



**POLITECHNIKA  
GDAŃSKA**

WYDZIAŁ ELEKTRONIKI,  
TELEKOMUNIKACJI I INFORMATYKI



Imię i nazwisko studenta: Dorian Krefft  
Nr albumu: 143263  
Studia drugiego stopnia  
Forma studiów: stacjonarne  
Kierunek studiów: Informatyka  
Specjalność/profil: Algorytmy i technologie internetowe

## **PRACA DYPLOMOWA MAGISTERSKA**

Tytuł pracy w języku polskim: Licznik uśmiechów

Tytuł pracy w języku angielskim: Smile counter

Potwierdzenie przyjęcia pracy	
Opiekun pracy  <i>podpis</i>	Kierownik Katedry/Zakładu  <i>podpis</i>
dr inż. Agnieszka Landowska	

Data oddania pracy do dziekanatu:



**POLITECHNIKA  
GDAŃSKA**

WYDZIAŁ ELEKTRONIKI,  
TELEKOMUNIKACJI I INFORMATYKI



## OŚWIADCZENIE

Imię i nazwisko: Dorian Krefft  
Data i miejsce urodzenia: 25.08.1992, Kościerzyna  
Nr albumu: 143263  
Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki  
Kierunek: informatyka  
Poziom studiów: II stopnia  
Forma studiów: stacjonarne

Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody\* na korzystanie z mojej pracy dyplomowej zatytułowanej: Licznik uśmiechów do celów naukowych lub dydaktycznych.<sup>1</sup>

Gdańsk, dnia .....

.....  
*podpis studenta*

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2006 r., nr 90, poz. 631) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),<sup>2</sup> a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

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

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

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia .....

.....  
*podpis studenta*

Upoważniam Politechnikę Gdańską do umieszczenia ww. pracy dyplomowej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jej procesom weryfikacji i ochrony przed przywłaszczeniem jej autorstwa.

Gdańsk, dnia .....

.....  
*podpis studenta*

\*) niepotrzebne skreślić

---

Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG.

Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:

Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

## STRESZCZENIE

Celem pracy magisterskiej jest utworzenie aplikacji zliczającej liczbę uśmiechów w ciągu dnia, miesiąca oraz od początku jej instalacji. Aplikacja powinna w czasie rzeczywistym analizować obraz z kamery podłączonej do stacji, na której została zainstalowana, a następnie w wyniku analizy wykryć twarz i określić, czy użytkownik się uśmiechnął. Ponadto, różne instancje aplikacji powinny posiadać wspólną bazę danych, co umożliwiłoby zliczanie wszystkich uśmiechów, jakie zostały wykryte. Aplikacja ma na celu przede wszystkim wspomóc rozwój dzieci z autyzmem i zachęcić je do uśmiechu.

W ramach projektu powstały dwie wersje aplikacji. Pierwszą z nich jest aplikacja webowa, oparta na technologii AngularJS po stronie klienta i usługach RESTowych po stronie serwera, które pozwalają na wykrywanie uśmiechu. Drugą wersją jest aplikacja desktopowa, w którą wbudowano usługi do rozpoznawania emocji w celu przyspieszenia jej wydajności. Obie wersje korzystają ze wspólnej bazy danych.

Aplikacja desktopowa oprócz zliczania liczby uśmiechów zachęca użytkowników do częstszego uśmiechania się. Taka funkcjonalność zrealizowana została przy pomocy prostych efektów wyświetlających się na obrazie z kamery (m.in. proste animacje podczas uśmiechania się, zabawne efekty do twarzy bez wykrytego uśmiechu).

**Słowa kluczowe:** affective programming, wykrywanie twarzy i uśmiechu

**Dziedzina nauki i techniki, zgodnie z wymogami OECD:** Nauki inżynieryjne i techniczne. Elektrotechnika, elektronika, inżynieria informatyczna.

## **ABSTRACT**

The goal of this MSc thesis is to design and implement an application for counting smiles during the day, the month and since its installation. The application should analyze the image from a webcam in real time, then detect a face and determine if a user has smiled. In addition, different instances of the application should share a database in order to allow collaborate counting of the detected smiles. The application is primarily designed to support the development of children with autism and encourage them to smile.

During the project two versions of the application were created. The first is a web application, based on technology AngularJS on the client side and server side with REST services, allowing the detection of a smile. The second version is a desktop application, which has built-in services recognizing emotions in order to speed up its performance. Both of them share a common database.

**Keywords:** affective programming, face and smile detection

**OECD field of science and technology (FOS) classification:** Engineering and technology. Electrical engineering, Electronic engineering, Information engineering.

## SPIS TREŚCI

Streszczenie .....	3
Abstract .....	4
Spis treści .....	5
1. Wstęp .....	7
2. Definicja problemu i jego rozwiązanie .....	8
2.1. Cele pracy .....	8
2.2. Motywacja podjęcia tematu .....	9
2.3. Zastosowanie pracy magisterskiej .....	9
2.4. Definicja uśmiechu .....	9
2.5. Sposoby detekcji uśmiechu .....	11
2.5.1. Przetwarzanie emocji i punkty kluczowe twarzy .....	11
2.5.2. Przetwarzanie obrazu w celu wyszukania podobnego fragmentu .....	12
2.6. Klasyfikatory .....	13
2.6.1. Metody boostingu .....	13
2.6.2. Klasyfikatory Haara .....	15
2.6.3. Klasyfikatory LBP .....	16
2.6.4. Trenowanie własnego klasyfikatora .....	19
2.7. Koncepcja rozwiązania .....	21
2.7.1. Przyjęte założenia .....	21
2.7.2. Architektura systemu .....	21
3. Implementacja rozwiązania .....	24
3.1. Przegląd istniejących bibliotek do rozpoznawania twarzy i emocji .....	24
3.1.1. OpenCV .....	24
3.1.2. OpenIMAJ .....	25
3.1.3. FaceSDK .....	25
3.2. Detekcja uśmiechu .....	26
3.3. Sposoby implementacji usług do detekcji uśmiechów .....	28
3.3.1. Usługa oparta o OpenCV .....	28
3.3.2. Usługa oparta o OpenIMAJ .....	29
3.3.3. Usługa oparta o FaceSDK .....	29
3.3.4. Usługa powstała poprzez połączenie OpenCV oraz FaceSDK .....	30
3.4. Przetwarzanie obrazu w czasie rzeczywistym .....	31
3.4.1. Wykrywanie nowego uśmiechu .....	31
3.4.2. Prezentowanie efektów zachęcających do uśmiechu .....	32
3.4.3. Lista zrealizowanych efektów .....	33
3.5. Aplikacja desktopowa .....	37
3.5.1. Ekran podglądu obrazu z kamery .....	37
3.5.2. Ekran ustawień .....	38
3.5.3. Konfiguracja i sposób uruchomienia aplikacji .....	38
3.6. Aplikacja webowa .....	40
3.6.1. Strona główna .....	41
3.6.2. Testowanie usług wykrywających uśmiech .....	43
3.6.3. Ekran pobierania aplikacji desktopowej .....	48
3.6.4. Konfiguracja i sposób uruchomienia aplikacji .....	48

4. Weryfikacja rozwiązania .....	51
4.1. Testy bibliotek.....	51
4.1.1. Własnoręcznie przygotowany zbiór obrazów .....	51
4.1.2. Testy na istniejących bazach twarzy ludzi .....	56
4.1.3. Porównanie klasyfikatora twarzy Haara z LBP .....	61
4.1.4. Analiza statystyk oraz wyników ankiety z aplikacji webowej.....	63
4.1.5. Wnioski z przeprowadzonych testów .....	69
4.2. Ewaluacja rozwiązania .....	70
4.2.1. Wdrożenie .....	70
4.3. Walidacja .....	70
5. Podsumowanie .....	71
Wykaz literatury .....	72
Wykaz tabel .....	74
Wykaz rysunków.....	75

## 1. WSTĘP

Wraz ze wzrostem liczby urządzeń z wbudowaną kamerą cyfrową rośnie popularność aplikacji oferujących analizę obrazu w czasie rzeczywistym. Użytkownicy chętnie sięgają po programy modyfikujące twarz i wyświetlające ciekawe efekty graficzne – najlepszym tego przykładem jest *Snapchat*, aplikacja na urządzenia mobilne, która od 2011 roku uzyskała ok. 100 milionów użytkowników korzystających z niej na co dzień [1]. Swoją popularność zawdzięcza algorytmom detekcji twarzy i jej charakterystyk – umożliwia użytkownikom nagrywanie filmów i robienie zdjęć wraz z automatycznie nanoszonymi efektami zależnymi między innymi od miny, jaką zrobi użytkownik.

Zastosowanie aplikacji analizujących obraz w czasie rzeczywistym nie musi dotyczyć jedynie rozrywki – rozpoznawanie twarzy można wykorzystać między innymi do zabezpieczania komputera przed dostępem niepowołanych osób. Innym przykładem zastosowania tego typu funkcjonalności jest detekcja uśmiechu podczas robienia zdjęć aparatem cyfrowym.

Głównym celem pracy magisterskiej jest zachęcenie dzieci z autyzmem do uśmiechania się – często występuje u nich problem z okazywaniem emocji. Analizując obraz z kamery internetowej można wykryć uśmiechy osób na obrazie – aplikacja może też przy pomocy prostej interakcji zachęcać użytkownika do uśmiechnięcia się, np. poprzez prezentowanie prostych, zabawnych efektów graficznych.

Licznik uśmiechów może być wykorzystany do prowadzenia statystyk i śledzenia poprawy w częstotliwości okazywania emocji przez takie dzieci. Ze względu na potrzebę jedynie zliczania uśmiechów, w projekcie nie występuje potrzeba rozpoznawania uśmiechu – takie założenie pozwoli na znaczące uproszczenie algorytmów, a więc także zwiększenie wydajności.

Ponieważ baza danych powinna być współdzielona przez wszystkie instancje aplikacji, można rozszerzyć funkcjonalność zliczania uśmiechów o zbieranie dodatkowych informacji, w tym zapisywanie (za zgodą użytkownika) klatek, na których wykryto uśmiech. Takie dane mogą być przydatne w przyszłych pracach magisterskich dotyczących tego tematu – być może pozwolą na analizę skuteczności i uzyskanie dokładniejszych algorytmów do rozpoznawania uśmiechu.

## 2. DEFINICJA PROBLEMU I JEGO ROZWIĄZANIE

### 2.1. Cele pracy

Głównym celem pracy jest utworzenie aplikacji desktopowej, która w czasie rzeczywistym pobiera obraz z kamery podłączonej do stacji, a następnie sygnalizuje wykrycie uśmiechów. Ważnym aspektem jest ich zliczanie – aplikacja powinna w prawidłowy sposób odbierać ciągłość uśmiechu (jego rozpoczęcie oraz zakończenie), czyli nie zliczać każdej klatki z uśmiechem osobno. Różne instancje aplikacji powinny mieć dostęp do współdzielonej bazy danych, a liczba rozpoznanych uśmiechów powinna być prezentowana w formie statystyk na ekranie z prezentowaniem obrazu z kamery – wartość ta powinna być na bieżąco aktualizowana. Program powinien także zachęcać użytkownika do wykonania uśmiechu, poprzez prezentowanie prostych efektów na obrazie z kamery.

