

**Jak statystycznie dobrać dane, aby uniknąć przedawniania się modelu?**

(OpenCV, emgu CV, Optical Flow, LBP)

**Etap 1**

|  |
| --- |
| Face recognition |
| Version 1.3 |
| On 2017-09-26 |

**Informacje o dokumencie**

|  |  |
| --- | --- |
| **Wersja** | **1.3** |
| **Data** | **2017-09-26** |

**Historia zmian**

|  |  |  |
| --- | --- | --- |
| **Wersja** | **Opis** | **Data** |
| **0.1** | Utworzenie dokumentu | 2017-09-14 |
| **0.2** | Wstępne informacje o EmguCV, LBP, Optical Flow, OpenCV | 2017-09-18 |
| **1.0** | Uzupełnienie dokumentacji o wyniki eksperymentów z algorytmem LBP | 2017-09-20 |
| **1.1** | Dodanie opisu algorytmu Haar | 2017-09-21 |
| **1.2** | Opracowanie wyników badań | 2017-09-25 |
| **1.3** | Ukończenie dokumentu | 2017-09-26 |

**Spis treści**

[**1. Optical Flow**](#_9bmq33ce1oi3) **5**

[1.1 Wstęp](#_8gxvf69e1afh) 5

[1.2 Problemy estymacji przepływu optycznego:](#_q0ptg6kbexqw) 5

[1.3 Metody wykrywania ruchu](#_gip7grjq3ork) 7

[1.3.1 Metody gradientowe](#_tj2n0az9zu9) 7

[1.3.2 Metody częstotliwościowe](#_rj48p4q3ez5j) 9

[1.3.3 Metody korelacyjne](#_18p37u6h1po7) 9

[**2. OpenCV**](#_774pgab2s7fb) **10**

[2.1 Wstęp](#_sftcqbqvsmxb) 10

[2.2 Historia](#_ydr0283f1u3s) 10

[2.3 Aplikacje](#_7bfsy5i62v02) 11

[2.4 Języki programowania](#_tzk21z15a4lf) 12

[**3. EmguCV**](#_k25yzjt6yd7c) **13**

[3.1 Wstęp](#_5joowpga5xj) 13

[3.2 Zalety Emgu CV](#_ifix3d9pj4mi) 13

[3.3 Budowa Emgu CV](#_3poed5rb95fn) 14

[3.4 Licencja EmguCV](#_w76gpzfbl9lo) 15

[3.5 Wydajność](#_opx2jz46rozq) 15

[3.6 Rozpoznawanie krawędzi i konturów](#_4efhe588ikhh) 15

[3.7 Rozpoznawanie twarzy](#_atffcjmmi4ag) 16

[3.8 Algorytmy nauczania maszynowego udostępniane przez EmguCV](#_ucdej43e37wd) 16

[**4. LBP**](#_dxjo92tjzc9n) **17**

[4.1 Wstęp](#_fhjgy81w4ma8) 17

[4.2 Działanie algorytmu](#_teordnwz51s6) 17

[4.3 LBP Extended](#_t6x4uls4sszf) 18

[4.4 Porównywanie histogramów](#_h4n37skz4v6o) 18

[4.5 Jednorodność](#_78m0lu5fqxau) 19

[4.6 LBP i rozpoznawanie twarzy](#_g1z5k848toz0) 20

[4.7 Zależność efektywności wykrywania obrazu od kolejnych parametrów](#_iwc6fbn6s763) 21

[4.7.1 Druga próba](#_dxfze0nmcj) 30

[**5. Haar**](#_luk66uimh021) **33**

[5.1 Wstęp](#_53ofwevg9p5k) 33

[5.2 Detekcja twarzy przy użyciu modelu kaskadowego](#_vbr8lzzci3ex) 34

[5.3 Ocena wad i zalet](#_6jp3l9z41wc9) 36

[**6. Porównanie algorytmów LBP i Haar**](#_smxajo7e21va) **37**

# 

# 

# 

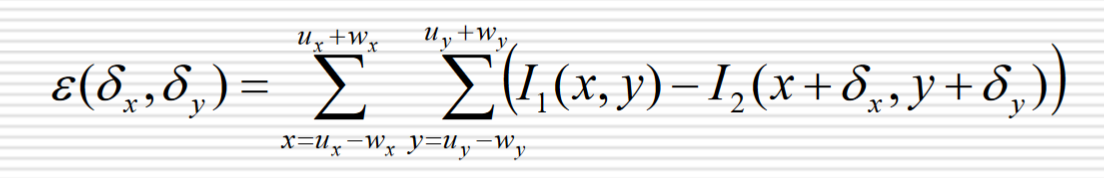
# 

# 

# **1. Optical Flow**

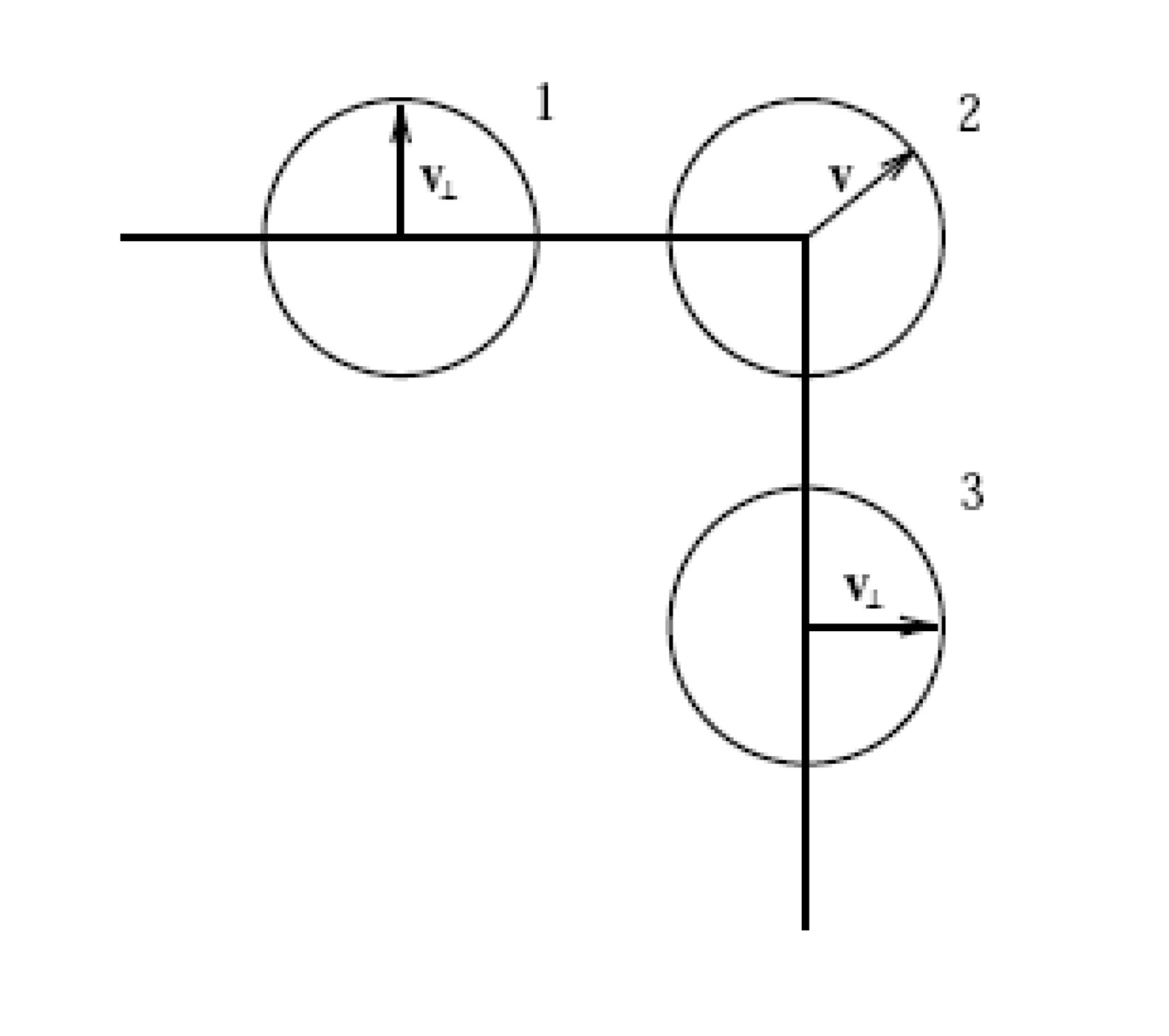
## **1.1 Wstęp**

Przepływ optyczny (ang. optical ﬂow) jest polem wektorowym zawierającym informacje pozwalające przekształcić obraz pierwszy w obraz drugi zgodnie z kierunkiem przepływu. Przepływ optyczny nie jest ściśle zdeterminowany, więc istnieje wiele pól przepływu pozwalających uzyskać obraz drugi z obrazu pierwszego.



