencja, rozpoznawanie obrazów, widzenie maszynowe, optymalizacja.

Dziedzina nauki i techniki, zgodnie z wymogami OECD: informatyka techniczna i telekomunikacja, informatyka (dziedzina nauk technicznych)

Abstract

The goal of this paper is to compare HAAR and LBP algorithms, to check the influence of number of positive and negative samples, their height and width or number of stages result in different training times and effectiveness which is achieved by trained classifier. Additional target of it, is to get more knowledge about tools offered by OpenCV library. The outcome of this paper is computer program which helps in mass learning the classifiers with different configuration. The results are displayed in tables. In this paper explained how the training is done, how the classifier is used and how his effectiveness was verified.

Keywords: artificial Intelligence, image recognition, computer vision, optimisation

Wykaz ważniejszych oznaczeń i skrótów	6
• Wstęp i cel pracy	7
• OpenCV	9
• Tytuł podrozdziału	numer strony
• Tytuł punktu podrozdziału	numer strony
• Tytuł podrozdziału	numer strony
----- kolejne rozdziały, podrozdziały i punkty	numer strony
x. Podsumowanie	numer strony
Wykaz literatury	numer strony
Wykaz rysunków	numer strony
Wykaz tabel	numer strony
Dodatek A	numer strony
Dodatek B	numer strony

Wykaz ważniejszych skrótów i oznaczeń

<i>LBP</i>	-	Local Binary Patterns
<i>-vec</i>	-	Plik wynikowy programu createsample, zawierające przygotowane próbki do treningu
<i>-info</i>	-	Plik adnotacji zawierający ścieżki obrazów i/lub ilość i położenie obiektów na nim.
<i>HOG</i>	-	(ang. Histogram of Oriented Gradients) Histogram zorientowanych gradientów
<i>SVM</i>	-	(ang. Support Vector Machine) Maszyna wektorów wspierających

1. Wstęp

Wraz z rozwojem technologii, naukowcy zaczęli zastanawiać się nad tym czy maszyny są w stanie myśleć. W 1956 roku podczas jednej z konferencji dotyczącej tego tematu, John McCarthy użył sformułowania 'sztuczna inteligencja'[1]. Od tego czasu minęło wiele lat a termin ten dotyczy już nie tylko podejmowania decyzji przez maszynę liczącą ale między innymi również rozpoznawaniem obrazów.

Widzenie maszynowe poruszone w niniejszej pracy jest złożonym tematem. W celu ułatwienia powstało wiele algorytmów pomagających w klasyfikacji obrazów jak i oprogramowania usprawniających ich implementację. Jednym z przykładów biblioteki wspomagających tworzenie oprogramowania do rozpoznawania obrazów jest OpenCV[2] (ang. Open Source Computer Vision). W skład niej wchodzi algorytmy optymalizujące, wspomagające uczenie maszynowe i mogą zostać użyte do rozpoznawania twarzy jak i śledzeniem ruchu źrenic czy usuwania efektu czerwonych oczu powstałych przy użyciu flesza.

Dużym problemem jest złożoność obliczeniowa, nawet małe zdjęcia składają się z wielu pikseli a przeprowadzenia operacji na nich wymaga wiele zasobów. Z tego powodu powstały sposoby klasyfikatory zajmujące się optymalizacją tego zadania. Wybór konkretnej metody nie jest prostym zadaniem, często ze skutecznością wiąże się dłuższy czas uczenia lub większe zapotrzebowanie na czas procesora czy pamięć. Z tego powodu dobór odpowiedniego klasyfikatora jest zadaniem nietrywialnym.

W ramach pracy porównane zostaną dwa algorytmy Viola-Jones'a zwany również Treningiem Haara i LBP (ang. Local Binary Patterns). Jej rezultatem będzie tabela zawierająca czas jaki był potrzebny do nauki algorytmu, ile próbek zostało użytych a także skuteczność z jaką udało się zakwalifikować zdjęcia.

2. Cel i zakres

Celem niniejszej pracy jest porównanie klasyfikatorów widzenia maszynowego i opisanie ich różnic.

- Przegląd literatury na temat widzenia komputerowego i OpenCV
- Przegląd literatury na temat algorytmów Viola-Jones'a i LBP
- Wybór implementacji wymienionych klasyfikatorów
- Stworzenie aplikacji umożliwiającej porównanie
- Wykonanie testów i utworzenie pomiarów

3. OpenCV

To biblioteka z licencją open source do widzenia komputerowego i uczenia maszynowego. Powstała w celu wprowadzenia wspólnej infrastruktury dla programów do widzenia komputerowego i percepcji maszynowej w komercyjnych rozwiązaniach[2]. Wspomaga w tworzeniu aplikacji w C++, Javie, Pythonie na platformy Windows, Android, Linux czy Mac OS.