Prócz wspomnianych wcześniej celów podstawowych, aplikacja desktopowa powinna dodatkowo udostępniać możliwość przypisania jej nazwy lokalizacji, przy pomocy której można pogrupować i zliczyć uśmiechy należące do konkretnej instancji. Liczniki prezentowane użytkownikowi powinny być także podzielone na trzy grupy: dzienny, miesięczny oraz ogólny (od momentu rozpoczęcia pracy instancji). Na ekranie głównym aplikacji powinien znajdować się baner reklamowy, prezentujący informację zdefiniowaną przez administratora.

Zaletą programu będzie jego wieloplatformowość – ze względu na fakt, że autor pracy magisterskiej w największym stopniu posiada znajomość języka programowania Java, aplikację będzie można uruchomić zarówno na systemach rodziny Windows, jak i Linux.

Aplikacja powinna zostać wdrożona w ośrodku dla dzieci z autyzmem w ramach terapii wspomagającej oddawanie emocji – szczególnie pokazywania uśmiechu. Po udanym wdrożeniu należy zweryfikować poprawność działania aplikacji.

Zakres pracy magisterskiej został rozszerzony o utworzenie także aplikacji webowej, której głównym zadaniem jest prezentowanie sposobu działania bibliotek wykrywających uśmiech. W trakcie tworzenia aplikacji należy przewidzieć możliwość wyboru biblioteki – implementacja powinna pozwolić na zmianę i wypróbowanie jednej z rozpoznanych bibliotek wspierających detekcję emocji użytkownika.

Zapisywanie uśmiechów do bazy danych można wzbogacić – za zezwoleniem użytkowników – o załączanie informacji o położeniu wykrytych uśmiechu wraz z kadrem z kamery. Zbiór takich danych można wykorzystać w następnych projektach oraz do znalezienia lepszych sposobów detekcji uśmiechu.

Efektom pracy magisterskiej powinny być dwie aplikacje posiadające wspólną bazę danych: aplikacja desktopowa i webowa. Obie powinny działać przy pomocy współdzielonych usług oferujących detekcję uśmiechu, co pozwoli na równoległą pracę przy obu wersjach programu.



## **2.2. Motywacja podjęcia tematu**

Główną motywacją podjęcia tematu jest zainteresowanie autora pracy magisterskiej dziedziną przetwarzania obrazu oraz chęć nauki rozpoznawania emocji u człowieka. Dziedzina ta jest wciąż młoda i dynamicznie się rozwija, posiadając olbrzymi potencjał na wykorzystanie w nowoczesnych aplikacjach – praca magisterska jest zatem także formą rozpoznania nowoczesnych technologii do przetwarzania emocji na obrazie w czasie rzeczywistym.

Dodatkową motywacją jest fakt, że badanie emocji wydaje się być coraz bardziej popularne, zwłaszcza na urządzeniach mobilnych. Wynik pracy będzie można wykorzystać w przyszłych projektach, a także rozwinąć go o dodatkowe cechy.

## **2.3. Zastosowanie pracy magisterskiej**

Podstawowym zastosowaniem produktu pracy jest rozrywka - prezentuje on rozpoznawanie emocji na obrazie w czasie rzeczywistym oraz przy pomocy prostych efektów zachęca użytkownika do uśmiechania się.

Aplikacja może być także wykorzystana do zbierania statystyk oraz materiałów do prac o podobnym temacie – może w tym pomóc zapisywanie klatek z wykrytym uśmiechem użytkownika wraz z szeregiem dodatkowych informacji.

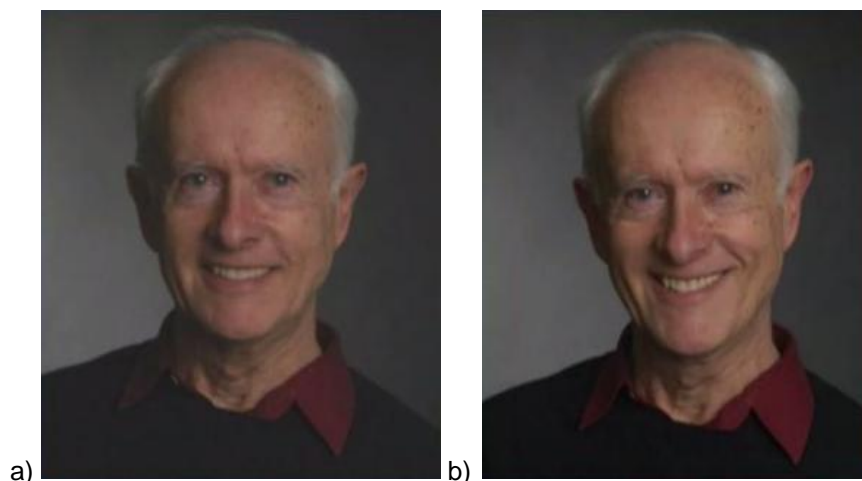
Ciekawym zastosowaniem może być także wykorzystanie usług do badania stopnia zadowolenia użytkownika podczas korzystania z innych programów – zebrana liczba uśmiechów może świadczyć o pozytywnym odebraniu np. filmu komediowego. Modułowość aplikacji znacząco ułatwia wdrożenie jej w innych projektach.

## **2.4. Definicja uśmiechu**

Przed zagłębieniem się w tajniki detekcji należy w pierwszej kolejności zdefiniować samo pojęcie uśmiechu.

Uśmiechem można nazwać napięcie mięśni znajdujących się po bokach ust w taki sposób, by unieść ich kąciaki ku górze [7]. Uśmiech najczęściej wiąże się z emocją zdefiniowaną jako radość – ponieważ jednak odgrywa on znaczącą rolę w komunikacji międzyludzkiej, często zdarza się, że jest wymuszony. Niekiedy w parze z takim sztucznym uśmiechem mogą pojawić się zupełnie inne emocje, jak na przykład zażenowanie czy uśmiech pojawiający się w postaci grymasu złości.

Istnieje możliwość rozpoznania faktu, czy uśmiech jest szczery, czy wymuszony. Określenie tego odbywa się przy pomocy obserwacji mięśni Duchenne (od imienia francuskiego neurologa, Guillaume Duchenne) znajdujących się w pobliżu oczu [8]. Zauważył on, że podczas szczerego uśmiechania się powieki oczu rozszerzają się, a w ich kąciakach pojawiają się zmarszczki.

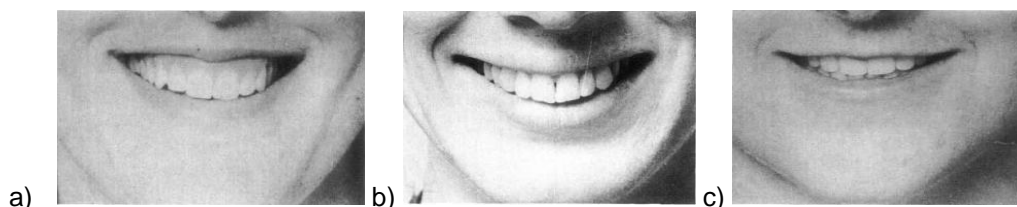


**Rys. 1. Różnice pomiędzy uśmiechem Pan Am oraz Duchenne**

a) Uśmiech Pan Am, b) Uśmiech Duchenne

Rys. 1. Pokazuje różnicę pomiędzy wspomnianym wcześniej uśmiechem Duchenne a tzw. uśmiechem Pan Am (tzw. „sztucznym”, zwanym także „*botox smile*”). Zyskał on swoją nazwę od amerykańskich linii lotniczych *Pan American World Airways*, których stewardessy stały się stereotypem tego uśmiechu [10].

Kolejną zmienną różniącą poszczególne uśmiechy jest ich stopień. Im większa emocja towarzyszy człowiekowi, tym jego uśmiech staje się szerszy. Prócz uśmiechu zamkniętego, w którym uniesione są jedynie kąciaki ust, istnieje także tzw. uśmiech otwarty, którego stopnie można podzielić na trzy grupy ([9]), co zostało przedstawione na przykładzie Rys. 2.



**Rys. 2. Przedstawienie stopni uśmiechu otwartego**

a) Szeroki uśmiech – widoczna jest całość zębów przednich oraz dziąseł, b) Średni uśmiech – widoczne jest od 75% do 100% zębów przednich oraz część dziąseł, c) Niewielki uśmiech – widoczne jest mniej niż 75% zębów przednich

Jak można się domyślić, najłatwiejszym do detekcji jest uśmiech otwarty, ze względu na widoczność zębów, które mocno wyróżniają się na tle reszty twarzy. Uśmiech zamknięty jest szczególnie trudny do wykrycia ze względu na konieczność określenia progu stopnia uniesienia kąciaków ust – dla wielu osób moment przejścia ust w stan uśmiechu jest różny i często trudno jednoznacznie określić, czy dana osoba się uśmiecha.

Warto także wspomnieć, że istnieje kilka rodzajów uśmiechu, zwłaszcza otwartego, jego cechy różnią się także pomiędzy poszczególnymi rasami ludzi. Wydaje się jednak, że nie będą one miały dużego wpływu na jakość detekcji.

W ramach pracy magisterskiej zdecydowano się wykrywać uśmiechy bez względu na to, czy uśmiech jest szczery, czy wymuszony – aplikacja ma zachęcać do okazywania emocji, jednak trudno wymagać od użytkownika uśmiechów typu Duchenne – częścią zabawy z aplikacją jest testowanie, czy policzy ona prawidłowo uśmiech.

## **2.5. Sposoby detekcji uśmiechu**

Problem rozpoznawania emocji u użytkownika systemu informatycznego może być rozwiązany przy pomocy różnych sposobów i polega na badaniu reakcji użytkownika na konkretną sytuację (np. tempo ruchów myszki czy kliknięć). Tematem pracy jest liczenie uśmiechów użytkownika, w związku z czym zdecydowano skupić się na analizie obrazu z kamery podłączonej do komputera.

Do detekcji uśmiechu wystarczy odnalezienie twarzy użytkownika – na niej znajdują się wszelkie informacje pozwalające określić, czy osoba się uśmiecha, czy nie. Głównymi metodami służącymi do odpowiedzi na to pytanie są: zbadanie emocji użytkownika (radość zazwyczaj definiowana jest przy pomocy uśmiechu) oraz porównywanie obrazu do interesujących nas fragmentów (np. twarzy uśmiechniętych osób).

### **2.5.1. Przetwarzanie emocji i punkty kluczowe twarzy**

Przetwarzanie emocji jest jednym z najskuteczniejszych sposobów do wykrywania uśmiechu. Polega ono na wykorzystaniu wszystkich informacji zawartych w twarzy, co pozwala na znaczące zwiększenie dokładności wykrywania uśmiechu dzięki wykorzystaniu danych o kontekście elementów twarzy. Minusem tego podejścia jest zazwyczaj wolniejsze działanie algorytmu w stosunku do porównywania fragmentów obrazów. Typowy proces wykorzystujący tę metodę dzieli się na cztery kroki: lokalizacja twarzy, wydobycie punktów kluczowych twarzy, ekstrakcja cech oraz predykcja emocji na podstawie zebranych informacji.