## **1.2 Problemy estymacji przepływu optycznego**

* Znalezienie wartości prędkości optycznej w najlepszym przypadku odzwierciedla jedynie składową ruchu obiektu prostopadłą do płaszczyzny obrazu. Poza tym, znalezienie wektora przepływu nie gwarantuje poprawnego wykrycia wektora ruchu. Może się zdarzyć, że w wyniku ruchu obrotowego, nagłych zmian oświetlenia sceny lub innych nieprzewidzianych zjawisk wektor zostanie wskazany nieprawidłowo.
* Problem szczelinowy (ang. aperture problem). Zjawisko występuje, gdy lokalne zmiany jasności obrazu mają charakter jednowymiarowy, wtedy wyznaczenie pełnego wektora prędkości jest niemożliwe. Doskonale to obrazuje rysunek poniżej, gdzie w szczelinach 1 i 3 możemy zaobserwować prędkości optyczne jedynie w kierunkach normalnych do krawędzi kwadratu. Przez szczelinę 2 umieszczoną w narożniku możemy zmierzyć pełną prędkość.



* Brak teksturowania lub przezroczystość obiektu może prowadzić do błędnego wykrycia ruchu, lub tylko częściowego – na konturach, w miejscu pojawiania się i znikania obiektu na obrazie. W ekstremalnym wypadku możemy wyobrazić sobie obracającą się kulę o jednolitym kolorze.
* Zmienne oświetlenie może doprowadzić do wykrycia pozornego ruchu niezależnie od zastosowanej metody. Można w tym wypadku kompensować oświetlenie, jeśli kamera jest statyczna lub posługiwać się kolorem obrazu w dziedzinie HSV, co poprawia odczyt.
* Specyficzne teksturowanie obiektu może być problematyczne. Jeśli wyobrazimy sobie obracający się walec z naniesionymi spiralnie pasami i oś obrotu będzie pionowa, to zwrot wektora prędkości pozornej (obserwowanej) będzie równoległy z osią pionową.
* Zachodzenie obiektów na siebie, powoduje znikanie i pojawianie się fragmentów obrazu lub niespodziewane zmiany jasności. W tym przypadku wskazanie wektorów prędkości może być kłopotliwe.

## **1.3 Metody wykrywania ruchu**

Wykrywanie ruchu można realizować na wiele sposobów. Cechą wspólną wszystkich metod

jest wykrywanie pola przepływu optycznego. Generalnie metody można podzielić na gradientowe, częstotliwościowe, korelacyjne. Choć różnią się one podejściem, można wydzielić podstawowe etapy estymacji:

* Filtrowanie oraz wygładzanie wstępne filtrami dolno- i pasmowo- przepustowymi w celu wyodrębnienia interesującego sygnału – zwiększenia odstępu sygnału od szumu SNR (ang. Signal to Noise Ratio),
* Uzyskanie podstawowych cech obrazu, t.j. pochodne lub powierzchnie korelacji,
* Integracji uzyskanych cech i wartości w celu stworzenia dwuwymiarowego obrazu.

### **1.3.1 Metody gradientowe**

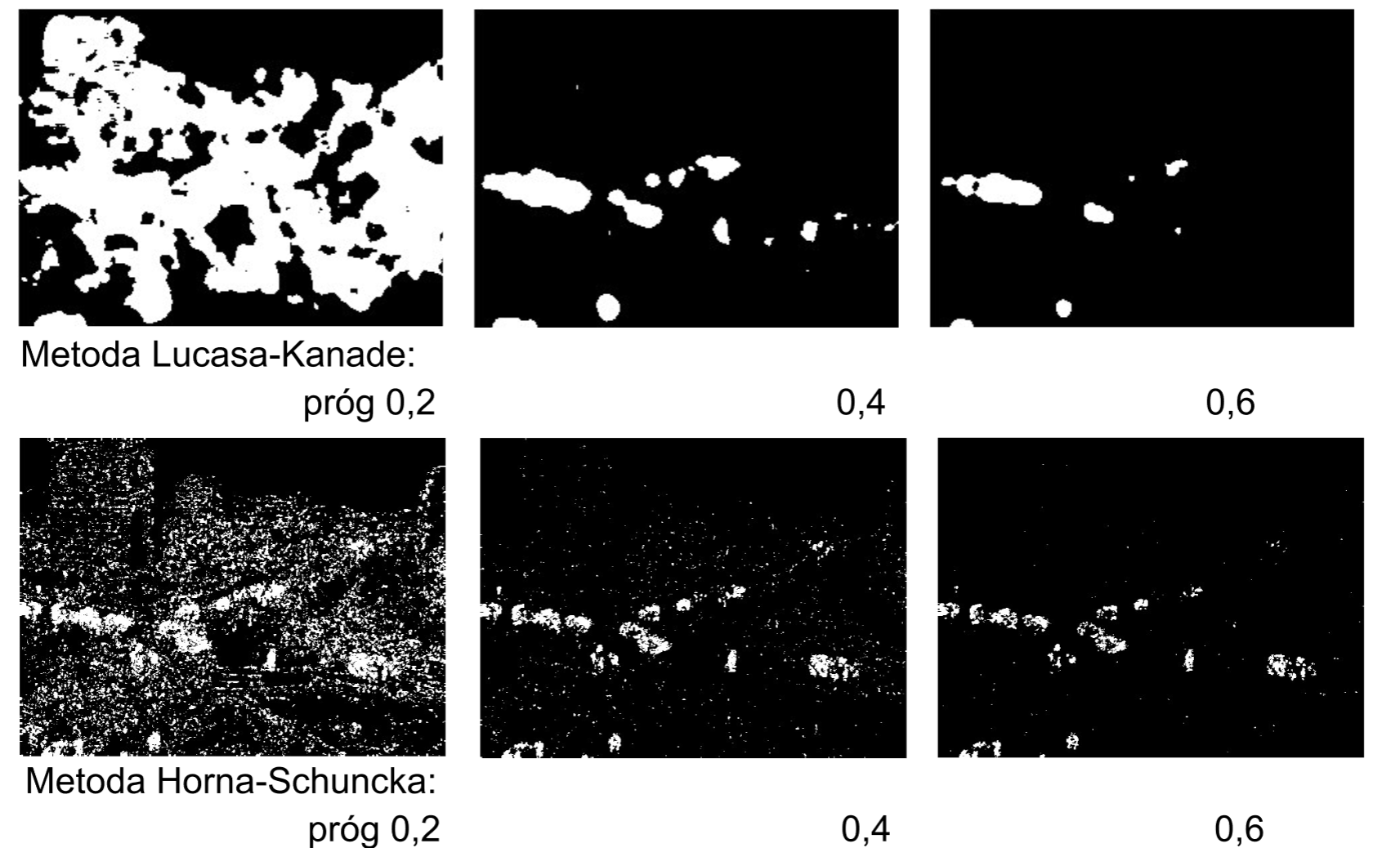
Opiera się na analizie pochodnych intensywności obrazu, do których reprezentacji są używane pochodne czasowe i przestrzenne. Problem związany z zastosowaniem tej metody dotyczy niejednoznaczności nowego położenia poszukiwanego punktu.

Przykładami użycia metody gradientowej w Optical flow są metoda lokalna Lucasa-Kanade i metoda globalna Horna-Schuncka.

Metoda Lucasa-Kanade opiera się na założeniu, że jasność punktu obrazu jest stała w czasie oraz ruch jest stosunkowo mały w następujących po sobie klatkach. Zaletami użycia tej metody jest możliwość szybkiego porównywania klatek i szybkie obliczenia. Główną wadą jest występowanie błędów na krawędziach poruszającego się obiektu. Metoda działania algorytmu polega na wybraniu pola wokół danego punktu, tak że wszystkie punkty mają ten sam ruch. Dla każdego z punktu jest obliczana ich wartość gradientowa.

W celu wyznaczenia wektora przepływu optycznego używa się iteracyjnej piramidalnej wersji tego algorytmu. Pozwala to na zniwelowaniu problemu gubienia większego ruchu. To rozwiązanie polega na badaniu obrazu z różną rozdzielczością, zaczynając od obrazu o najniższej liczbie detali (najwyższy poziom piramidy), a kończąc na obrazie o największej liczbie szczegółów.

Metoda globalna Horna-Schuncka opiera się na wyznaczeniu wektora przesunięcia na podstawie przemieszczenia się wybranych obszarów obrazu. Ta metoda zakłada, że jasność danego punktu należącego do poruszającego się obszaru jest stała oraz sąsiadujące ze sobą punkty należące do jednego ruchomego obiektu mają zbliżone do siebie prędkości. W związku z czym nieciągłości pola ruchu występują jedynie w obszarach, gdzie obiekty zachodzą na siebie. Obliczanie pola przepływu odbywa się poprzez minimalizację funkcjonału zawierającego w sobie oba te założenia.



Przykłady obrazów po binaryzacji modułu optical flow, który został obliczony za pomocą różnych metod gradientowych.

### **1.3.2 Metody częstotliwościowe**

Jest związana z filtrami operującymi w dziedzinie częstotliwości. Opiera się na działaniu algorytmu DFT (Dyskretna Transformata Fouriera) w celu detekcji przesunięcia przestrzennego. Są one uznawane za najbardziej efektowne i zbliżone do modelu ludzkiego sposobu spostrzegania ruchów. Jednakże są one najbardziej złożone obliczeniowo oraz ciężkie do zaimplementowania ze względu na skomplikowany model matematyczny.

Dzielą się one na dwie główne podgrupy - filtry częstotliwościowe i fazowe. Filtry częstotliwościowe opierają się na badaniu rozkładu energii w przestrzeni fourierowskiej. Natomiast filtry fazowe polegają na definowaniu prędkości w kategori zmian fazy sygnału.