3.1. Historia OpenCV

Zapoczątkowany przez firmę Intel projekt, początkowo miał usprawnić zadania o wysokiej złożoności obliczeniowej. W projekt zaangażowani byli głównie eksperci od optymalizacji i wydajności których celem było:

- stworzenie otwartej i zoptymalizowanej infrastruktury do przetwarzania obrazów,
- rozpowszechnić wiedzę i narzędzia na których programiści mogą polegać i bazować,

Od 2015 roku OpenCV zostało przejęte przez organizację non-profit OpenCV.org[4].


3.2. Aplikacje wchodzące w skład OpenCV

Po zbudowaniu biblioteki OpenCV mamy dostęp do kilku aplikacji które wspomagają szkolenie w ich skład wchodzi:

3.1.1. Annotation (ang. adnotacja)

Aby uruchomić to narzędzie z linii komend potrzebujemy podać ścieżkę do pliku w którym znajdują się obrazy, a także gdzie i pod jaką nazwą chcemy zapisać plik -info. Jeżeli podamy poprawne parametry na ekranie powinno pokazać się pierwsze z naszych zdjęć na którym mamy możliwość zaznaczyć, zatwierdzić wyznaczony obszar lub usunąć ostatni a następnie przejść do kolejnego zdjęcia, tak jak pokazano na rysunku 3.1. Jeżeli chcemy utworzyć plik zawierający obrazy tła czyli takie które nie zawierają obiektu którego chcemy nauczyć obiekt, wystarczy nic nie zaznaczać. Kompletny dla zdjęć zawierających obiekt, powinien składać się ze ścieżki do zdjęcia, ilości obiektów jakie znajdują się na wybranym obrazie i ich współrzędne.

```
>opencv_annotation.exe --images=Images/Positive --annotations=pos.info
* mark rectangles with the left mouse button,
* press 'c' to accept a selection,
* press 'd' to delete the latest selection,
* press 'n' to proceed with next image,
* press 'esc' to stop.
```



Images/Negative\Negative10.jpg	0	Images/Positive\Positive10.jpg	1	10	2	85	34
Images/Negative\Negative100.jpg	0	Images/Positive\Positive100.jpg	1	12	8	80	28
Images/Negative\Negative101.jpg	0	Images/Positive\Positive101.jpg	1	10	5	84	31
Images/Negative\Negative102.jpg	0	Images/Positive\Positive102.jpg	1	9	5	83	30
Images/Negative\Negative103.jpg	0	Images/Positive\Positive103.jpg	1	10	5	87	30
Images/Negative\Negative104.jpg	0	Images/Positive\Positive104.jpg	1	9	7	88	28
Images/Negative\Negative105.jpg	0	Images/Positive\Positive105.jpg	1	9	7	84	29
Images/Negative\Negative106.jpg	0	Images/Positive\Positive106.jpg	1	7	9	88	30

