

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria systemów informatycznych

Wykorzystanie kamery przez aplikacje w systemie Android

Adamski Maciej
Numer albumu 300184

promotor
prof. Przemysław Rokita

WARSZAWA 2021

Wykorzystanie kamery przez aplikacje w systemie Android

Streszczenie. _____ TODO _____

Słowa kluczowe: XXX, XXX, XXX

Agnieński tytuł

Abstract. _____ TODO _____

Keywords: XXX, XXX, XXX

Spis treści

1. Sposób badania algorytmów	7
1.1. Dataset	7
1.2. Urządzenie do testów	7
2. Detekcja twarzy	8
2.1. Algorytmy detekcji twarzy	8
2.1.1. Klasyfikator kaskadowy	8
2.1.1.1. Haar	8
2.1.1.2. Local binary patterns	8
2.1.2. Histogram zorientowanych gradientów	9
2.1.3. CNN + MMOD	10
2.1.4. Głębokie sieci neuronowe	10
2.2. Filtrowanie wyników	10
3. Porównanie algorytmów detekcji twarzy	12
3.1. Odrzucenie algorytmu Dlib CNN MMOD	12
3.2. Testowanie na statycznych zdjęciach	12
3.2.1. Oczekiwany wynik	13
3.2.2. Warunki testowania	13
3.2.3. Badanie skuteczności detekcji	13
3.2.4. Badanie szybkości detekcji	16
3.2.5. Wpływ wielkości zdjęcia na szybkość algorytm <i>DNN Caffe</i>	17
3.2.6. Precyzyja detekcji algorytmu <i>DNN Caffe</i> zależnie od sposobu filtracji .	18
3.3. Testowanie na obrazie z kamery na żywo	18
3.3.1. Skuteczność detekcji	19
3.3.2. Szybkość detekcji	19
3.4. Wybór algorytmu	19
4. Detekcja oczu	20
4.1. Obcięcie obszaru detekcji	20
4.2. Filtrowanie wyników	21
5. Detekcja źrenic	22
5.1. Algorytm CDF	22
5.1.1. Kroki algorytmu	23
5.1.2. Wynik kolejnych etapów algorytmu	24
5.2. Algorytm PF	24
5.3. Algorytm EA	24
6. Landmarks	25
6.1. OpenCV-contrib	25
6.1.1. FacemarkLBF	25

0. Spis treści

7. EAR - Eye Aspect Ratio	26
7.1. Wzór obliczania EAR	26
7.2. Zasada działania EAR w kontekście mrugania	26
7.3. Testowanie z użyciem landmarków LBF opencv-contrib	27
7.3.1. Test z użyciem kamery na żywo	27
7.3.2. Niedokładność nakładania landmarków	30
Bibliografia	31
Spis rysunków	34
Spis tabel	35

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

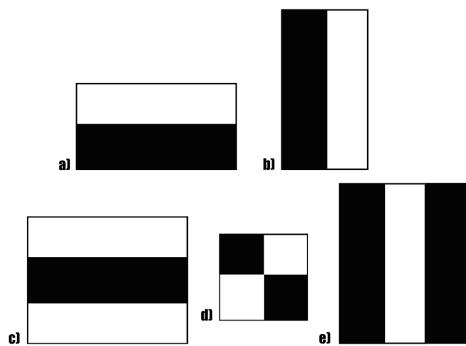
Cascade Classifier to jedno z podejść do zadania klasyfikacji obiektów. Kaskadowość przejawia się tym, że klasyfikator składa się z łańcucha mniejszych klasyfikatorów. Z danych wejściowych jednego może korzystać następnym jako dodatkowe źródło informacji i użyć je do własnej klasyfikacji. Z tego powodu 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 8×8 . 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]

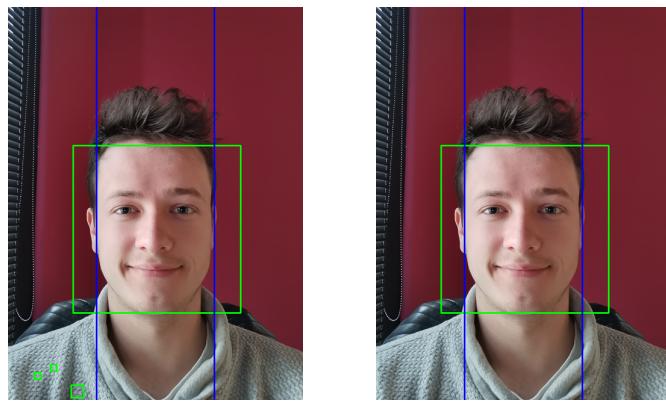
Jednym z modeli dostępnych do detekcji twarzy przy pomocy głębokich sieci neuronowych są modele Caffe (*Convolutional Architecture for Fast Feature Embedding*) [21]. W projekcie zostanie użyty wzorzec *caffe res10_300x300_ssd_iter_140000_fp16* [22].