### **1.3.3 Metody korelacyjne**

Opierają się na podziale obrazu na bloki pikseli, które są porównywane w celu wykrycia korelacji. Jest to dopasowanie blokowe, która opiera się na algorytmie Block-matching algorithm. Algorytm dzieli bieżący obraz na klatki rozmiaru MxM, które są porównywane z odpowiednimi klatkami i ich otoczeniem w celu stworzenia wektora, opisującego ruch jednej klatki.

Po wyznaczeniu otoczenia poszukiwania wokół wybranego bloku, które składa się z p pixeli z każdej z 4 stron, liczy się funkcje kosztu. Jedna z najczęściej używanych jest funkcja MAD ( Mean Absolute Difference) czy MSE ( Mean Squared Error). Blok, który ma najmniejszą wartość funkcji kosztu jest właśnie poszukiwanym blokiem, który wyznaczy ruch danej klatki obrazu.

Zaletami korzystania z metody korelacyjnej jest możliwość wyboru ilości i dowolnego rozmieszczenia wektorów ruchu.

# **2. OpenCV**

## **2.1 Wstęp**

OpenCV (ang. Open Source Computer Vision Library) - to biblioteka funkcji, przeznaczonych głównie do przetwarzania obrazu w czasie rzeczywistym. Początkowo opracowana przez firmę Intel, później wspierana przez Willow Garage, a teraz prowadzona przez Itseez. Biblioteka jest multiplatformowa i dopuszczona do wolnego użytku na licencji BSD, dlatego jest bezpłatna do celów akademickich, jak i komercyjnych. OpenCV wspiera frameworki Deep Learning, TensorFlow, Torch/PyTorch i Caffe.

Biblioteka posiada interfejsy do języków: C++, C, Java i Python oraz obsługuje systemy Windows, Linux, Mac OS, iOS i Android. Napisana jest w języku C/C++, co pozwala korzystać z przetwarzania wielordzeniowego.

## **2.2 Historia**

Oficjalnie uruchomiony w 1999 roku projekt OpenCV był początkowo inicjatywą Intel Research w celu przyspieszenia aplikacji wymagających dużej ilości CPU.

Głównymi jego celami były:

* Zaawansowane badania wizualne, które zapewnią nie tylko otwarty, ale także zoptymalizowany kod, dla podstawowej infrastruktury wizualnej.
* Propagowanie wiedzy na temat rozpoznawania obrazu poprzez udostępnianie wspólnej infrastruktury, którą programiści mogliby wykorzystać, dzięki czemu kod byłby bardziej czytelny i przenośny.
* Zaawansowane aplikacje komercyjne oparte na przetwarzaniu obrazu, dzięki udostępnieniu bezpłatnego, przenośnego kodu, zoptymalizowanego pod kątem wydajności.

Pierwsza wersja alfa OpenCv została udostępniona opinii publicznej w 2000 roku podczas konferencji IEEE Conference on Computer Vision and Pattern Recognition. Kolejne pięć wersji beta powstały w latach 2001 - 2005. Wersja 1.0 została wydana w 2006 roku. Wersja 1.1 "pre-release" w 2008 r.

W październiku 2009 roku wydano drugą wersję biblioteki OpenCV, która zawierała w sobie duże zmiany dotyczące interfejsu w C++, które miały na celu uproszczenia go, usprawnienie i dodanie nowych funkcji.

W sierpniu 2012 roku wsparcie dla OpenCV zostało przejęte przez organizację non-profit OpenCV.org.

## **2.3 Aplikacje**

Obszary zastosowań OpenCV obejmują:

* Zestawy narzędzi 2D i 3D
* Oszacowanie egomotyczne
* System rozpoznawania twarzy
* Rozpoznawanie gestów
* Interakcja człowiek-komputer
* Robotyka mobilna
* Zrozumienie ruchu
* Identyfikacja obiektu
* Segmentacja i rozpoznawanie
* Stereofis stereofoniczny wizji: percepcja głębokości z dwóch kamer
* Struktura z ruchu
* Rozszerzona rzeczywistość

Aby wspierać niektóre z powyższych obszarów, OpenCV ma w sobie statystyczną bibliotekę nauczania maszynowego, która zawiera:

* Nauka drzewa decyzyjnego
* Gradient pobudzający drzewa
* Algorytm maksymalizacji oczekiwania k-najbliższy algorytm sąsiedztwa
* Klasyfikator Naive Bayes
* Sztuczne sieci neuronowe
* Losowy las
* Maszyna wektora nośnego
* Głębokie sieci neuronowe

## **2.4 Języki programowania**

OpenCV jest napisane w języku C++, który jest równocześnie jego głównym interfejsem, lecz wciąż zachowuje mniej wszechstronny, ale szeroki interfejs w języku C. Istnieją powiązania z językiem Python, Java i MATLAB/OCTAVE. API tych interfejsów może być znalezione w dokumentacji online. Wrappery do innych języków takich jak C#, Perl, Ch, Haskell i Ruby zostały opracowane w celu zachęcenia szerszej publiczności.

Wszystkie nowe algorytmy oraz usprawnienia w OpenCV są teraz rozwijane w interfejsie C++.

# **3. EmguCV**

## **3.1 Wstęp**

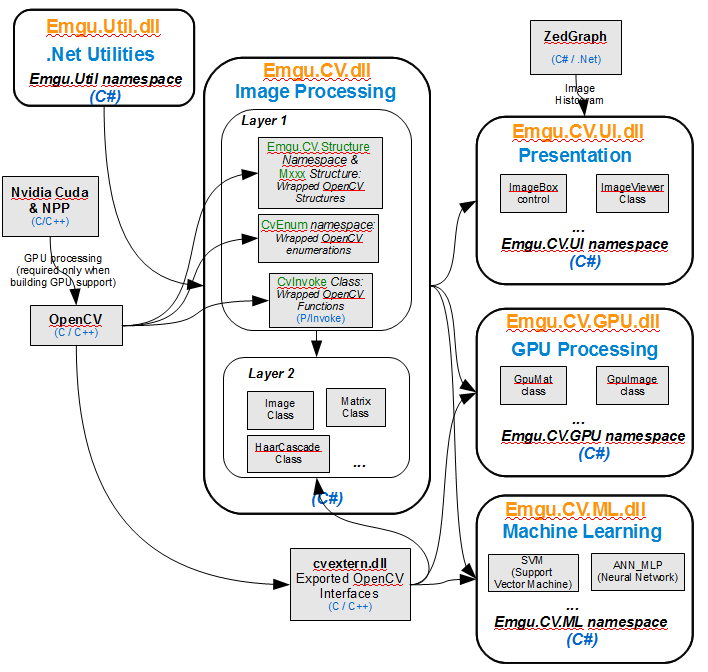
Emgu CV to platforma cross-platform .Net do biblioteki przetwarzania obrazu OpenCV. Dzięki niej możemy przetwarzać i analizować statyczne i ruchome obrazy oraz inne sygnały cyfrowe.

Korzystając z takiego wrappera jak Emgu CV możliwe jest wykorzystanie bardzo zoptymalizowanej biblioteki bezpośrednio z C#, bez konieczności schodzenia do niskopoziomowej implementacji w C.

Do przykładowych zastosowań Emgu CV należą aplikacje do sprawdzania jakości produktów w fabrykach, medyczne przetwarzanie obrazów, kalibracje kamer i zastosowania w szeroko pojętej robotyce. Poza przetwarzaniem obrazu wrapper również pozwala korzystać z biblioteki nauczania maszynowego pochodzącej z OpenCV. (“Emgu CV Essentials”, Shin Shi, 2013)  
Pakiet może być kompilowany przez Visual Studio, Xamarin Studio i Unity, może działać na systemach Windows, Linux, Mac OS X, iOS, Android i Windows Phone.

## **3.2 Zalety Emgu CV**

* Emgu CV jest w całości napisany w C#. Korzyścią jest to, że można go skompilować w Mono, a zatem może działać na dowolnych platformach Mono, w tym iOS, Android, Windows Phone, Mac OS X i Linux.
* Image Class z ogólnym kolorem i głębokością
* Automatyczne gromadzenie śmieci.
* XML Serializable Image
* Dokumentacja XML i pomoc techniczna



## **3.3 Budowa Emgu CV**

Emgu CV ma dwie główne warstwy, jak pokazano na schemacie powyżej.

* Warstwa podstawowa (layer 1) zawiera odwzorowania funkcji, struktur i wyliczeń, które bezpośrednio odzwierciedlają te w OpenCV
* Druga warstwa (layer 2) zawiera klasy łączące zalety świata .NET

## **3.4 Licencja EmguCV**

EmguCV oparta jest na licencji GPLv3, co znaczy, że produkt oparty na niej musi być open-source i darmowy. Możliwe jest uzyskanie licencji komercyjnej po umieszczeniu opłaty. (“Emgu CV Essentials”, Shin Shi, 2013).

## **3.5 Wydajność**

Według (“Emgu CV Essentials”, Shin Shi, 2013) na podstawie prostych operacji na obrazach czarno-białych Emgu CV zajmuje trzecie miejsce pod względem szybkości przetwarzania (po bibliotece OpenCV napisanej w C oraz trudnej metody wywoływania jej za pomocą P/Invoke w C#).

## **3.6 Rozpoznawanie krawędzi i konturów**

W EmguCV możliwe jest korzystanie z detektora Canny Edge w celu rozpoznawania krawędzi. Poniższy kod z książki (“Emgu CV Essentials”, Shin Shi, 2013) pozwoli na stworzenie prostego czarno-białego outline’u obrazu.

