

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI  
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

# Praca dyplomowa inżynierska

na kierunku Informatyka  
w specjalności Inżynieria systemów informatycznych

Zastosowanie analizy obrazu twarzy do sterowania aplikacją na  
urządzeniach z systemem Android

Adamski Maciej

Numer albumu 300184

promotor  
prof. Przemysław Rokita

WARSZAWA 2021



# **Zastosowanie analizy obrazu twarzy do sterowania aplikacją na urządzeniach z systemem Android**

**Streszczenie.** \_\_\_\_\_ TODO \_\_\_\_\_

**Słowa kluczowe:** XXX, XXX, XXX

# Agnieński tytuł

**Abstract.** \_\_\_\_\_ TODO \_\_\_\_\_

**Keywords:** XXX, XXX, XXX

# Spis treści

<b>1. Sposób badania algorytmów</b> . . . . .	9
1.1. Dataset . . . . .	9
1.2. Urządzenie do testów . . . . .	9
<b>2. Detekcja twarzy</b> . . . . .	10
2.1. Algorytmy detekcji twarzy . . . . .	10
2.1.1. Klasyfikator kaskadowy . . . . .	10
2.1.1.1. Haar . . . . .	10
2.1.1.2. Local binary patterns . . . . .	10
2.1.2. Histogram zorientowanych gradientów . . . . .	11
2.1.3. CNN + MMOD . . . . .	12
2.1.4. Głębokie sieci neuronowe . . . . .	12
2.2. Filtrowanie wyników . . . . .	12
<b>3. Porównanie algorytmów detekcji twarzy</b> . . . . .	14
3.1. Odrzucenie algorytmu Dlib CNN MMOD . . . . .	14
3.2. Testowanie na statycznych zdjęciach . . . . .	14
3.2.1. Oczekiwany wynik . . . . .	15
3.2.2. Warunki testowania . . . . .	15
3.2.3. Badanie skuteczności detekcji . . . . .	15
3.2.4. Badanie szybkości detekcji . . . . .	18
3.2.5. Wpływ wielkości zdjęcia na szybkość algorytm <i>DNN Caffe</i> . . . . .	19
3.2.6. Precyzja detekcji algorytmu <i>DNN Caffe</i> zależnie od sposobu filtracji .	20
3.3. Testowanie na obrazie z kamery na żywo . . . . .	20
3.3.1. Skuteczność detekcji . . . . .	21
3.3.2. Szybkość detekcji . . . . .	21
3.4. Wybór algorytmu . . . . .	21
<b>4. Facemark</b> . . . . .	22
4.1. Local binary features . . . . .	22
4.2. Kazemi . . . . .	22
<b>5. Porównanie algorytmów detekcji facemarków</b> . . . . .	23
5.1. Testowanie na statycznych zdjęciach . . . . .	23
5.1.1. Usunięcie części zdjęć z datasetu . . . . .	23
5.1.2. Badanie skuteczności detekcji . . . . .	23
5.1.3. Badanie szybkości detekcji . . . . .	24
5.2. Testowanie na obrazie z kamery na żywo . . . . .	24
5.2.1. Skuteczność detekcji . . . . .	24
5.2.2. Szybkość detekcji . . . . .	25
5.3. Wybór algorytmu . . . . .	25

<b>6. EAR - Eye Aspect Ratio . . . . .</b>	26
6.1. Wzór obliczania EAR . . . . .	26
6.2. Zasada działania EAR w kontekście mrugania i określenia czy oko jest otwarte/zamknięte . . . . .	26
6.3. Wyznaczenie progu EAR . . . . .	27
6.3.1. Statyczne zdjęcia . . . . .	27
6.3.2. Obraz z kamery . . . . .	28
6.4. Wnioski . . . . .	29
<b>7. Detekcja oczu . . . . .</b>	30
7.1. Algorytmy detekcji oczu . . . . .	30
7.1.1. Klasyfikator kaskadowy Haar . . . . .	30
7.1.2. Facemarki . . . . .	30
7.2. Filtrowanie wyników metody Haar . . . . .	31
7.3. Obcięcie obszaru detekcji dla metody Haar . . . . .	31
7.4. Dostosowanie wielkości obszaru facemarków oczu . . . . .	33
<b>8. Porównanie algorytmów detekcji oczu . . . . .</b>	34
8.1. Testowanie na statycznych zdjęciach . . . . .	34
8.1.1. Oczekiwany wynik . . . . .	34
8.2. Warunki testowania . . . . .	34
8.3. Badanie skuteczności detekcji . . . . .	35
8.3.1. Badanie szybkości detekcji . . . . .	35
8.4. Testowanie na obrazie z kamery na żywo . . . . .	36
8.4.1. Skuteczność detekcji . . . . .	36
8.4.2. Szybkość detekcji . . . . .	37
8.5. Wybór algorytmu . . . . .	37
<b>9. Detekcja żrenic . . . . .</b>	38
9.1. Algorytm CDF (Cumulative Distribution Function) . . . . .	38
9.1.1. Kroki algorytmu . . . . .	38
9.1.2. Wynik kolejnych etapów algorytmu . . . . .	39
9.2. Algorytm PF (Projection Function) . . . . .	39
9.2.1. Funkcja projekcji . . . . .	40
9.2.1.1. Funkcja całkowa . . . . .	40
9.2.1.2. Funkcja wariancji . . . . .	40
9.2.1.3. Funkcja ogólna . . . . .	40
9.2.2. Kroki algorytmu . . . . .	41
9.2.3. Wynik kolejnych etapów algorytmu . . . . .	41
9.3. Algorytm EA (Edge Analysis) . . . . .	42
9.3.1. Kroki algorytmu . . . . .	42
9.3.2. Wynik kolejnych etapów algorytmu . . . . .	42

<b>Bibliografia</b>	43
<b>Spis rysunków</b>	46
<b>Spis tabel</b>	47



# 1. Sposób badania algorytmów

Algorytmy będę badał i porównywał przy pomocy zarówno statycznych zdjęć z przygotowanego datasetu, jak i korzystając z obrazu na żywo z przedniej kamery urządzenia.

## 1.1. Dataset

Na potrzeby badań i porównania algorytmów przygotowałem dataset składający się z 80 zdjęć zawierających twarze.

Źródła zdjęć:

- 50 zdjęć wybranych z datasetu *Young and Old Images Dataset* [1]
- 30 zdjęć mojego autorstwa

Dataset został przygotowany w taki sposób, żeby zawierał zróżnicowane zdjęcia pod względem względami, takimi jak: jakość obrazu, oświetlenie, powierzchnia zajmowana przez twarz, kolorystyka, płeć, kolor skóry, częściowe zakrycie twarzy, okulary czy odwrócona w bok głowa. Wszystkie 80 zdjęć zostało opisanych przeze mnie na potrzeby badań, w szczególności: obszar twarzy, oczu czy środek źrenic.

Dodatkowo do badania detekcji źrenic zostanie użyty *MRL Eye Dataset* [2] zawierający zdjęcia oczu wraz z pozycją środka źrenicy.

## 1.2. Urządzenie do testów

Wszystkie testy przeprowadzę będą na urządzeniu Huawei P30 Pro w normalnym trybie wydajności i bez włączonego oszczędzania energii.

## 2. Detekcja twarzy

W przetwarzaniu cyfrowym obrazu stosujemy technikę od ogółu do szczegółu. Przykładowo chcąc uzyskać barwę nadwodzia najpierw musimy wykryć samochód itp. W pracy dyplomowej by uzyskać informacje o oczach czy ustach potrzebujemy najpierw informacji o wystąpieniu twarzy i gdzie się ona znajduje. Dlatego pierwszym etapem przetwarzania jest detekcja ludzkiej twarzy. Chcemy z obrazu uzyskać informację o jej położeniu i w następnych etapach operować tylko na wycinku zdjęcia.

### 2.1. Algorytmy detekcji twarzy

Na potrzeby projektu zostanie zaimplementowane pięć algorytmów, których celem jest wykrycie twarzy na zdjęciu. Krótki opis poszczególnych metod znajduje się w następnym podrozdziale. Z zaproponowanych rozwiązań zostanie finalnie wybrane jedno po serii badań i testów mających na celu wybór zarówno skutecznego jak i szybkiego.

#### 2.1.1. Klasyfikator kaskadowy

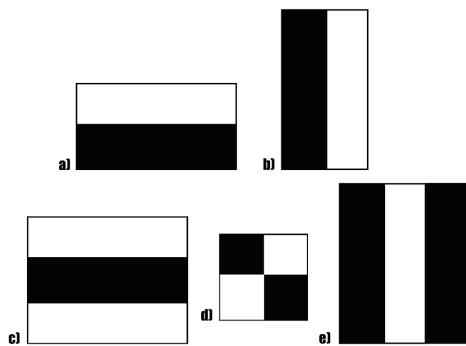
*Cascade Classifier* to jedno z podejść do zadania klasyfikacji obiektów. Kaskadowość przejawia się tym, że klasyfikator składa się z łańcucha mniejszych klasyfikatorów. Z danych wejściowych jednego może korzystać następnym jako dodatkowe źródło informacji i użyć je do własnej klasyfikacji. Z tego powodu kolejne elementy są bardziej zaawansowane i operują na większym zestawie danych. Dzięki swojej kaskadowej naturze modele takie mogą być lepiej trenowane i dawać lepsze rezultaty niż klasyfikatory typu monolity.

Do ładowania i przetwarzania kaskadowych klasyfikatorów w projekcie zostanie użyty moduł *CascadeClassifier* [3] biblioteki *OpenCV* [4].

**2.1.1.1. Haar** Jednym z najbardziej znanych modeli klasyfikacji kaskadowej jest *Haar*, który został opisany po raz pierwszy w 2001 roku [5]. Może on być używany do klasyfikacji różnych obiektów, ale autorzy skupiali się głównie na detekcji twarzy. Algorytm [6] [7] [8] bazuje na podzieleniu zdjęcia na regiony i wykorzystaniu w każdej z nich pięciu cech krawędzi (patrz *rysunek 2.1*). Algorytm porównując jasność pikseli w białej i czarnej części stwierdza czy istnieją krawędzie lub linie. Cechy składające się tylko z dwóch regionów odpowiadają za wykrycie pionowych i poziomych krawędzi. Zestaw trzech za wykrycie linii. Natomiast kwadratowa cecha za zmiany przekątne. W dzisiejszych czasach *Haar* nie jest już tak często stosowany jak jeszcze parę lat temu.

Na potrzeby pracy dyplomowej zostanie wykorzystany model *Haarcascade Frontalface Default* [9] autorstwa Rainera Lienharta.

**2.1.1.2. Local binary patterns** Metoda ta porównuje piksele z ośmioma swoimi najbliższymi sąsiadami w ustalonej kolejności. Jeśli jasność głównego piksela jest większa



**Rysunek 2.1.** Obliczane cechy w modelu Haar. Źródło: [6]

niż porównywanego to na odpowiedniej pozycji 8-bitowej liczby wstawia 1, inaczej 0. Następnie z uzyskanych w ten sposób liczb tworzy histogram używany jako deskryptor cech. Te same dane mogą być użyte do uczenia maszynowego. [10]

Jest to metoda cechująca się wysoką szybkością działania i z tego powodu stosowana w systemach z ograniczonymi zasobami sprzętowymi. Niestety kosztem efektywności.

W projekcie użyty będzie model *LBP Cascade Frontalface* [11].

### 2.1.2. Histogram zorientowanych gradientów

Metoda *HOG* (*Histograms of Oriented Gradients*) [12] została opracowana kilkanaście lat temu przez Navneet Dalal i Bill Triggs celem detekcji ludzkiego ciała. Aktualnie, mimo upływu lat, jest wciąż szeroko wykorzystywana do klasyfikacji obrazów czy wykrywania twarzy.

Uzyskanie histogramu HOG składa się z kilku etapów. Metoda [13] [14] [15] ta bazuje na obliczeniu gradientów poziomych i pionowych. Możliwe jest to przez filtrowanie za pomocą odpowiedniego jądra lub przez operator Sobela [16]. Dla tak wyodrębnionych gradientów oblicza się ich długość i kierunek (kąt). Następnie dzielimy zdjęcie na obszary o wielkości  $8 \times 8$ . Dla każdego regionu tworzymy jednowymiarowy wektor o 9 komórkach, w których będzie zapisany histogram HOG. Pola wektora odzwierciedlają kierunek gradientu i odpowiadają kolejnym wielokrotnościami kąta  $\angle 20^\circ$ . Wypełniamy go dodając do pól odpowiadającym danemu kątowi wartość gradientu kolejnych pikseli. Jeśli kierunek znajduje się pomiędzy dwoma kątami to wartość dzieli się zależnie od różnicy między dwiema komórkami. Celem wyeliminowania wpływu jasności i oświetlenia przeprowadza się normalizację wartości. Gdy obliczy się już histogram dla każdego regionu, łączy się je w wektor deskryptora cech HOG. Tak uzyskany wektor możemy wykorzystać jako dane uczące algorytmów klasyfikujących. W przypadku metody HOG często wykorzystuje się maszynę wektorów nośnych (SVM) [17].

