

# [INZ] Notatki

Adamski Maciej

Październik 2021

## 1 Detekcja twarzy

### 1.1 Cascading Classifier

#### 1.1.1 Haar Cascade

#### 1.1.2 LBP Cascade

### 1.2 Deep Neural Network

Jeden z modułów Opencv-contrib. Służy do ładowania modeli głębkich sieci neuronowych i przepuszczeniu przez nie obrazów (lub ich części) celem wykrycia różnych obiektów.

#### 1.2.1 Caffemodel

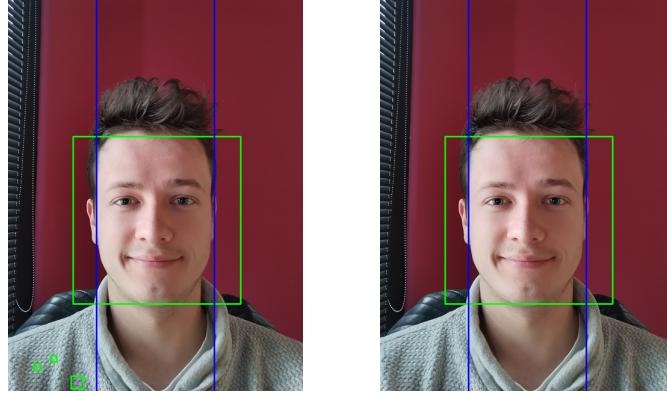
Jednym z modeli dostępnych do detekcji twarzy przy pomocy głębkich sieci neuronowych są modele Caffe (*Convolutional Architecture for Fast Feature Embedding*).

Aktualnie używany w projekcie wzorzec caffe to  
`res10_300x300_ssd_iter_140000_fp16`.

### 1.3 Filtrowanie wyników

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

- Pierwszym etapem jest odrzucenie obszarów, 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.

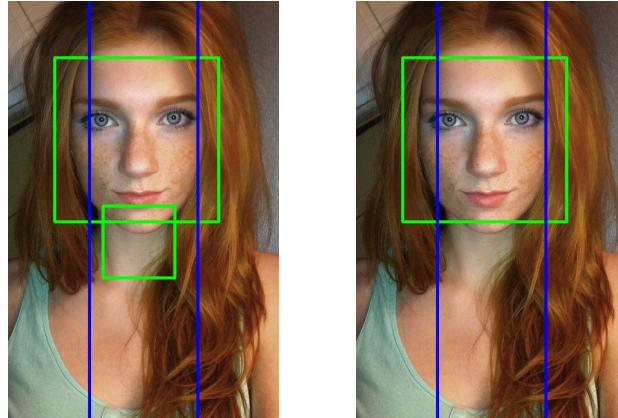


(a) Przed filtrowaniem zależnym od położenia. Wykryte dodatkowe błędne obszary

(b) Po filtrowaniu. Błędne obszary odrzucone

Rysunek 1: Zielone obszary - obszary, w których według klasyfikatora może znajdować się twarz. Niebieskie pasy - obszar, w którym musi znajdować się środek twarzy.

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



(a) Przed filtrowaniem zależnym od wielkości. Wykryty dodatkowy błędny obszar

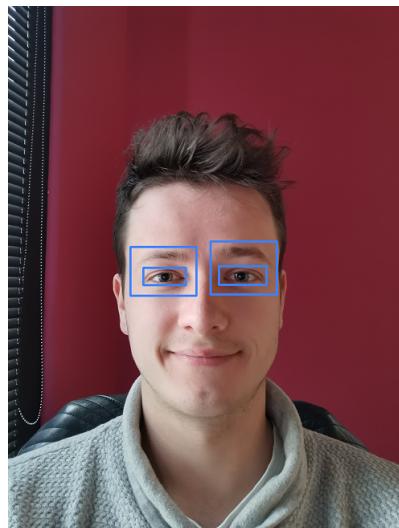
(b) Po filtrowaniu. Błędny obszar odrzucony

Rysunek 2: Kilka wykrytych obszarów w środkowej części. Wybieram największy. [4]

## 2 Detekcja oczu

Przed przystąpieniem do detekcji oczu należy wyznaczyć obszar, na którym wykryta została twarz. Następnie obcinając klatkę tylko do ustalonego prostokąta wykrywam oczy za pomocą Cascading Classifier - podobnie jak twarz.