Image<Gray, Byte> cannyGray = gray1.Canny(120, 180);

pictureBox2.Image = cannyGray.ToBitmap();

Po wykryciu krawędzi w formie osobnych pixeli możliwe jest też ich połączenie w kontury. Odpowiada za to funkcja FindCountours(). Wynik tej funkcji może zostać przetransformatowany w łańcuch Freemana, czyli sekwencji kolejnych kroków wartości.   
Dodatkową funkcjonalnością EmguCV jest możliwość konwertowania konturów do polygonów.

## **3.7 Rozpoznawanie twarzy**

Proces rozpoznawania twarzy składa się z Preprocessingu obrazu lub wideo, detekcji twarzy, rozpoznawania cech charakterystycznych oraz dopasowywania ich do danych z bazy.

Ważną częścią rozpoznawania twarzy są algorytmy nauczania maszynowego. Są one w stanie na podstawie obrazów treningowych nauczyć się rozróżniać pomiędzy obrazami zawierającymi twarze a inne obiekty.

## **3.8 Algorytmy nauczania maszynowego udostępniane przez EmguCV**

* ANN\_MLP
* Boost
* DTree
* EM
* GBTrees
* KNearest
* NormalBayesClassifier
* HaarCascade
* CascadeClassifer
* Facerecognizer

# **4. LBP**

## **4.1 Wstęp**

Algorytm LBP służy do opisu lokalnych właściwości tekstury. Przekształca on wejściowy, czarno - biały, obraz w nowy, przyporządkowując każdemu pikselowi i jego otoczeniu pewną liczbę, wyznaczoną za pomocą określonej metody. W zależności od użytej metody wyróżnia się podstawowy LBP oraz warianty Extended LBP (inaczej zwane Circular LBP).

W wersji podstawowej obliczenia dla każdego piksela bazują na jego naturalnym otoczeniu 3x3. Ogólniejsze algorytmy posługują się parametrami r oraz p, oznaczającymi odpowiednio promień i liczbę punktów. Zamiast sąsiadów danego piksela, wyznaczają one p równomiernie rozłożonych punktów na okręgu o promieniu r o środku w danym pikselu, a następnie interpolują ich kolory. Po wyznaczeniu otoczenia, algorytmy przyporządkowują jego punktom wartości binarne, z których powstaje jedna liczba zapisana dwójkowo. W ten sposób uzyskuje się wartość piksela wynikowego obrazu. Zaletą rozszerzonej wersji jest to, że lepiej koduje cechy niezmiennicze na skale czy obroty.

## **4.2 Działanie algorytmu**

Algorytm liczy lokalne przedstawienie tekstury. Najpierw obraz zostaje przekształcony do skali szarości. Następnie dla każdego piksela zostanie wybrany obszar o promieniu r wokół centralnego piksela. Każdemu pikselowi jest przypisywana liczba dziesiętna po binaryzacji liczby otrzymanej z otoczenia tego piksela. Wartości są przetrzymywane w macierzy. Na ich podstawie tworzony jest histogram. Wartości tak wyliczone są konkatenowane do postaci wektora. Wektory histogramów są porównywane z wektorami obrazów w bazie.

## **4.3 LBP Extended**

Dzięki wykorzystaniu LBP Extended możliwe jest zakodowanie większej ilości wzorców, które będą się różnić poziomem wygładzenia i dokładności detali. Im większy promień r, tym bardziej jest wygładzony obraz LBP.

* ror
* Uniform
* nri\_Uniform
* var

## **4.4 Porównywanie histogramów**

Podobieństwo (lub niepodobieństwo) histogramów uzyskanych ze zdjęć po zastosowaniu LBP można badać różnymi miarami statystycznymi. W OpenCV istnieje funkcja

*double compareHist(InputArray H1, InputArray H2, int method);*

spełniająca to zadanie. Uzyskana przez nią metryka jest znajdowana na podstawie statystycznej metody *int method,* która może być jedną z poniższych:

* Odległość (*HISTCMP\_CHISQR* w openCV)  
  Jest to odległość oparta na teście , którą oblicza się ze wzoru

* Korelacja (*HISTCMP\_CORREL* w openCV)  
  Jest to zwykła korelacja między histogramami:

gdzie *cov* oznacza kowariancję, a *std* odchylenie standardowe:

a oznacza średnią arytmetyczną.

* Odległość Hellingera (*HISTCMP\_HELLINGER* w openCV)  
  Wyraża się ona wzorem:

* Alternatywna odległość (*HISTCMP\_CHISQR\_ALT* w openCV)  
  Wyraża się wzorem

* Przekrój (*HISTCMP\_INTERSECT* w openCV)  
  Zwraca sumę minimów odpowiadających wartości:

* Dywergencja Kullbacka-Leiblera (*HISTCMP\_KL\_DIV* w openCV)  
  Mierzy ona względną entropię histogramów . Im jest wyższa, tym większą informacje daje od . Z uwagi na to, że nie jest ona symetryczna, nie można jej traktować jako metryki statystycznej. Dywergencję Kullbacka-Leiblera można obliczyć ze wzoru

## **4.5 Jednorodność**

Natomiast liczba punktów p jest ważna ze względu na jednorodność. Jednorodność decyduje o opisie rotacji i poziomach wariacji w skali szarości. Funkcja LBP jest uznawana za jednorodną, gdy liczba binarna, która jest przypisywana każdemu pikselowi, ma co najwyżej 2 zmiany 0-1 lub 1-0. Im więcej punktów p jest wybieranych, tym gęstość ostatecznego histogramu wzrasta. Dla każdego punktu p istnieje p + 1 jednorodnych tekstur.

## **4.6 LBP i rozpoznawanie twarzy**

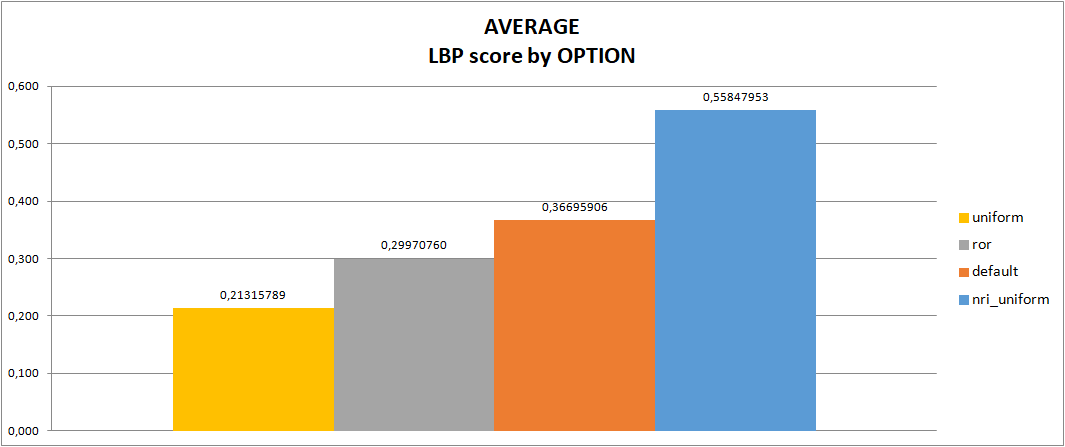
Metodę LBP wykorzystuje się do rozpoznawania twarzy ze względu na możliwość porównywania histogramów struktur dla danego obrazu. Metoda ta wykorzystuje różnice w odległości między histogramami i następnie je porównuje.  
  


Poprzednie zdjęcia należą do kolekcji Train, na której **nasz program** uczył się rozpoznawać osobę o indeksie 54. Nasze doświadczenia zostały przeprowadzone na bazie około **650 zdjęć** przygotowanych dla **55 osób**. Zostały one podzielone w stosunku około 5:1 na zdjęcia Train (służące do nauki rozpoznawania danej osoby), a Test (służące do testowania skuteczności algorytmu).

Osobę “54” i jej powyższe zdjęcia testowe będziemy śledzić na przestrzeni różnych opcji i metod algorytmu LBP. Pomoże ona nam wizualizować i prześledzić jak różne parametry wpływają na rozpoznawanie twarzy.

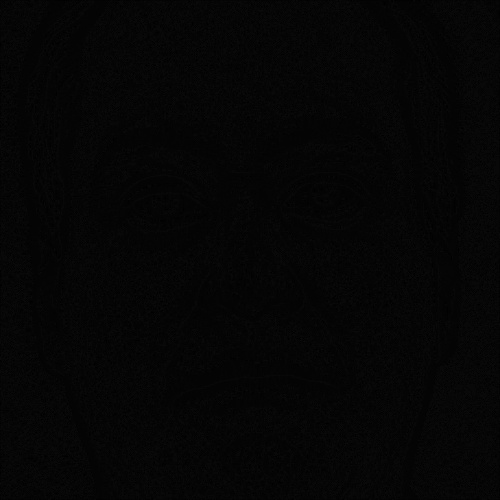
## **4.7 Zależność efektywności wykrywania obrazu od kolejnych parametrów**tabela.png

Powyższa tabela przedstawia wyniki pierwszego z dwóch robionych przez nas testów. Można z niej wywnioskować zależność pomiędzy rodzajem algorytmu LBP (Option), metodą porównywania histogramów (Method), szybkością jego działania (Speed) i efektywnością rozpoznawania twarzy (Result).