W projekcie zostanie użyta metoda *HOG* z biblioteki *dlib*, która uczona była z użyciem liniowego *SVM*.

### 2.1.3. CNN + MMOD

Konwolucyjne sieci neuronowe (CNN) uczą się jakie cechy obrazu pozwalają sklasyfikować widoczne na nim obiekty. Za pomocą operacji splotowych, nakładając odpowiednie filtr są wstanie je uwypuklić i uzyskać istotne informacje. To właśnie w warstwie konwolucyjnej używane są odpowiednie jądra. Sieć przez trening sama dobiera optymalne filtry oraz ich wartości. Dodatkowo istnieją tutaj warstwy próbkowania (downsampling), których celem jest zmniejszenie wielkości obrazu przez pominięcie części pikseli. Pomaga to uprościć sieć, ale kosztem utraty pewnej ilości informacji. Czasem zamiast pomijać piksele brane są wartości uśredniane lub maksymalne z pewnego sąsiedztwa. [18]

W bibliotece *dlib* sieć CNN często jest zestawiona z metodą *Max-Margin Object Detection (MMMOD)* [19]. Służy ona do optymalizacji i zwiększenia prędkości detekcji obiektów.

Taka implementacja CNN+MMOD dostępna w *dlib* zostanie użyta w projekcie.

### 2.1.4. Głębokie sieci neuronowe

Głębokie sieci neuronowe (DNN) różnią się od klasycznych tym, że mają większą liczbę warstw ukrytych. Taki algorytm tworzy plamki o ustalonej wielkości ze zdjęć wejściowych, a następnie przepuszcza je przez kolejne warstwy sieci celem wykrycia pożądanych obiektów. Na wyjściu podaje prawdopodobieństwo, że na obrazie znajduje się interesujący nas element.

W projekcie zostanie wykorzystany do tego jeden z modułów biblioteki *OpenCV* zawierający implementację DNN [20]

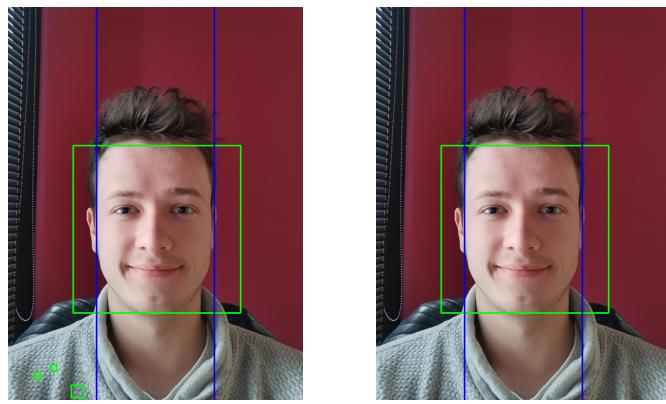
Jednym z modeli dostępnych do detekcji twarzy przy pomocy głębokich sieci neuronowych są modele Caffe (*Convolutional Architecture for Fast Feature Embedding*) [21]. W projekcie zostanie użyty wzorzec *caffe res10\_300x300\_ssd\_iter\_140000\_fp16* [22].

## 2.2. Filtrowanie wyników

Użyte algorytmy mogą dawać w wyniku błędnie określone obszary twarzy. Z tego względu zwróconą tablicę obszarów poddaję filtrowaniu.

Proces ten składa się z następujących etapów:

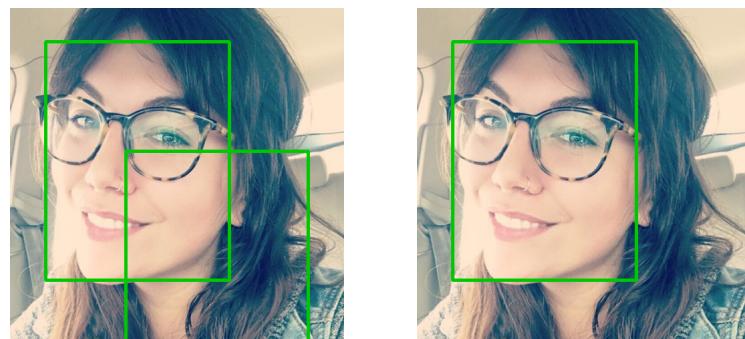
- Na początku odrzucam obszary, których środek znajduje się poza ustalonym pionowym obszarem (przyjąłem przedział [0.25, 0.75] szerokości). Wynika to z założeń, że osoba używająca telefonu, korzysta z niego patrząc na wprost, a nie z boku. Natomiast odchył od pionu to indywidualne preferencje - dlatego nie określам poziomego obszaru. (patrz *Rysunek 2.2*)
- Kolejnym etapem jest odrzucenie tych detekcji, które wychodzą zbyt daleko poza zdjęcie. Jeśli którykolwiek z boków prostokąta wystaje pionowo/poziomo o odległość większą niż 10% odpowiednio wysokości/szerokości to zostaje odrzucony. (patrz *Rysunek 2.3*)



(a) Przed filtrowaniem zależnym od położenia

(b) Po filtrowaniu

**Rysunek 2.2.** Działanie filtrowania detekcji twarzy w oparciu o położenie twarzy w centralnej części zdjęcia.

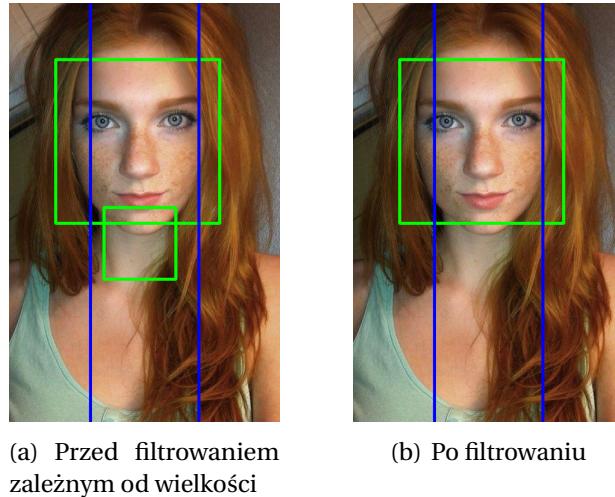


(a) Przed filtrowaniem zależnym od wystawania poza obraz

(b) Po filtrowaniu

**Rysunek 2.3.** Działanie filtrowania detekcji twarzy w oparciu o odległość wykrytego obszaru poza zdjęcie.

- Z pozostałych obszarów wybieram ten, który zajmuje największą powierzchnię. Taki wybór motywuję własnymi obserwacjami zachowania algorytmów detekcji twarzy oraz tym, że głowa użytkownika telefonu na obrazie z kamery przedniej zajmuje większą część płaszczyzny, ponieważ korzystając z urządzenia nie trzymamy go bardzo daleko od siebie. (patrz Rysunek 2.4)



**Rysunek 2.4.** Działanie filtrowania detekcji twarzy w oparciu o wielkość wykrytego obszaru. Źródło zdj.: [23]

### 3. Porównanie algorytmów detekcji twarzy

W rozdziale 2. *Detekcja twarzy* przedstawiłem kilka metod, które zostały zaimplementowane w projekcie. Ze względu, że dla prawidłowego i akceptowalnego działania aplikacji potrzebna jest odpowiednia skuteczność i szybkość detekcji twarzy, przetestuję i porównam wszystkie metody (z wyjątkiem *CNN MMOD* - patrz rozdz. 3.1). Na podstawie wyników wybiorę jedną, której będę używał w dalszej części projektu.

#### 3.1. Odrzucenie algorytmu Dlib CNN MMOD

Ze względu na bardzo wolne działanie algorytmu dlib opartego na konwolucyjnych sieciach neuronowych MMOD nie zostanie on przetestowany i zostaje od razu odrzucony. Czas przetwarzania jednego zdjęcia 500x500 wynosił kilka sekund, co całkowicie uniemożliwia działanie aplikacji w czasie rzeczywistym. Bardzo niska prędkość detekcji prawdopodobnie wynika z niemożności skorzystania z obliczeń na karcie graficznej na urządzeniach mobilnych. Metoda ta jest bardzo szybka gdy wykorzystuje do działania takie architektury jak CUDA [24], natomiast dużo gorzej radzi sobie z obliczeniami wykonywanymi na procesorach CPU.

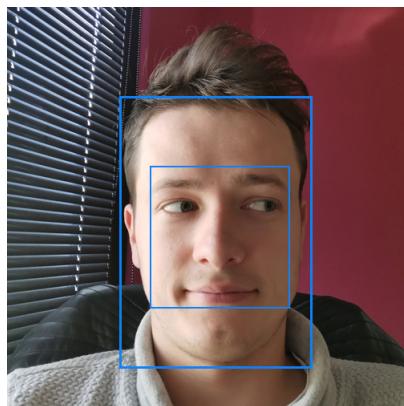
#### 3.2. Testowanie na statycznych zdjęciach

Pierwszy etap testowania algorytmów detekcji twarzy będzie bazował na statycznych zdjęciach z głównego datasetu (patrz rozdz. 1.1. *Dataset*). Pozwoli to przetestować na jednakowych danych wszystkie metody pod względem ich skuteczności wykrywania docelowych obszarów w różnych warunkach oraz uzyskać miarodajne wyniki.

### 3.2.1. Oczekiwany wynik

Każde zdjęcie z datasetu do tego etapu opisałem przez dwa prostokąty między którymi powinna się znaleźć wykryta przez algorytm twarz. Obszar ten został dobrany w następujący sposób:

- Wewnętrzna część obejmuje minimalny obszar, na którym znajdują się brwi, oczy, nos i usta.
- W zewnętrznym prostokącie powinna znaleźć się cała twarz. Powiększony jest on o pewną tolerancję.



Rysunek 3.1. Oczekiwany obszar detekcji twarzy.

### 3.2.2. Warunki testowania

Dla każdego algorytmu zostaną przeprowadzone testy na zestawach obrazów o następujących rozdzielczościach i przestrzeniach barw:

- 300x300 RGB
- 500x500 RGB
- 300x300 skala szarości
- 500x500 skala szarości

W przypadku *DNN Caffe* nie jest możliwe przeprowadzenie badań dla zdjęć w skali szarości, ponieważ wymaga on obrazu z trzema kanałami barw.

Testy będą przeprowadzone w trybie *release*, ponieważ w trybie *debuggowania* algorytm *Dlib HOG* działał nawet 180 razy wolniej. W przypadku pozostałych algorytmów tryb budowania nie miał większego wpływu na prędkość obliczeń, ale żeby wyniki były jak najbardziej miarodajne to każdy test musi być przeprowadzony w tych samych warunkach.

### 3.2.3. Badanie skuteczności detekcji

W tym teście zostaną zebrane i porównane następujące dane:

- **Prawidłowe detekcje** - suma perfekcyjnych i częściowo dobrych detekcji

### 3. Porównanie algorytmów detekcji twarzy

---

- **Perfekcyjne detekcje** - jeśli wykryty obszar w pełni znajduje się pomiędzy oczekiwany prostokątami
- **Częściowo dobre detekcje** - jeśli są krawędzie, które znajdują się poza oczekiwany obszarem, ale w zadowalającej odległości (patrz niżej - *Uwaga 1.*)
- **Z 3 na 4 krawędzie perfekcyjne detekcje** - jeśli tylko jedna krawędź znajduje się poza oczekiwany obszarem w zadowalającej odległości (patrz niżej - *Uwaga 2.*)
- **Złe detekcje** - jeśli twarz nie została wykryta lub wskazany obszar jest niezadowalający
- **Twarze niewykryte** - jeśli całkowicie nie udało się wykryć twarzy (patrz niżej - *Uwaga 3.*)

*Uwaga 1.* Obszar uznany jest za częściowo dobry jeśli żadna krawędź nie jest oddalona o więcej niż  $1.2x$  i maksymalnie jedna oddalona jest o długość z przedziału  $[1.1x, 1.2x]$ . Odległość  $x$  to szerokość lub wysokość (zależnie od krawędzi) maksymalnego oczekiwanej obszaru twarzy.

*Uwaga 2.* Obszar zaliczony jest do grupy 3/4 perfekcyjnych detekcji, jeśli 3 krawędzie znajdują się w oczekiwany obszarze, a czwarta odchylona od normy w przedziale  $[1.0x, 1.2x]$ .

*Uwaga 3.* Dodatkowy podział zlej detekcji na niewykryte twarze wynika z faktu, że metody oparte o *Cascading Classifier* na wyjściu podają obszar kwadratowy i przy rozciągniętej lub pochylonej twarzy boczne obszary mogą być bardzo oddalone od oczekiwanej wartości, ale dalej wykryć twarz.