Wynik który chcę uzyskać - wykryte oczy powinny się znaleźć pomiędzy naniesionymi prostokątami:



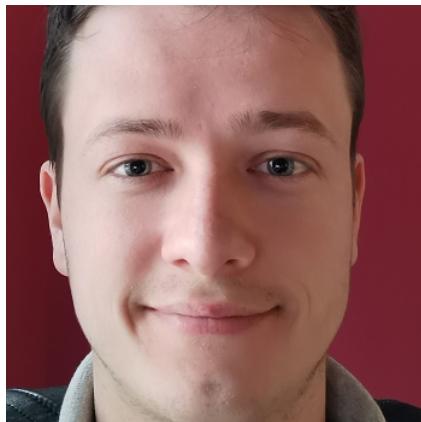
Rysunek 3: Przybliżony obszar oczu, który chcę wykrywać

### 2.1 Obcięcie obszaru detekcji

Dodatkowo - oprócz detekcji jedynie na obszarze twarzy - zdecydowałem się zaćwierdzić 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.



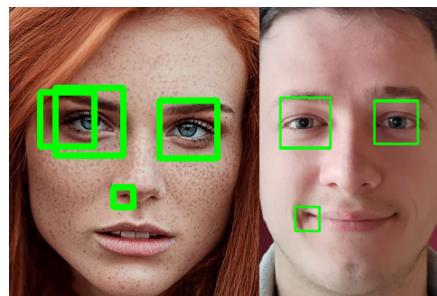
(a) Wykryty obszar twarzy



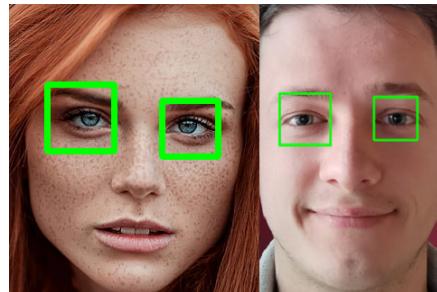
(b) Wycięty obszar oczu

Rysunek 4: Obcięcie obszaru detekcji oczu

Takie zawężenie obszaru detekcji pozwoliło wyeliminować część z błędnie oznaczonych oczu - poniżej środka twarzy czy w bok od rzeczywistego położenia:



(a) Wykrywanie oczu bez dodatkowego obcięcia obszaru



(b) Wykrywanie oczu z dodatkowym obcięciem obszaru

Rysunek 5: Poprawienie rezultatu detekcji oczu po dodatkowym obcięciu obszaru. [6]

— przyszłości zrobić testy na wielu zdjęciach wraz z wynikami przed/po w formie liczbowej. —

## 2.2 Filtrowanie wyników

Ze względu na możliwość błędnych wskazań warto będzie wprowadzić filtrowanie wyników podobne jak w przypadku detekcji twarzy.

Jednym z pomysłów jest odległość między oczami

## 3 Detekcja źrenic

Do wykrywania źrenic, najpierw musimy wyznaczyć obszar oczu. Następnie korzystając z jednych z poniższych metod ustalam interesujący nas środek.

Wynik, który w przybliżeniu chcę uzyskać:



Rysunek 6: W przybliżeniu środek źrenic, który chcę uzyskać

### 3.1 Algorytm CDF

Algorytm zaimplementowany na podstawie dwóch artykułów o detekcji źrenic [1] [2]. 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.



(a) Oko skierowane w prawo      (b) Oko skierowane na wprost



(c) Oko skierowane w lewo

Rysunek 7: Rezultat wykrywania źrenic metodą CDF

### 3.1.1 Kroki algorytmu

- Za pomocą progowania z użyciem dystrybuanty CDF 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

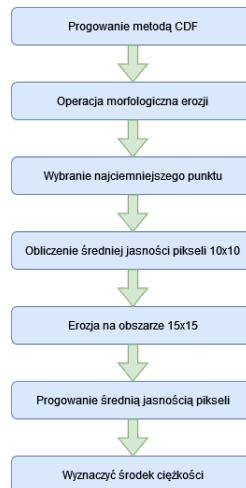
- Na uzyskany obraz binarny nakładamy operację morfologiczną erozji (filtr minimalny), celem usunięcia pojedynczych ciemnych pikseli