Biblioteki przetwarzające emocje na zadanym obrazie najczęściej posiadają ich zdefiniowaną i z góry ograniczoną listę – prawdopodobieństwo wystąpienia każdej z nich zazwyczaj jest przedstawione w macierzy. Typowymi emocjami uwzględnianymi przez takie biblioteki są: smutek, gniew, strach, zaskoczenie, wstręt oraz radość. Do wykrywania uśmiechu najbardziej przydatną emocją jest radość – przy założeniu szczerości uśmiechania się, można ją rozpoznać m.in. po występowaniu uśmiechu na twarzy użytkownika. Zgodnie z takim podejściem wystarczy odczytać wartość prawdopodobieństwa występowania radości – po przekroczeniu odpowiedniego progu można założyć, że w danym fragmencie obrazu występuje uśmiech.

Rozwiązania implementujące przetwarzanie emocji do określenia stopnia występowania danej emocji u osoby na zdjęciu najczęściej wykorzystują punkty kluczowe twarzy. Mogą się one znajdować np. na brwiach, oczach, nosie, policzkach oraz ustach, a ich liczba różni się pomiędzy konkretnymi implementacjami i ma wpływ na dokładność oraz szybkość działania algorytmu.

Ponieważ zadaniem niniejszej pracy magisterskiej jest detekcja uśmiechu (a nie emocji radości), do pracy algorytmu nie są potrzebne informacje na temat wszystkich występujących emocji. Niektóre biblioteki, oprócz macierzy prawdopodobieństwa, zwracają także informacje o samych punktach kluczowych – odrzucenie etapu ekstrakcji cech pozwala znacząco na przyspieszenie pracy programu. Informacje o występowaniu uśmiechu można odczytać z brwi, oczu, policzków oraz ust – ponieważ jednak założono wykrywanie nie tylko uśmiechów Duchenne ale także sztucznych, można skupić się na badaniu punktów kluczowych samych ust. Po zdefiniowaniu krzywych opisujących obie wargi można ustalić próg wygięcia dolnej i górnej, po przekroczeniu którego usta będą znajdowały się w stanie uśmiechu. Dzięki takiemu podejściu można dużo łatwiej (w porównaniu do drugiej metody) wykrywać uśmiechy zamknięte.

#### *2.5.2. Przetwarzanie obrazu w celu wyszukania podobnego fragmentu*

Proces przetwarzania obrazu w celu wyszukiwania podobnego fragmentu twarzy jest obecnie jednym z najpopularniejszych sposobów detekcji uśmiechu. Polega on na zebraniu próbek pozytywnych (fragmenty innych obrazów zawierające interesujący nas obiekt) oraz negatywnych (fragmenty nie zawierające poszukiwanego obiektu) – przy ich pomocy algorytm wyszukuje w zadanym obrazie fragmenty, które spełniają warunki podobieństwa.

Jedną z metod implementacji takiego przetwarzania obrazu są klasyfikatory (kaskady) Haara oraz LBP. Klasyfikatory są prostym sposobem na przedstawienie informacji interesującym nas obiekcie w postaci pliku XML, co znacząco zmniejsza rozmiar materiałów niezbędnych do wykrycia uśmiechu.

Na chwilę obecną sposób ten jest najczęściej wykorzystywany na urządzeniach mobilnych, zazwyczaj wykorzystując możliwości biblioteki *OpenCV* – przy jego pomocy proces wyszukiwania uśmiechu jest stosunkowo szybki, w porównaniu do poprzednich metod. Dokładność wskazywanych uśmiechów w głównej mierze zależy od klasyfikatorów (od liczby zebranych próbek pozytywnych i negatywnych, ich jakości, zróżnicowaniu, stosunku itd.), ale także od ustawień wprowadzonych w samej bibliotece (np. stosunek wysokości do szerokości w wykrywanym uśmiechu). Warto jednak zauważyć, że mimo wszystko dokładność będzie mniejsza niż przy wykorzystaniu np. rozpoznawania kluczowych punktów twarzy – klasyfikator nie informuje o kontekście fragmentu (np. „usta muszą znajdować się w twarzy”). Bez zastosowania dodatkowych kroków weryfikujących poprawność wykrycia uśmiechu algorytm może zwrócić błędne (często nawet bezsensowne) pozycje ust. Dodatkowo, przy pomocy tego sposobu najtrudniej jest wykryć uśmiech zamknięty, ze względu na fakt, że próbki pozytywne takiego rodzaju uśmiechu są bardzo podobne do próbek negatywnych (np. ust bez stanu uśmiechu) – wymusza to zastanowienie się nad stopniem wykrywania uśmiechu już na poziomie przygotowywania próbek.

## 2.6. Klasyfikatory

Klasyfikatory (kaskady) są najpopularniejszym sposobem implementacji metody przetwarzania obrazów w celu wyszukiwania interesujących fragmentów (tzw. *region of interest* – ROI). Przykładem takich fragmentów mogą być twarze, piesi na chodniku, czy też tablice rejestracyjne pojazdów. Klasyfikatory są przedstawiane w postaci pliku XML zawierającego informacje o zebranych próbkach pozytywnych oraz negatywnych – czyli o tym, co algorytm ma znaleźć i czego unikać na zdjęciu.

Najpopularniejsze biblioteki implementują dwa formaty klasyfikatorów – Haara oraz LBP. Różnica pomiędzy nimi polega głównie na sposobie przedstawiania informacji, co prowadzi do różnic w dokładności i szybkości działania algorytmu.

### 2.6.1. Metody boostingu

Boosting to metoda, która przy pomocy  $n$  słabych klasyfikatorów (czyli takich, które samodzielnie nie są w stanie odpowiedzieć na zadane im pytanie) tworzy jeden silny [15, 16], a zatem służy do zwiększania skuteczności dowolnego algorytmu uczenia. Polega to na tworzeniu prostego modelu z danych do trenowania klasyfikatora, by następnie na jego podstawie utworzyć kolejny model, próbujący poprawić błędy poprzedniego – proces ten powtarzany jest tak długo, aż osiągnie się określony próg błędu lub wykona się założona z góry liczba kroków  $T$ . Istnieje dużo różnych algorytmów boostingu, jednak omówiony zostanie algorytm AdaBoost, ze względu na wykorzystanie w bibliotekach do wykrywania twarzy (np. OpenCV). Został on utworzony przez Yoava Freunda oraz Roberta Schapire'a w 2003 r., którzy nazwali swój uczący się algorytm adaptive boosting, w skrócie AdaBoost.

W przypadku klasyfikatorów, AdaBoost najczęściej otrzymuje na wejściu zbiór treningowy  $\Omega = [(x_1, y_1), \dots, (x_n, y_n)]$ , gdzie  $x_i$  to instancja problemu z dziedziny  $X$ , a  $y_i$  to binarna wartość decyzyjna dla tej instancji ( $Y \in \{-1, 1\}$ ). Na przykładzie niniejszej pracy magisterskiej można założyć, że każda instancja problemu  $x_i$  to pojedyncze zdjęcie (lub jego fragment), natomiast wartość decyzyjna  $y_i$  pozwala określić, czy na danym obrazie znajduje się uśmiech, czy też nie. Podstawowym założeniem działania algorytmu AdaBoost jest stwierdzenie, że dla dowolnego rozkładu, jeżeli otrzymamy wystarczająco dużą liczbę próbek pozytywnych i negatywnych, z dużym prawdopodobieństwem można w wielomianowym czasie otrzymać klasyfikator, który odpowie na pytania lepiej niż podejście losowania odpowiedzi [24]. Można zatem stwierdzić, że błąd takiego klasyfikatora musi opisywać równanie  $\epsilon < \frac{1}{2}$ .

Podejście algorytmu AdaBoost zakłada, że chociaż dowolny problem może być trudny do rozwiązania, to znacznie łatwiej można go rozbić na szereg mniejszych problemów. Aby lepiej zobrazować to podejście, można posłużyć się przykładem: zamiast odpowiadać od razu na pytanie, czy na zdjęciu znajduje się uśmiech, można w pierwszej kolejności zapytać np. „czy wskazany rejon posiada kolor zębów” oraz „czy rejon otoczony jest ciemniejszym kolorem (wargą)?”. Owe mniejsze problemy nazywane są *słabymi* klasyfikatorami,

które samodzielnie nie są w stanie odpowiedzieć prawidłowo na pytanie. Dopiero odpowiednio liczna grupa takich klasyfikatorów w połączeniu może stworzyć klasyfikator *silny*. Działanie algorytmu w postaci pseudokodu przedstawia Rys. 3 [15].

```

Dane:  $\Omega = [(x_1, y_1), \dots, (x_n, y_n)], Y \in \{-1, 1\}$ 

for  $i = 1$  to  $n$  do
     $w_i = \frac{1}{n}$ 
end
for  $t = 1$  to  $T$  do
    znajdź nowy słaby klasyfikator  $K_t$  przy pomocy wag  $w$ 
    
$$\epsilon_t = \frac{\sum_{i=1}^n w_i I(y_i \neq K_t(x_i))}{\sum_{i=1}^n w_i}$$

    
$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

     $p = w$ 
    oblicz współczynnik normalizacji  $Z_t$ 
    for  $i = 1$  to  $n$  do
        
$$w_i = \frac{p_i^{-\alpha_t y_i K_t(x_i)}}{Z_t}$$

    end
end
    znajdź i zwróć klasyfikator silny  $K_{\text{silny}}$ :
    
$$K_{\text{silny}}(x) = \text{sign}(\sum_{t=1}^T \alpha_t K_t(x))$$


```

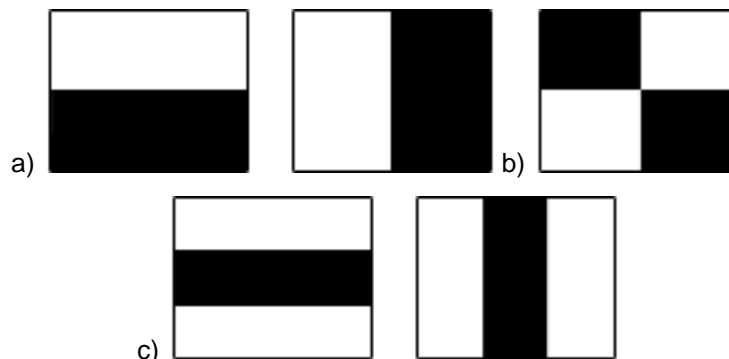
**Rys. 3. Pseudokod algorytmu AdaBoost**

Każdy krok algorytmu tworzy nowy słaby klasyfikator, który stara się lepiej rozwiązać problemy poprzedniego. W pierwszym etapie każda instancja problemu otrzymuje równą wagę ( $\forall_{1 \leq i \leq n+1} w_x(i) = \frac{1}{n}$ ), a we wszystkich kolejnych iteracjach wagi są zwiększane instancjom, na które klasyfikatory nie były w stanie prawidłowo odpowiedzieć. Dzięki temu każdy kolejny krok algorytmu skupi się na trudniejszych problemach, co powinno pozytywnie wpłynąć na jakość detekcji żądanych obiektów na obrazie. Każdemu z utworzonych klasyfikatorów jest ustawiony współczynnik  $\alpha_t$  na podstawie jego błędu  $e_t$  obliczonego z uwzględnieniem wag wszystkich instancji. Na zakończenie algorytmu zostaje zwrócony klasyfikator silny, który odpowiada na pytanie korzystając z utworzonych wcześniej klasyfikatorów słabych. Można ująć, że klasyfikator ten jako odpowiedź zwraca „głos większości” swoich składowych z uwzględnieniem ich ważności przy pomocy współczynnika  $\alpha_t$ .