**Tabela 3.1.** Skuteczność algorytmów detekcji twarzy dla obrazów RGB

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
<b>Haar Cascade 500x500</b>	69	4	65	32	11	9
<b>Haar Cascade 300x300</b>	68	5	63	33	12	9
<b>LBP Cascade 500x500</b>	58	4	54	29	22	22
<b>LBP Cascade 300x300</b>	61	4	57	34	19	17
<b>DNN Caffe 500x500</b>	80	68	12	11	0	0
<b>DNN Caffe 300x300</b>	80	62	18	18	0	0
<b>Dlib HOG 500x500</b>	78	31	47	36	2	2
<b>Dlib HOG 300x300</b>	76	24	52	33	4	4

Metoda *Haar Cascade* daje średnio  $\sim 69/80$  (86%) dobrych detekcji. Jest to dosyć przecienny wynik. Na taki rezultat składa się kilka problemów tej metody. Nie radzi sobie ona dobrze z częściowo zakrytymi twarzami lub gdy głowa jest pochylona w bok. Kolejnymi czynnikiem wpływającym negatywnie na detekcję jest światło - problem z wykrywaniem występuje gdy zdjęcie jest zbyt jasne, twarz oświetlona lub źródło światła

**Tabela 3.2.** Skuteczność algorytmów detekcji twarzy dla obrazów w skali szarości

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
<b>Haar Cascade 500x500</b>	69	4	65	31	11	9
<b>Haar Cascade 300x300</b>	67	4	63	34	13	9
<b>LBP Cascade 500x500</b>	60	5	55	30	20	17
<b>LBP Cascade 300x300</b>	60	4	56	31	20	17
<b>DNN Caffe 500x500</b>	nd.	nd.	nd.	nd.	nd.	nd.
<b>DNN Caffe 300x300</b>	nd.	nd.	nd.	nd.	nd.	nd.
<b>Dlib HOG 500x500</b>	75	27	48	35	5	5
<b>Dlib HOG 300x300</b>	72	23	49	31	7	7

świeci prosto w obiektyw. Zaletą tej metody można zapisać małą ilość zwróconych przez nią dodatkowych, błędnych obszarów, które musiały zostać odfiltrowane.

*Klasyfikator kaskadowy* bazując na modelu *LBP* miał najgorsze wyniki detekcji twarzy, na poziomie ~ 60/80 (75%). Jednak co zwraca uwagę to fakt, że bardzo duży odsetek twarzy nie został w ogóle wykryty. Występują tu te same problemy co w *Haar Cascade*, ale dodatkowo algorytm nie radzi sobie gdy twarz zajmuje prawie całe zdjęcie.

Najlepszy wynik detekcji uzyskał bezdyskusyjnie *DNN Caffe*. Fakt, że w każdym z dwóch testów wykrył on 100% twarzy jest warty odnotowania. Co więcej perfekcyjne detekcje były na poziomie ~ 65/80 (81,25%). Nie występują tu problemy takie jak w poprzednich algorytmach. Radzi sobie on dobrze w złych warunkach oświetleniowych. Częściowe zakrycie twarzy nie wpływa na detekcję. Wykrywa on dobrze zarówno pochycone jak i odwrócone twarze. Jedynym negatywnym zjawiskiem, które zaobserwowałem w tej metodzie to zwracanie wielu dodatkowych obszarów, które są błędne. Zastosowanie filtrowania pozwoliło jednak odrzucić wszystkie błędne obszary.

Bardzo dobre wyniki detekcji uzyskał również algorytm *Dlib HOG*, w szczególności w przypadku zdjęć RGB 500x500 - jego skuteczność była na poziomie 78/80 (97,5%). Zaletą tej metody jest zwracanie tylko jednego wykrytego obszaru - na żadnym z 80 zdjęć nie zwrócił ani jednego dodatkowego miejsca, które uznał za twarz. Nie udało się mu się wykryć twarzy gdy była ona w połowie zakryta. Zakładając jednak, że aplikacja będzie wykorzystywać oczy, usta itd. użytkownika przed telefonem można przyjąć, że jego twarz będzie w wystarczającym stopniu widoczna. W przeciwnieństwie do pozostałych metod, które zwracają obszar całej twarzy, ta wykrywa częściowo obcięty rejon - np. pomijając czoło. Nie jest to w ogólności wadą, ponieważ te części twarzy nie są konieczne w pozostałych etapach.

Różnica w procencie perfekcyjnych detekcji pomiędzy *DNN Caffe*, a pozostałymi wynika z rodzaju obszarów zwracanych przez te algorytmy. Metoda oparta na głębokich sieciach neuronowych zwraca prostokąt o dowolnym stosunku boków, natomiast reszta

### 3. Porównanie algorytmów detekcji twarzy

---

zwraca kwadrat. Dzięki temu *DNN* lepiej dopasowuje się do kształtu twarzy niż *Cascading Classifier* i *HOG*.

Zmiana różnicy barw nie przyniosła istotnych zmian w skuteczności działania poszczególnych algorytmów. Jedynie zauważalne obniżenie detekcji w skali szarości w porównaniu do RGB widoczne jest dla metody opartej na *histogramie gradientów zorientowanych Dlib*.

#### 3.2.4. Badanie szybkości detekcji

W tym teście zostaną zebrane i porównane następujące dane:

- **Całkowity czas przetwarzania** - suma czasów wszystkich 20 iteracji, całkowity czas testu.
- **Średni czas przetwarzania pojedynczej iteracji** - uśredniony czas pojedynczej iteracji
- **Średni czas przetwarzania jednego zdjęcia** - uśredniony czas przetwarzania pojedynczego zdjęcia

*Uwaga 1.* Celem miarodajnego wyniku czasu przetwarzania każdy test zostanie przeprowadzony 20 razy.

**Tabela 3.3.** Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
<b>Haar Cascade 500x500</b>	116,72 s	5,836 s	0,072 s
<b>Haar Cascade 300x300</b>	46,24 s	2,312 s	0,028 s
<b>LBP Cascade 500x500</b>	63,27 s	3,163 s	0,039 s
<b>LBP Cascade 300x300</b>	21,25 s	1,062 s	0,013 s
<b>DNN Caffe 500x500</b>	106,17	5,308	0,066 s
<b>DNN Caffe 300x300</b>	103,56 s	5,178 s	0,064 s
<b>Dlib HOG 500x500</b>	71,10 s	3,555 s	0,044 s
<b>Dlib HOG 300x300</b>	26,08 s	1,304 s	0,016 s

Najszybszy okazał się algorytm operujący na histogramach gradientowych. Niewiele wolniej przetwarzał algorytm kaskadowy *LBP*. *DNN Caffe* dla zdjęć 500x500 był porównywalnie szybki jak *Haar Cascade*, natomiast już w przypadku 300x300 około 2.5 razy wolniejszy.

Co ciekawe i wartere odnotowania to fakt, że algorytm *DNN* przetwarzał prawie tak samo szybko obie rozdzielcości zdjęć. Można wysnuć tezę, że dla tej metody wielkość obrazu nie ma wpływu na szybkość przetwarzania. Ze względu, że taka właściwość może

**Tabela 3.4.** Czas przetwarzania algorytmów detekcji twarzy dla obrazów w skali szarości

	<b>Całkowity czas przetwarzania</b>	<b>Średni czas przetwarzania pojedynczej iteracji</b>	<b>Średni czas przetwarzania pojedynczego zdjęcia</b>
<b>Haar Cascade 500x500</b>	116,52 s	5,826 s	0,072 s
<b>Haar Cascade 300x300</b>	45,93 s	2,296 s	0,028 s
<b>LBP Cascade 500x500</b>	62,34 s	3,117 s	0,038 s
<b>LBP Cascade 300x300</b>	21,64 s	1,082 s	0,013 s
<b>DNN Caffe 500x500</b>	nd.	nd.	nd.
<b>DNN Caffe 300x300</b>	nd.	nd.	nd.
<b>Dlib HOG 500x500</b>	58,60 s	2,930 s	0,036 s
<b>Dlib HOG 300x300</b>	21,88 s	1,094 s	0,013 s

okazać się przydatna w perspektywie dalszych etapów projektu, zamierzam zbadać tę zależność w następnym rozdziale.

Na szybkość detekcji algorytmu *HOG* niewątpliwie miało wpływ użycie go w języku C++ mimo narzutu związanego z wywoływaniem go przez interfejs *Java Native Interface*.

Zmiana detekcji z trójkanałowej RGB na skalę szarości w przypadku *Haar* i *LBP* nie skróciła czasu detekcji. W przypadku metody z biblioteki *Dlib* algorytm przetwarzał te zdjęcia ~ 15 – 20% krócej niż w wersji kolorowej.

### 3.2.5. Wpływ wielkości zdjęcia na szybkość algorytm *DNN Caffe*

**Tabela 3.5.** Wpływ rozdzielczości zdjęcia na detekcję DNN

	<b>Prawidłowe detekcje</b>	<b>Perfekcyjne detekcje</b>	<b>Częściowo dobre detekcje</b>	<b>Średni czas przetwarzania pojedynczej iteracji</b>	<b>Średni czas przetwarzania pojedynczego zdjęcia</b>
<b>300x300</b>	80	62	18	5,148 s	0,064 s
<b>500x500</b>	80	68	12	5,239 s	0,065 s
<b>1000x1000</b>	80	67	13	5,29 s	0,066 s
<b>2000x2000</b>	80	65	15	4,988 s	0,062 s

Test ten potwierdza postawioną przeze mnie wcześniej tezę, że wielkość zdjęcia nie ma wpływu na szybkość przetwarzania algorytmu *DNN Caffe*. Testy w każdej rozdzielczości zostały wykonane mniej więcej w tym samym czasie, a różnica zapewne jest skutkiem obciążenia urządzenia w danej chwili i jest pomijalna.

Prawdopodobnie wynika to z faktu, że metoda ta tworzy na podstawie zdjęcia wejściowego plamki o podanej wielkości niezależnie od rozdzielczości. W zaimplementowanym algorytmie jest to rozmiar 300x300. Dzięki temu zawsze ma on do przetworzenia taką samą ilość danych, więc czas powinien być w przybliżeniu stały.

### 3. Porównanie algorytmów detekcji twarzy

---

Skuteczność detekcji w każdym wariantie była przybliżona i uzyskiwała 100% prawidłowych wskazań.

#### 3.2.6. Precyza detekcji algorytmu *DNN Caffe* zależnie od sposobu filtracji

Metoda oparta na głębokich sieciach neuronowych na wyjściu zwraca wiele obszarów wraz ze wskaźnikiem pewności detekcji. Im większy współczynnik tym w teorii większa szansa, że jest to obiekt, który chcieliśmy wykryć.

Z tego powodu postanowiłem porównać autorskie filtrowanie opisane wcześniej (patrz rozdz. 2.2.*Filtrowanie wyników*) i wybór detekcji z największym procentem pewności.

**Tabela 3.6.** Wynik porównania sposobów filtrowania detekcji

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
Autorskie filtrowanie 300x300	80	62	18	18	0	0
Autorskie filtrowanie 500x500	80	68	12	11	0	0
Najwyższy współczynnik pewności 300x300	76	58	18	18	4	4
Najwyższy współczynnik pewności 500x500	76	64	12	11	4	4

Wyniki w tabeli 3.6 pokazują, że zaproponowana przeze mnie wcześniej sekwencja filtrowania wykrytych obszarów daje lepsze rezultaty niż wybór najwyższego współczynnika pewności.

### 3.3. Testowanie na obrazie z kamery na żywo

W teście opartym na statycznych zdjęciach najlepsze okazały się algorytmy *DNN* i *HOG*, a dodatkowo ten drugi był również najszybszy. Z tego względu w próbie wykorzystującej obraz na żywo badane będą wyłącznie te dwa algorytmy, a pozostałe odrzucone.

Obraz przechwytywany będzie w domyślnej rozdzielcości dla modułu CameraX - 640x480. [25]

Oba algorytmy zostaną przetestowane w czterech różnych warunkach:

- 1. w zwykłych, domowych warunkach oświetleniowych
- 2. w ciemnym miejscu
- 3. z intensywnym oświetleniem zza osoby
- 4. z intensywnym oświetleniem zza urządzenia

Zostaną zebrane informacje o procencie klatek z wykrytą twarzą oraz chwilową i średnią ilość klatek na sekundę.

Każdy test będzie trwał 210 klatek, ale pierwsze i ostatnie 5 nie będzie branych pod uwagę przy wynikach. Związane są one z inicjalizacją algorytmów oraz ręcznego wyłączenia aplikacji przez co nie przenoszą w pełni wartościowych informacji.

### 3.3.1. Skuteczność detekcji

Sprawdzenie skuteczności algorytmów polega na zebraniu informacji na ilu klatkach z obrazu na żywo udało się wykryć twarz.

**Tabela 3.7.** Skuteczność algorytmów detekcji twarzy dla obrazu na żywo z kamery

Warunki	1.	2.	3.	4.
<b>DNN rgb</b>	200	200	200	200
<b>HOG rgb</b>	200	200	200	200
<b>HOG sk. szaro.</b>	200	200	200	200

Jak widać oba algorytmy wykrywały w każdej odebranej klatce twarz. Potwierdzają to zarówno logi jak i podgląd na żywo podczas testu.

### 3.3.2. Szybkość detekcji

Szybkość detekcji będzie porównana za pomocą średniej ilości klatek na sekundę. Jako, że w statystykach nie jest uwzględniany tylko czas detekcji, a działanie całej aplikacji to występują tu pewne narzuty czasowe związane z wyświetlaniem obrazu i rysowaniem na nich wykrytego obszaru celem podglądu testowanych parametrów na żywo. Jednak opóźnienie to występuje w każdym algorytmie i można uznać je za takie same.