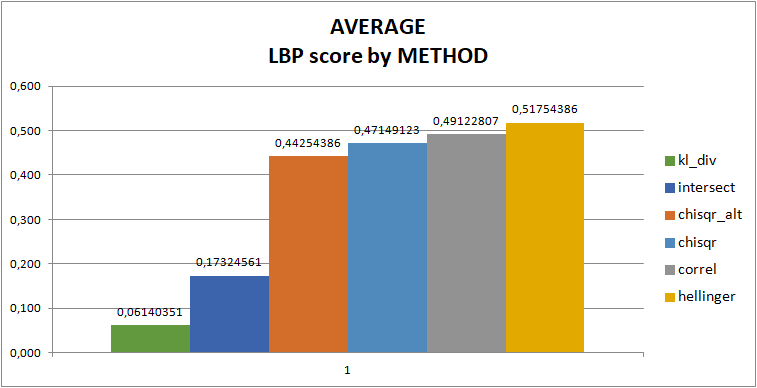
* Średni wynik względem LBP Option:

Jak widać niezależnie od prędkości czy metody najlepsze wyniki osiągane są przy pomocy opcji nri\_uniform (ponad połowa trafień nawet przy niesprzyjających innych parametrach). Najgorzej zaś wypadła opcja uniform - średnio około 21% poprawnych wyników.

Poniżej znajdują przykładowe zdjęcia osoby ***"54"*** w trakcie działania różnych opcji algorytmu LBP. Wszystkie inne parametry zostały zachowane identyczne *(promień 1, metoda porównywania histogramów chisqr\_alt, szybkie obliczenia, rozmiar 500 x 500px*).

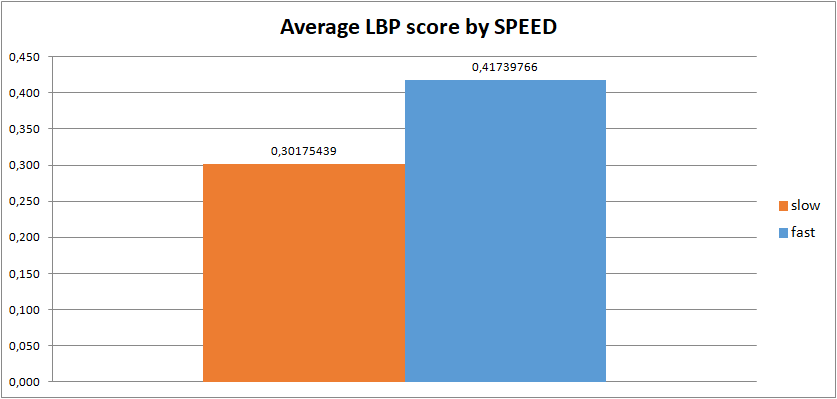
Po lewej ilustracja opcji nri\_uniform, osiągająca statystycznie najlepsze wyniki w trakcie naszych próbek. Obok opcja domyślna default.

Opcje ror i uniform. **Uniform dla wybranej osoby *“54”* jak i dla całej naszej próbki najczęściej ma problemy z rozpoznawaniem twarzy.** W tym wypadku pogląd jest prawie całkowicie bezużyteczny. Dla opcji uniform będzie to ciągłe zjawisko, co zaobserwujemy w późniejszych częściach raportu.

* Średni wynik względem metody porównywania histogramów (Method):

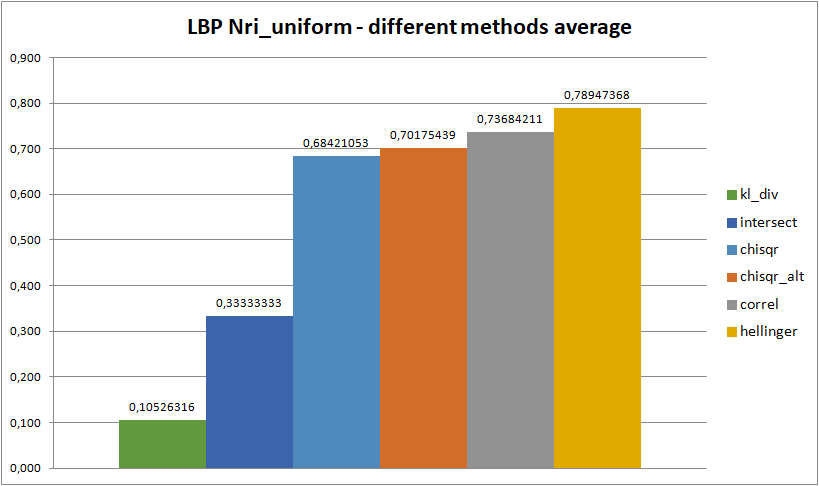
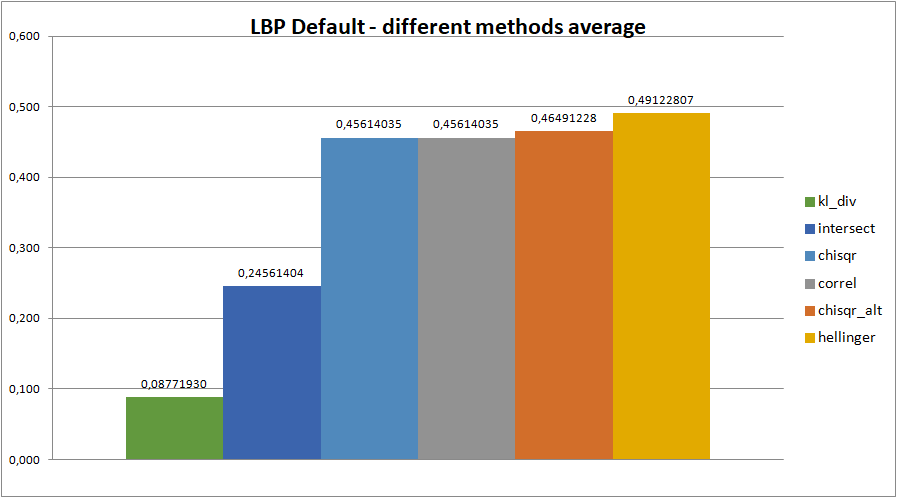
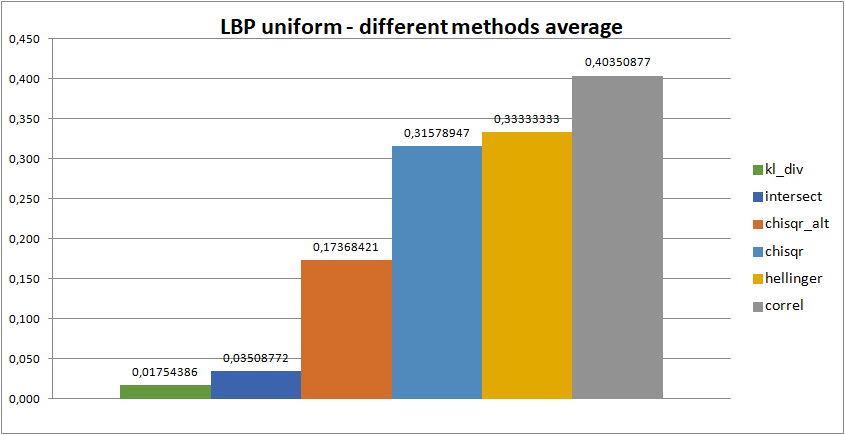
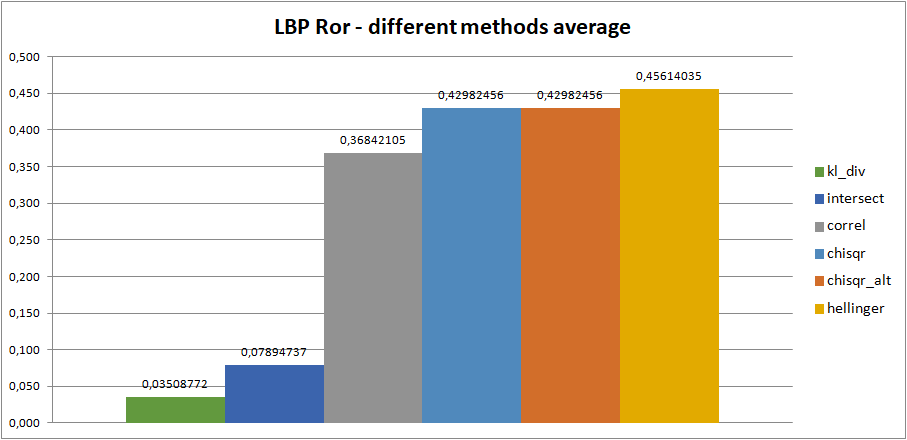
Można zaobserwować, że metody kl\_div oraz intersect osiągają znacząco gorsze wyniki od pozostałych metod niezależnie od innych parametrów. Pozostałe metody uzyskały średni wynik w przedziale 44% - 52%.