- Znajdujemy najciemniejszy piksel na oryginalnym obrazie wśród tych, które mają wartość 255 (są białe) na obrazie binarnym
- Obliczamy średnią jasność pikseli w kawdracie 10x10 wokół wybranego najciemniejszego punktu
- Nakładamy erozję na obszarze 15x15 wokół wybranego punktu
- Na tym obszarze stosujemy progowanie

$$I^*(x, y) = \begin{cases} 255, & I(x, y) < AVG_I \\ 0, & \text{wpp} \end{cases} \quad (3)$$

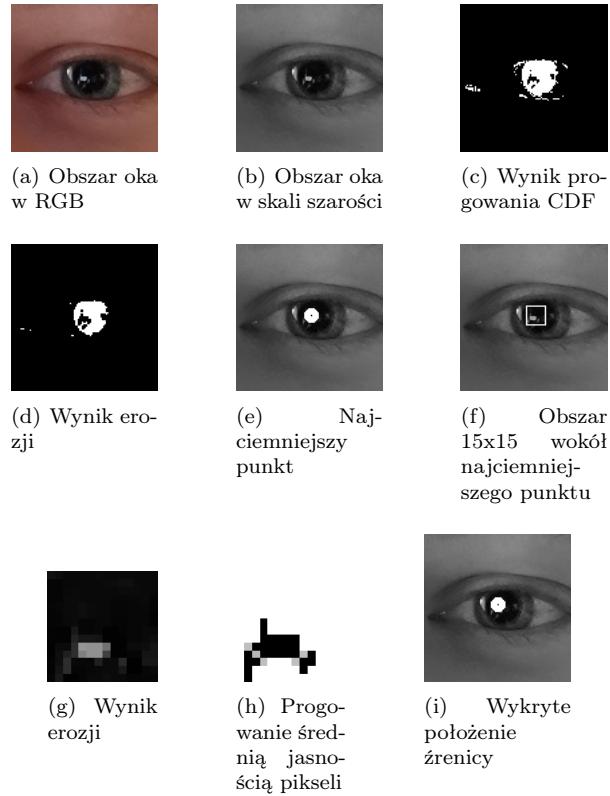
Gdzie  $AVG_I$  to średnia jasność obszaru obliczona wcześniej

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



Rysunek 8: Kroki algorytmu metodą CDF

### 3.1.2 Wynik kolejnych etapów algorytmu



Rysunek 9: Kolejne etapy wykrywania żrenic metodą CDF

### 3.2 Algorytm PF

\_\_\_\_\_ W przyszłości do testów zaimplementować algorytm PF[2] \_\_\_\_\_

### 3.3 Algorytm EA

\_\_\_\_\_ W przyszłości do testów zaimplementować algorytm EA[2] \_\_\_\_\_

## 4 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.

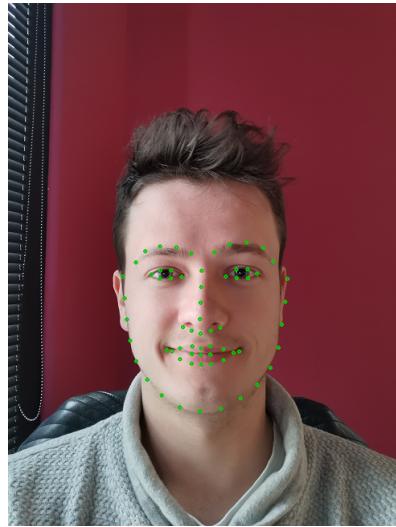
## 4.1 OpenCV-contrib

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

- Kazemi
- AAM
- LBF

### 4.1.1 FacemarkLBF

Używając metody FacemarkLBF oraz modelu *lbfmodel.yaml* określiłem punkty orientacyjne twarzy. [5]



Rysunek 10: Twarz z naniesionymi landmarkami

## 5 EAR - Eye Aspect Ratio

Metoda wykorzystująca landmarki na twarzy. Posiadając oznaczone za pomocą tej metody oczy możemy obliczyć tzw. *EAR*, czyli stosunek otwarcia oczu - wysokość do szerokości widocznej części gałki ocznej. [3] [7]