**Tabela 3.8.** Szybkość algorytmów detekcji twarzy dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
<b>DNN rgb</b>	14,004	14,298	14,175	14,270	14,186
<b>HOG rgb</b>	16,488	16,224	16,337	16,400	16,362
<b>HOG sk. szaro.</b>	19,405	19,546	19,367	19,796	19,528

Podobnie jak w testach na statystycznych zdjęciach algorytm *HOG* okazał się szybszy od *DNN Caffe*. W przypadku histogramu gradientów w oparciu o zdjęcie w skali szarości była to prędkość większa aż o 37%.

## 3.4. Wybór algorytmu

W dalszej części projektu zdecydowałem się na korzystanie z algorytmu *dlib HOG* do detekcji twarzy. W testach skuteczności okazał się on prawie tak samo skuteczny jak *DNN Caffe*, ale zdecydowanie od niego szybszy.

Dodatkowo wybór padł na dostarczanie do detektora obrazu w skali szarości ze względu na zysk w ilości klatek na sekundę - dla barw mono był szybszy o 19% od wersji trójkanałowej.

## 4. Facemark

*Facemarks* (lub *Face landmarks*) to punkty nakładane na twarz wokół interesujących obszarów - takich jak oczy, nos czy usta. Pozwalają określić położenie, rozmiar czy kształt tych obiektów. Mogą być również użyte do predykcji czy mamy zamknięte/otwarte oczy (patrz rozdz. 6.EAR) lub czy się uśmiechamy.



Rysunek 4.1. Przykład zdjęcia twarz z naniesionymi facemarkami

### 4.1. Local binary features

Metoda oparta o histogram LBF [26]. W projekcie została użyta implementacja tego algorytmu z modułu *face* [27], który zawarty jest w dodatkach do biblioteki OpenCV zbiorczo nazwanych *contirb* [28]. Do jej działania korzystałem z gotowego modelu [29], który był trenowany na datasetie *HELEN* [30].

### 4.2. Kazemi

Kolejnym algorytmem służącym do estymacji facemarków jest Kazemi [31], który wykorzystuje drzewa regresyjne. Jest on zaimplementowany w bibliotece *dlib*. Gotowy model był trenowany na podstawie datasetu *iBUG 300-W face landmark* [32].

## 5. Porównanie algorytmów detekcji facemarków

Podobnie jak w przypadku detekcji twarzy przeprowadziłem testy algorytmów wykrywania facemarków, przedstawionych w rozdziale 4.Facemark. Ze względu na specyfikę nanoszenia punktów charakterystycznych oraz ich ilość, trudno jest określić dokładność działania korzystając z matematycznych i liczbowych form wyrazu. Z tego powodu jakość obu algorytmów to moja subiektywna ocena na podstawie obserwacji facemarków obszaru oczu i ust. Dużą uwagę podczas opiniowania poświęcałem dokładnemu odwzorowaniu punktów w przypadku przyjmniętych lub całkiem zamkniętych oczu, ponieważ ma to istotny wpływ na inne aspekty pracy dyplomowej. Natomiast złożoność czasowa jest już mierzalna i została wyrażona liczbowo.

### 5.1. Testowanie na statycznych zdjęciach

Oba algorytmy były przetestowane na statycznych zdjęciach z datasetu w zakresie skuteczności i szybkości działania.

#### 5.1.1. Usunięcie części zdjęć z datasetu

Ze względu na wybór algorytmu *HOG* do detekcji twarzy zmuszony byłem odrzucić 2 z 80 przygotowanych zdjęć, ponieważ metodzie tej nie udało się wykryć na nich twarzy (patrz rozdz. 3.2.3. *Badanie skuteczności detekcji twarzy*).

#### 5.1.2. Badanie skuteczności detekcji

Podczas testu zostały zebrane następujące dane:

- **Prawidłowe detekcje** - pokrycie twarzy facemarkami, które uznałem za dobre
- **Złe detekcje** - pozostałe, które nie uznałem za dobre
- **Detekcje lepsze niż drugiego algorytmu** - który z dwóch algorytmów poradził sobie lepiej w danym przypadku testowym.

Zebrane dane są całkowicie subiektywnym odczuciem i inne osoby mogłyby mieć odmienne wyniki.

Oba algorytmy dawały taki sam rezultat zarówno w skali szarości jak i w trzy kanałowym zestawie barw, dlatego tabela wynikowa została uproszczona przez usunięcie takiego podziału.

**Tabela 5.1.** Skuteczność algorytmów detekcji landmarków

	Prawidłowe detekcje	Złe detekcje	Detekcje lepsze niż drugiego algorytmu
LBF	35	42	16
Kazemi	66	12	62

## 5. Porównanie algorytmów detekcji facemarków

---

Zebrane dane pokazują jasno, że model oparty na metodzie *Kazemi* dał zdecydowanie lepsze wyniki niż drugi badany algorytm. W 62 przypadkach testowych pokrycie twarzy facemarkami było subiektywnie dokładniejsze niż w metodzie *LBF*. Tylko 12 z 78 detekcji uznałem za błędne. Można przyjąć, że jest to wynik co najmniej poprawny. Natomiast algorytm zaimplementowany w bibliotece OpenCV mylił się w ponad połowie przypadków.

### 5.1.3. Badanie szybkości detekcji

W tym teście zostały zebrane i porównane następujące dane:

- **Całkowity czas przetwarzania** - suma czasów detekcji facemarków dla wszystkich 20 iteracji
- **Średni czas przetwarzania pojedynczej iteracji** - uśredniony czas detekcji facemarków dla pojedynczej iteracji
- **Średni czas przetwarzania jednego zdjęcia** - uśredniony czas detekcji facemarków dla pojedynczego zdjęcia

**Tabela 5.2.** Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
<b>Kazemi RGB</b>	5,717 s	0,285 s	0,00366 s
<b>LBF RGB</b>	6,545 s	0,327 s	0,00419 s
<b>Kazemi sk. szaro.</b>	5,472 s	0,273 s	0,00351 s
<b>LBF sk. szaro.</b>	6,084 s	0,304 s	0,00391 s

Algorytm Kazemi z użyciem biblioteki dlib i języka C++ okazał się szybszy o ponad 10% od odpowiednika w postaci LBF. Obie metody uzyskały lepszy czas w teście opartym na obrazach w skali szarości niż w RGB o kilka procent. Istotnym faktem jest, że te algorytmy potrzebują mało czasu na przetworzenie pojedynczego zdjęcia, dzięki czemu w warunkach czasu rzeczywistego nie spowodują znacznego spadku klatek na sekundę.

## 5.2. Testowanie na obrazie z kamery na żywo

Kolejnym etapem testowania detekcji facemarków będzie wykorzystanie obrazu z kamery na żywo. Warunki przeprowadzenia eksperymentu zostały opisane w rozdz. 3.3.

### 5.2.1. Skuteczność detekcji

Ze względu na opisaną wyżej trudność matematycznego wyrażenia skuteczności nakładania punktów charakterystycznych, wyniki porównania zostały przedstawione w formie opisowej i są moim subiektywnym odczuciem obserwacji działania algorytmu na żywo.

Algorytm Kazemi bez zarzutu poradził sobie w trzech scenariuszach. Natomiast w jednym - przy mocnym oświetleniu padającym na obiektyw i na twarz - występowało wtedy chwilowe niedokładne dopasowanie. Poza tym problemem radził sobie on bardzo dobrze. Ruchy twarzy nie przeszkały w prawidłowym ułożeniu facemarków. Punkty były bardzo stabilne, a podczas sztywnego położenia twarzy nie występowały ich drgania.

Gorzej poradził sobie algorytm LBF. Podobnie jak Kazemi miał pewne problemy podczas scenariusza opartego na mocnym oświetleniu. Występowało ciągłe drwanie punktów, nawet podczas sztywnego położenia twarzy. Metoda ta oznaczała twarz jako szerszą niż w rzeczywistości była. Podczas ruchów twarzy algorytm gubił prawidłowe położenie punktów.

### 5.2.2. Szybkość detekcji

Test był przeprowadzony używając detekcji twarzy *HOG*, do którego dostarczano obraz w przestrzeni barw RGB.

**Tabela 5.3.** Szybkość algorytmów detekcji facemarków dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
<b>LBF RGB</b>	16,076	15,960	15,820	16,079	15,983
<b>LBF sk. szaro.</b>	15,959	16,016	15,829	15,793	15,899
<b>Kazemi RGB</b>	15,887	15,941	16,077	16,171	16,019
<b>Kazemi sk. szaro.</b>	15,747	16,146	15,866	16,096	15,963

Mniejsza ilość klatek w przypadku testów w skali szarości prawdopodobnie jest związana z dodatkowym narzutem czasowym w postaci konwersji obrazu z trójkanałowej barwy na jednokanałową.

Oba algorytmy uzyskały bardzo zbliżone wyniki. Niewiele szybsza okazała się jednak metoda Kazemi. Rezultat jest porównywalny z testem przeprowadzonym na statycznych zdjęciach.

### 5.3. Wybór algorytmu

*Kazemi* okazał się przede wszystkim dużo skuteczniejszym i stabilnym algorytmem niż facemarki oparte na *Local Binary Features*. Dodatkowo jest około 10% szybszy. Wszystkie testy wskazują na wyższość *Kazemi* i z tych powodów jest on używany w dalszej części pracy dyplomowej i projektu.

## 6. EAR - Eye Aspect Ratio

Metoda polegająca na obliczeniu *EAR* [33] [34], czyli stosunek otwarcia oczu - wysokość do szerokości widocznej części gałki ocznej. Wykorzystuje się tu facemarki (patrz rozdz. 4.*Facemarks*) naniesione dookoła oczu.

### 6.1. Wzór obliczania EAR

Zależnie od ilości punktów wokół oka będzie różny wzór obliczania EAR.

Dla 6 punktów:

$$EAR = \frac{dist(L_0, L_1) + dist(L_2, L_4)}{2 * dist(L_3, L_5)} \quad (1)$$

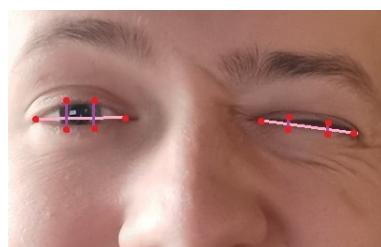
Natomiast, dla 4 punktów:

$$EAR = \frac{dist(L_0, L_2)}{dist(L_1, L_3)} \quad (2)$$

Gdzie  $L_x$  to kolejne landmarki dokoła oczu, a *dist* to odległość między dwoma punktami (odległość euklidesowa).

### 6.2. Zasada działania EAR w kontekście mrugania i określenia czy oko jest otwarłe/zamknięte

W teorii otwarte oczy będą miały większy wymiar liczbowy EAR, niż oczy zamknięte. Na rysunku 6.1 widać, że oko otwarte ma większe odległości między punktami pionowymi niż w przypadku oka zamkniętego. Dzięki takim różnicom możemy wykryć spadek wskaźnika EAR poniżej pewnego ustalonego poziomu, oznaczający zamknięcie oka, natomiast wzrost, otwarcie oka. Całościowo obserwując zmiany np. za pomocą pochodnej jesteśmy w stanie stwierdzić mrugnięcie.



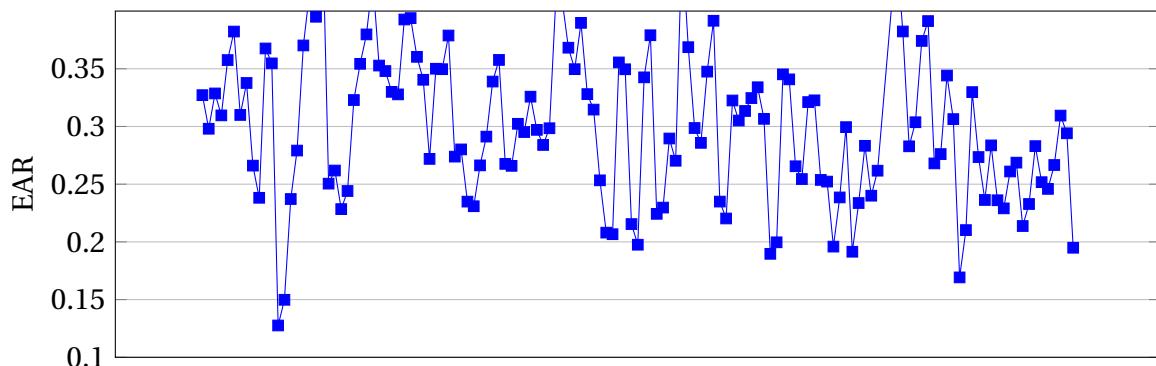
Rysunek 6.1. Teoretyczny rozmieszczenie landmarków wokół oczu wraz z naniesionymi połączciami do obliczenia EAR

### 6.3. Wyznaczenie progu EAR

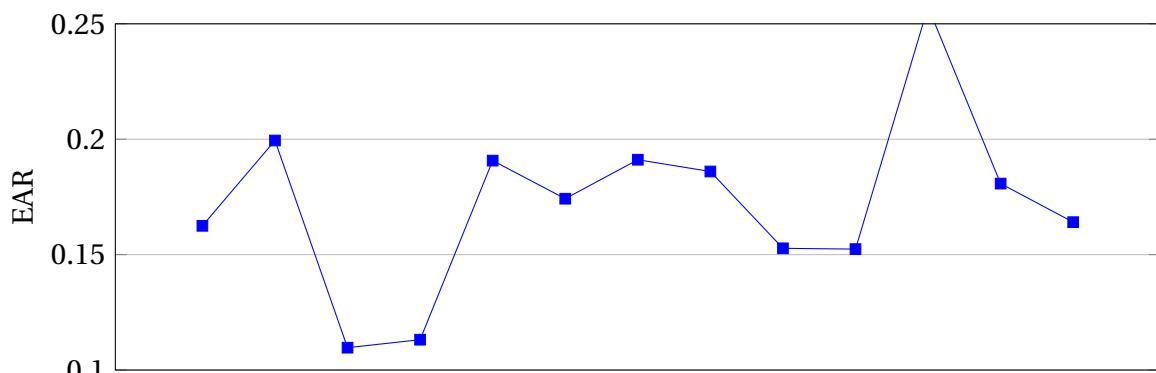
Korzystając z tej metody potrzebujemy określić pewien próg wskaźnika EAR, poniżej którego będziemy klasyfikować oczy jako zamknięte lub mrugające. W tym celu zebrałem dane dotyczące wyliczonego stosunku zarówno na statycznych zdjęciach, jak i obrazie na żywo.