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:

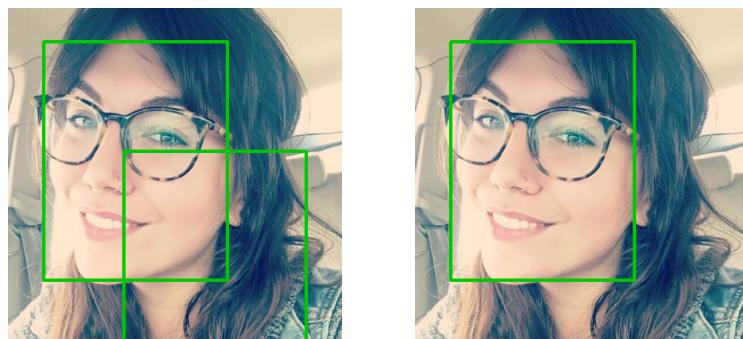
- Na początku odrzucam obszary, których środek znajduje się poza ustalonym pionowym obszarem (przyjąłem przedział [0.25, 0.75] szerokości). Wynika to z założeń, że osoba używająca telefonu, korzysta z niego patrząc na wprost, a nie z boku. Natomiast odchył od pionu to indywidualne preferencje - dlatego 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.

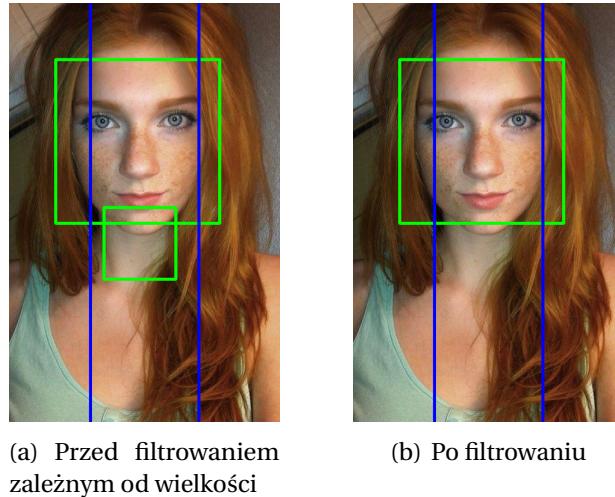


(a) Przed filtrowaniem zależnym od wystawiania poza obraz

(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.

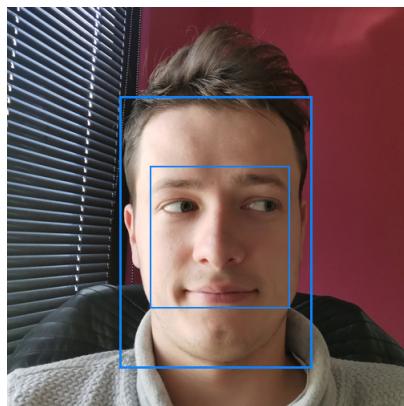
3.2. Testowanie na statycznych zdjęciach

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
Haar Cascade 500x500	69	4	65	32	11	9
Haar Cascade 300x300	68	5	63	33	12	9
LBP Cascade 500x500	58	4	54	29	22	22
LBP Cascade 300x300	61	4	57	34	19	17
DNN Caffe 500x500	80	68	12	11	0	0
DNN Caffe 300x300	80	62	18	18	0	0
Dlib HOG 500x500	78	31	47	36	2	2
Dlib HOG 300x300	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
Haar Cascade 500x500	69	4	65	31	11	9
Haar Cascade 300x300	67	4	63	34	13	9
LBP Cascade 500x500	60	5	55	30	20	17
LBP Cascade 300x300	60	4	56	31	20	17
DNN Caffe 500x500	nd.	nd.	nd.	nd.	nd.	nd.
DNN Caffe 300x300	nd.	nd.	nd.	nd.	nd.	nd.
Dlib HOG 500x500	75	27	48	35	5	5
Dlib HOG 300x300	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
Haar Cascade 500x500	116,72 s	5,836 s	0,072 s
Haar Cascade 300x300	46,24 s	2,312 s	0,028 s
LBP Cascade 500x500	63,27 s	3,163 s	0,039 s
LBP Cascade 300x300	21,25 s	1,062 s	0,013 s
DNN Caffe 500x500	106,17	5,308	0,066 s
DNN Caffe 300x300	103,56 s	5,178 s	0,064 s
Dlib HOG 500x500	71,10 s	3,555 s	0,044 s
Dlib HOG 300x300	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

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

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
300x300	80	62	18	5,148 s	0,064 s
500x500	80	68	12	5,239 s	0,065 s
1000x1000	80	67	13	5,29 s	0,066 s
2000x2000	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.
DNN rgb	200	200	200	200
HOG rgb	200	200	200	200
HOG sk. szaro.	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:
DNN rgb	14,004	14,298	14,175	14,270	14,186
HOG rgb	16,488	16,224	16,337	16,400	16,362
HOG sk. szaro.	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.

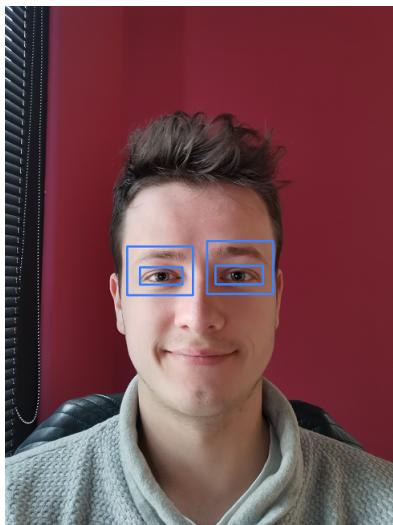
Dodatkowo wybór padł na dostarczanie do detektora obrazu w skali szarości ze względu na zysk w ilości klatek na sekundę - dla barw mono był szybszy o 19% od wersji trójkanałowej.