Zależnie od ilości punktów wokół oka będzie różny wzór obliczania EAR.  
Dla 6 punktów:

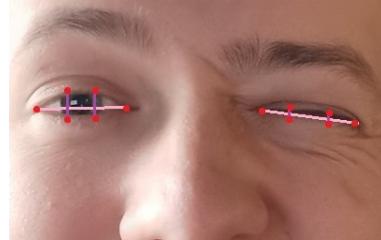
$$EAR = \frac{dist(L_0, L_1) + dist(L_2, L_4)}{2 * dist(L_3, L_5)} \quad (4)$$

Natomiast, dla 4 punktów:

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

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

W teorii otwarte oczy będą miały większy wymiar liczbowy EAR, niż oczy zamknięte. Na zdjęciu poniżej widać, że oko otwarte ma większe odległości między punktami pionowymi niż w przypadku oka zamkniętego.



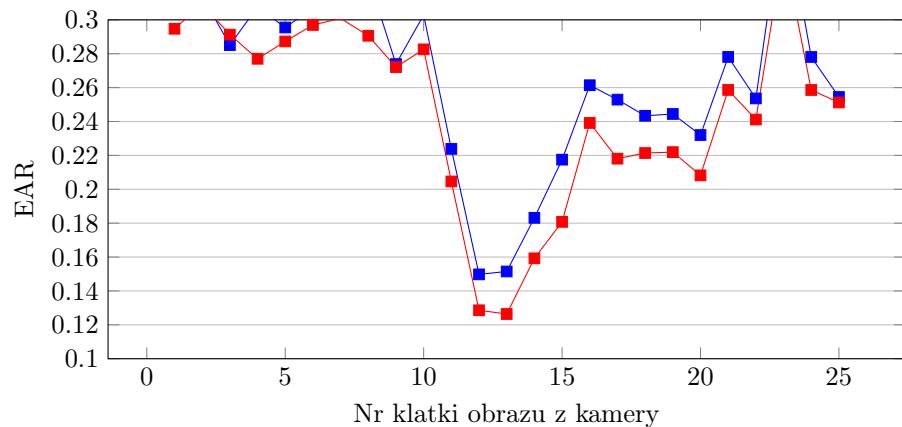
Rysunek 11: Teoretyczny rozmieszczenie landmarków wokół oczu wraz z nansionymi połączonymi do obliczenia EAR