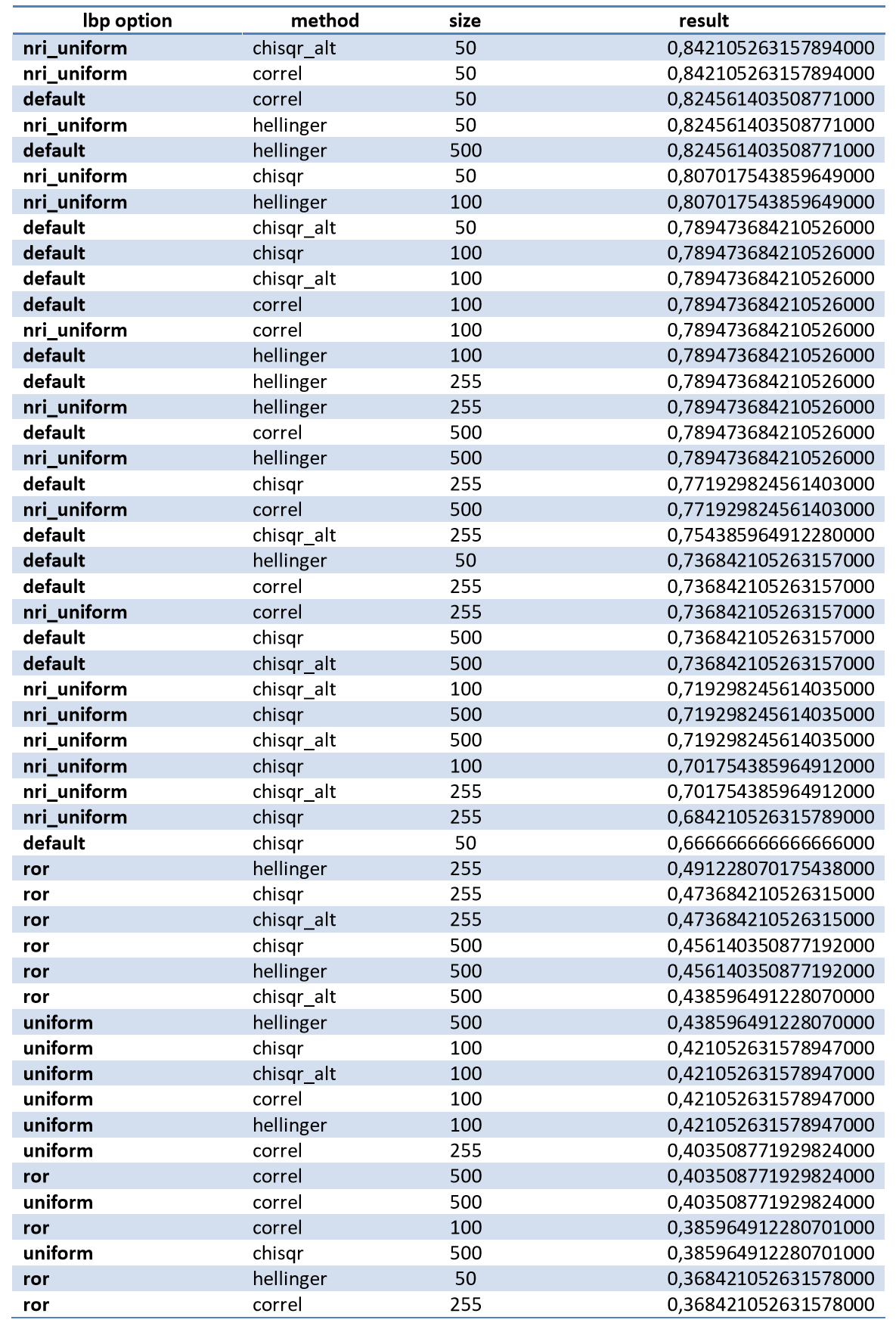
* Średni wynik względem prędkości (Speed):

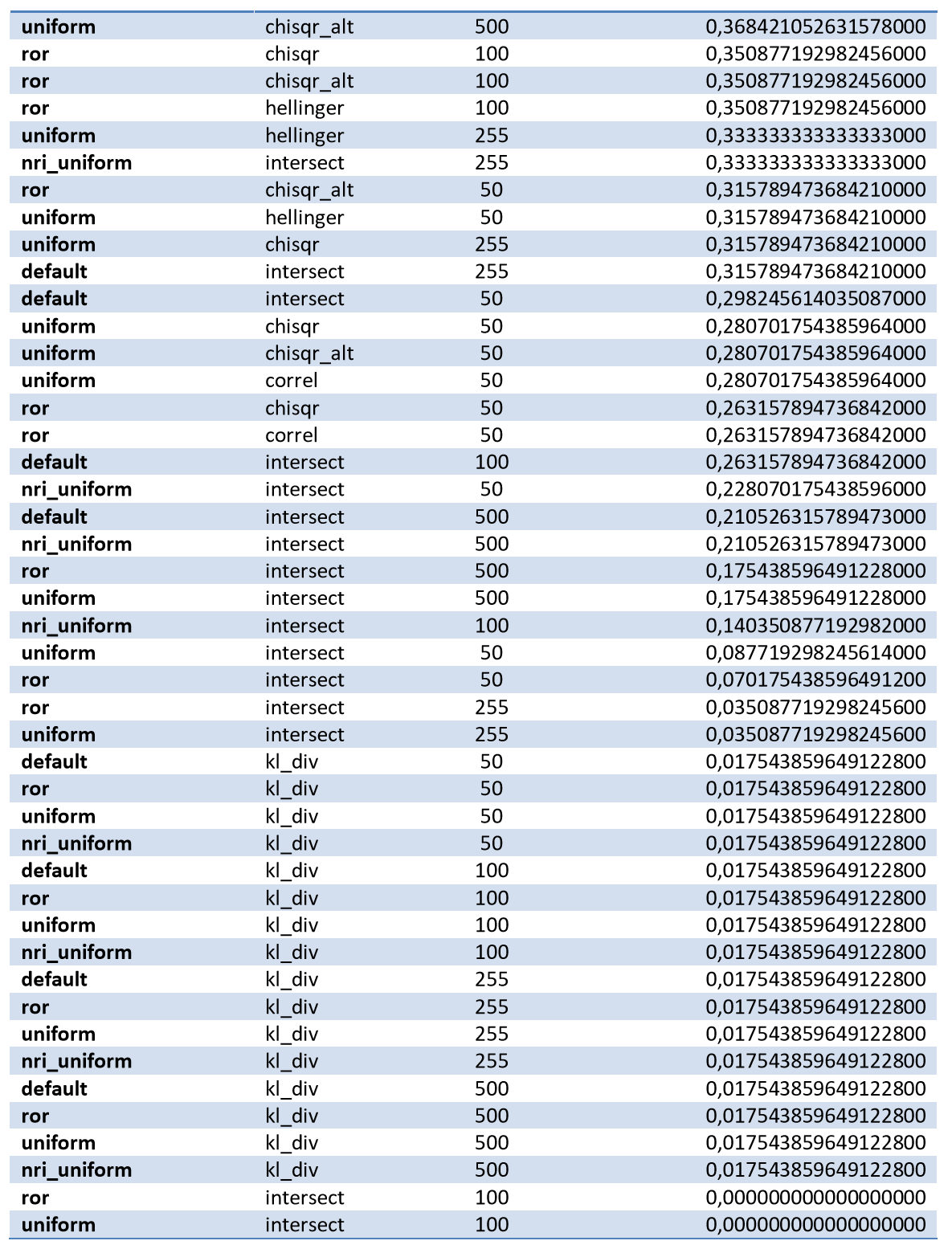
****

Lepsze wyniki osiągane były przy większej prędkości wykonywania algorytmu.

Biorąc pod uwagę te wnioski warto przyjrzeć się dokładniejszym korelacjom Opcji i Metod. Poniżej cztery wykresy opisują dobrze zależności pomiędzy nimi - można zauważyć, że najkorzystniej wypada połączenie nri\_uniform - Hellinger, zostawiając daleko w tyle wyniki pozostałych Opcji i Metod.



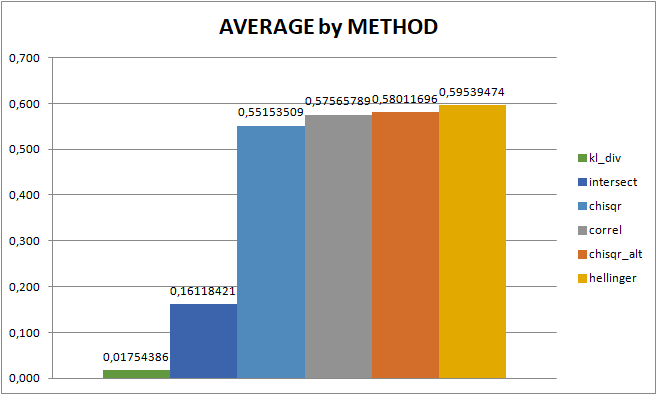
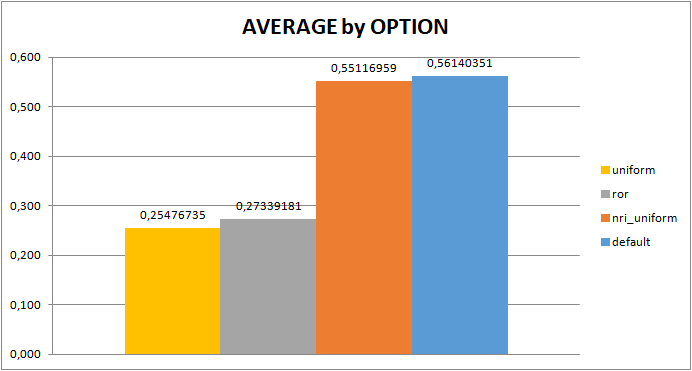


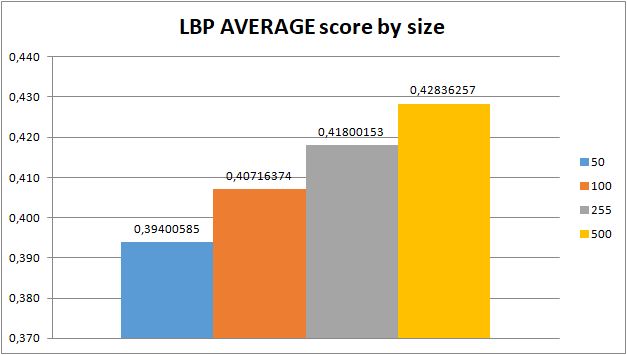


### **4.7.1 Druga próba**

Wyniki pierwszego eksperymentu pokazały, że wolna prędkość obliczeń dawała znacząco gorsze wyniki, niż opcja szybka. W drugiej serii testów pominęliśmy więc parametr prędkości wykonywania algorytmu - wszystkie działania wykonane były wykonane z opcją szybką. Pozwoliło to też znacząco przyspieszyć czas obliczeń i ograniczyć go do ok. 30 minut.  
  