Rys. 3.1. Przykład użycia programu annotation

3.1.2. Createsample (ang. stwórz próbkę)

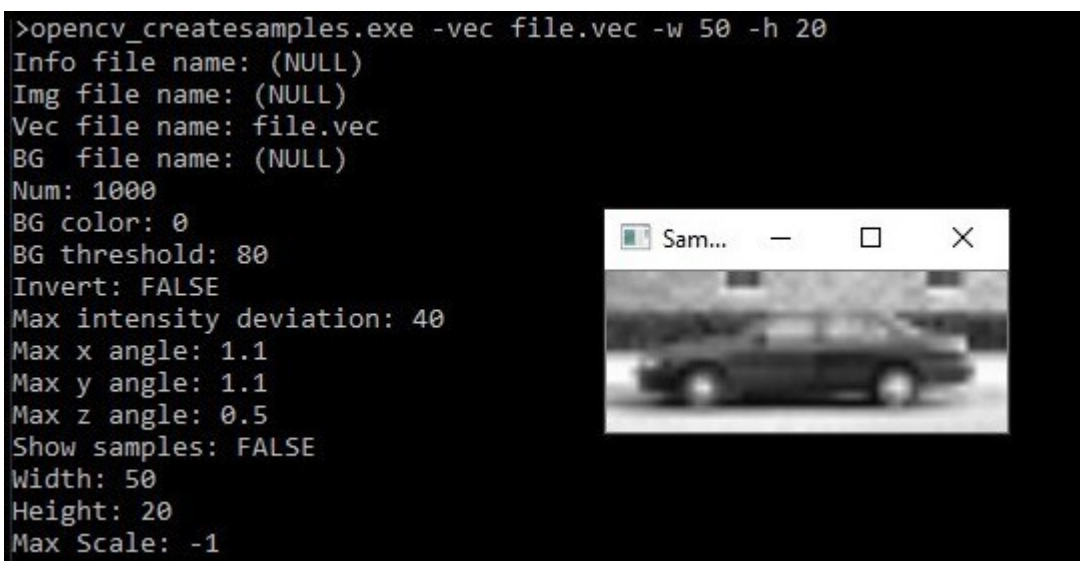
To narzędzie służy do wykonania pliku -vec zawierającego obrazy przygotowane do wyszkolenia klasyfikatora. Można go wygenerować na dwa sposoby, podając jeden obrazek na którym zawieramy nasz obiekt a następnie plik -info z obrazkami lub zamiast pojedynczego zdjęcia podać również plik -info z pozytywnymi obrazkami. Dobrą praktyką jest mimo wszystko używanie pliku z wieloma różnymi obrazami przedstawiającymi obiekt w ten sposób zwiększamy szanse na lepsze nauczanie klasyfikatora. Program wymaga także podania wysokości i szerokości próbek jakie chcemy uzyskać. Przykład użycia narzędzia do tworzenia próbek pokazano na obrazku 3.2.1. Jeżeli nie podamy plików -info lub pojedynczego obrazka a także ścieżka -vec będzie zawierała już istniejący plik to program wyświetli nam przygotowane próbki, tak jak przedstawiono na obrazku 3.2.2.

```

>opencv_createsamples.exe -vec file.vec -info pos.info -bg neg.info
-num 550 -w 50 -h 20
Info file name: pos.info
Img file name: (NULL)
Vec file name: file.vec
BG file name: neg.info
Num: 550
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 50
Height: 20
Max Scale: -1
RNG Seed: 12345
Create training samples from images collection...
Done. Created 550 samples

```

Rys. 3.2.1. Przykład utworzenia próbek za pomocą narzędzia createsample



Rys. 3.2.2. Wyświetlanie próbek za pomocą narzędzia createsample

3.1.3. *Traincascade* (ang. *Trenuj klasyfikator*)

Za jego pomocą można stworzyć klasyfikator. Jako parametry należy podać mu ścieżkę do katalogu w którym zostaną utworzone pliki klasyfikatora, plik `-vec` i `-info` z zdjęciami tła, liczbę pozytywnych i negatywnych próbek użytych w każdym etapie szkolenia, ilość faz, typ klasyfikatora a także wysokość i szerokość która powinna być zgodna z tą która została użyta do utworzenia pliku `-vec`, przykład użycia został przedstawiony na obrazku 3.3. Podczas każdego etapu, zostaje przygotowany plik który pozwala nam na przerwanie szkolenia i

przywrócenia go na takim etapie na którym został ukończony.

```
>opencv_traincascade.exe -data path_to_classifier/ -vec file.vec
-bg neg.info -numPos 550 -numNeg 500 -w 50 -h 20 -numStages 1
PARAMETERS:
cascadeDirName: path_to_classifier/
vecFileName: file.vec
bgFileName: neg.info
numPos: 550
numNeg: 500
numStages: 1
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : -1
stageType: BOOST
featureType: HAAR
sampleWidth: 50
sampleHeight: 20
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [50,20] : 487255

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed    550 : 550
NEG count : acceptanceRatio    500 : 1
Precalculation time: 8.769
+---+---+---+---+
| N |   HR   |   FA   |
+---+---+---+---+
| 1 |       1 |       1 |
+---+---+---+---+
| 2 |       1 |       1 |
+---+---+---+---+
| 3 |       1 |       1 |
+---+---+---+---+
| 4 | 0.998182 | 0.548 |
+---+---+---+---+
| 5 | 0.996364 | 0.376 |
+---+---+---+---+
END>
Training until now has taken 0 days 0 hours 0 minutes 51 seconds.
```

Rys. 3.3. Przykład użycia narzędzia do trenowania klasyfikatora

3.1.4. Visualisation (ang. Narzędzie do wizualizacji)

Kiedy posiadamy wytrenowany klasyfikator, jednak chcielibyśmy sprawdzić jakie cechy wybrał, możemy użyć właśnie tego programu. Jako parametry należy podać obraz, na który zostaną naniesione cechy, ścieżkę do wytrenowanego klasyfikatora a także gdzie chcielibyśmy zapisać zdjęcie z naniesionymi cechami. Przykład dla klasyfikatora HAAR'a został przedstawiony na obrazku 3.4.