Główną zaletą algorytmu AdaBoost jest jego uniwersalność zastosowania, szybkość i prostota implementacji - nie wymaga on żadnej dodatkowej wiedzy na temat słabych klasyfikatorów, ponieważ uczą się one same na wskazanym zbiorze [24].

### 2.6.2. Klasyfikatory Haara

Podstawową metodą porównywania obrazów przy użyciu klasyfikatora jest obliczanie tzw. cech Haara. Są one badane przy użyciu kilku sąsiadujących ze sobą prostokątów, ułożonych na trzy różne sposoby, jak zaprezentowano na Rys. 4.

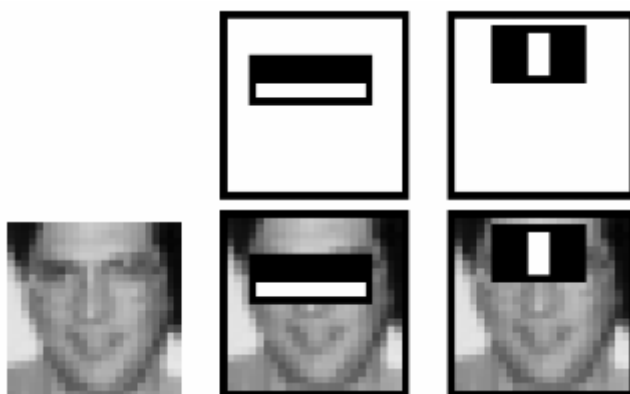


**Rys. 4. Cechy obrazu badane przez klasyfikator**

a) krawędzie b) cztery prostokąty c) linie

Jednostki cech Haara składają się z kilku prostokątów podzielonych na dwie równe grupy (na rysunku grupa „biała” oraz „czarna”). Wynikiem jej zastosowania jest uzyskanie prostej wartości liczbowej, zdobytej poprzez porównanie (odjęcie) ze sobą sumy pikseli w części białej ze sumą pikseli z części czarnej [13].

W ten sposób otrzymuje się prostą informację typu „dolna część jest jaśniejsza niż górna”. Tak uzyskane dane są następnie wykorzystywane do skategoryzowania procesowanego fragmentu obrazu. Dla przykładu, jeżeli klasyfikator wyszukuje ludzkie twarze, można wykorzystać obserwację, że region zawierający oko będzie zazwyczaj ciemniejszy od regionu zawierającego część czoła i policzka. W takim przypadku przy pomocy klasyfikatora wykorzysta się jednostkę typu linia, która w swej czarnej części będzie zawierała oko, a w białej odpowiednio czoło oraz policzek. Przykładowe wykorzystanie cech Haara na zdjęciu twarzy pokazuje Rys. 5.



**Rys. 5. Przykład nałożenia cech Haara na zdjęciu**

Oprócz sposobu ułożenia sąsiadujących prostokątów kolejnymi ważnymi parametrami jednostek cech Haara są ich rozmiar oraz położenie. Zwyczajne przeszukiwanie obrazu

poprzez rozpoczęcie z jednostką 2x2 pikseli i przesuwaniu jej po całym obrazie byłoby bardzo nieefektywne – już dla obrazu 24x24 pikseli istnieje ponad 160 000 możliwości. Aby znacząco zmniejszyć liczbę przeszukiwań zdecydowano się na wykorzystanie opisanego we wcześniejszym rozdziale algorytmu AdaBoost, w którym słabymi klasyfikatorami są właśnie jednostki cech Haara, a w kolejnych iteracjach są wyznaczane takie cechy, które dają najmniejszą wartość błędu. Klasyfikator silny jest ważonym złożeniem takich klasyfikatorów słabych. Dodatkowym sposobem na przyspieszenie pracy algorytmu jest założenie, że większość fragmentów obrazu nie posiada w sobie twarzy. W takim przypadku można uruchomić znacznie prostszy mechanizm, który stwierdzi, czy w danym rejonie *może* znajdować się twarz – jeżeli stwierdzi on, że nie, można przejść do następnego fragmentu obrazu.

W przypadku biblioteki OpenCV wprowadzono dodatkowo pojęcie *kaskady* klasyfikatorów. Normalnie pojedynczy klasyfikator silny składa się z  $T$  klasyfikatorów słabych – aby uzyskać odpowiedź, wykonywana jest suma ważona ze wszystkich jego składowych. Przy zastosowaniu kaskady klasyfikatorów owe składowe są dodatkowo grupowane względem ogólności w pomniejsze *etapy* (ang. *stages*). Podczas procesowania wskazanego fragmentu obrazu odpytywane są jedynie etapy najbardziej ogólne i algorytm przejdzie do następnego etapu tylko w przypadku otrzymania pozytywnej odpowiedzi. Takie podejście pozwala znacząco zminimalizować czas na procesowanie fragmentów, w których na pewno nie ma twarzy. Przykład wizualizacji działania klasyfikatora Haara został przedstawiony na prostym filmiku z popularnego serwisu YouTube pod adresem [14].

### 2.6.3. Klasyfikatory LBP

Innym sposobem implementacji klasyfikatora jest LBP – *Local Binary Pattern*. Służy on do badania zależności pomiędzy intensywnością pikseli na teksturze [26]. W przypadku LBP wykorzystano fakt, że wspólne cechy (takie jak krawędzie, linie czy punkty) mogą być przedstawione przy pomocy wartości liczbowej w odpowiedniej skali, a następnie porównane ze sobą w celu określenia podobieństwa. Dużą zaletą w działaniu tego klasyfikatora jest fakt, że operuje on w całości na liczbach całkowitych. Prostota w operacjach tego algorytmu i jego szybkość sprawiają, że jest on szczególnie często wykorzystywany na niewielkich urządzeniach mobilnych.

Główną ideą klasyfikatora LBP jest badanie intensywności pikseli na obrazie. W podstawowej wersji algorytmu klasyfikator operuje na kwadratach o boku 3 px, badając relację pomiędzy ich środkiem a ośmioma pikselami z nim sąsiadującymi. Dla każdego z sąsiadujących pikseli, jeżeli jest on intensywniejszy od piksela środka, zostaje dodana odpowiadająca mu waga w postaci potęgi 2, co pokazuje wzór:

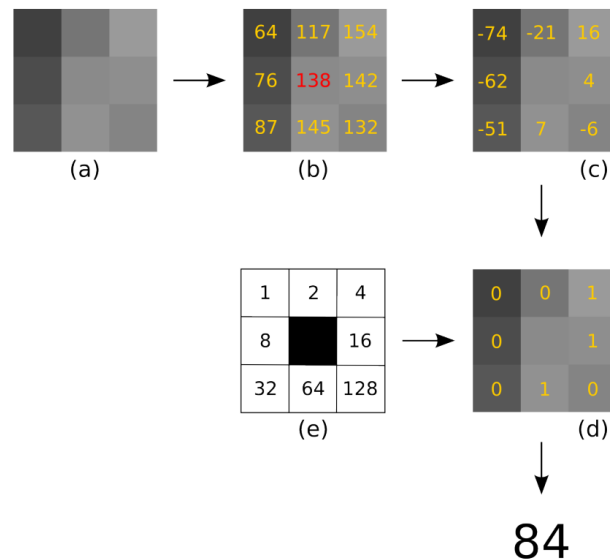
$$LBP(p_{sr}) = \sum_{i=0}^7 \text{czyDodatnie}(\text{intensywność}_{p_i} - \text{intensywność}_{p_{sr}}) * 2^i,$$

gdzie funkcja *czyDodatnie* zwraca wartość 1 w przypadku nieujemnej wartości oraz wartość 0, jeżeli różnica intensywności jest ujemna. Wynik przeprowadzonej operacji



w wersji kwadratu 3x3 znajduje w zakresie od 0 do 255 ze względu na istnienie 8 sąsiadujących pikseli.

Przykładowy proces operacji LBP pokazano na Rys. 6 [26].



**Rys. 6. Przykład działania klasyfikatora LBP**

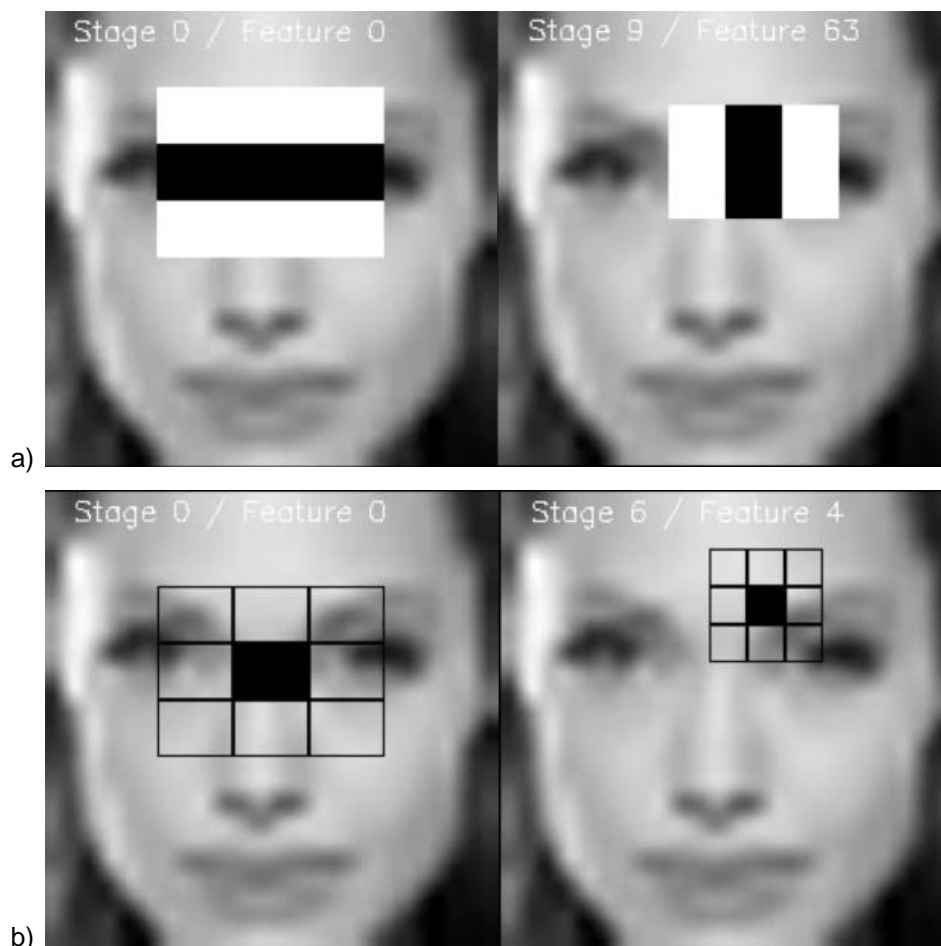
Pierwszy etap procesowania fragmentu (a) pokazuje, jak wygląda fragment obrazu przed procesowaniem LBP. Zawiera on jeden piksel środka, oznaczony jako  $p_{sr}$  oraz osiem pikseli sąsiadujących  $p_i$ . Kolejność sąsiadów na liście zazwyczaj jest ustalana jako zgodna z ruchem wskazówek zegara. W kolejnym kroku (b) na każdym z pikseli umieszczono poziom jego intensywności. Etap (c) pokazuje naniesione różnice pomiędzy pikselem środka a jego sąsiadami. Następnie (d) pokazuje, jakie wartości zostały przypisane każdemu z pikseli po zastosowaniu funkcji *LBP*. Macierz oznaczona jako (e) prezentuje wagi każdego piksela zgodnie z jego kolejnością ( $2^i$ ). Ostatnim etapem jest dodanie do siebie wag pikseli, które w wyniku zastosowania funkcji *LBP* otrzymały wartość 1.