4. Detekcja oczu

Przed przystąpieniem do detekcji oczu należy wyznaczyć obszar, na którym wykryta została twarz (patrz rozdz. 2.*Detekcja twarzy*).

Następnie obcinając klatkę tylko do ustalonego prostokąta wykrywam oczy za pomocą *Cascading Classifier* - podobnie jak twarz. ——— to będzie usunięte jeśli dam więcej algorytmów ———

Wynik który chcę uzyskać - wykryte oczy powinny się znaleźć pomiędzy naniesionymi prostokątami: ——— to będzie przeniesione do badania ewentualnie, jeśli dam więcej algorytmów ———



Rysunek 4.1. Przybliżony obszar oczu, który chcę wykrywać

4.1. Obcięcie obszaru detekcji

Dodatkowo - prócz detekcji jedynie na obszarze twarzy - zdecydowałem się zawęzić płaszczyznę przeszukiwań. Wstępnie metodą prób i błędów dobrałem następujące parametry obcięcia obszaru:

- Góra: 0.1
- Dół: 0.45
- Lewo: 0.1
- Prawo: 0.1

Parametr określa jaka część obszaru zostaje pominięta z poszczególnych stron. Wynik takiego obcięcia widoczny jest na *rysunku 4.2*.

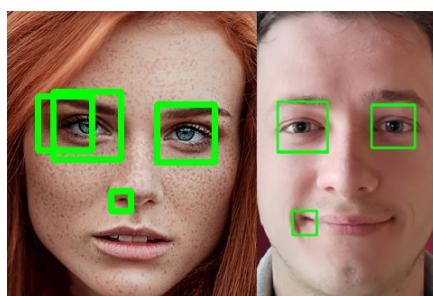
Takie zawężenie obszaru detekcji pozwoliło wyeliminować część z błędnie oznaczonych oczu. Na *rysunku 4.2* widoczne są dodatkowe wykryte obszary oraz ich odrzucenie dzięki temu przekształceniu.



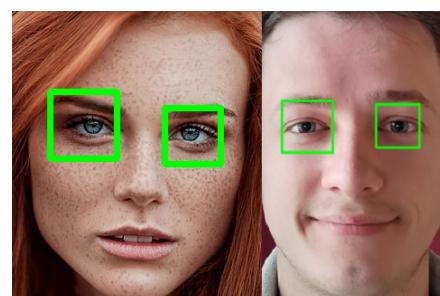
(a) Wykryty obszar twarzy



(b) Wycięty obszar oczu

Rysunek 4.2. Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami

(a) Wykrywanie oczu bez dodatkowego obcięcia obszaru



(b) Wykrywanie oczu z dodatkowym obcięciem obszaru

Rysunek 4.3. Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru.
Źródło pierwszego zdj.:[26]

_____ przyszłości zrobić testy na wielu zdjęciach wraz z wynikami przed/po w formie liczbowej. Wtedy dobrać testowo nowe parametry _____

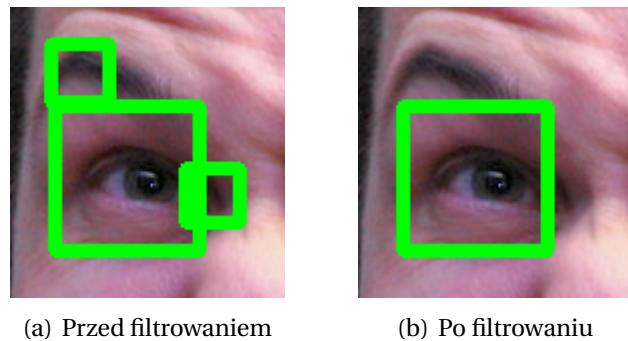
4.2. Filtrowanie wyników

Ze względu na możliwość błędnych wskazań 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 grup wybranie największego obszaru

Przykład rezultatu takiego filtrowania pokazany jest na *rysunku 4.4*. Dodatkowo pozwoliło to na łatwe zidentyfikowanie który obszar to które oko i ich posortowanie.



Rysunek 4.4. Efekt filtrowania obszarów detekcji oczu

5. Detekcja źrenic

Do wykrywania źrenic, najpierw musimy wyznaczyć obszar oczu (patrz rozdz. 4.*Dekodowanie obrazu*). Następnie korzystając z jednych z poniższych metod ustalamy interesujący nas środek źrenicy.

Wynik, który w przybliżeniu chcę uzyskać: — jeśli będzie więcej metod to przesunąć to do testów —