#### 6.3.1. Statyczne zdjęcia

Na podstawie używanego datasetu wyliczyłem EAR dla wszystkich oczu i podzieliłem je na dwie grupy: oczy otwarte (rys. 6.2) i oczy zamknięte (rys. 6.3).



Rysunek 6.2. EAR dla oczu otwartych



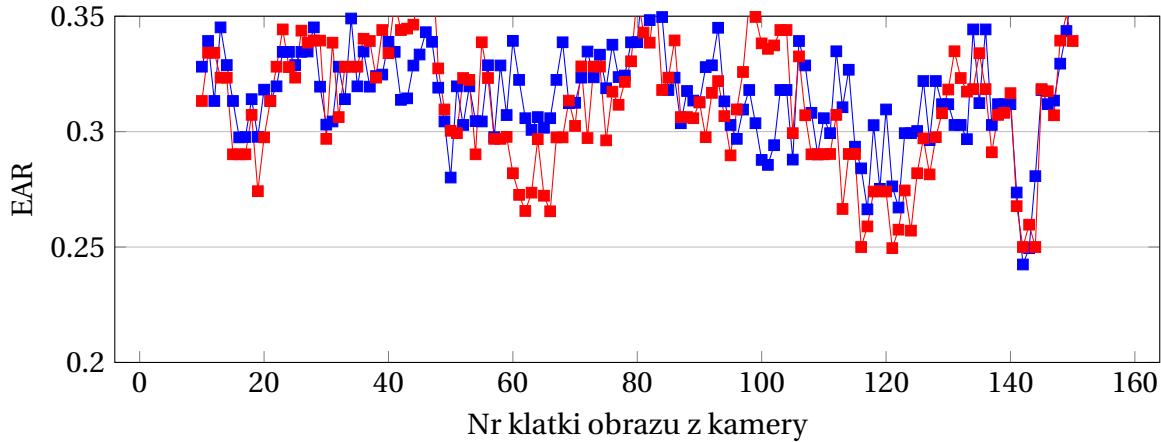
Rysunek 6.3. EAR dla oczu zamkniętych

W przypadku oczu otwartych widać, że poza kilkoma wyjątkami wskaźnik EAR znajduje się powyżej 0.20, a czasem przekraczając nawet 0.35. Przypadek gdzie wartość jest poniżej 0.15 wynika najprawdopodobniej z faktu, że oczy były mocno przymknięte i ze względu na perspektywę rozciągnięte horyzontalnie. Odchylenie takie traktuję jako przypadek skrajny i nie wpływający na wyniki.

Na wykresie dla oczu zamkniętych ponownie próg zdaje się być na poziomie 0.20, a większość wskaźników osiąga wartość około 0.19.

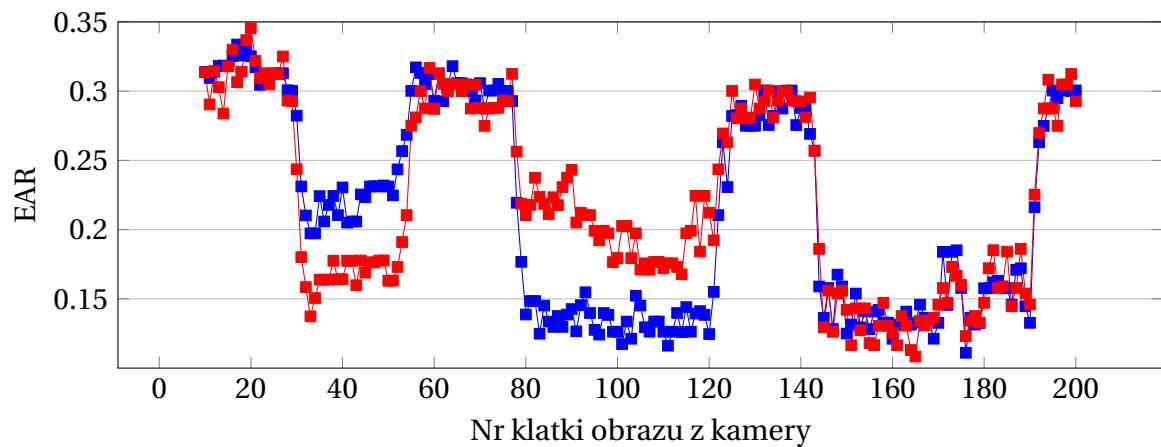
### 6.3.2. Obraz z kamery

Na obrazie z kamery urządzenia przeprowadziłem trzy testy wskaźnika EAR: oczy otwarte (rys. 6.4), oczy chwilowo zamknięte (rys. 6.5) oraz mrugnięcie (rys. 6.6). na wykresach czerwony kolor oznacza EAR dla prawego oka, natomiast niebieski dla lewego.



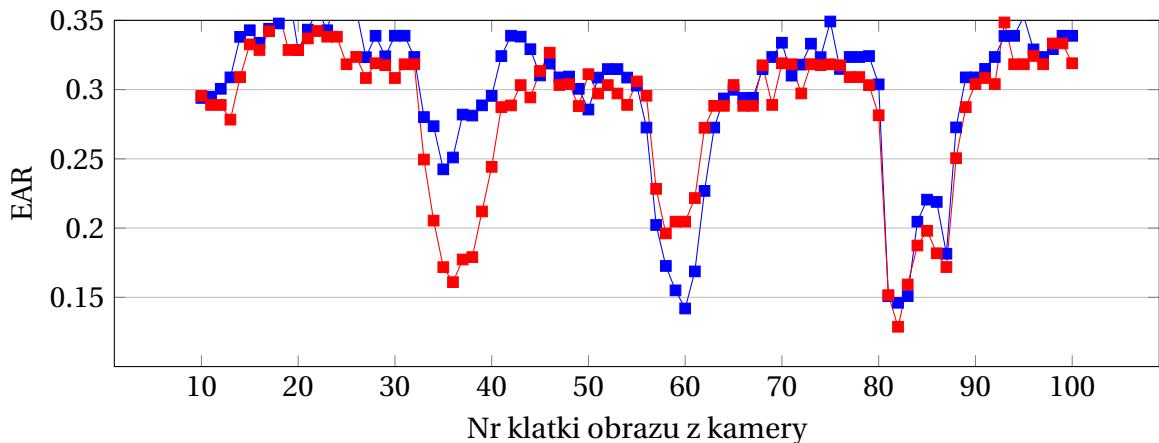
Rysunek 6.4. Oczy otwarte na obrazie z kamery

W tym przypadku oczy były przez cały czas otwarte. Liczbowy wymiar EAR nie spadał poniżej 0.25, a przez większość czasu trzymał się nawet powyżej 0.30.



Rysunek 6.5. Oczy czasowo zamknięte na obrazie z kamery

W tym teście oczy były czasowo zamknięte. Najpierw w przedziale klatek [25, 60] zamknięte było tylko oko prawe, następnie w przedziale [80, 120] tylko lewe, a na końcu w okresie [140, 190] oba oczy jednocześnie. Zmiana EAR jest tutaj mocno wyraźna, a w tych trzech przedziałach miał wartość między 0.10, a 0.20. Dodatkowo na podstawie tych danych trzeba również odnotować spadek EAR dla obu oczu nawet w przypadku zamknięcia tylko jednego. Może to wynikać z faktu, że człowiek zamykając jedno przykrywa lekko również drugie. Ale przyczyną może być też sam algorytm facemarków, w którym punkty dla różnych oczu mogą być ze sobą skorelowane.



**Rysunek 6.6.** Mruganie na obrazie z kamery

W trzeciej wariacji testu na obrazie z kamery występuowały mrugnięcia: w przedziale [30, 40] prawego oka, [55, 65] lewego, a w przedziale [80, 90] oboma oczami na raz. Szybka zmiana wymiaru EAR jest w tych chwilach bardzo wyraźna i łatwa do wykrycia. W szczytowych chwilach mrugnięcia wartość spadała prawie do poziomu 0.15. Ponownie występuje to zmniejszenie EAR dla obu oczu nawet w przypadku mrugnięcia wyłącznie jednym.

#### 6.4. Wnioski

Zmiany wartości EAR na obrazie z kamery są bardzo wyraźne i łatwe do wykrycia. Dzięki temu metoda ta nadaje się do detekcji mrugania i tego czy oczy są zamknięte. Jednak problem ze zmianą EAR dla obu oczu nawet w przypadku zamknięcia tylko jednego sprawia, że będzie odnotowany fakt mrugnięcia bez podziału na to którym okiem została wykonana czynność.

Na podstawie testów progowy poziom EAR dla oka zamkniętego ustaliłem na  $\leq 0.19$ . Powyżej tej wartości traktuję oko jako otwarte.

## 7. Detekcja oczu

W tej pracy dyplomowej dużą rolę odgrywają oczy i ich wpływ na sterowanie aplikacją. Z tego powodu musiała zostać określona i zaimplementowana skuteczna metoda ich detekcji.

### 7.1. Algorytmy detekcji oczu

Na potrzeby tego etapu zostały zastosowane dwie metody opierające się na algorytmach opisanych wcześniej - klasyfikatory kaskadowe oraz facemarki.

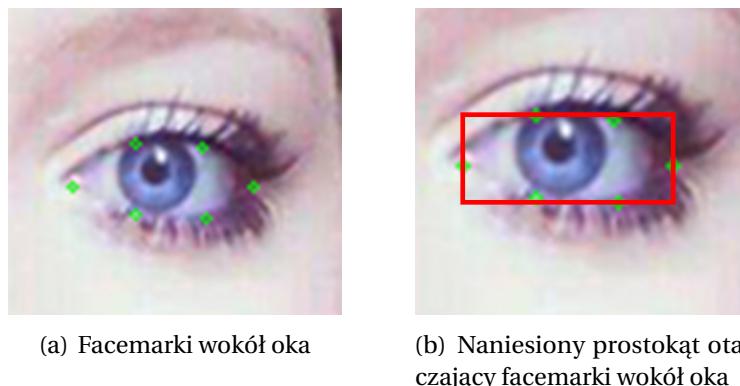
#### 7.1.1. Klasyfikator kaskadowy Haar

Metody detekcji oparte na klasyfikatorach kaskadowych zostały opisane w rozdz. 2.1.1. Do detekcji oczu z użyciem tego algorytmu zostanie wykorzystany model autorstwa Sharameem Hameed [35].

#### 7.1.2. Facemarki

Opisane w rozdz. 4 facemarki nanoszą punkty charakterystyczne na obraz twarzy. Znajdują się one m.in. wokół oka. Fakt ten można wykorzystać do detekcji ich obszaru. Celem uzyskania takiego efektu należy wyznaczyć prostokąt, który będzie otaczał wszystkie sześć facemarków danego oka. [36]

Przykład takiego działania widoczny jest na rysunku 7.1.



Rysunek 7.1. Wykorzystanie facemarków do detekcji obszaru oczu

Wykorzystywany jest tu też wskaźnik *EAR* (patrz rozdz. 6. *EAR - Eye Aspect Ratio*) do wykrycia czy oko jest zamknięte czy otwarte.

Skuteczność tej metody zależy od dokładności algorytmu odwzorowującego facemarki. Ewentualne niedoskonałości można niwelować zwiększając wielkość prostokąta o pewną tolerancję. Taki współczynnik w projekcie został testowo ustalony, a wyniki i wartości parametrów opisane w rozdz. 7.4.