W drugiej próbie został również dodany nowy parametr, będący rozmiarem obrazka. Zdjęcia twarzy przed wykonaniem algorytmów będą zmniejszane do ustalonych z góry rozmiarów. Pierwsza próba opierała się na stałych rozmiar 255 na 255 pikseli.  
Do drugiej dodaliśmy opcję 50, 100 oraz 500 pikseli.   
Rozpoznawanie większych obrazów zajmuje więcej czasu, ale powinno dać lepszą skuteczność dla wszystkich pozostałych parametrów.   
Ciekawiło nas, czy odwrotne stwierdzenie dla mniejszych obrazów również będzie prawdziwe.

Wyniki względem Opcji i Metody były zbliżone do danych uzyskanych przy poprzednim podejściu, co ilustrują wykresy poniżej.   
**Opcja Default okazała się więc być najdokładniejsza z ograniczeniem szybkości fast oraz modyfikując rozmiary obrazków.**





Na powyższym wykresie wyraźnie widać, że rozmiar obrazka wpływa na wynik działania algorytmu LBP. Skuteczność rozpoznawania twarzy wzrasta, gdy użyte zostaje zdjęcie o większym rozmiarze. Różnica pomiędzy najgorszym a najlepszym wynikiem wynosi zaledwie 3,435672%, dlatego też ta zmienna nie została uznana przez nas za bardzo znaczącą.



Podgląd działania algorytmów w zależności od rozmiaru dla parametrów ror, promień 1, fast.

Wniosek: *Przy przetwarzaniu znaczącej ilości zdjęć słuszne więc wydaje się skalowanie ich do najmniejszych rozmiarów w celu przyspieszenia obliczeń.*

### **4.7.2 Trzecia i dalsze próby**

Ze względu na dużą złożoność obliczeniową i czasową eksperymentów dla promieni nie udało nam się przeprowadzić wielu doświadczeń badających zależność pomiędzy nimi i skutecznością.  
Wiadome jest, że dla promienia 0 obliczenia nie dochodzą do skutku.  
Obliczenia dla promienia równego dwa zajęły nam około 2 godziny, po ograniczeniu naszych badań tylko do szybkiej metody (patrz [Druga próba](#_dxfze0nmcj)).   
Wyniki dla eksperymentów, w tym zdjęcia śledzące zmiany dla osoby ‘***54’*** znajdują się na repozytorium:

<https://github.com/MikolajBalcerek/FaceDetection/tree/master/Wyniki%20LBP>

W ramach upływu czasu wyniki dalszych badań będą publikowane w tym miejscu.

### **4.7.3 Posłowie**

LBP było jedynym algorytmem, który udało nam się dokładnie przeanalizować i zbadać wpływ jego parametrów na wyniki. Naszą implementację wykonaną w **Pythonie 3** za pomocą biblioteki **OpenCV, numpy, skimage oraz scipy** można podejrzeć na<https://github.com/MikolajBalcerek/FaceDetection>, gdzie wraz z kodem znajdują się wyniki naszych eksperymentów.   
  
Program umożliwia dalsze prowadzenie badań nad wpływem parametrów na wyniki zmieniając zmienne w pliku **lbp.py**:  
*# parametry dla calc\_lbps(X)*

*#TUTAJ MOŻNA ZMIENIAĆ PARAMETRY*

radius\_max\_range **=** 4; *#maksymalna wartość dla radius którą osiągnie w testowaniu*

radius\_min\_range **=** 3; *# minimalna wartość dla radius od której się zacznie sprawdzanie możliwości*

radius\_increment **=** 1; *#inkrement po którym ma się zmieniać radius*

listSizes **=** [50, 100, 255, 500];

options **=** [**"default"**, **"ror"**, **"uniform"**, **"nri\_uniform"**]

methods **=** [**"chisqr"**, **"chisqr\_alt"**, **"correl"**, **"hellinger"**, **"kl\_div"**, **"intersect"**]

*# metody porównywania histogramów*

speeds **=** [**'fast'**] *#, 'slow']*

W pliku **load\_faces.py** poza pomocniczymi funkcjami znajduje się również program testujący działanie ***CascadeClassifier*** i ***detect Multiscale*** na podstawie bazy ok. **700 zdjęć**. Mimo manipulacjami ich parametrami nie udało nam się zmienić ich skuteczności i wynosiła ona stałe ~65% w rozpoznawaniu twarzy.

# **5. Haar**

## **5.1 Wstęp**

Funkcje typu Haar są cyfrowymi funkcjami obrazu używanymi do rozpoznawania obiektów. Swoją nazwę zawdzięczają podobieństwu z falami Haar, które zostały użyte w pierwszym detektorze twarzy w czasie rzeczywistym.

Funkcja Haar uwzględnia przyległe regiony prostokątne w określonej lokalizacji w oknie wykrywania, podsumowuje intensywność pikseli w każdym regionie i oblicza różnicę między tymi sumami. Ta różnica jest następnie wykorzystywana do klasyfikacji podsekcji obrazu. Na przykład powiedzmy, że mamy bazę obrazów z ludzkimi twarzami. Powszechnie wiadomo, że ​​wśród wszystkich twarzy obszar oczu jest ciemniejszy niż okolica policzków. Dlatego wspólną cechą Haara do wykrywania twarzy jest zestaw dwóch sąsiadujących prostokątów, które leżą nad okiem i w okolicy policzków. Położenie tych prostokątów jest określone względem okna detekcji, które działa jak ograniczenie do obiektu docelowego (twarz w tym przypadku).

W fazie wykrywania obiektów okno obrazu docelowego przesuwa się nad obrazem wejściowym, a dla każdej podsekcji obrazu obliczana jest funkcja Haar. Różnica ta jest następnie porównywana z wyuczonym progiem, który dzieli nie-obiekty od obiektów. Ponieważ taka pojedyncza funkcja Haar jest słabym klasyfikatorem (jej jakość detekcji jest nieco lepsza niż przypadkowe zgadywanie), wiele obiektów Haar jest niezbędnych do opisania obiektu z wystarczającą dokładnością. W ramach wykrywania obiektów szkieletem Viola-Jones cechy typu Haar są zorganizowane w model klasyfikatora kaskadowego, tworząc silny uczący się klasyfikator.

## **5.2 Detekcja twarzy przy użyciu modelu kaskadowego**

Metoda wykrywania obiektów za pomocą klasyfikatorów kaskadowych opartych na cechach Haar została przedstawiona przez Paula Viola i Michaela Jonesa w ich artykule "Wykrywanie obiektów szybkich przy użyciu wzbogaconej kaskady prostych funkcji" w 2001 roku. Jest to podejście oparte na komputerach maszynowych, gdzie funkcja kaskady jest wyszkolona z wielu pozytywnych i negatywnych obrazów. Następnie jest używany do wykrywania obiektów na innych obrazach.

Początkowo algorytm potrzebuje dużo pozytywnych obrazów (obrazów twarzy) i negatywnych obrazów (obrazów bez twarzy) w celu wyszkolenia klasyfikatora. Potem musimy wyodrębnić z niego funkcje. W tym celu zastosowano cechy Haar przedstawione na rysunku poniżej. Każda cecha jest pojedynczą wartością otrzymaną przez odjęcie sumy pikseli w białym prostokącie od sumy pikseli pod czarnym prostokątem.



Wszystkie możliwe rozmiary i lokalizacje każdego jądra są używane do obliczania wielu funkcji. (Wystarczy wyobrazić sobie, jak dużo obliczeń zajmuje taka operacja. Okno 24x24 daje nawet ponad 160000 funkcji). Dla każdego obliczenia funkcji musimy znaleźć sumę pikseli pod białymi i czarnymi prostokątami.

Aby rozwiązać ten problem, wprowadzono integralne obrazy. Upraszczają one operację obliczania sumy po wszystkich pikselach, do operacji obejmującej tylko cztery piksele. Operacja ta znacząco usprawnia działanie algorytmu.