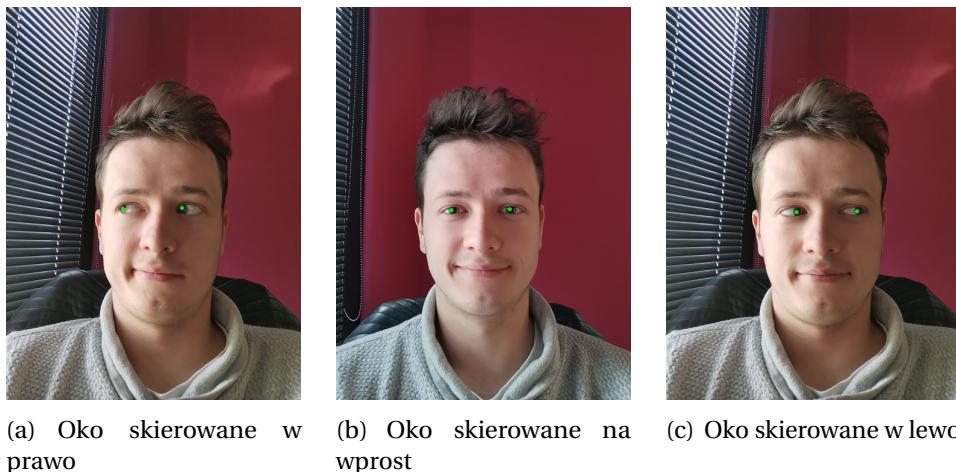


Rysunek 5.1. W przybliżeniu środek źrenic, który chcę uzyskać

5.1. Algorytm CDF

Algorytm zaimplementowany na podstawie dwóch artykułów o detekcji źrenic [27] [28]. Opiera się w głównej mierze na progowaniu za pomocą dystrybuanty. Cały algorytm przetwarza obszar oka w skali szarości.

Metoda ta daje całkiem dobre i prawdopodobnie wystarczające rezultaty.



Rysunek 5.2. Rezultat wykrywania źrenic metodą CDF

5.1.1. Kroki algorytmu

- Za pomocą progowania z użyciem dystrybuanty tworzymy obraz binarny.

$$CDF(r) = \sum_{w=0}^r p(w) \quad (1)$$

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

$$I'(x, y) = \begin{cases} 255, & CDF(I(x, y)) < a \\ 0, & \text{inne} \end{cases} \quad (2)$$

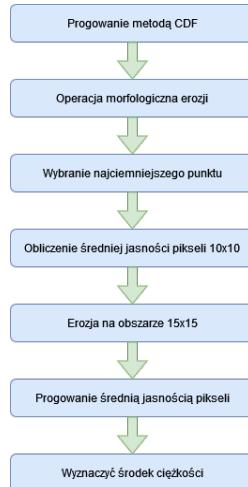
Gdzie I to jasność piksela, natomiast a to ustalony próg

- 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

$$I'(x, y) = \begin{cases} 255, & I(x, y) < AVG_I \\ 0, & w.p.p. \end{cases} \quad (3)$$

Gdzie AVG_I to średnia jasność obszaru obliczona wcześniej

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



Rysunek 5.3. Kroki algorytmu metodą CDF

5.1.2. Wynik kolejnych etapów algorytmu

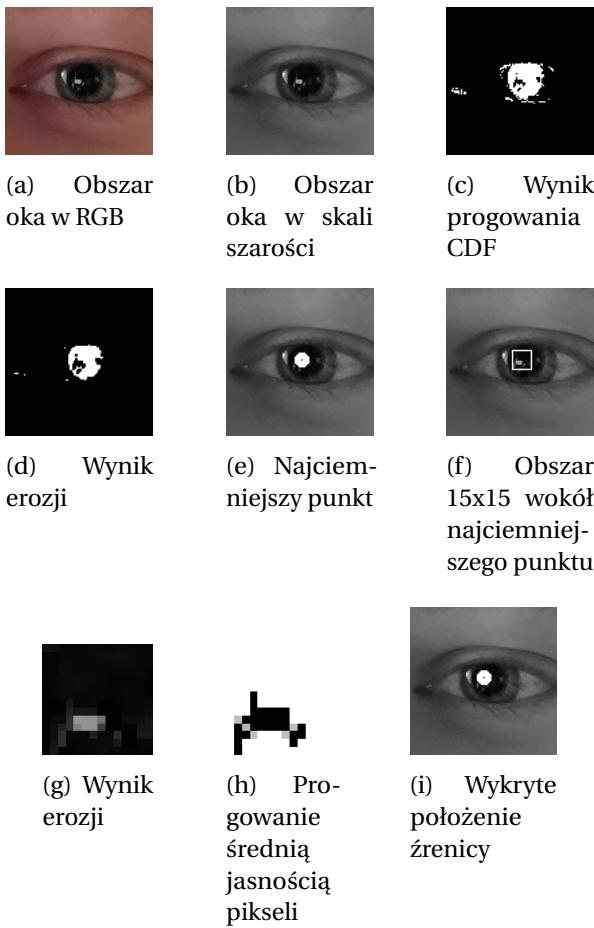
Na *rysunku 5.4* przedstawiony jest na przykładzie wynik kolejnych etapów detekcji źrenic za pomocą metody CDF.

5.2. Algorytm PF

____ W przyszłości do testów zaimplementować algorytm PF[28] ____

5.3. Algorytm EA

____ W przyszłości do testów zaimplementować algorytm EA[28] ____



Rysunek 5.4. Kolejne etapy wykrywania źrenic metodą CDF

6. Landmarks

Są 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 lub czy się uśmiechamy.