Zwyczajna wersja klasyfikatora jest zbyt lokalna i nie wystarczy do detekcji twarzy oraz uśmiechów. Rozszerzeniem LBP jest wieloblokowy LBP (*Multi-block LBP – MBLBP*), który zamiast operować na pojedynczych pikselach (środek i sąsiedzi), operuje na równych grupach pikseli (najczęściej wciąż 3x3), a wartości intensywności w każdej grupie uśrednia – zostało to pokazane na Rys. 7.

0	1	2
7		3
6	5	4

Rys. 7. Przykład działania MBLBP

Klasyfikatory LBP są na ogół znacznie szybsze od klasyfikatorów Haara. Dzieje się tak między innymi dlatego, że w całości działają w oparciu o proste operacje na liczbach całkowitych. Chociaż używanie tego typu operacji powoduje na ogół spadek dokładności – w większości przypadków klasyfikatory Haara będą zwracały informacje z mniejszą liczbą błędów – w przypadku przygotowania odpowiedniej jakości próbek jest możliwe wytrenowanie klasyfikatora LBP, który będzie miał przynajmniej zbliżoną dokładność do Haara [27].



Rys. 8. Różnice w określaniu cech twarzy przez klasyfikatory

a) klasyfikator Haara, b) klasyfikator LBP

Rys. 8. pokazuje na przykładzie zdjęcia twarzy różnice w stosowaniu jednostek klasyfikatora do określania cech obiektów.

#### 2.6.4. Trenowanie własnego klasyfikatora

Chociaż istnieje wiele gotowych klasyfikatorów różnego rodzaju, czasem istniejący klasyfikator jest zbyt mało wiarygodny w założonym zastosowaniu lub zachodzi potrzeba wykrywania nietypowych obiektów na zdjęciu. W takiej sytuacji najlepszym rozwiązaniem jest utworzenie własnego klasyfikatora przy pomocy gotowych i darmowych narzędzi.

Pierwszym krokiem do wytrenowania własnego klasyfikatora jest zebranie odpowiedniej jakości próbek pozytywnych i negatywnych. Próbki negatywne są obrazami, które instruują klasyfikator, czego nie może wykryć – bezwzględnie nie mogą zatem zawierać poszukiwanego obiektu. Próbki pozytywne powinny natomiast zawierać tylko i wyłącznie obiekt, który należy znaleźć.

Zebrane próbki odpowiedniej jakości są najważniejszym składnikiem wytrenowania klasyfikatora. Powinny one być dobrze przemyślane. Zazwyczaj im większy jest zbiór próbek, tym większa będzie jego dokładność, o ile próbki nie są do siebie bardzo zbliżone (np. 10 sąsiednich klatek tego samego filmu niewiele różniących się od siebie). Często zdarza się też, że zbiory próbek są rozszerzane na podstawie działania klasyfikatora na nim opartych. Dla przykładu, jeżeli klasyfikator zwraca dużo *false alarmów* podczas detekcji ust wskazując dodatkowo na oczy, należy zbiory próbek negatywnych wzbogacić o fragmenty zawierające w sobie np. pojedyncze oko, a następnie ponownie wyszkolić nowy klasyfikator. Dobrym zwyczajem jest także prezentowanie próbek negatywnych i pozytywnych jako fragmentów prawdziwych zdjęć (łącznie z tłem, bez wycinania konturów) [12].

Istnieje wiele gotowych narzędzi umożliwiających trenowanie klasyfikatora. Jednym z najbardziej popularnych jest narzędzie udostępniane przez twórców biblioteki OpenCV. Składa się ono z kilku modułów: *opencv\_createsamples*, *opencv\_annotation*, *opencv\_traincascade*, a także *opencv\_visualisation*. Oprócz zadanego zbioru próbek pozytywnych oraz negatywnych posiada ono szereg przydatnych parametrów, pozwalających na łatwiejsze utworzenie wydajnych kaskad klasyfikatorów. Warto także wspomnieć, że trenuje ono zarówno klasyfikatory Haara, jak i LBP.

Sam schemat tworzenia klasyfikatora można przedstawić w postaci listy kroków:

1. Ustal średni rozmiar obiektu, który należy wykrywać
2. Przedstaw zbiór negatywny i pozytywny w postaci pliku *.txt*, w którym każda linia zawiera ścieżkę do obrazu
3. Przy pomocy narzędzia *opencv\_createsamples* wygeneruj dodatkowe próbki pozytywne poprzez ich np. obracanie lub zmianę rozmiaru. Zostaną one zapisane w postaci pliku *.dat*.

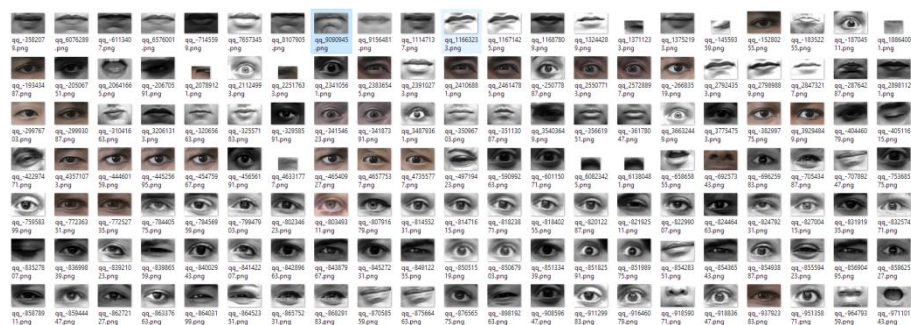
4. Wytrenuj nowy klasyfikator, korzystając z narzędzia *opencv\_traincascade*, używając utworzonego wcześniej pliku z próbkami negatywnymi (krok 2) oraz pozytywnymi (krok 3)

W ramach pracy magisterskiej zdecydowano się spróbować wygenerować własny klasyfikator. Jako zbiór próbek pozytywnych ustalono fragmenty zdjęć zawierające jedynie uśmiech zarówno w postaci zamkniętej, jak i otwartej. Zbiór ten zawierał łącznie 379 obrazów, a jego fragment został pokazany na Rys. 9.



**Rys. 9. Fragment zebranego zbioru próbek pozytywnych**

Jako zbiór próbek negatywnych ustalono wszystkie pozostałe elementy twarzy, które nie są uśmiechem. Można w nim znaleźć m.in. brwi, oczy o różnym stopniu przymrużenia, nos oraz usta w stanie nie będącym uśmiechem. Zbiór łącznie liczył 3 725 próbek, a jego fragment został zaprezentowany na Rys. 10.



**Rys. 10. Fragment zebranego zbioru próbek negatywnych**

Do wzbogacenia zbioru negatywnego zdecydowano się wykorzystać istniejące klasyfikatory. W tym celu napisano program, który pośród zadanej listy zdjęć uruchomi klasyfikatory nosa, oczu, brwi oraz ust, a następnie znalezione fragmenty zapisze w podanym katalogu. Zbiór taki został następnie dokładnie zweryfikowany, czy żaden fragment nie zawiera uśmiechu – mogłoby to spowodować błędną pracę działania klasyfikatora.

Trenowanym rodzajem klasyfikatora był klasyfikator Haara, szerokość wyszukiwanego obiektu ustalono na 40, natomiast wysokość na 20. Proces trenowania klasyfikatora trwał ponad 6 dni, po czym na wykorzystywanym do tego celu sprzęcie zabrakło zasobów – udało się

wytrenować tylko 15 z 20 etapów. Niestety, ze względu na czasochłonność tego procesu i brak lepszego sprzętu zdecydowano się zaniechać próby trenowania własnego klasyfikatora.

## **2.7. Koncepcja rozwiązania**

### **2.7.1. Przyjęte założenia**

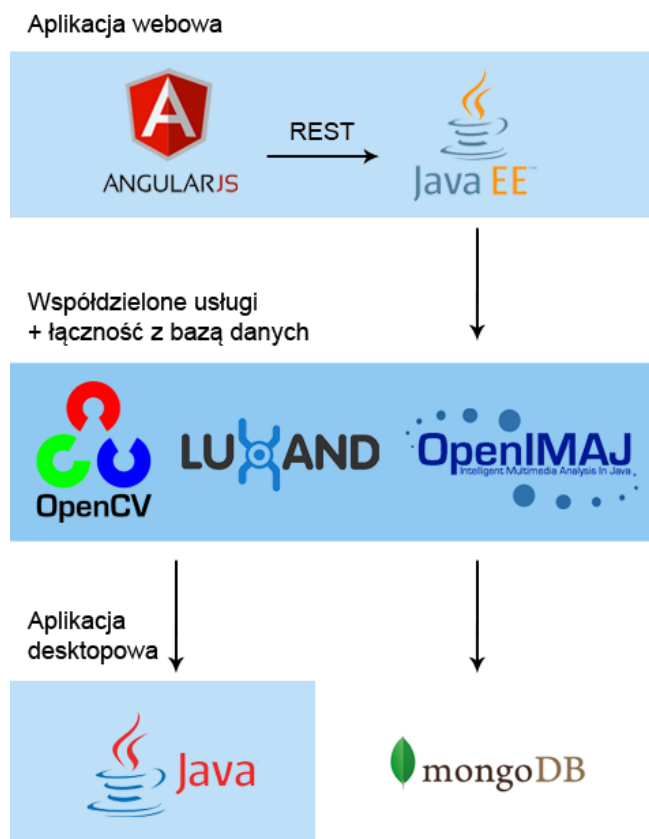
W celu łatwiejszego określenia zakresu i sposobu implementacji pracy magisterskiej zdecydowano się na wprowadzenie kilku założeń. Najważniejsze z nich dotyczą procesu detekcji uśmiechu. Przede wszystkim, uśmiechy nie powinny być rozróżniane jako prawdziwe i sztuczne (wspomniane wcześniej uśmiechy Duchenne oraz Pan Am). Kolejnym założeniem jest skupienie się na detekcji przede wszystkim uśmiechów otwartych ze względu na trudności z ustaleniem uniwersalnego progu pewności dla uśmiechów zamkniętych – należy jednak spróbować wykrywać oba typy uśmiechów. Aplikacja powinna umożliwić wykrywanie wielu uśmiechów na raz, odpowiednio radząc sobie z ich zliczaniem.