Spośród wszystkich obliczonych funkcji, większość z nich jest bez znaczenia. Poniższa grafika ilustruje problem. W górnym rzędzie pokazano dwie dobre cechy. Pierwsza cecha skupia się na porównaniu odcienia obszaru oczu (ciemniejszy) i obszaru nosa i policzków (jaśniejszy). Druga wybrana cecha polega na zauważeniu, że oczy są ciemniejsze niż mostek nosa. Ale te same okna, które nakładają się na policzki lub w innym miejscu, są nieistotne. Więc jak wybrać najlepsze funkcje z ponad 160000 funkcji?

Można to osiągnąć dzięki metodzie Adaboost - algorytmowi, dzięki któremu z dużej liczby słabych klasyfikatorów można otrzymać jeden lepszy.



W tym celu stosujemy każdą cechę na wszystkich obrazach szkoleniowych. Dla każdej funkcji algorytm znajduje najlepszy próg, który klasyfikuje twarze do pozytywnych i negatywnych. Wybieramy funkcje z minimalną liczbą błędów, co oznacza, że ​​są to cechy, które najlepiej klasyfikują obrazy zawierające twarz lub pozbawione jej. Każdy obraz ma taką samą wagę na początku. Po każdej klasyfikacji, wagi obrazów błędnie sklasyfikowanych wzrastają. Proces jest kontynuowany dopóki nie zostanie osiągnięta wymagana dokładność, stopień błędu lub wymagana liczba funkcji. Klasyfikator końcowy jest ważoną sumą tych słabych klasyfikatorów. Nazywa się słabym, ponieważ sam nie może klasyfikować obrazu, ale razem z innymi stanowi silny klasyfikator. Dzięki zastosowaniu takiego klasyfikatora, można zredukować liczbę funkcji z ponad 160000 do około 6000. Jest to duży zysk obliczeniowy.

Pomimo znacznego zredukowania obliczeń, zastosowanie 6000 funkcji dla okna 24x24 do obrazu może okazać się nieefektywne. Nie każde okno będzie zawierało twarz.   
W przykładowym obrazku większość obszaru jest inna niż twarz. Lepszym rozwiązaniem będzie zatem wprowadzić metodę sprawdzania, czy okno nie jest obszarem twarzy. Jeśli nie, odrzucamy je bez ponownego przetwarzania.

W tym celu wprowadzono koncepcję kaskady klasyfikatorów. Zamiast stosowania wszystkich 6000 funkcji w oknie, grupuje się funkcje na różnych etapach klasyfikatorów

i stosuje się je etap po etapie. Jeśli okno nie powiedzie się na pierwszym etapie, odrzucamy je. Nie uwzględniamy na nim pozostałych funkcji. Jeśli pierwszy etap się powiedzie, program przechodzi na etap drugi funkcji i proces jest kontynuowany. Okno, które przejdzie wszystkie etapy, jest obszarem twarzy. Początkowe etapy zawierają dość niewielką liczbę cech (1, 10, 25, 40, 50 cech w pierwszych pięciu krokach). Zastosowanie takiego podejścia sprawia, że na pojedynczym oknie zostaje ocenianych średnio 10 z 6000 funkcji.

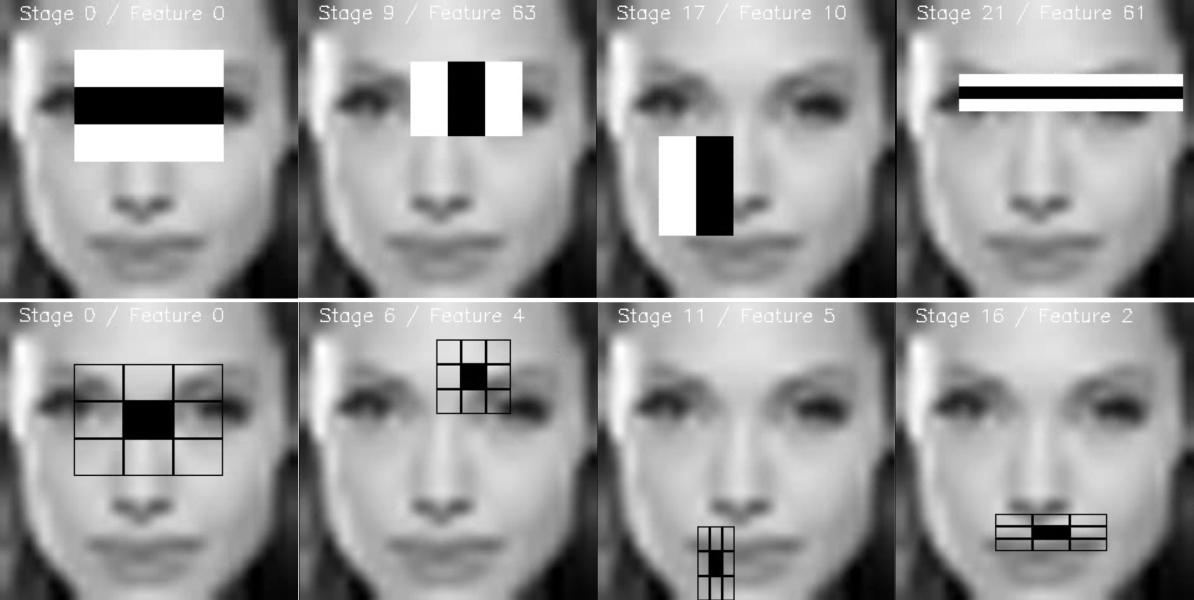
## **5.3 Ocena wad i zalet**

Sprawność detekcji twarzy dla takiego podejścia jest bardzo wysoka ( większa niż 95%). Mimo tego zdarza się, że okno zakomunikuje nas o wykryciu twarzy mimo tego że twarz na obrazku nie występuje. Aby zminimalizować występowanie takiej sytuacji stosuje się zasadę, że dopiero trzy nakładające się na siebie ramki z wykrytą twarzą ostatecznie decydują czy twarz faktycznie widnieje na obrazku. Proces ten doskonale widać na poniższej ilustracji.

Kluczową zaletą Haar w porównaniu do większości innych funkcji jest jej szybkość obliczeniowa. Dzięki zastosowaniu integralnych obrazów, w dowolnym rozmiarze można obliczyć dowolną wielkość Haar (w przybliżeniu 60 instrukcji mikroprocesora dla funkcji 2-prostokątnej).

# **6. Porównanie algorytmów LBP i Haar**

Oba algorytmy wykrywania twarzy występujące w bibliotece OpenCV działają na podobnej zasadzie, lecz każdy z nich ma swoje mocniejsze i słabsze strony. Cechą łączącą oba algorytmy jest to, że muszą zostać wytrenowane, aby mogły spełniać swoją funkcję. Przebieg pracy obydwu algorytmów przedstawia poniższa ilustracja.



Pomimo kilku podobieństw obydwu algorytmów łatwo zauważyć też ich różnice w budowie, użyteczności, czy wykorzystaniu. Zbiór tych różnic przedstawia poniższa tabela.

|  |  |  |
| --- | --- | --- |
| **Algorytm** | **Zalety** | **Wady** |
| Haar | * Wysoka dokładność wykrywania * Niska liczba błędnych odczytów | * Duża złożoność algorytmu i wolna praca * Dłuższy czas szkolenia algorytmu * Niska dokładność przy słabym oświetleniu * Ograniczone działanie przy trudnych warunkach oświetleniowych * Mniej odborny na okluzje |
| LBP | * Obliczeniowo prosty i szybki w działaniu * Krótszy czas szkolenia * Odporny na chwilowe zmiany oświetlenia * Odporny na okluzje | * Mniejsza dokładność * Wyższa liczba błędnych odczytów |

Każdy klasyfikator wykrywania twarzy OpenCV ma swoje wady i zalety, ale największe różnice to dokładność i szybkość. W przypadku konieczności dokładniejszego wykrycia, klasyfikator Haar wydaje się być lepszym wyborem. Algorytm ten wydaje się być bardziej przydatny w takich technologiach, jak systemy zabezpieczeń czy wysokiej klasy stalking.

Klasyfikator LBP charakteryzuje się dużą szybkością, dlatego powinien być używany w aplikacjach mobilnych lub wbudowanych systemach.

# Literatura

1. Szczepan Kurnyta „Optymalizacja algorytmu wyznaczającego przepływ optyczny dla obrazów kolorowych”
2. David Stavens „The OpenCV Library: Computing Optical Flow”
3. David J. Fleet, Yair Weiss „Optical Flow Estimation”
4. Dr. Florian Raudies „Optical Flow”
5. Marcin Nazimek, „Porównanie skuteczności algorytmów detekcji ruchu dla systemów wizyjnych ruchu ulicznego w wykrywaniu pojazdów”
6. Marcin Bzdawski, „Śledzenie obiektów w sekwencjach obrazów”
7. Andrzej Głowacz, Zbigniew Mikrut, Piotr Pawlik, „Algorytm wideodetekcji korzystający z metody obliczenia przepływu optycznego”
8. Aroh Barjatya, „Block Matching Algorithms For Motion Estimation”
9. Dhara Patel, „Optical Flow Measurement using Lucas kanade Method”
10. Wikipedia, „Haar-like feature”
11. Adam Harvey, „Adam Harvey Explains Viola-Jones Face Detection”
12. OpenCV, „Face Detection using Haar Cascades“
13. OpenCV, „Cascade Classifier Training”
14. Ramiz Raja, „Face detection using OpenCV and Python: a beginner’s guide”