## 7.2. Filtrowanie wyników metody Haar

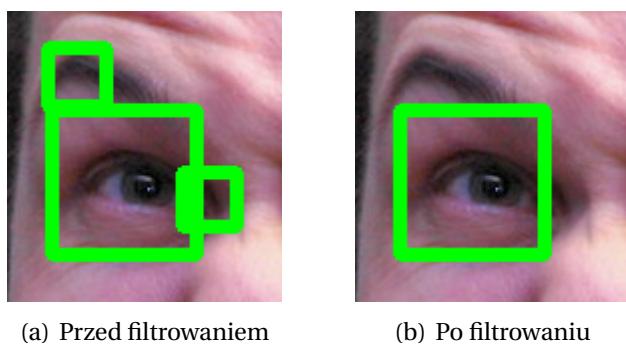
Ze względu, że algorytmu Haar może zwracać większą ilość prawdopodobnych obszarów, w których spodziewa się on wykryć pożądany obiekt, wprowadziłem filtrowanie wyników detekcji oczu.

Algorytm filtrowania składa się z dwóch etapów:

- Podzielenie wykrytych obszarów na dwie grupy - na lewą i prawą stronę twarzy
- W obu grupach wybranie największego obszaru

Dodatkowo pozwoliło to na łatwe zidentyfikowanie który obszar to które oko i ich posortowanie.

Przykładowy rezultat takiego filtrowania pokazany jest na *rysunku 7.2*.



**Rysunek 7.2.** Efekt filtrowania obszarów detekcji oczu

## 7.3. Obcięcie obszaru detekcji dla metody Haar

Dla metody opartej na Haar zdecydowałem się dodatkowo zawęzić płaszczyznę przeszukiwań na niecałą twarz, celem uzyskania wyników lepszych niż bez takiego zmniejszenia.

Ustalenie jaki obszar da najlepszy rezultat odbyło się przez testowe sprawdzanie kombinacji trzech parametrów oznaczających jaka część wykrytej twarzy zostaje obcięta z poszczególnej strony. Mogą one przyjmować wartości w następujących przedziałach:

- Góra: [0.0, 0.02]
- Dół: [0.2, 0.6]
- Boki: [0.0, 0.02]

Każdy parametr mógł osiągać wartości będące wielokrotnością 0.05 w poszczególnych przedziałach.

Ze względu na dużą liczbę kombinacji (225) nie zostaną podane wyniki, a jedynie wybrana najlepsza kombinacja:

- Góra: 0.05
- Dół: 0.3

## 7. Detekcja oczu

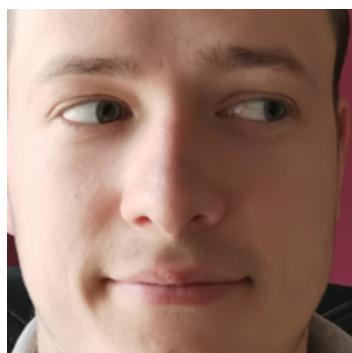
- Boki: 0.0

Algorytm Haar uzyskał lepszą skuteczność detekcji wykorzystując dodatkowe obcięcie niż bez niego:

**Tabela 7.1.** Skuteczność algorytmu detekcji oczu Haar z dodatkowym obcięciem i bez

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
<b>Haar bez obcięcia</b>	124	85	39	34	8	32
<b>Haar z obcięciem</b>	133	95	38	23	9	11

Przykład takiego obcięcia z wybranymi parametrami widoczny jest na *rysunku 7.3*.



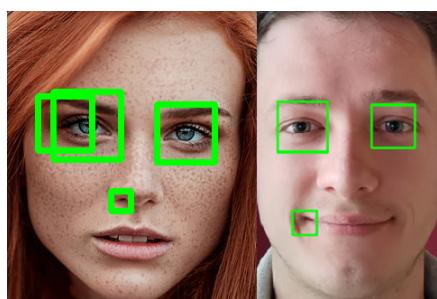
(a) Wykryty obszar twarzy



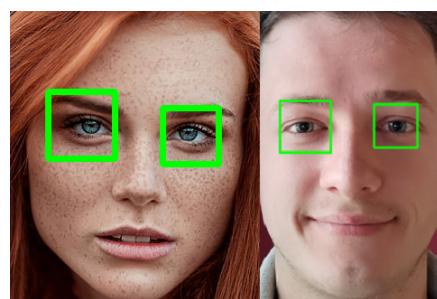
(b) Obszar po obcięciu

**Rysunek 7.3.** Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami

Wprowadzenie takiej modyfikacji pozwoliło odrzucić część błędnie ustalanych obszarów, szczególnie w dolnej części twarzy. Przykład poprawionej detekcji oczu dzięki temu zabiegowi widoczny jest na rysunku 7.4



(a) Wykrywanie oczu bez dodatkowego obcięcia obszaru



(b) Wykrywanie oczu z dodatkowym obcięciem obszaru

**Rysunek 7.4.** Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru.  
Źródło pierwszego zdj.:[37]

#### 7.4. Dostosowanie wielkości obszaru facemarków oczu

Dla metody opartej o punkty charakterystyczne twarzy zdecydowałem się dodać pewną tolerancję do obszaru wynikającego jedynie z połączenia tych punktów.

Ustalenie jakie zwiększenie zwracanego regionu da najlepsze rezultaty detekcji oczu odbyło się przez testowe sprawdzenie kombinacji trzech parametrów, które oznaczały wielkość tolerancji z danej strony. W teści przyjmowały one następujące wartości

- Góra: z przedziału [0.0, 1.0], będące wielokrotnością 0.1
- Dół: z przedziału [0.0, 1.0], będące wielokrotnością 0.1
- Boki: z przedziału [0.0, 0.5], będące wielokrotnością 0.05

Ze względu na dużą ilość kombinacji (1331) nie zostaną podane wyniki, a jedynie wybrana najlepsza kombinacja:

- Góra: 0.7
- Dół: 0.5
- Boki: 0.2

**Tabela 7.2.** Skuteczność algorytmu detekcji oczu wykorzystując facemarki z dodatkowym zwiększeniem obszaru i bez

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
<b>Facemarki oczu bez zwiększenia obszaru</b>	142	17	125	14	6	6
<b>Facemarki oczu ze zwiększeniem obszaru</b>	144	142	2	12	6	6

Zmiany wielkości zwracanego obszaru oczu nie zwiększył znaczco liczby dobrych detekcji. Natomiast największy zysk widoczny jest w perfekcyjnych detekcjach. Dzięki takiemu dostosowaniu udało się osiągnąć prawie 100% wskaźnik perfekcyjnych względem prawidłowych detekcji. Pozwoli to uzyskać lepiej wykryty obszar oczu, co może mieć przełożenie na skuteczność detekcji źrenic.

## 8. Porównanie algorytmów detekcji oczu

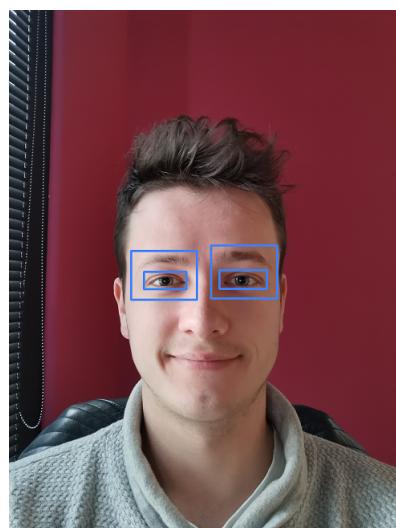
### 8.1. Testowanie na statycznych zdjęciach

Testowanie detekcji oczu wykorzystując statyczne zdjęcia z datasetu (odrzucone 2, dla których wybrany algorytm nie wykrył prawidłowo twarzy). Znajduje się na nich w sumie 166 oczu do wykrycia, w tym 14 zamkniętych.

#### 8.1.1. Oczekiwany wynik

Podobnie jak w przypadku testowania detekcji twarzy na statycznych zdjęciach, tak w przypadku wykrywania oczu akceptowalny obszar był opisany dwoma prostokątami - minimalny i maksymalny. Przykład takiego oznaczenia widoczny jest na rys. 8.1. Został on dobrany w następujący sposób:

- prostokąt wewnętrzny obejmuje jedynie widoczną część gałki ocznej
- prostokąt zewnętrzny jest powiększony na boki i w dół o pewną odległość od gałki, a od góry zawiera w sobie również brwi.



Rysunek 8.1. Przybliżony obszar oczu, który chcę wykrywać

### 8.2. Warunki testowania

Oba algorytmy zostaną przetestowane na wykrytych przez algorytm HOG twarzach ze zdjęć 500x500. W tym teście postanowiłem odrzucić już rozdzielcość 300x300, ze względu, że na części z nich osoba jest już bardzo oddalona, co skutkuje bardzo małym rozmiarem oczu. W przypadku korzystania z przedniej kamery telefonu nie będą występowały takie odległości między twarzą, a urządzeniem. Z tego powodu takie warunki nie przyniosłyby znaczących i wartościowych w kontekście pracy dyplomowej wyników.

Testy zostaną przeprowadzone w przestrzeni barw RGB oraz w skali szarości.

### 8.3. Badanie skuteczności detekcji

Na początku porównywana była skuteczność zaprezentowanych metod. Chociaż w pewien sposób procent detekcji był już przedstawiony podczas dostrajania algorytmów, to tutaj zostały skonfrontowane ze sobą oba algorytmy.

W przypadku metody opartej o facemarki, niewykryte oczy zarówno otwarte jak i zamknięte mogą oznaczać źle określenie położenia lub złą klasyfikację do jednej z tych grup. Przykładowo oko otwarte mające *EAR* mniejszy niż ustalony próg klasyfikowane było jako zamknięte, co skutkowało oznaczeniem jako błędna detekcja.

**Tabela 8.1.** Skuteczność algorytmów detekcji twarzy

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
Facemarki oczu RGB	144	142	2	12	6	6
Facemarki oczu sk. szaro.	145	140	5	11	1	6
Haar RGB	133	95	38	23	9	11
Haar sk. szaro.	130	95	35	26	11	7

Najlepsze wyniki uzyskała metoda oparta na punktach charakterystycznych twarzy w trójkanałowej przestrzeni barw. Udało się jej wykryć średnio 87% wszystkich oczu. A w przypadku przestrzeni RGB aż 98,6% z nich perfekcyjnie. Ta druga statystyka jest dużo lepsza niż w przypadku metod Haar które uzyskały raptem 57%. Ogólnie algorytm wykorzystujący facemarki był lepszy od drugiego o około 10%.

Co ciekawe większą liczbę dobrych detekcji metoda oparta o facemarki uzyskała w przypadku obrazów w skali szarości, ale mniej perfekcyjnych. Natomiast, algorytm Haar lepiej poradził sobie w przypadku trójkanałowego zestawu barw, ale wykrywał ona więcej zamkniętych oczu przez co w tej statystyce wypadł gorzej niż w skali szarości.

Waźną różnicą w działaniu obu algorytmów jest kształt i rozmiar oznaczanego obszaru. Metoda oparta o klasyfikatory zwraca kwadratowy region, zawierający dużo większą część twarzy niż same oczy. Natomiast facemarki tworzą obszar o kształcie prostokąta o dowolnym stosunku boków i zawierają głównie samą gałkę oczną. Zmniejsza to ilość punktów, które mogą przeszkadzać np. w detekcji źrenic.

#### 8.3.1. Badanie szybkości detekcji

Dla detekcji oczu z użyciem facemarków zostały przedstawione dwa wyniki czasowe - jeden uwzględniający czas wykrycia punktów charakterystycznych, a drugi bez. Związane jest to z faktem, że mimo iż do wykorzystania tej metody niezbędna jest detekcja facemarków, to etap taki i tak będzie wykonany, ponieważ punkty te są używane np. do stwierdzenia mrugnięcia. W przypadku czasów zawierających algorytm punktów charakterystycznych test był przeprowadzony dla dwóch przestrzeni barw. Natomiast, na

## 8. Porównanie algorytmów detekcji oczu

---

szbkość przekształcenia facemarków w obszar oka nie wpływa liczba kanałów, ponieważ nie operuje on na pikselach, tylko na zwróconych punktach kartezjańskich. Z tego powodu przedstawiony jest tylko jeden wynik.

**Tabela 8.2.** Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
<b>Facemarki oczu RGB (z detekcją facemarków)</b>	5,638 s	0,281 s	0,00361 s
<b>Facemarki oczu sk. szaro. (z detekcją facemarków)</b>	5,803 s	0,290 s	0,00372 s
<b>Facemarki oczu (bez detekcją facemarków)</b>	0,024 s	0,00122 s	0,00000157 s
<b>Haar RGB</b>	26,219 s	1,310 s	0,0168 s
<b>Haar sk. szaro.</b>	25,830 s	1,291 s	0,0165 s

Wyniki bezdyskusyjnie pokazują dużo większą szybkość detekcji oczu z użyciem facemarków. Nawet biorąc pod uwagę czas wykrycia punktów charakterystycznych metoda ta była ponad 4 razy szybsza od algorytmu opartego na klasyfikatorach kaskadowych Haar.

Tworzenia obszarów z punktów charakterystycznych osiągnęło czas rzędu  $10^{-5}$ , co w porównaniu do pozostałych algorytmów wykorzystywanych w projekcie jest praktycznie zerowy. Tak krótki czas przetwarzania sprawia, że nie będzie on miał żadnego wpływu na ilość klatek na sekundę podczas odbierania obrazu na żywo z kamery.

Przestrzeń barw nie miała znaczącego wpływu na czas detekcji, a wyniki w przypadku trójkanałowych kolorów były porównywalne do skali szarości.