## 5.1 Testowanie z użyciem landmarków LBF opencv-contrib

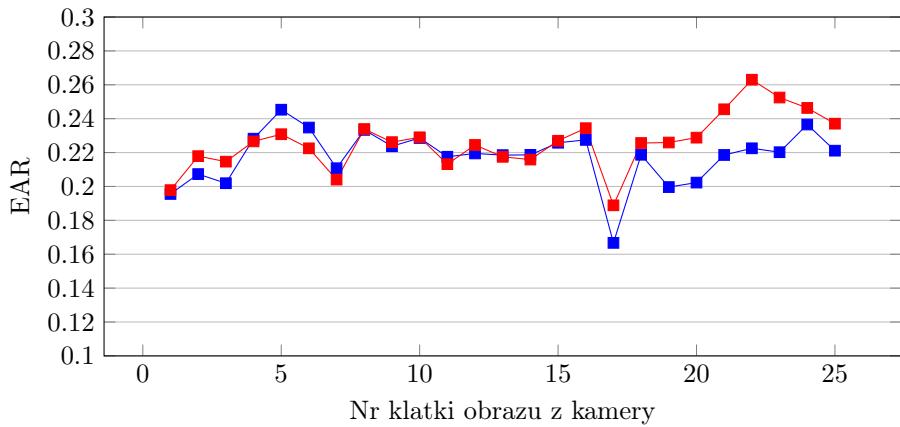
### 5.1.1 Test z użyciem kamery na żywo

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

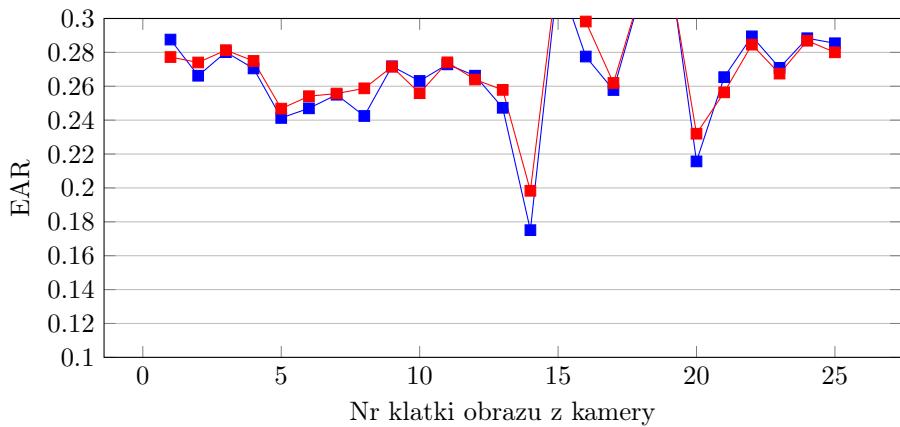
Poniżej znajdują się trzy testy, na których mrugnąłem tylko raz - lewym okiem, prawym i oboma na raz.



Rysunek 12: Mrugnięcie lewym okiem



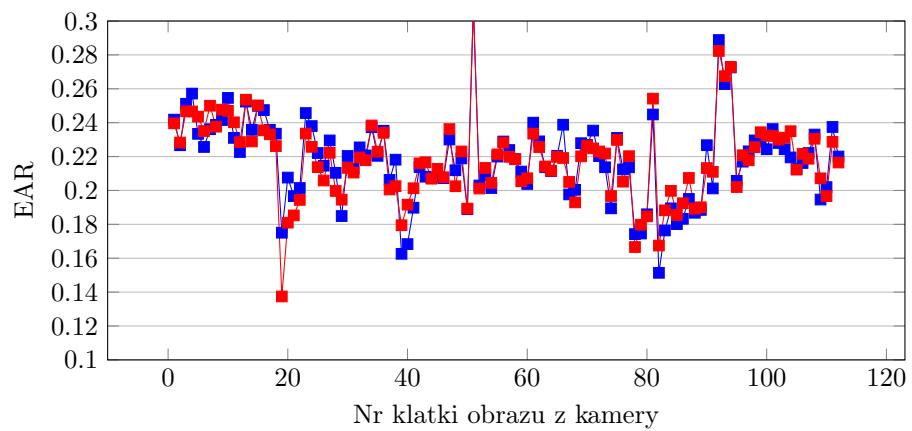
Rysunek 13: Mrugnięcie prawym okiem



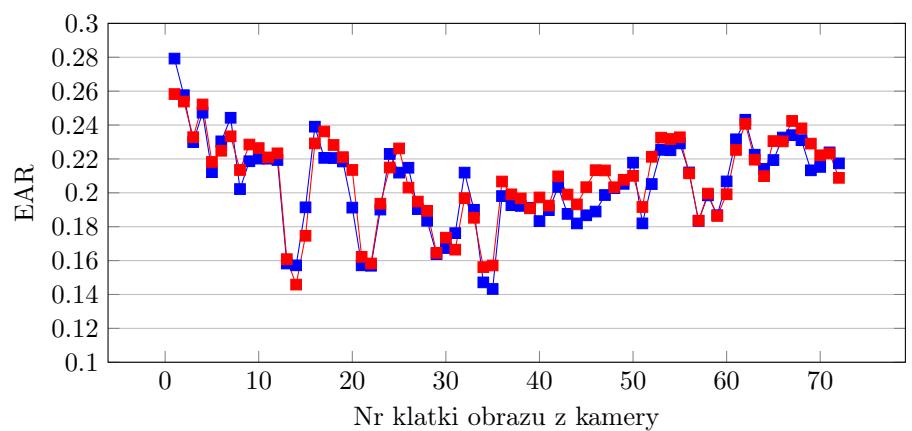
Rysunek 14: Mrugnięcie oboma oczami na raz