Rys. 3.4. Przykład pokazania cech za pomocą narzędzia do wizualizacji

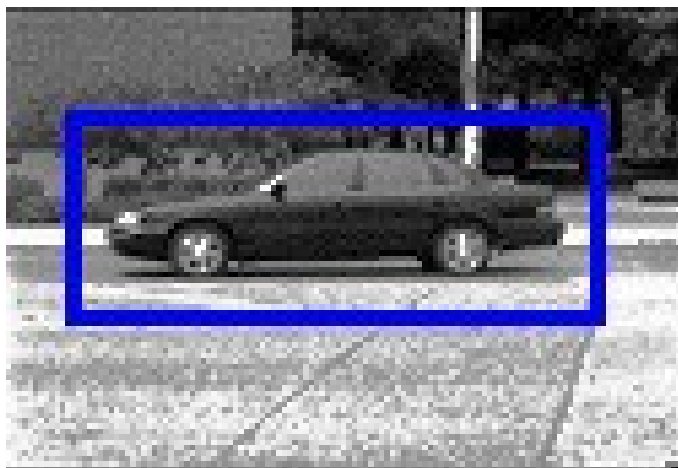
3.2. Przykład znajdowania obiektu przy użyciu klasyfikator

```
1      Mat image = imread(pathToPicture, CV_LOAD_IMAGE_COLOR);
2      if (!image.data)
3      {
4          cout << "Could not open or find the image\n";
5          exit(-1);
6      }
7      Mat frame_gray;
8      cvtColor(image, frame_gray, COLOR_BGR2GRAY);
9      equalizeHist(frame_gray, frame_gray);
10     vector<Rect> object;
11     cascadeClassifier.detectMultiScale(frame_gray, object);
12     for (size_t i = 0; i < object.size(); i++)
13     {
14         Point upperLeftCorner(object[i].x, object[i].y);
15         Point lowerRowCorner(object[i].x + object[i].width - 20, object[i].y +
16         object[i].height - 20);
17         rectangle(image, upperLeftCorner, lowerRowCorner, (255, 0, 255), 2);
18     }
19     string toSave = s->generateResultPath() + "/result" + resultId + string(".jpg");
```


W wierszu 1 użyta została struktura `Mat` jest to macierz którą można wykorzystać do przechowywania zdjęć, jest trójwymiarowa. Pierwsze dwa wymiary opisują współrzędne poszczególnych pikseli a trzeci jest obiekt wektora który przechowuje informacje o wartościach kolorów. `Imread` to funkcja która służy do wczytywania obrazów, jako parametry przyjmuje ścieżkę do rysunku i sposób w jakim chcemy go zapisać, w tym wypadku w formacie BGR (z ang. blue green red). Od 2 do 6 linii, sprawdzamy czy obiekt został znaleziony i poprawnie wczytany. W miejscu oznaczonym numerem 7 tworzymy nową macierz by za pomocą metody `cvtColor` z wiersza 8 przekształcić go z oryginalnego formatu kolorów do skali szarości jej argumentami są zdjęcie które chcemy przekształcić, zmienna do której chcemy zapisać przekonwertowany obraz a następnie sposób zmiany. Operacja ta jest wymagana ponieważ rysunki użyte do szkolenia również powinny być monochromatyczne. Funkcja z wiersza numer 9 `equalizeHist` jest używana w celu zwiększenia kontrastu. Jej pierwszym parametrem jest zdjęcie źródłowe, a drugim jest gdzie chcemy zapisać rezultat operacji. Poprzez użycie tej metody możemy lepiej wyeksponować różnice między elementami obrazu. Przykład przedstawiono na zdjęciu 3.5. Klasa `vector<Rect>` z linii 10 zostanie wykorzystana do przechowywania prostokątów wewnątrz którego funkcja `detectMultiScale` z wiersza 11 znalazła wyuczony przedmiot. Ta metoda może być wykonana tylko na zmiennej przechowującej wczytany klasyfikator. Jej argumentami są obraz który chcemy przeszukać oraz wektor do którego może zapisywać znalezione obiekty. W liniach od 12 do 17 iterujemy po wszystkich odzyskanych miejscach i nanosimy na nasz rysunek ramkę w której on się znajduje. W wiersie 19 użyta została metoda do zapisu obrazu na dysk twardy. Jej pierwszym od lewej parametrem jest lokalizacja łącznie z nazwą pliku a następną struktura zawierająca to zdjęcie. Po poprawnej detekcji powinniśmy uzyskać rezultat podobny do tego przedstawionego na zdjęciu 3.6.