### 8.4. Testowanie na obrazie z kamery na żywo

Detekcja oczu została przetestowana również na obrazie z kamery na żywo. Warunki przeprowadzenia eksperymentu zostały opisane w rozdz. 3.3.

#### 8.4.1. Skuteczność detekcji

Obserwując na żywo detekcje oczu byłem w stanie wysnuć kilka wniosków na temat badanych algorytmów.

Przede wszystkim oba działały bardzo stabilnie. Nie występowało nieuzasadnione gubienie obszaru oczu podczas statycznego położenia głowy.

W metodzie Haar występował problem przy mocnym skręceniu głowy w bok. Zwracany był wtedy często błędny i bardzo powiększony obszar dalszego oka.

Natomiast detekcja oczu na podstawie facemarków miała problemu z mocno przymkniętymi oczami. Wskaźnik EAR sygnalizował wtedy je jako zamknięte przez co algorytm nie zwracał ich regionu.

#### 8.4.2. Szybkość detekcji

Ze względu na praktycznie zerowy obciążenie podczas przekształcania facemarków w obszar twarzy (patrz rozdz. 8.3.1) wyniki dla facemarków zostały skopiowane z rezultatów uzyskanych przez algorytm Kazemi podczas porównania metod nakładania punktów charakterystycznych na obraz z kamery na żywo w rozdz. 5.2.2.

Ponownie detekcja twarzy odbywała się przy zastosowaniu algorytmu *HOG* dostarczając obraz RGB.

**Tabela 8.3.** Szybkość algorytmów detekcji oczu dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
<b>Haar RGB</b>	11,698	13,600	12,290	12,942	12,632
<b>Haar sk. szaro.</b>	11,956	12,403	12,333	12,472	12,290
<b>Facemarki RGB</b>	15,887	15,941	16,077	16,171	16,019
<b>Facemarki sk. szaro.</b>	15,747	16,146	15,866	16,096	15,963

Test ten pokazał znaczną przewagę szybkościową metody opartej na facemarkach. Klasyfikatory kaskadowe uzyskały około 23% mniejszą liczbę klatek na sekundę.

#### 8.5. Wybór algorytmu

Porównanie dwóch algorytmów detekcji oczu bezdyskusyjnie pokazało wyższość metody opartej na facemarkach nad klasyfikatorem kaskadowym Haar. Punkty charakterystyczne uzyskały lepszy procentowo wynik prawidłowych detekcji, a dodatkowo w teście na statycznych zdjęciach prawie wszystkie były perfekcyjne. Próby szybkościowe jednoznacznie wskazały na facemarki, które były 4 razy szybsze od Haar, a jeśli brać pod uwagę jedynie czas tworzenia obszarów to ponad tysiąckrotnie.

Na podstawie przeprowadzonych testów zostały wybrane facemarki do detekcji oczu i to ten algorytm będzie wykorzystywany w projekcie i pracy dyplomowej.

## 9. Detekcja źrenic

W pracy dyplomowej występuje sterowanie aplikacją za pomocą ruch gałek ocznych, w szczególności analizując położenie źrenic. Mając wykryty obszar oczu (patrz rozdz. 7.*Detekcja oczu*) metodami klasycznymi przetwarzania obrazów określając położenie środka źrenicy.

W projekcie zaimplementowane zostały trzy algorytmy do testów i wybrany jeden, który dawał najlepsze rezultaty.

### 9.1. Algorytm CDF (Cumulative Distribution Function)

Algorytm zaimplementowany na podstawie dwóch artykułów o detekcji źrenic [38] [39]. Opiera się w głównej mierze na progowaniu za pomocą dystrybuanty. Następnie analizuje najciemniejsze obszary, w których wyznacza środek źrenicy.

#### 9.1.1. Kroki algorytmu

- Za pomocą progowania z użyciem dystrybuanty tworzymy obraz binarny.

$$CDF(r) = \sum_{w=0}^r p(w) \quad (3)$$

Gdzie  $p(w)$  to prawdopodobieństwo znalezienia punktu o jasności równej  $w$  - określone przy pomocy dystrybuanty.

$$I'(x, y) = \begin{cases} 255, & CDF(I(x, y)) < a \\ 0, & wpp \end{cases} \quad (4)$$

Gdzie  $I$  to jasność piksela, natomiast  $a$  to ustalony próg

- Na uzyskany obraz binarny nakładamy operację morfologiczną erozji (filtr minimałny), celem usunięcia pojedynczych ciemnych pikseli
- Znajdujemy najciemniejszy piksel na oryginalnym obrazie wśród tych, które mają wartość 255 (są białe) na obrazie binarnym
- Obliczamy średnią jasność pikseli w kwadracie 10x10 wokół wybranego najciemniejszego punktu
- Nakładamy erozję na obszarze 15x15 wokół wybranego punktu
- Na tym obszarze stosujemy progowanie

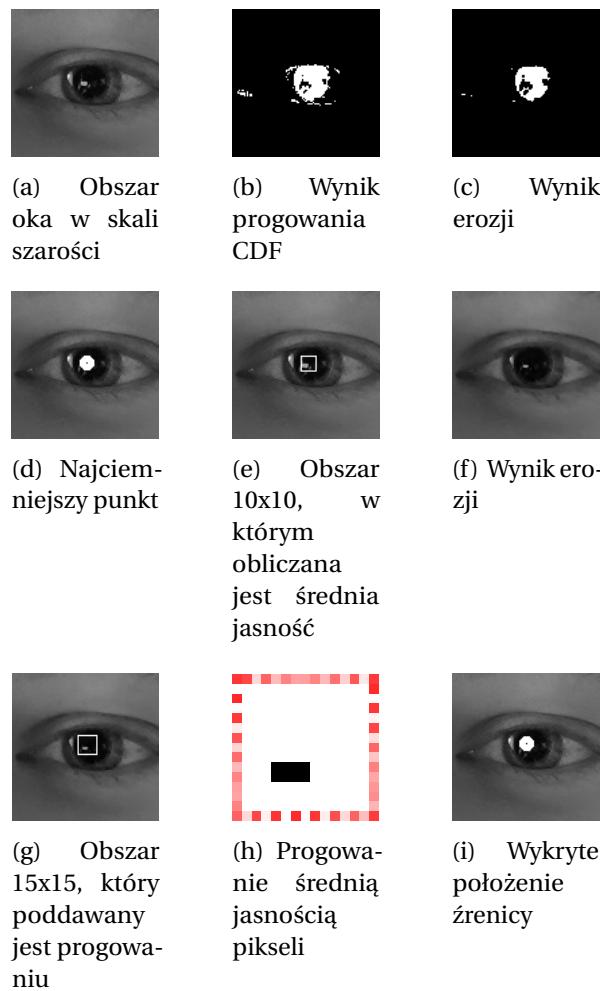
$$I'(x, y) = \begin{cases} 255, & I(x, y) < I_{AVG} \\ 0, & w.p.p. \end{cases} \quad (5)$$

Gdzie  $I_{AVG}$  to średnia jasność obszaru obliczona wcześniej

- Środkiem źrenicy będzie środek ciężkości białych punktów na binarnym obszarze, który uzyskaliśmy

### 9.1.2. Wynik kolejnych etapów algorytmu

Na rys. 9.1 przedstawiony jest wynik kolejnych etapów detekcji źrenic za pomocą metody CDF na przykładowym zdjęciu oka.



**Rysunek 9.1.** Kolejne etapy wykrywania źrenic metodą CDF

## 9.2. Algorytm PF (Projection Function)

Algorytm po raz pierwszy opisany w artykule zatytułowanym *Projection Functions for Eye Detection* [40] z 2004 roku. Jest oparty na rzutowaniu jasności pikseli na składowe poziome i pionowe. Do obliczenia tych wartości wykorzystuje się funkcje projekcji, która może przyjmować różne postaci - z czego trzy zostały opisane w tej pracy dyplomowej. Największe różnice wartości oznaczają szybkie zmiany jasności, które mogą być konturami oka. [39]

### 9.2.1. Funkcja projekcji

Funkcja projekcji służąca do rzutowania jasności rzędów i kolumn może przyjmować różne formy. Najczęściej stosowana jest funkcja całkowa, jednak ze względu na swoje niedociągnięcia i kłopoty z wykrywaniem wariantacji powstały także inne algorytmy. Trzy różne funkcje opisane są poniżej.

**9.2.1.1. Funkcja całkowa** Wylicza się za jej pomocą średnią jasność pikseli w danym rzędzie lub kolumnie. W teorii jest to całka:

$$IPF_h(y) = \frac{1}{x_b - x_a} \int_{x_a}^{x_b} I(x, y) dx \quad (6)$$

$$IPF_v(x) = \frac{1}{y_b - y_a} \int_{y_a}^{y_b} I(x, y) dy \quad (7)$$

Ale w praktyce degeneruje się do funkcji sumy:

$$IPF_h(y) = \frac{1}{x_b - x_a} \sum_{x=x_a}^{x_b} I(x, y) \quad (8)$$

$$IPF_v(x) = \frac{1}{y_b - y_a} \sum_{y=y_a}^{y_b} I(x, y) \quad (9)$$

**9.2.1.2. Funkcja wariantacji** Średnia różnica między jasnością danego piksela, a wyliczonego  $IPF$  danego rzędu lub kolumny

$$VPF_h(y) = \frac{1}{x_b - x_a} \sum_{x=x_a}^{x_b} (I(x, y) - IPF_h) \quad (10)$$

$$VPF_v(x) = \frac{1}{y_b - y_a} \sum_{y=y_a}^{y_b} (I(x, y) - IPF_v) \quad (11)$$

Chociaż we wskazanym wyżej artykule funkcja ta występuje w przedstawionej formie to spotykane są również postacie z różnicą kwadratową oraz z pierwiastkami z obliczonej wartości.

**9.2.1.3. Funkcja ogólna** Parametryzowana suma wyliczonych wartości  $IPF$  i  $VPF$ .

$$GPF_h(y) = (1 - \alpha) * IPF_h + \alpha * VPF_h \quad (12)$$

$$GPF_v(x) = (1 - \alpha) * IPF_v + \alpha * VPF_v \quad (13)$$

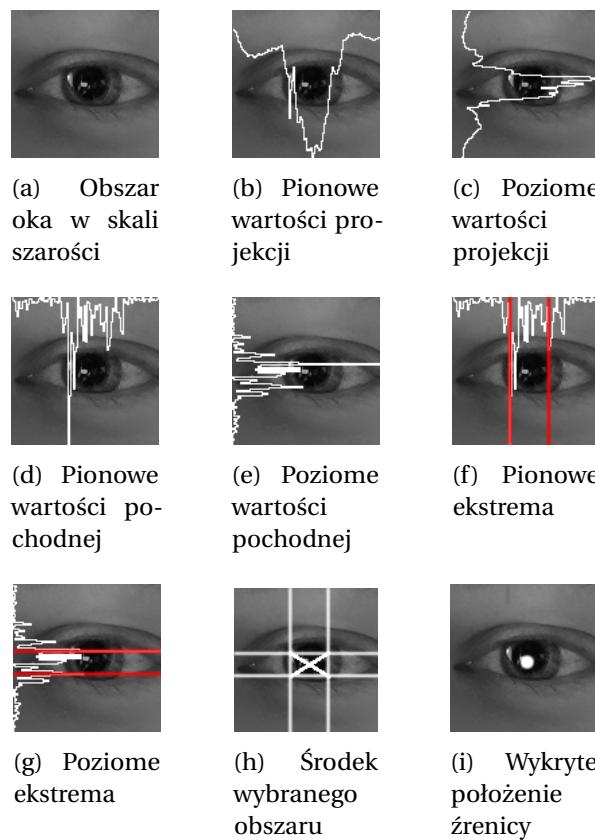
Autorzy tej metody zalecają parametr  $\alpha = 0.6$  jako dający najlepsze rezultaty.

### 9.2.2. Kroki algorytmu

- Dla każdego rzędu i kolumny obliczamy wartość funkcji projekcji
- Dla wyliczonych funkcji projekcji obliczamy wartości pochodnej w każdym rzędzie i kolumnie
- Wybieramy po dwa ekstrema dla poziomego i pionowego wymiaru
- Przecięcie wybranych kolumn i rzędów utworzy prostokąt, którego środkiem będzie poszukiwany punkt na źrenicy

### 9.2.3. Wynik kolejnych etapów algorytmu

Na rys. 9.2 przedstawiony jest wynik kolejnych etapów detekcji źrenic za pomocą metody PF na przykładowym zdjęciu oka.