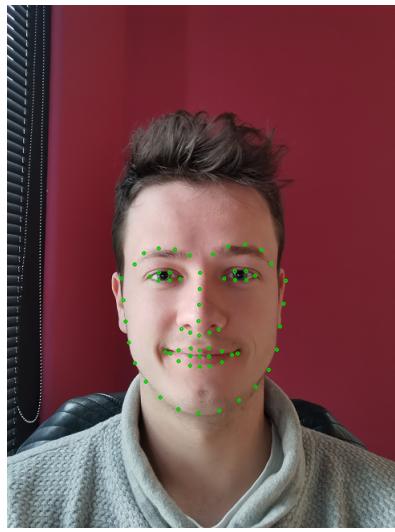
6.1. OpenCV-contrib

Dodatkowe moduł opencv facemark (*OpenCV-contrib*) zawiera trzy algorytmy detekcji landmarków:

- Kazemi
- AAM
- LBF

6.1.1. FacemarkLBF

Używając metody FacemarkLBF oraz modelu *lbfmodel.yaml* określiłem punkty orientacyjne twarzy. [29]



Rysunek 6.1. Twarz z naniesionymi landmarkami

7. EAR - Eye Aspect Ratio

Metoda polegająca na obliczeniu *EAR* [30] [31], czyli stosunek otwarcia oczu - wysokość do szerokości widocznej części gałki ocznej. Wykorzystuje się tu landmarki (patrz rozdz. 6.*Landmarks*) naniesione dookoła oczu.

7.1. Wzór obliczania EAR

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

Dla 6 punktów:

$$EAR = \frac{dist(L_0, L_1) + dist(L_2, L_4)}{2 * dist(L_3, L_5)} \quad (4)$$

Natomiast, dla 4 punktów:

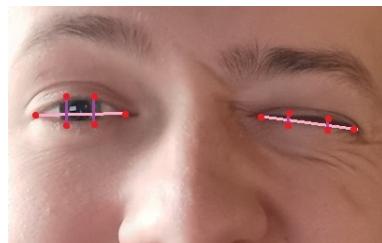
$$EAR = \frac{dist(L_0, L_2)}{dist(L_1, L_3)} \quad (5)$$

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

7.2. Zasada działania EAR w kontekście mrugania

W teorii otwarte oczy będą miały większy wymiar liczbowy *EAR*, niż oczy zamknięte. Na rysunku 7.1 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. Całkowicie obserwując zmiany np. za pomocą pochodnej jesteśmy w stanie stwierdzić mrugnięcie.



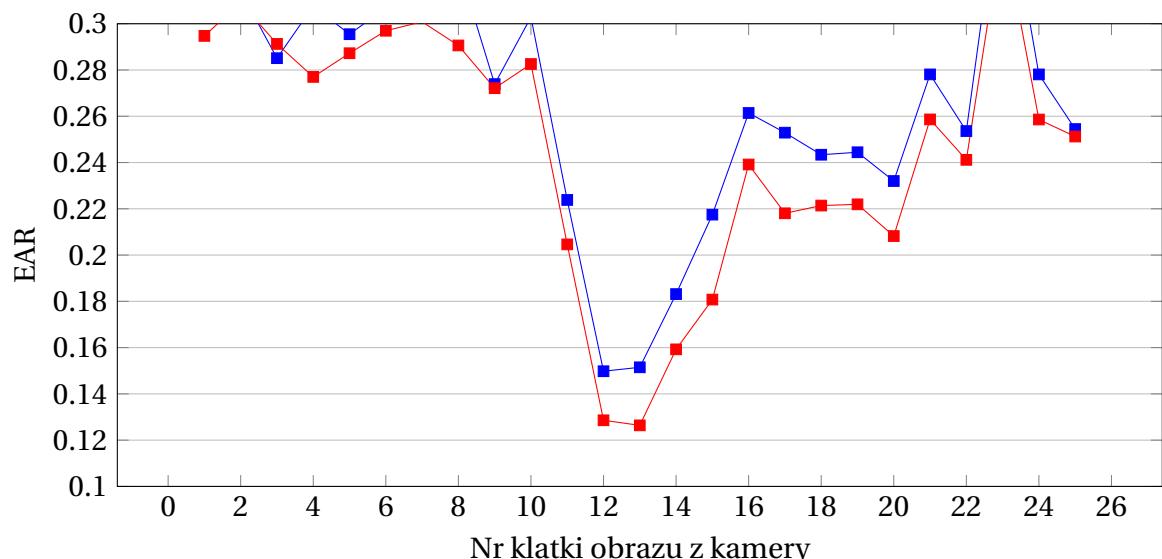
Rysunek 7.1. Teoretyczny rozmieszczenie landmarków wokół oczu wraz z naniesionymi połączonymi do obliczenia EAR

7.3. Testowanie z użyciem landmarków LBF opencv-contrib

7.3.1. Test z użyciem kamery na żywo

Wykonałem kilka krótkich testów z użyciem obrazu pochodzącego z przedniej kamery telefonu.