W ramach pracy magisterskiej należy zaimplementować usługi różnego rodzaju – zarówno oparte o klasyfikatory, jak i te wykorzystujące punkty kluczowe twarzy do określenia uśmiechu. Użytkownik powinien w łatwy sposób móc przełączać się pomiędzy różnymi implementacjami. Należy dobrać odpowiedni rodzaj klasyfikatorów do prawidłowej pracy usług.

Testy aplikacji powinny odbyć się w trzech etapach. Pierwszy, lokalny, powinien przetestować działanie usług wykrywających uśmiech na wybranych bazach twarzy dostępnych w Internecie. Drugi etap powinien obejmować testy usług przy pomocy aplikacji webowej, dzięki czemu w testach może wziąć udział większa liczba osób (co pozwala na przetestowanie działania mechanizmów detekcji w różnych warunkach). Ostatnim krokiem testów powinny być testy wdrożeniowe.

### **2.7.2. Architektura systemu**

Zgodnie z przedstawionymi założeniami aplikację zdecydowano się podzielić trzy główne moduły, co przedstawiono na Rys. 11.



Rys. 11. Koncepcja architektury systemu

Pierwszy z nich, będący rdzeniem całej aplikacji, stanowi logika wykrywania uśmiechów oraz połączeń z bazą danych. Wydzielenie jej do niezależnej części systemu pozwoli na implementowanie wszystkich głównych funkcjonalności w dowolnej innej aplikacji. Technologią, w której zrealizowana jest warstwa usług jest Java 8, która umożliwia także dołączanie i wykonywanie bibliotek dynamicznych napisanych w języku C, co może być wykorzystane w celu przyspieszenia pracy z usługami do przetwarzania obrazu.

Ponieważ w aplikacji nie przewiduje się przechowywania dużej ilości powiązanych ze sobą złożonymi relacjami danych, zdecydowano się na wykorzystanie MongoDB – bazy *no-sql* (*not only SQL*), opartej o przechowywanie dokumentów. Zaletą takiego rozwiązania jest jego prostota oraz szybkość – dzięki zrezygnowaniu z transakcyjności oraz wielu innych elementów bazy relacyjnej można uzyskać nawet dziesięciokrotne przyspieszenie w stosunku do bazy Oracle [4].

Aplikacja desktopowa realizowana jest w technologii Java 8, a jej głównym celem jest wyświetlanie obrazu z kamery i zliczanie uśmiechów w czasie rzeczywistym. Elementem wyróżniającym aplikację desktopową na tle webowej jest fakt prezentowania efektów zachęcających użytkownika do uśmiechu – dzięki wbudowanej warstwie usług jest znacznie wydajniejsza niż jej internetowy odpowiednik, co pozwala na rozbudowanie funkcjonalności.

Aplikacja webowa składa się z dwóch części: frontendu oraz backendu. Pierwsza z nich zbudowana jest w oparciu o framework *Angular 1.6* i pozwala na wykonanie testów bibliotek

do wykrywania uśmiechu. Dodatkowo powinna ona umożliwiać pobranie aplikacji desktopowej w zależności od systemu operacyjnego użytkownika oraz wyświetlać zebrane statystyki w postaci przejrzystych wykresów (np. prezentujących wyniki z testów usług). Część backendu zrealizowana jest w technologii *Java EE* i udostępnia usługi REST-owe wykorzystywane w części frontendu.

### 3. IMPLEMENTACJA ROZWIĄZANIA

#### 3.1. Przegląd istniejących bibliotek do rozpoznawania twarzy i emocji

Koncepcja rozpoznawania emocji użytkownika na podstawie obrazu z zainstalowanej kamery w jego systemie nie jest nowa – na rynku istnieje mnóstwo różnych bibliotek oferujących mechanizmy rozpoznawania twarzy i emocji. Problem wykrywania uśmiechu w czasie rzeczywistym wymaga jednak szybkiego działania algorytmu, aby aplikacja działała płynnie i stabilnie. Podczas poszukiwań biblioteki do rozpoznawania emocji skupiono się zatem nie tylko na oferowanych przez nią funkcjonalnościach, ale przede wszystkim na wysokiej wydajności jej działania.

Pośród znalezionych bibliotek wybierano takie, które da się użyć w języku programowania Java, ze względu na postawione założenia i znajomość tego języka przez autora pracy magisterskiej. Ze względu na konieczność przetwarzania w czasie rzeczywistym oraz potrzebę wysokiej wydajności, szczególnie interesujące były biblioteki napisane w języku C, które można było wykorzystać jako bibliotekę dynamiczną (w postaci plików *.dll* dla systemów rodziny Windows oraz *.so* dla systemów Unixowych).

##### 3.1.1. OpenCV

OpenCV jest darmową (także dla zastosowania komercyjnego) biblioteką napisaną w języku C oraz opracowaną przez firmę Intel. Zawiera w sobie wiele skomplikowanych mechanizmów do analizy i transformacji obrazu, jednak jej główną funkcjonalnością (pod którą została zoptymalizowana dla procesorów Intel) jest detekcja oraz rozpoznawanie obiektów. OpenCV zostało podzielone na kilka modułów, z których najważniejsze to *core* (zawiera modele danych i podstawowe funkcje, jest współdzielony przez pozostałe moduły), *highgui* (wbudowane wsparcie do przechwytywania obrazów w czasie rzeczywistym) oraz *imgproc* (zaawansowane funkcjonalności do przetwarzania obrazów, m.in. przekształcenia, ustawienie skali szarości).

Dużą zaletą biblioteki jest możliwość wykorzystania jej jako biblioteki dynamicznej dla wielu różnych języków programowania – twórcy udostępnili tzw. *wrappery* dla m.in. Pythona, Javy, C#, Perla oraz Matlaba. Dzięki takiemu podejściu można z powodzeniem uruchomić ją na systemach nie tylko stacji komputerowych, ale także urządzeń mobilnych. Na liście wspieranych systemów znajdują się m.in. Windows, Linux, MacOS, Android, iOS oraz BlackBerry. Od końcówki roku 2013 biblioteka posiada natywne API dla języka Java ([6]), wcześniej ten cel spełniała biblioteka JavaCV.

Działanie biblioteki opiera się o zastosowanie klasyfikatorów Haara, które wykorzystuje do wszystkich operacji detekcji fragmentów obrazu. Ze względu na popularność w sieci można znaleźć wiele gotowych klasyfikatorów. Twórcy biblioteki udostępnili zbiór gotowych klasyfikatorów ([3]), jednak znaleźć można również ciekawe typy klasyfikatorów na stronach tworzonych przez społeczność ([5]). Choć większość z nich działa poprawnie, kilku sporo



brakuje do ideału – przykładem może być dostarczony przez twórców klasyfikator uśmiechu, który podczas testów bardziej sprawiał wrażenie klasyfikatora ust.

Największą zaletą tej biblioteki okazuje się być jej wydajność i szybkość działania – dzięki wsparciu sprzętowemu na procesorach rodziny Intel oraz zastosowaniu szybkich klasyfikatorów biblioteka dobrze radzi sobie z procesowaniem wideo w czasie rzeczywistym.

### 3.1.2. *OpenIMAJ*

OpenIMAJ jest darmową biblioteką open-source w całości napisaną w Javie, dzięki czemu jest niezwykle łatwa w wykorzystaniu – wystarczy dodać ją jako zależność projektu np. przy pomocy *Mavena* [11], nie trzeba też łączyć żadnych bibliotek dynamicznych. Przez swoją prostotę w wykorzystaniu biblioteka nieco słabszymi wynikami wydajnościowymi – mimo wszystko implementacja funkcjonalności w Javie jest wolniejsza niż w języku C.

Dużą zaletą biblioteki jest rozbudowane API. Oprócz nieprzydatnych w niniejszej pracy funkcji do zarządzania obrazem z kamery oraz dźwiękami zawiera ono wiele funkcji do przetwarzania obrazów, jak np. modyfikacja kanałów obrazu, wykrywanie krawędzi oraz metody do rysowania na obrazie, zarządzanie histogramem obrazu czy też porównywanie obrazów (wraz z ustaleniem punktów kluczowych odpowiadających na drugim obrazie).

Najważniejszą funkcjonalnością biblioteki jest detekcja zadanych elementów przy pomocy klasyfikatorów Haara. Choć OpenIMAJ posiada działający dosyć sprawnie własny klasyfikator twarzy, można także załadować kaskadę z pliku – obsługiwany jest m.in. ten sam format, co w przypadku biblioteki OpenCV, dzięki czemu wybór klasyfikatorów jest równie urozmaicony. Warto wspomnieć także o wykrywaniu punktów kluczowych twarzy – twórcy biblioteki udostępnili bardzo łatwy w wykorzystaniu mechanizm pozwalający na uzyskanie pozycji oczu, nosa oraz ust (łącznie 9 punktów).

### 3.1.3. *FaceSDK*

FaceSDK jest jedną z komercyjnych bibliotek udostępnionych przez firmę Luxand [19]. Firma udostępnia możliwość otrzymania 30-dniowego, bezpłatnego klucza do korzystania z biblioteki (bez niego jakiegokolwiek korzystanie z metod wystawionych przez twórców jest niemożliwe). FaceSDK jest wielopatformowe – do pobrania są wersje skompilowane m.in. dla Androida, iOS, Linuxa oraz Windows (w wersjach 32- i 64-bitowych). Dużą zaletą jest także możliwość jej wykorzystania w formie dynamicznej biblioteki w wielu różnych językach: na liście wspieranych widnieje między innymi C#, Java oraz Objective C.

Biblioteka Luxandu opiera się o mechanizm wykrywania punktów kluczowych twarzy – oprócz informacji o położeniu samej twarzy, udostępnia ona pozycję ponad 70 punktów, co pozwala między innymi na określenie miejsca oczu, brwi, ust, nosa i konturów twarzy.