**Rysunek 9.2.** Kolejne etapy wykrywania źrenic metodą PF

### 9.3. Algorytm EA (Edge Analysis)

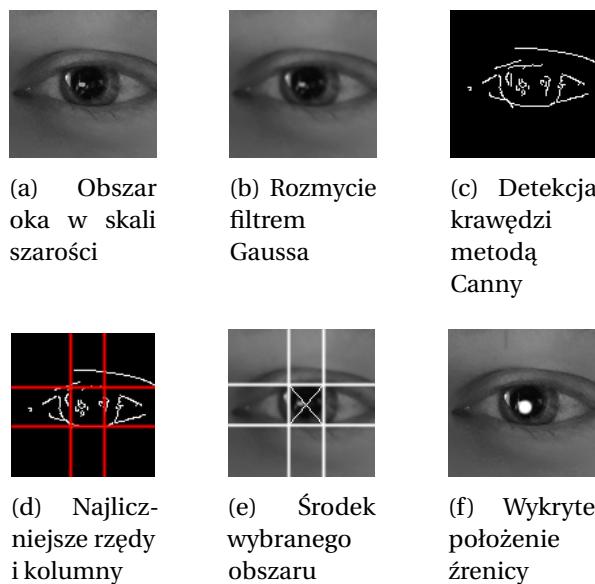
Algorytm detekcji źrenic [39] opierający się na wykryciu i analizie krawędzi. W teorii krawędzie najbardziej pionowe i poziome na zdjęciu powinny należeć do tęczówki i źrenicy.

#### 9.3.1. Kroki algorytmu

- Dla obszaru oka w skali szarości nakładamy filtr rozmywający, np. Gaussa. Pozwoli to pozbyć się drobnych szumów i wygładzić obraz.
- Wykrywamy krawędzie i tworzymy obraz binarny. Można zastosować np. algorytm Canny, który jest jedną z najpopularniejszych metod detekcji krawędzi
- Wybieramy dwa rzędy i dwie kolumny z największą liczbą punktów o wartości 255 (białe piksele na obrazie binarnym)
- Przecięcie wybranych rzędów i kolumn utworzy prostokąt, którego środkiem będzie poszukiwany punkt na źrenicy

#### 9.3.2. Wynik kolejnych etapów algorytmu

Na rys. 9.3 przedstawiony jest wynik kolejnych etapów detekcji źrenic za pomocą metody EA na przykładowym zdjęciu oka.



Rysunek 9.3. Kolejne etapy wykrywania źrenic metodą EA

## Bibliografia

- [1] A. Yanamandra. "Young and Old Images Dataset". (2019), adr.: <https://www.kaggle.com/abhishekryana/young2old-dataset> (term. wiz. 18.10.2021).
- [2] M. R. L. (MRL). "MRL Eye Dataset". (), adr.: <http://mrl.cs.vsb.cz/eyedataset> (term. wiz. 18.10.2021).
- [3] OpenCV. "Class CascadeClassifier". (), adr.: <https://docs.opencv.org/3.4.15/javadoc/org/opencv/objdetect/CascadeClassifier.html> (term. wiz. 30.10.2021).
- [4] ——, "Home: OpenCV". (), adr.: <https://opencv.org/> (term. wiz. 30.10.2021).
- [5] P. Viola i M. Jones, "Rapid object detection using a boosted cascade of simple features", w *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, t. 1, 2001, s. I–I. DOI: 10.1109/CVPR.2001.990517.
- [6] G. S. Behera. "Face Detection with Haar Cascade". (2020), adr.: <https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08> (term. wiz. 30.10.2021).
- [7] A. Rosebrock. "OpenCV Haar Cascades". (2021), adr.: <https://www.pyimagesearch.com/2021/04/12/opencv-haar-cascades/> (term. wiz. 25.09.2021).
- [8] A. Obukhov, "Chapter 33 - Haar Classifiers for Object Detection with CUDA", w *GPU Computing Gems Emerald Edition*, ser. Applications of GPU Computing Series, W.-m. W. Hwu, red., Boston: Morgan Kaufmann, 2011, s. 517–544, ISBN: 978-0-12-384988-5. DOI: <https://doi.org/10.1016/B978-0-12-384988-5.00033-4>. adr.: <https://www.sciencedirect.com/science/article/pii/B9780123849885000334>.
- [9] R. Lienhart. "Haarcascade Frontalface Default". (), adr.: [https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade\\_frontalface\\_default.xml](https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml) (term. wiz. 20.09.2021).
- [10] J. E. C. Cruz, E. H. Shiguemori i L. N. F. Guimarães, "A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery", 2016.
- [11] "LBP cascade frontalface model". (), adr.: [https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade\\_frontalface.xml](https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade_frontalface.xml) (term. wiz. 01.10.2021).
- [12] N. Dalal i B. Triggs, "Histograms of oriented gradients for human detection", w *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, t. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.
- [13] ichi.pro. "Delikatne wprowadzenie do histogramu zorientowanych gradientów". (), adr.: <https://ichi.pro/pl/delikatne-wprowadzenie-do-histogramu-zorientowanych-gradientow-279201309061802> (term. wiz. 31.10.2021).
- [14] S. Mallick. "Histogram of Oriented Gradients explained using OpenCV". (2016), adr.: <https://learnopencv.com/histogram-of-oriented-gradients/> (term. wiz. 31.10.2021).

## 9. Bibliografia

---

- [15] M. Fabien. "A full guide to face detection". (2019), adr.: <https://maelfabien.github.io/tutorials/face-detection/#4-which-one-to-choose-> (term. wiz. 31.10.2021).
- [16] M. S. Nixon i A. S. Aguado, *Feature Extraction & Image Processing for Computer Vision*, 3 wyd. Academic Press, 2013, ISBN: 0123965497.
- [17] R. Gandhi. "Support Vector Machine — Introduction to Machine Learning Algorithms". (2018), adr.: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> (term. wiz. 31.10.2021).
- [18] M. Mamczur. "Jak działają konwolucyjne sieci neuronowe (CNN)". (), adr.: <https://mirosławmamczur.pl/jak-działają-konwolucyjne-sieci-neuronowe-cnn/> (term. wiz. 20.10.2021).
- [19] D. E. King, "Max-Margin Object Detection", *CoRR*, t. abs/1502.00046, 2015. arXiv: 1502.00046. adr.: <http://arxiv.org/abs/1502.00046>.
- [20] openCV. "OpenCV: Deep Neural Network module". (2021), adr.: [https://docs.opencv.org/3.4.15/d6/d0f/group\\_dnn.html](https://docs.opencv.org/3.4.15/d6/d0f/group_dnn.html) (term. wiz. 10.09.2021).
- [21] Y. Jia, E. Shelhamer, J. Donahue i in., "Caffe: Convolutional Architecture for Fast Feature Embedding", *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, czer. 2014. DOI: 10.1145/2647868.2654889.
- [22] S. Mallick. "Face Detection comparison - models". (), adr.: <https://github.com/spmallick/learnopencv/blob/master/FaceDetectionComparison/models/> (term. wiz. 28.10.2021).
- [23] L. P. LLC. "Always Bet on Red". (), adr.: <https://pl.pinterest.com/pin/169025792249522554/> (term. wiz. 20.09.2021).
- [24] WikiPedia. "nVidia CUDA". (2021), adr.: <https://pl.wikipedia.org/wiki/CUDA> (term. wiz. 31.10.2021).
- [25] Google. "CameraX overview". (2021), adr.: <https://developer.android.com/training/camerax> (term. wiz. 30.08.2021).
- [26] S. Ren, X. Cao, Y. Wei i J. Sun, "Face Alignment at 3000 FPS via Regressing Local Binary Features", w *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, s. 1685–1692. DOI: 10.1109/CVPR.2014.218.
- [27] OpenCV. (), adr.: [https://github.com/opencv/opencv\\_contrib/tree/master/modules/face](https://github.com/opencv/opencv_contrib/tree/master/modules/face) (term. wiz. 12.11.2021).
- [28] —, "OpenCV-contrib". (), adr.: [https://github.com/opencv/opencv\\_contrib](https://github.com/opencv/opencv_contrib) (term. wiz. 24.10.2021).
- [29] L. Kurnianggoro. (), adr.: <https://github.com/kurnianggoro/GSOC2017/blob/master/data/lbfmodel.yaml> (term. wiz. 12.11.2021).
- [30] V. Le. "HELEN dataset". (2012), adr.: <http://www.ifp.illinois.edu/~vuongle2/helen/> (term. wiz. 12.11.2021).

- [31] V. Kazemi i J. Sullivan, "One millisecond face alignment with an ensemble of regression trees", w *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, s. 1867–1874. DOI: 10.1109/CVPR.2014.241.
- [32] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou i M. Pantic, "300 Faces In-The-Wild Challenge: database and results", *Image and vision computing*, t. 47, s. 3–18, mar. 2016, ISSN: 0262-8856. DOI: 10.1016/j.imavis.2016.01.002.
- [33] N. Kamarudin, N. A. Jumadi, L. M. Ng i in., "Implementation of Haar Cascade Classifier and Eye Aspect Ratio for Driver Drowsiness Detection Using Raspberry Pi", *Universal Journal of Electrical and Electronic Engineering*, t. 6, s. 67–75, grud. 2019. DOI: 10.13189/ujeee.2019.061609.
- [34] A. Rosebrock. "Eye blink detection with OpenCV, Python, and dlib". (2017), adr.: <https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/> (term. wiz. 13.09.2021).
- [35] S. Hameed. "Haar eye detection model". (), adr.: [https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade\\_eye.xml](https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_eye.xml) (term. wiz. 14.11.2021).
- [36] A. Rosebrock. "Detect eyes, nose, lips, and jaw with dlib, OpenCV, and Python". (2017), adr.: <https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/> (term. wiz. 14.11.2021).
- [37] Nikolaj. "2018". (2013), adr.: [https://www.zastavki.com/eng/Girls/Beautiful\\_Girls/wallpaper-126539.htm](https://www.zastavki.com/eng/Girls/Beautiful_Girls/wallpaper-126539.htm) (term. wiz. 20.09.2021).
- [38] M. Asadifard i J. Shanbezadeh, "Automatic Adaptive Center of Pupil Detection Using Face Detection and CDF Analysis", w *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I*, IMCES, Hong Kong, 2010, March 17–19.
- [39] M. Cieśla i P. Kozioł, *Eye Pupil Location Using Webcam*, Wydział Fizyki, Astronomii i Informatyki Stosowanej, Uniwersytet Jagielloński, ul. Reymonta 4, 30-059, Kraków, Polska, 2012.
- [40] Z.-H. Zhou i X. Geng, "Projection functions for eye detection", *Pattern Recognition*, t. 37, s. 1049–1056, maj 2004. DOI: 10.1016/j.patcog.2003.09.006.

## **Spis rysunków**

2.1	Obliczane cechy w modelu Haar. Źródło: [6]	11
2.2	Działanie filtrowania detekcji twarzy w oparciu o położenie twarzy w centralnej części zdjęcia.	13
2.3	Działanie filtrowania detekcji twarzy w oparciu o odległość wykrytego obszaru poza zdjęcie.	13
2.4	Działanie filtrowania detekcji twarzy w oparciu o wielkość wykrytego obszaru. Źródło zdj.: [23]	14
3.1	Oczekiwany obszar detekcji twarzy.	15
4.1	Przykład zdjęcia twarz z naniesionymi facemarkami	22
6.1	Teoretyczny rozmieszczenie landmarków wokół oczu wraz z naniesionymi połączeniami do obliczenia EAR	26
6.2	EAR dla oczu otwartych	27
6.3	EAR dla oczu zamkniętych	27
6.4	Oczy otwarte na obrazie z kamery	28
6.5	Oczy czasowo zamknięte na obrazie z kamery	28
6.6	Mruganie na obrazie z kamery	29
7.1	Wykorzystanie facemarków do detekcji obszaru oczu	30
7.2	Efekt filtrowania obszarów detekcji oczu	31
7.3	Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami	32
7.4	Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru. Źródło pierwszego zdj.: [37]	32
8.1	Przybliżony obszar oczu, który chcę wykrywać	34
9.1	Kolejne etapy wykrywania źrenic metodą CDF	39
9.2	Kolejne etapy wykrywania źrenic metodą PF	41
9.3	Kolejne etapy wykrywania źrenic metodą EA	42

## **Spis tabel**

3.1	Skuteczność algorytmów detekcji twarzy dla obrazów RGB	16
3.2	Skuteczność algorytmów detekcji twarzy dla obrazów w skali szarości	17
3.3	Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB	18
3.4	Czas przetwarzania algorytmów detekcji twarzy dla obrazów w skali szarości	19
3.5	Wpływ rozdzielczości zdjęcia na detekcję DNN	19
3.6	Wynik porównania sposobów filtrowania detekcji	20
3.7	Skuteczność algorytmów detekcji twarzy dla obrazu na żywo z kamery	21
3.8	Szybkość algorytmów detekcji twarzy dla obrazu na żywo z kamery [klatki/s]	21

5.1	Skuteczność algorytmów detekcji landmarków . . . . .	23
5.2	Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB . . . . .	24
5.3	Szybkość algorytmów detekcji facemarków dla obrazu na żywo z kamery [klatki/s] . . . . .	25
7.1	Skuteczność algorytmu detekcji oczu Haar z dodatkowym obcięciem i bez . . . . .	32
7.2	Skuteczność algorytmu detekcji oczu wykorzystując facemarki z dodatkowym zwiększeniem obszaru i bez . . . . .	33
8.1	Skuteczność algorytmów detekcji twarzy . . . . .	35
8.2	Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB . . . . .	36
8.3	Szybkość algorytmów detekcji oczu dla obrazu na żywo z kamery [klatki/s] . . . . .	37