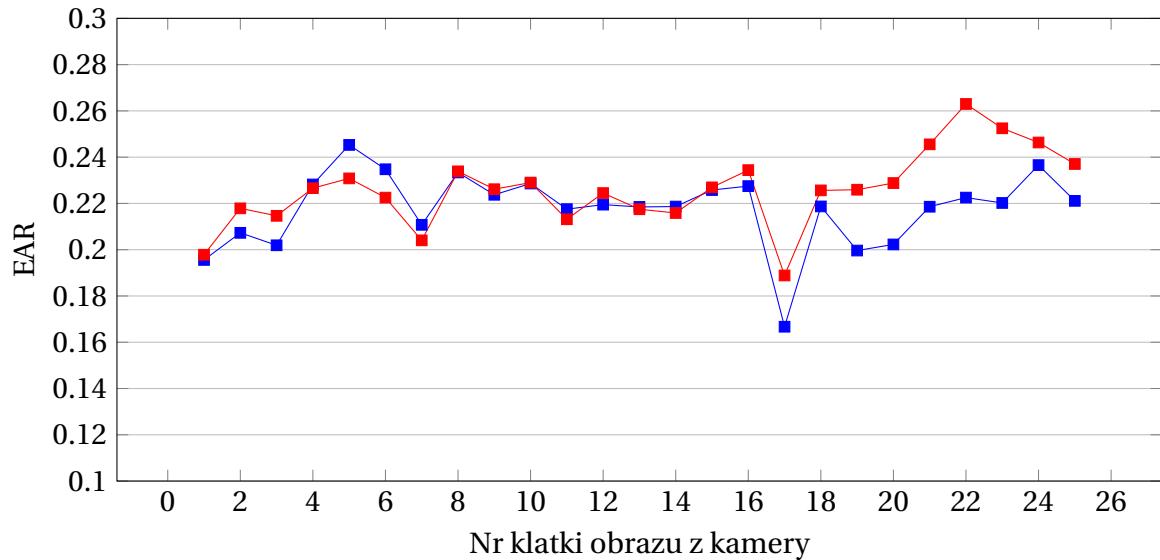
Poniżej znajdują się trzy testy, na których mrugnąłem tylko raz - lewym okiem, prawym i oboma na raz.



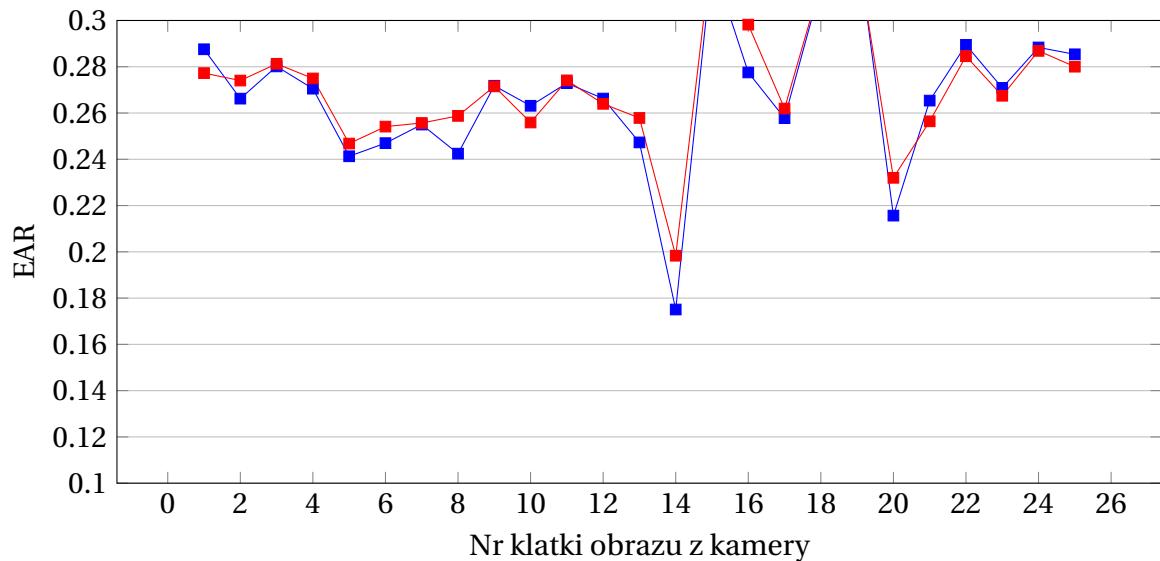
Rysunek 7.2. Mrugnięcie lewym okiem

Testy z pojedynczym mruganiem w krótkim okresie czasu dają przyzwoite wyniki i można na nich określić moment mrugania. W szczególności przy mrugnięciu jednym okiem.

7. EAR - Eye Aspect Ratio



Rysunek 7.3. Mrugnięcie prawym okiem



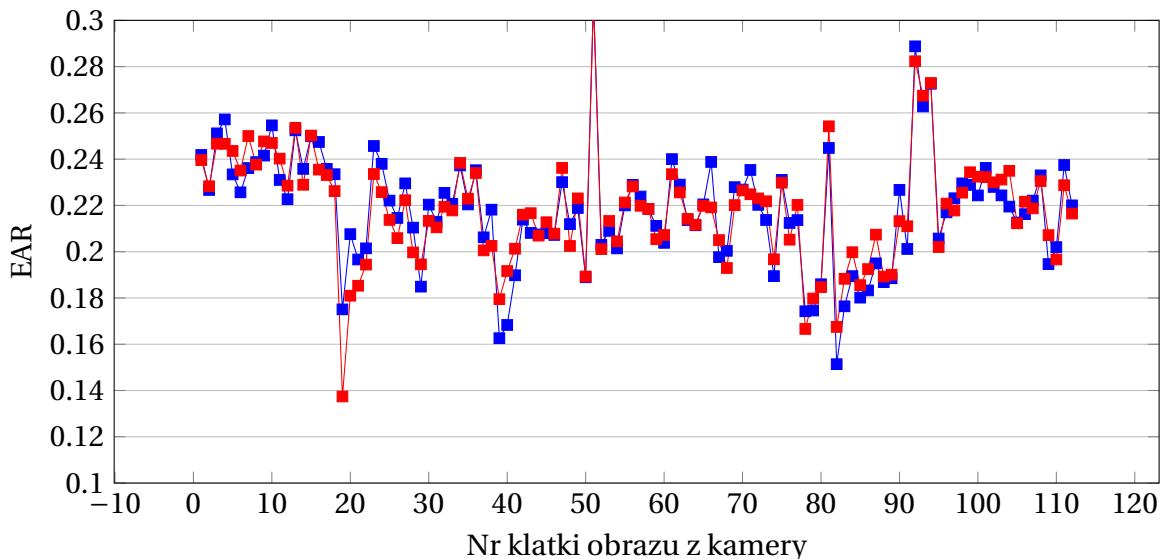
Rysunek 7.4. Mrugnięcie oboma oczami na raz