Rys. 3.5. Przykład transformacji za pomocą metody `equalizeHist`



Rys. 3.6. Znaleziony i oznaczony obiekt

4. LBP

Algorytm wywodzi się pierwotnie od powstałego w 1990 widma tekstury[7]. Jeżeli mamy obraz to każdy jego piksel może zostać opisany zależnie od swojego otoczenia. Aby to zrobić należy dla każdego piksela utworzyć macierz 3x3 z natężeniami koloru (Rys. 4.1), zawierającą otaczające go piksele. Każdy z elementów macierzy musi zostać porównany do centralnego i w zależności od tego czy jest mniejszy, tożsamy czy większy otrzymujemy kolejno wartości 0,1 i 2 (Rys. 4.2a). Wartości sprawdzamy od lewego górnego rogu zgodnie ze wskazówkami zegara, tworząc w ten sposób wektor. Następnie jeżeli zastosujemy wzór:

$$\sum_{i=1}^8 E_i * 3^{i-1}$$

gdzie E_i to kolejne wartości tablicy, otrzymamy nową wartość dla środkowego piksela. Przykład obliczania przedstawiono na rysunku 4.2. W 1994 roku, grupa naukowców z uniwersytetu Oulu w Finlandii, na podstawie widma tekstury utworzyła definicję LBP[6]. Główną różnicą są dwa stany zamiast trzech co zmniejszyło wartość jaką przyjmuje środkowy piksel po transformacji z 6561 do 256 co znacząco upraszcza obliczenia. Inną zmianą jest odczytywanie wartości wierszami zamiast zgodnie z ruchem wskazówek zegara. Przykład klasyfikacji tekstury za pomocą LBP przedstawiono na obrazie 4.3.

102	230	184
58	104	104
16	170	20

Rys. 4.1. Przykładowa macierz

a)	b)	c)	d)																																				
<table><tr><td>0</td><td>2</td><td>2</td></tr><tr><td>0</td><td></td><td>1</td></tr><tr><td>0</td><td>2</td><td>0</td></tr></table>	0	2	2	0		1	0	2	0	<table><tr><td>1</td><td>3</td><td>9</td></tr><tr><td>2187</td><td></td><td>27</td></tr><tr><td>729</td><td>243</td><td>81</td></tr></table>	1	3	9	2187		27	729	243	81	<table><tr><td>0</td><td>2*3</td><td>2*9</td></tr><tr><td>0</td><td></td><td>1*27</td></tr><tr><td>0</td><td>2*243</td><td>0</td></tr></table>	0	2*3	2*9	0		1*27	0	2*243	0	<table><tr><td>0</td><td>6</td><td>18</td></tr><tr><td>0</td><td></td><td>27</td></tr><tr><td>0</td><td>486</td><td>0</td></tr></table>	0	6	18	0		27	0	486	0
0	2	2																																					
0		1																																					
0	2	0																																					
1	3	9																																					
2187		27																																					
729	243	81																																					
0	2*3	2*9																																					
0		1*27																																					
0	2*243	0																																					
0	6	18																																					
0		27																																					
0	486	0																																					

Rys. 4.2. Etapy obliczania widma natężenia

a) Porównanie wartości natężenia, b) tabela wartości poszczególnych sąsiadów, c) pomnożone przez siebie tabele a i b, d) obliczone wartości

a)	b)	c)																											
<table border="1"> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td></td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	1	0		1	0	1	0	<table border="1"> <tr><td>1</td><td>2</td><td>4</td></tr> <tr><td>8</td><td></td><td>16</td></tr> <tr><td>32</td><td>64</td><td>128</td></tr> </table>	1	2	4	8		16	32	64	128	<table border="1"> <tr><td>0</td><td>2</td><td>4</td></tr> <tr><td>0</td><td></td><td>16</td></tr> <tr><td>0</td><td>64</td><td>0</td></tr> </table>	0	2	4	0		16	0	64	0
0	1	1																											
0		1																											
0	1	0																											
1	2	4																											
8		16																											
32	64	128																											
0	2	4																											
0		16																											
0	64	0																											

Rys. 4.3. Etapy klasyfikacji tekstury za pomocą LBP

a) Porównanie wartości natężenia, b) tabela wartości poszczególnych sąsiadów, c) wartości po pomnożeniu przez siebie tabel a i b