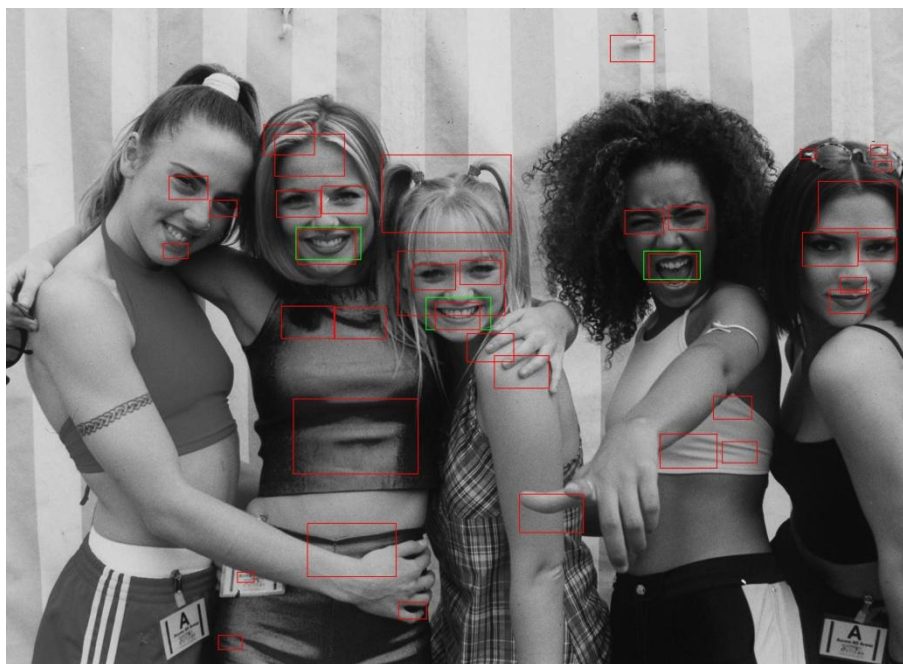
FaceSDK jest zbiorem wielu przydatnych funkcji przedstawionych w formie wygodnego API nie tylko do detekcji twarzy, ale też ich rozpoznawania. Na podstawie wykrytych punktów kluczowych twarzy, korzystając z wbudowanych metod, można w bardzo łatwy sposób otrzymać

informacje o płci osoby, do której należy twarz, czy osoba ma otwarte oczy oraz *stopień* uśmiechu danej osoby. Ostatni wymieniony parametr jest bardzo ważny w kontekście niniejszej pracy magisterskiej – dzięki skali liczb rzeczywistych z zakresu  $<0, 1>$  można w bardzo dokładny sposób stwierdzić, czy dana osoba się uśmiecha. Jest to znaczna przewaga nad rozwiązaniami opartymi o klasyfikatory, które zwracają prostą informację: *tak* lub *nie*. Takie rozwiązanie pozwala także na ustalenie progu uśmiechania się, od którego będzie zależeć zwiększanie licznika uśmiechów.

### 3.2. Detekcja uśmiechu

Detekcja uśmiechów na obrazie w czasie rzeczywistym wymaga przetwarzania każdej klatki filmu osobno. Na każdej z nich znajdować się może wiele użytkowników, jednak nie wszyscy muszą się uśmiechać.

Wyszukiwanie ust w stanie uśmiechu może nie wystarczyć – biblioteki oparte o mechanizm klasyfikatorów Haara informacje posiadają wysokie ryzyko wystąpienia tzw. *false alarmów* (np. detekcja uśmiechów na ścianie) – częstym problemem jest uznawanie oczu jako ust, co przedstawiono na przykładzie Rys. 12. Przedstawia on pięć uśmiechniętych kobiet z popularnego zespołu *Spice Girls*.



Rys. 12. Przykład działania klasyfikatora uśmiechu przy użyciu OpenCV

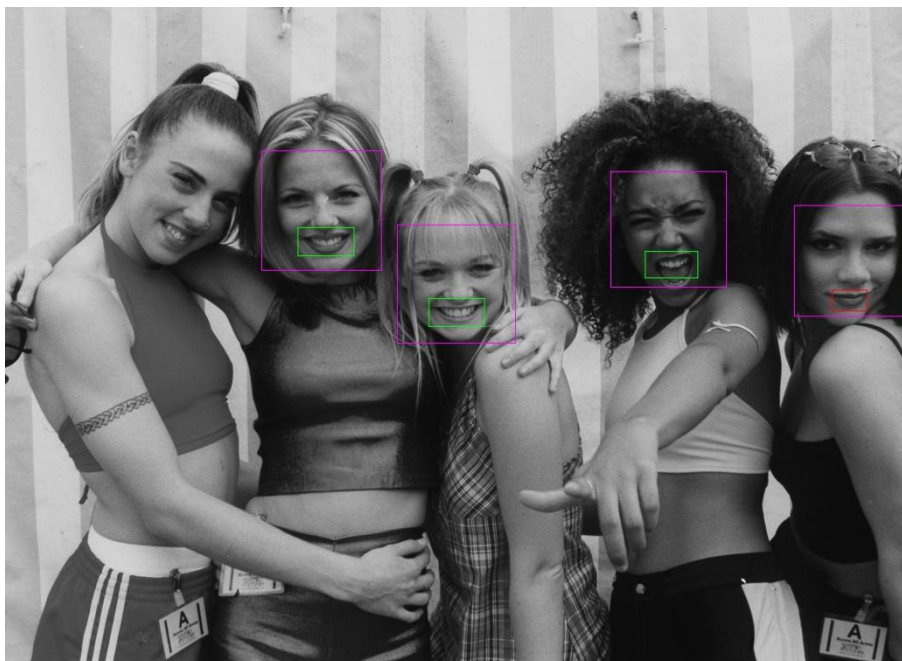
Zielone ramki na zdjęciu oznaczają wykryte uśmiechy, natomiast czerwone – wykryte usta ogółem.

Jak można zauważyć, o ile klasyfikator uśmiechu działa w poprawnie (wykryte zostały tylko faktyczne usta w stanie uśmiechu), to klasyfikator ust ogółem pozostawia wiele do życzenia – jako usta zostały oznaczone nie tylko oczy, ale także czoło, czubek nosa czy broda, nie wspominając o całkowicie błędnych odpowiedziach, jak na przykład kawałek

bluzki czy ściany. Łącznie wykryto 41 ust (3 w stanie uśmiechu), pomimo istnienia tylko pięciu osób na zdjęciu – część ramek występuje na tyle bardzo blisko siebie, że zostały ze sobą nałożone.

Aby uniknąć takiej sytuacji, należy wprowadzić dodatkowe ograniczenia do detekcji uśmiechu. Warto zauważyć, że detekcja twarzy jest dużo dokładniejsza – testowane biblioteki w większości przypadków znajdują ją prawidłowo, dlatego też można wykorzystać ten fakt podczas wyszukiwania uśmiechu. Jednym z podejść wykorzystujących ten pomysł jest sposób opisany przez Amine Sehiliego na swoim blogu [2]. Twierdzi on, że należy w pierwszej kolejności wyszukać twarze na obrazie, następnie wewnątrz nich odnaleźć nos - wówczas wystarczy wyszukać usta, które powinny znajdować się tuż pod nim. Takie podejście wymaga jednak zastosowania klasyfikatora nosa, co w czasie rzeczywistym może powodować większe opóźnienia w procesie detekcji uśmiechów i niekoniecznie oferować znacząco lepszą dokładność.

Chcąc uniknąć stosowania trzeciego klasyfikatora można uprościć proces wyszukiwania ust – można założyć, że usta powinny znajdować się jedynie w dolnej części twarzy. Wyniki takiego sposobu zostały zaprezentowane na Rys. 13. Można na nim zobaczyć, że usługa poprawnie odnalazła pozycję ust dla każdej znalezionej twarzy (zaznaczonej na fioletowo).



**Rys. 13. Przykład detekcji uśmiechu dla połączonych klasyfikatorów**

Od razu można zauważyć różnicę – tym razem klasyfikator wykrył 4 usta, z czego trzy są w stanie uśmiechu. Problemem w wykorzystaniu takich usług są twarze, których algorytm nie był w stanie wykryć (np. obrócona twarz kobiety z lewej strony) – w takim wypadku potencjalne uśmiechy nie zostaną pokazane. Wydaje się jednak, że nie jest to błąd w działaniu usługi – skoro nie jesteśmy w stanie zweryfikować, czy usta należą do jakiegokolwiek twarzy, to nie mamy żadnej pewności, że są to prawdziwe usta. Dodatkowo, uśmiech kobiety z prawej

strony nie został wykryty – stało się tak niezależnie od sposobu implementacji usługi, przyczyną jest jakość wykorzystanego klasyfikatora uśmiechu.

Warto także zauważyć, że w podobny sposób działa biblioteka *FaceSDK* – zanim zbierze informacje o stopniu uśmiechu, wymaga najpierw detekcji twarzy.

### **3.3. Sposoby implementacji usług do detekcji uśmiechów**

Podczas tworzenia usług służących do wykrywania uśmiechów na obrazie założono, że każda z nich implementować będzie wspólny interfejs, dzięki czemu można w łatwy sposób przełączać się pomiędzy różnymi implementacjami. Najważniejsza metoda wspomnianego wcześniej interfejsu zwraca listę wykrytych twarzy na pojedynczym zdjęciu. Każda twarz posiada swoją pozycję (położenie lewego górnego rogu), rozmiar (szerokość i wysokość) oraz informacje dotyczące ust, jeżeli je wykryto. Usta oprócz danych analogicznych do twarzy zawierają także pole *pewność uśmiechu*, przechowujące liczbę rzeczywistą i pozwalające na określenie, czy znajdują się w stanie uśmiechu.

W ramach pracy magisterskiej zaimplementowano cztery rodzaje usług, każdą z nich opisano szerzej w odpowiednim podrozdziale.

#### **3.3.1. Usługa oparta o OpenCV**

Usługa została napisana w oparciu o OpenCV w wersji 3.3 i wymaga dołączenia bibliotek dynamicznych. Korzysta ona z trzech klasyfikatorów: klasyfikatora twarzy widzianej „od przodu” typu Haar [20] oraz LBP [25], klasyfikatora ust w stanie zwykłym [21] oraz klasyfikatora uśmiechu [22]. Algorytm, zgodnie z którym pracuje usługa, wygląda następująco:

1. Znajdź wszystkie twarze na zdjęciu
2. Wewnątrz twarzy w dolnej części poszukaj uśmiechu
3. Jeżeli nie znaleziono uśmiechu, znajdź usta w stanie zwykłym

Przed przekazaniem zdjęcia do usługi zdecydowano się na ustawienie mu skali szarości oraz wyrównanie histogramu (metody *Imgproc.cvtColor* oraz *Imgproc.equalizeHist* zdefiniowane w OpenCV), co zwiększa dokładność pracy klasyfikatora.

Wyszukiwanie ust pomimo braku znalezienia uśmiechu jest konieczne ze względu na prezentowanie efektów zachęcających do uśmiechu – niektóre z nich powinny znać pozycję ust.

Przy wykrywaniu wskazanych przez klasyfikator elementów na obrazie OpenCV pozwala na ustawienie szeregu parametrów konfigurujących pracę biblioteki. Najważniejsze z nich to skala, minimalna liczba sąsiadów (czyli liczba wykrytych obiektów w pobliżu), tryby detekcji (np. „wyszukaj największy”, „skaluj obraz” itd., wykorzystywane tylko w przypadku starszych formatów kaskad), minimalny rozmiar obiektu oraz maksymalny rozmiar obiektu. Wszystkie opcje ustawiono na podstawie eksperymentów oraz analogicznych fragmentów kodu znalezionych w różnego rodzaju poradnikach.