Poniżej rozciągnąłem w czasie test na sekwencje kilku mrugnięć.

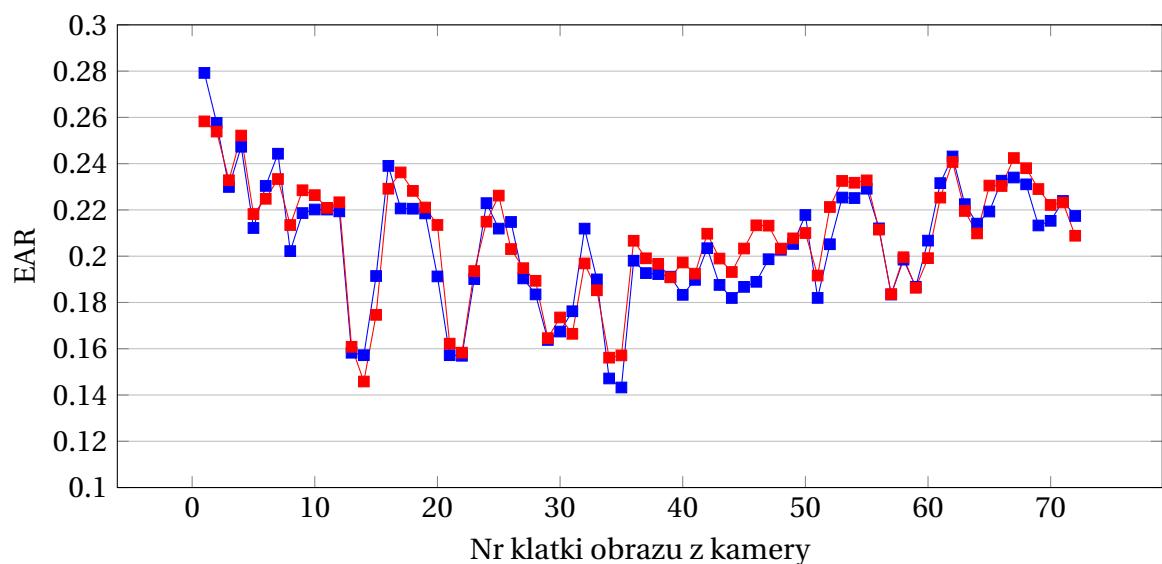
Tu również z dużym prawdopodobieństwem można określić, w których klatkach wystąpiło mrugnięcie - widać gwałtowne obniżenie wartości EAR.

Najlepsze wyniki były w przypadku pierwszego testu, ponieważ widać wtedy znaczną różnicę EAR między otwartym ($\sim 0.26 - 0.30$), a zamkniętym okiem ($\sim 0.12 - 0.18$). Na pozostałych testach różnice nie były już tak znaczące - na ostatnich dwóch wykresach otwarte i zamknięte oko ma niewielką różnicę w EAR. Takie małe różnice wyników mogą uniemożliwić prawidłową detekcję.