4.1. Usprawniony LBP[14]

Ze względu na swoją budowę, LBP zaproponowany w 1994 roku, bardzo słabo radził sobie ze zmianami w oświetleniu, rozmyciu i przybliżeniom. W 2009 roku pojawiła się propozycja usprawnienia do oryginalnego pomysłu. Zamiast porównywać każdy piksel, z centralnym w nowej wersji sprawdzane są tylko przeciwległe piksele (rys. 4.4b), nie tylko zmniejsza to ilość możliwych kombinacji z 256 do 16 ale również skraca czas obliczeń i sprawia, że algorytm jest mniej podatny na różnego rodzaju zakłócenia na obrazie. Obliczanie nowej wartości rozpoczyna się od lewego sąsiada i postępuje zgodnie z ruchem wskazówek zegara. Przykład przedstawiono na rysunku 4.3.

a)	b)	c)									
<table border="1"> <tr><td>n5</td><td>n6</td><td>n7</td></tr> <tr><td>n4</td><td></td><td>n0</td></tr> <tr><td>n3</td><td>n2</td><td>n1</td></tr> </table>	n5	n6	n7	n4		n0	n3	n2	n1	$\begin{aligned} n0 >= n4 &= 1 \\ n1 >= n5 &= 0 \\ n2 >= n6 &= 0 \\ n3 >= n7 &= 0 \end{aligned}$	$\begin{aligned} 1*2^0 + \\ 0*2^1 + \\ 0*2^2 + \\ 0*2^3 + \end{aligned}$
n5	n6	n7									
n4		n0									
n3	n2	n1									

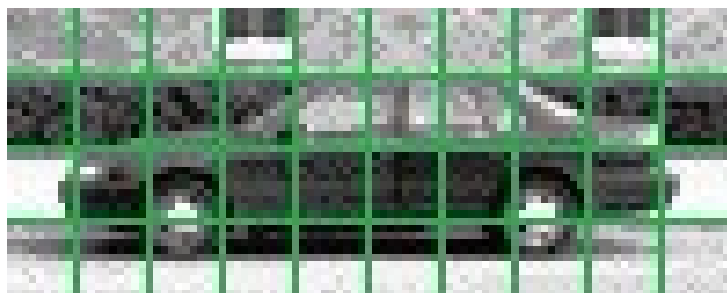
Rys. 4.4. Przykład usprawnionego LBP dla macierzy z rysunku 4.1.

a) Indeksowanie tabel, b) Porównanie indeksów, c) Obliczanie nowej wartości

4.2. Histogram zorientowanych gradientów[9]

W celu stworzenia narzędzia do detekcji połączono klasyfikator tekstu LBP razem z Histogramem zorientowanych gradientów. HOG służy do wyznaczania cech na podstawie kierunku gradientów. Wykorzystuje on fakt, że różnica w natężeniu pikseli jest większa przy krawędziach i rogach obiektu, co daje więcej informacji o obiekcie niż równe regiony.

W celu opisania zdjęcia, algorytm musi wykonać kilka kroków. W pierwszej kolejności domyślny obraz należy podzielić na wiele mniejszych fragmentów, tak jak na rysunku 4.4. Kolejnym etapem jest policzenie histogramu, którym jest wektor zawierający informacje o kątach danego gradientu. Skala może być od 0 do 180 stopni jeżeli wartości ujemne rozpatrujemy jak dodatnie lub do 360 stopni. Następnym etapem jest normalizacja otrzymanych tablic, zaczynając od lewego górnego rogu, uwzględniane są cztery mniejsze regiony, które razem tworzą jedną większą strefę i na takim obiekcie dokonywane są działania. Operacja zostaje powtórzona kolejno dla sąsiedniego rozszerzonego dystryktu po prawej stronie, a następnie dla tych poniżej. Otrzymany HOG naniesiony na zdjęcie z którego został wyliczony powinien być wyraźny na konturach obiektu.



Rys. 4.5. Przykład podziału zdjęcia

4.3. Uczenie

Przetworzone przez klasyfikator zdjęcia poddawane są analizie statystycznej przez maszynę wektorów wspierających. Na podstawie tych danych mechanizm buduje klasę obiektu za pomocą której można rozpoznawać obiekt[10].