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

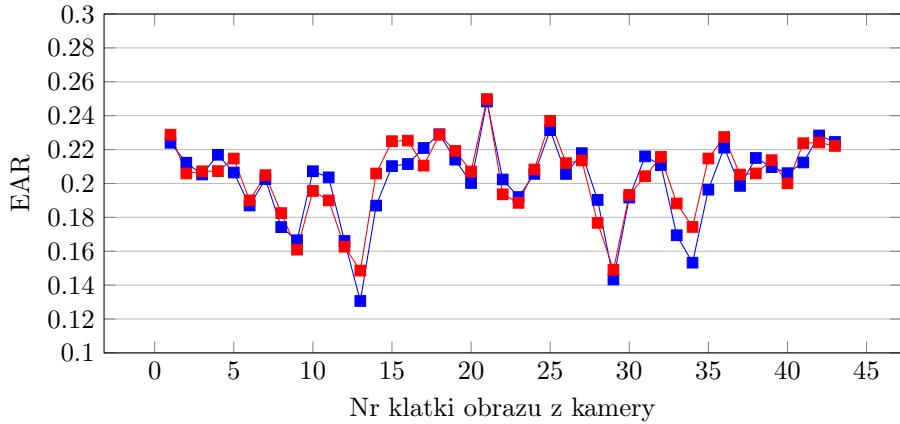
Poniżej rozciągnąłem w czasie test na sekwencje kilku mrugnięć.



Rysunek 15: Kilka mrugnięć



Rysunek 16: Kilka mrugnięć



Rysunek 17: Kilka mrugnięć

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

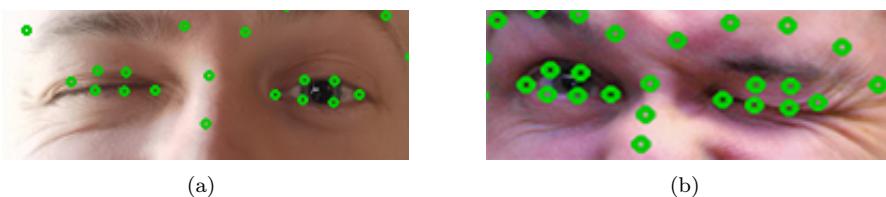
Najlepsze wyniki były w przypadku pierwszego testu, ponieważ widać wtedy znaczną różnicę EAR między otwartym ( $\sim 0.26 - 0.30$ ), a zamkniętym okiem ( $\sim 0.12 - 0.18$ ). Na pozostałych testach różnice nie były już tak znaczące - na ostatnich dwóch wykresach otwarte i zamknięte oko ma niewielką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

W każdym z przypadków testowych EAR dla jednego i drugiego oka prawie się pokrywają. 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 by przyjąć, że jest to wartość koło 0.18. Jednak w przypadku ostatecznego wyboru tej metody wymagałoby to dodatkowych badań celem określenia tej wartości.

### 5.1.2 Niedokładność nakładania landmarków

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



Rysunek 18: 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] Mansour Asadifard i Jamshid 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.
- [2] Michał Cieśla i Przemysław Kozioł. *Eye Pupil Location Using Webacam*. Wydział Fizyki, Astronomii i Informatyki Stosowanej, Uniwersytet Jagielloński. ul. Reymonta 4, 30-059, Kraków, Polska, 2012.
- [3] Nora Kamarudin i in. „Implementation of Haar Cascade Classifier and Eye Aspect Ratio for Driver Drowsiness Detection Using Raspberry Pi”. W: *Universal Journal of Electrical and Electronic Engineering* 6.5B (2019), s. 67–75.
- [4] Legacy Photography LLC. *Always Bet on Red*. URL: <https://pl.pinterest.com/pin/169025792249522554/> (term. wiz. 20.09.2021).
- [5] Satya Mallick. *Facemark: Facial Landmark Detection using OpenCV*. 2018. URL: <https://learnopencv.com/facemark-facial-landmark-detection-using-opencv/> (term. wiz. 10.09.2021).
- [6] Nikolaj. 2018. 2013. URL: [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).
- [7] Adrian Rosebrock. *Eye blink detection with OpenCV, Python, and dlib*. 2017. URL: <https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/> (term. wiz. 13.09.2021).