W każdym z przypadków testowych EAR dla jednego i drugiego oka prawie się pokry-



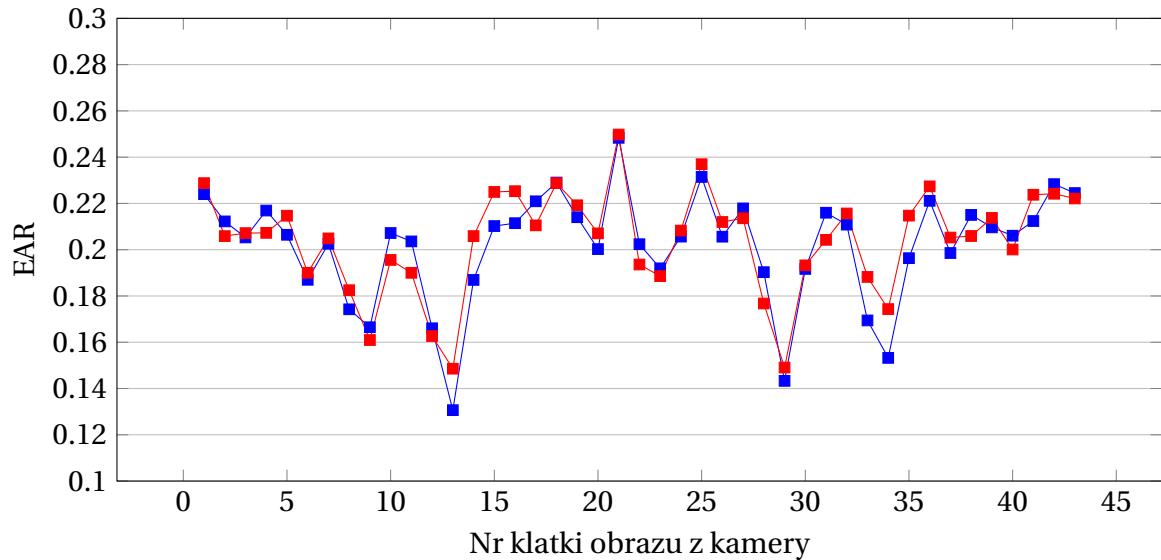
Rysunek 7.5. Kilka mrugnięć



Rysunek 7.6. Kilka mrugnięć

wają. Nawet przy mruganiu tylko jednym. Nie pozwoli to więc określić, którym okiem użytkownik mrugał.

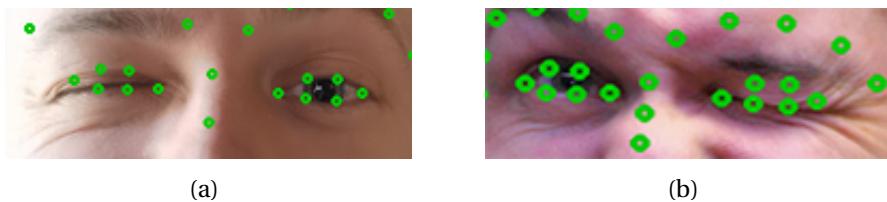
Niewątpliwą trudnością w przypadku tej metody byłoby określenie progu wartości EAR, które zakwalifikowałbym jako mrugnięcie. Patrząc na wykresy powyżej można przyjąć, że jest to wartość koło 0.18. Jednak w przypadku ostatecznego wyboru tej metody wymagałoby to dodatkowych badań celem określenia tej wartości.



Rysunek 7.7. Kilka mrugnięć

7.3.2. Niedokładność nakładania landmarków

Patrząc jednak na położenie tych landmarków wokół oczu mam wątpliwości co do skuteczności tej metody:



Rysunek 7.8. Landmarki na oczach otwartych/zamkniętych

Jak widać algorytm całkiem dobrze radzi sobie z rozmieszczeniem landmarków w przypadku otwartych oczu. Natomiast gdy oczy są zamknięte widać dużą niedokładność, która na pewno w dużym stopniu utrudnia prawidłową detekcję mrugnięcia.

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 (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 (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).

7. 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 (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] Nikolaj. "2018". (2013), adr.: <https://www.zastavki.com/eng/Girls/Beautiful-Girls/wallpaper-126539.htm> (term. wiz. 20.09.2021).
- [27] 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.
- [28] 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.
- [29] S. Mallick. "Facemark: Facial Landmark Detection using OpenCV". (2018), adr.: <https://learnopencv.com/facemark-facial-landmark-detection-using-opencv/> (term. wiz. 10.09.2021).

- [30] 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.
- [31] 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).

Spis rysunków

2.1	Obliczane cechy w modelu Haar. Źródło: [6]	9
2.2	Działanie filtrowania detekcji twarzy w oparciu o położenie twarzy w centralnej części zdjęcia.	11
2.3	Działanie filtrowania detekcji twarzy w oparciu o odległość wykrytego obszaru poza zdjęcie.	11
2.4	Działanie filtrowania detekcji twarzy w oparciu o wielkość wykrytego obszaru. Źródło zdj.: [23]	12
3.1	Oczekiwany obszar detekcji twarzy.	13
4.1	Przybliżony obszar oczu, który chcę wykrywać	20
4.2	Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami	21
4.3	Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru. Źródło pierwszego zdj.: [26]	21
4.4	Efekt filtrowania obszarów detekcji oczu	22
5.1	W przybliżeniu środek źrenic, który chcę uzyskać	22
5.2	Rezultat wykrywania źrenic metodą CDF	23
5.3	Kroki algorytmu metodą CDF	24
5.4	Kolejne etapy wykrywania źrenic metodą CDF	25
6.1	Twarz z naniesionymi landmarkami	26
7.1	Teoretyczny rozmieszczenie landmarków wokół oczu wraz z naniesionymi połączeniami do obliczenia EAR	27
7.2	Mrugnięcie lewym okiem	27
7.3	Mrugnięcie prawym okiem	28
7.4	Mrugnięcie oboma oczami na raz	28
7.5	Kilka mrugnięć	29
7.6	Kilka mrugnięć	29
7.7	Kilka mrugnięć	30
7.8	Landmarki na oczach otwartych/zamkniętych	30

Spis tabel

3.1	Skuteczność algorytmów detekcji twarzy dla obrazów RGB	14
3.2	Skuteczność algorytmów detekcji twarzy dla obrazów w skali szarości	15
3.3	Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB	16
3.4	Czas przetwarzania algorytmów detekcji twarzy dla obrazów w skali szarości	17
3.5	Wpływ rozdzielczości zdjęcia na detekcję DNN	17
3.6	Wynik porównania sposobów filtrowania detekcji	18

3.7 Skuteczność algorytmów detekcji twarzy dla obrazu na żywo z kamery	19
3.8 Szybkość algorytmów detekcji twarzy dla obrazu na żywo z kamery [klatki/s] .	19