Każdy z obrazów zawiera cechy, w wypadku LBP są to pojedyncze piksele poddane transformacji a w przypadku HOG są zorientowane gradienty małych regionów. Na podstawie porównania wielu próbek zawierających obiekt, SVN próbuje zdefiniować te wartości które najlepiej reprezentują dany obiekt[11]. Ważnym elementem jest dobra standaryzacja próbek, jeżeli będą się zbyt różniły klasyfikator może się nauczyć rozpoznawać szum. Kolejnymi rzeczami z jakimi zmagają się twórcy klasyfikatorów jest niedouczenie lub przeuczenie. Pierwszy z nich polega na posiadaniu zbyt skromnej bazy treningowej, co skutkuje w słabym wyuczeniu cech, co może powodować znajdowanie obiektów tam gdzie ich nie ma. Drugi natomiast charakteryzuje się tym, że niemal perfekcyjnie znajduje obiekty na obrazach na których się nauczył, natomiast słabo radzi sobie z innymi zdjęciami. Dzieje się tak ponieważ klasyfikator zbyt dobrze nauczył się rozpoznawać konkretne przypadki, bardziej niż obiekt który reprezentują[12]. Ogólnym celem jest dobranie takiego zestawu próbek który będzie pomiędzy tymi dwoma problemami.

4.4. Połączenie LBP + HOG

Udowodniono, że połączenie tych dwóch klasyfikatorów usprawnia detekcję[8][13]. Zaletą histogramu zorientowanych gradientów jest fakt, że dobrze radzi sobie z wykrywaniem krawędzi czy kształtu. Jego minusem natomiast jest problem z zagraconym tłem, który rozwiązuje LBP dobrze filtrując szumy. To powoduje, że obie metody są dla siebie komplementarne.

5. HAAR

Klasyfikator swój początek ma w 1997 roku, w algorytmie zaproponowanym przez Papageorgiou[16]. Jego implementacja zawierała użycie falek haara, dzięki czemu zawdzięcza swoją nazwę. Działanie polegało na zbudowaniu zestawu cech składających się z dwu wymiarowych falek haara które opisują stosunek głębie koloru pomiędzy poszczególnymi regionami. Na rysunku przedstawiono przykład cech, kolorem czarnym oznaczono regiony o niskim natężeniu a białym wysokie.



Rys. 5.0. Podstawowe regiony zaproponowane w 1997r.

5.1. Viola Jones[17]

W 2001 roku pojawił się artykuł autorstwa Paul'a Viola'i i Michael'a Jones'a na temat nowego klasyfikatora również wykorzystujący Haar'a. Jego działanie opiera się na trzech filarach. Pierwszy z nich to utworzenie całki z obrazu, co przyspiesza obliczanie regionów na zdjęciu. Drugim jest wykorzystanie zmodyfikowanej wersji AdaBoost'a do wybrania cech i wyuczeniu klasyfikatora. Ostatnim wykorzystuje metody wspomagające wybór rejonów o wysokim prawdopodobieństwie występowania obiektu.

5.1.1. Całka z obrazu

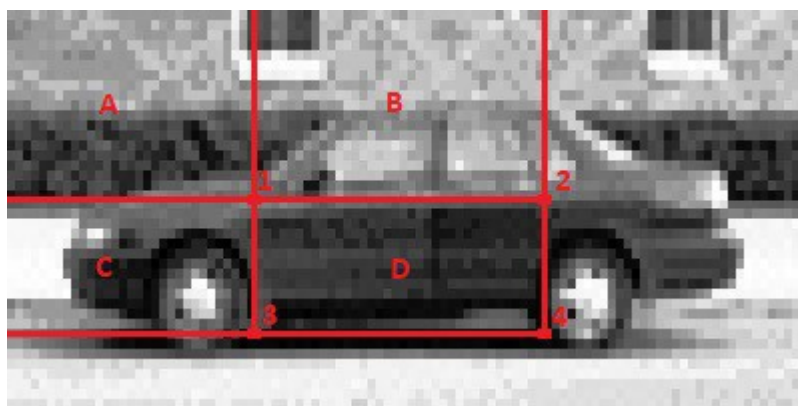
Obliczenie całki odbywa się za pomocą wzorów:

$$\begin{aligned}s(x, y) &= s(x, y-1) + i(x, y) \\ ii(x, y) &= ii(x-1, y) + s(x, y)\end{aligned}$$

gdzie, x i y określają kolejno poziome i pionowe koordynaty położenia punktu, $i()$ wartość natężenia dla aktualnego piksela, $s()$ suma dla wiersza i $ii()$ to wynik. Wystarczy jedna iteracja przez wszystkie piksele by otrzymać tablicę zawierającą skonwertowane zdjęcie. Aby obliczyć przykładowy region D dla rys. 5.1. wystarczy obliczyć sumę różnic:

$$D = (2 - 1) + (4 - 3)$$

Dzięki temu zadanie obliczenia pojedynczej strefy ma stałą złożoność obliczeniową.



Rys. 5.1. Przykład obliczania pojedynczego cechy Haar'a

5.2.2. Znajdowanie cech

Na pojedynczym obrazie można wyznaczyć niemal nieskończenie wiele cech. W 1998 roku P. Viola i J. S. DeBonet zaproponowali algorytm do rozpoznawania miejsc charakterystycznych[19]. W implementacji użyto 25 filtrów, na podstawie których tworzone są nowe obrazy lepiej reprezentujące wrażliwe miejsca. Dla przykładu nowo utworzony rysunek może mieć bardziej widoczne linie horyzontalne i linie wertykalne a jeszcze inne ukosy.

W późniejszej pracy P. Viola i K. Tieu[20] połączono filtry wraz z implementacją AdaBoost'a. Na podstawie skonwertowanych obrazów wyznaczane są cechy Haara które następnie otrzymują wagi które są modyfikowane przy porównaniu z próbkami pozytywnymi i negatywnymi. Duża ilość cech znacząco wpływa na jakość detekcji a także na zmniejszenie fałszywych trafień jednak znacząco też wydłuża czas detekcji dlatego istotnym jest też wybór najlepszych.

X. Wykaz Literatury

1. 1 <https://pdfs.semanticscholar.org/d486/9863b5da0fa4ff5707fa972c6e1dc92474f6.pdf>
2. 2 <https://opencv.org/about/>
3. 3 <https://pdfs.semanticscholar.org/40b1/0e330a5511a6a45f42c8b86da222504c717f.pdf>
4. 4 https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html
5. 5 https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html
6. 6 T. Ojala, M. Pietikäinen, and D. Harwood (1994), "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions", Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994), vol. 1, pp. 582 - 585.
7. 7DC. He and L. Wang (1990), "Texture Unit, Texture Spectrum, And Texture Analysis", Geoscience and Remote Sensing, IEEE Transactions on, vol. 28, pp. 509 - 512.
8. 14 M. Heikkilä, M. Petikäinen, C. Schmid, "Description of interest regions with local binary patterns", "Pattern Recognition" Volume 42, Issue 3, March 2009, Pages 425-436
9. 8 "An HOG-LBP Human Detector with Partial Occlusion Handling", Xiaoyu Wang, Tony X. Han, Shuicheng Yan, ICCV 2009
10. 9 <https://www.learnopencv.com/histogram-of-oriented-gradients/>
11. 10 <https://docs.marklogic.com/guide/search-dev/classifier>
12. 11 http://www.improvedoutcomes.com/docs/WebSiteDocs/Classification_and_Prediction/S_LAM/An_Introduction_to_Classification.htm
13. 12 <https://elitedatascience.com/overfitting-in-machine-learning>
14. 13 M. Ghorbani, A. Tavakoli, M.M. Dehshibi (2015) „HOG and LBP towards a robust recognition system”, 2015 Tenth International Conference on Digital Information Management (ICDIM)
15. 15 Papageorgiou, Oren and Poggio, "A general framework for object detection", International Conference on Computer Vision, 1998.
16. 16 M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. „Pedestrian detection using wavelet templates”. In Computer Vision and Pattern Recognition, pages 193-99, 1997.
17. 17 P. Viola, M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features"
18. 18 Yoav Freund and Robert E. Schapire. „A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt '95,

pages 23–37. Springer-Verlag, 1995.

19. J. S. DeBonet and P. Viola. Structure driven image database retrieval. In Adv. Neur. Info. Proc. Sys., volume 10, 1998.
20. K. Tieu and P. Viola. Boosting Image Retrieval. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000

19

XI Spis rysunków

- 3.1. Przykładowe użycie programu annotation
- 3.2.1. Przykład utworzenia próbek za pomocą narzędzia createsample
- 3.2.2. Wyświetlanie próbek za pomocą narzędzia createsample
- 3.3. Przykład użycia narzędzia do trenowania klasyfikatora
- 3.4. Przykład pokazania cech za pomocą narzędzia do wizualizacji
- 3.5. Przykład transformacji za pomocą metody equalizeHist
- 3.6. Znaleziony i oznaczony obiekt
- 4.1. Przykładowa macierz.
- 4.2. Etapy obliczania widma natężenia
- 4.3. Etapy klasyfikacji tekstury za pomocą LBP
- 4.4. Przykład usprawnionego LBP dla macierzy z rysunku 4.1.
- 4.5. Przykład podziału zdjęcia
- 5.0. Podstawowe regiony zaproponowane w 1997r.
- 5.1. Przykład obliczania pojedynczego cechy Haar'a

xx