Dla detekcji twarzy czynnik skali ustawiono na wartość 1.05, minimalny rozmiar obiektu na (30,30), natomiast maksymalny rozmiar na wymiary zadanego zdjęcia. Minimalny rozmiar twarzy jest ustawiony na niewielką wartość stałą ze względu na fakt, że na obrazie może znajdować się ich wiele – każda z nich w różnej odległości (a zatem i rozmiarze), przez co trudno jest zdefiniować bezpieczne minimum w taki sposób, aby prawidłowo wykrywać wszystkie twarze. Detekcja uśmiechów dla twarzy mniejszych niż 30 px będzie nieefektywna ze względu na niewielki rozmiar ust i przez to małą dokładność wykrywania. Ustawienie niezerowej wartości ma znaczny wpływ na wydajność algorytmu (średni czas procesowania pojedynczego zdjęcia skrócił się o ok. 50-100 ms) – wyklucza ono bowiem wyszukiwanie wszystkich fragmentów mniejszych niż 30 px.

Przy wykrywaniu ust lub uśmiechu sprawa wygląda nieco inaczej, ponieważ dany jest obszar twarzy, do której usta mają należeć. Dzięki temu minimum można ustalić jako  $\frac{1}{8} \times \frac{1}{10}$  rozmiarów twarzy – maksimum natomiast jako połowę jej szerokości i  $\frac{1}{3}$  jej wysokości. Dodatkowo licznik skali ustawiono na wartość 1.05, a minimalną liczbę sąsiadów na 5.

Ponieważ klasyfikatory nie pozwalają na dokładne określenie dokładności detekcji uśmiechu, w przypadku wykrycia uśmiechu pewność ustawiono na 1, natomiast w przypadku wykrycia ust wartość tego parametru wynosi 0.

### 3.3.2. Usługa oparta o OpenIMAJ

Usługa oparta o OpenIMAJ w wersji 1.3 nie wymaga dołączenia żadnych bibliotek zewnętrznych, dzięki czemu powinna działać na każdym systemie operacyjnym z maszyną Javy. Ponieważ API tej biblioteki pozwala na korzystanie z niej analogicznie w stosunku do OpenCV, ich implementacje są bardzo do siebie podobne.

OpenIMAJ także wykorzystuje trzy klasyfikatory – pierwszy, wbudowany, to klasyfikator wykrywania twarzy „od przodu”. Kolejne są wspomnianymi wcześniej klasyfikatorami uśmiechu oraz ust.

Algorytm wykrywania uśmiechów i ustawiania pewności wykrycia jest identyczny z poprzednią implementacją. Jediną różnicą jest fakt, że OpenIMAJ nie przyjmuje tak rozbudowanej liczby parametrów – wspierany jest jedynie minimalny rozmiar obiektów. Ponieważ jest on przyjmowany jedynie jako konstruktor detektora, ustawiony jest zawsze na wartość (30,30), z tych samych względów, co w przypadku usługi OpenCV.

### 3.3.3. Usługa oparta o FaceSDK

Usługa wykorzystuje FaceSDK w wersji 6.2 i wymaga do pracy nie tylko dołączenia dynamicznych bibliotek zewnętrznych, ale także klucza z licencją – bez niego usługa zwraca zawsze puste dane. Do detekcji uśmiechu są wykorzystywane wyłącznie wbudowane funkcje biblioteki. Ponieważ FaceSDK zwraca zawsze informacje o położeniu i stanie ust, algorytm do detekcji uśmiechu jest znacznie uproszczony:

1. Wykryj wszystkie twarze na zdjęciu

## 2. Dla każdej twarzy wykryj jej cechy i odczytaj położenie oraz stan ust

Odczytanie pozycji i rozmiaru uśmiechu odbywa się poprzez pobranie informacji o punktach kluczowych położonych na ustach (lewy górny róg ukryty pod stałą o nazwie *FSDK.FSDKP\_MOUTH\_LEFT\_TOP* oraz prawy dolny, którego dotyczy stała o nazwie *FSDK.FSDKP\_MOUTH\_RIGHT\_BOTTOM*). Wartość szerokości i wysokości otrzymano poprzez odjęcie od siebie obu punktów.

Wartość pewności uśmiechu pobrano z udostępnionej przez bibliotekę cechy o nazwie *Expression*, która dostępna jest po odnalezieniu punktów kluczowych twarzy (czyli też po próbie odczytania pozycji ust). Doświadczalnie sprawdzono, że wartość w okolicy 0.40 oznacza początek uśmiechu zamkniętego – dotyczy to jednak wyłącznie twarzy autora pracy magisterskiej.

Proces detekcji twarzy w FaceSDK można dodatkowo skonfigurować poprzez wykorzystanie funkcji *FSDK.SetFaceDetectionParameters*, która przyjmuje trzy parametry: *HandleArbitraryRotations*, określający, czy biblioteka ma wykrywać twarze obrócone o maksymalnie 30 stopni, *DetermineFaceRotationAngle*, który sprawia, że dla każdej twarzy będzie obliczony kąt obrotu w płaszczyźnie czołowej (przydatny przy zaawansowanych pracach z punktami kluczowymi) oraz *InternalResizeWidth*, który określa szerokość, do której zostanie przeskalowane każde zdjęcie [23]. Pierwszy parametr zdecydowano się włączyć, pomimo niewielkiego kosztu w wydajności. Drugi ze względu na fakt, że punkty kluczowe poza rogami ust nie są wykorzystywane, ustawiono na *false*. Wartość ostatniego parametru pozostawiono domyślną (384 px), stosując się do rady twórców.

### 3.3.4. Usługa powstała poprzez połączenie OpenCV oraz FaceSDK

Ostatnia zaimplementowana usługa powstała poprzez obserwacje wad i zalet pozostałych usług. Zauważono, że OpenCV w bardzo dokładny sposób pozwala na określenie pozycji i rozmiaru twarzy (rzadko wzbudzając tzw. *false alarmy* – czyli wskazując twarze w miejscach, w których ich nie ma), ale ma większe problemy z detekcją ust i uśmiechu. Dodatkowo, przez fakt, że w przypadku braku uśmiechu wykorzystywany jest trzeci klasyfikator, usługa ta niepotrzebnie traciła czas na ponowne przeszukiwanie zdjęcia. Drugą obserwacją był fakt, że FaceSDK zdarza się wykrywać dodatkowe twarze w losowych miejscach, ale za to bardzo dokładnie pozwala na określenie stopnia pewności detekcji uśmiechu.

Korzystając z wymienionych wcześniej obserwacji zdecydowano się na zaimplementowanie czwartej usługi, łączącej zalety OpenCV oraz FaceSDK. Konfiguracja obu bibliotek jest analogiczna do podstawowych wersji. Algorytm pracy tej usługi wygląda następująco:

1. Przy pomocy OpenCV wykryj twarze na zdjęciu
2. Przy pomocy FaceSDK wyszukaj punkty kluczowe twarzy w wykrytych przez OpenCV fragmentach obrazu

Proces dodatkowo przyspieszono poprzez wyłączenie detekcji twarzy przez FaceSDK. Mechanizm taki pozwalał na większą pewność w detekcji twarzy, lecz powodował zbyt duży spadek wydajności.

### **3.4. Przetwarzanie obrazu w czasie rzeczywistym**

Algorytmy wykorzystywane w pracy magisterskiej realizują detekcję uśmiechu, ale nie jego rozpoznawanie. Dzięki takiemu podejściu można znacząco przyspieszyć działanie aplikacji – zapamiętywanie uśmiechów i przypisywanie ich konkretnej osobie jest znacznie bardziej złożone czasowo. Ponieważ jednak liczenie uśmiechów ma odbywać się w czasie rzeczywistym, występuje potrzeba kojarzenia uśmiechów na poszczególnych klatkach.

#### **3.4.1. Wykrywanie nowego uśmiechu**

Pojedynczy uśmiech trwa więcej niż jedną klatkę – prosty sposób ze zliczaniem uśmiechów na każdym kadrze nie spełniłby zatem swojej roli, ponieważ jeden faktyczny uśmiech byłby co każdą klatkę swojego trwania rozpoznawany jako nowy. W związku z tym powstają dwa problemy: rozpoznanie na kilku sąsiadujących klatkach tego samego uśmiechu oraz uzyskanie informacji, czy dany uśmiech jest nowy i powinien zostać zliczony.

Dodatkową trudnością przy realizacji rozwiązania kojarzenia uśmiechów może być losowość algorytmów – niedokładność lub nawet niewykrycie uśmiechu na jednym kadrze (np. przez zmianę oświetlenia czy szum na obrazie kamery). Użytkownik może nieznacznie się poruszać, nie można więc założyć, że uśmiech będzie zawsze w tym samym miejscu. Także same usta w czasie uśmiechu często zmieniają swój kształt i rozmiar. Algorytm powinien być odporny na drobne zakłócenia w detekcji twarzy i uśmiechu, w innym przypadku liczniki będą niedokładne.

Podczas analizy obrazu w czasie rzeczywistym należy zapamiętać  $n$  sąsiednich klatek wstecz - im większa liczba klatek, tym większa odporność na losowość. Wartość tę należy tak dobrać, by osiągnąć wymaganą jakość – trzeba mieć jednak na uwadze, że jeśli będzie ona zbyt duża, mogą pojawiać się problemy z oznaczeniem uśmiechu jako nowy. Domyślnie parametr  $n$  ustawiono na zapamiętywanie 5 poprzednich kadrów. Dla każdej nowej klatki trzeba wykonać detekcję uśmiechów, a następnie sprawdzić, czy na jednej z zapamiętanych klatek znajduje się już uśmiech podobny do istniejącego.

Pierwszym podejściem do szukania podobieństw wykrytych osób było sprawdzenie pozycji i rozmiaru ust. Podczas testów takiego rozwiązania szybko okazało się, że detekcja uśmiechu zachowuje się w sposób dosyć chaotyczny – nawet sąsiednie klatki mogą mieć dużą różnicę w rozmiarach wykrytego uśmiechu. Zgodnie z taką obserwacją nie można sprawdzać podobieństwa na podstawie wykrytych ust - lepszym rozwiązaniem wydaje się być porównanie twarzy, ponieważ nie tylko dane zwracane przez biblioteki są dosyć stabilne, ale także jej wymiary nie ulegają drastycznej zmianie podczas wykonywania uśmiechu. W związku z takim

założeniem, logicznym wydaje się sprawdzanie podobieństwa pomiędzy uśmiechami na podstawie twarzy, do których należą.

Samo sprawdzanie podobieństwa nie musi być skomplikowane. Można założyć, że pomiędzy dwiema sąsiednimi klatkami pozycja twarzy użytkowników będzie bardzo podobna – w przypadku gwałtownych ruchów większość kamerek internetowych zwraca rozmyty obraz, który i tak nie mógłby być zinterpretowany. Wystarczy zatem proste sprawdzanie, czy pozycja i rozmiar twarzy jest podobny – z dokładnością do  $\frac{1}{4}$  wymiarów twarzy. Algorytm sprawdzający podobieństwo wygląda zatem następująco:

$$d_x = \frac{(f_1^{szerokość} + f_2^{szerokość})}{8}, d_y = \frac{(f_1^{wysokość} + f_2^{wysokość})}{8}$$

$$podobieństwo_x \Leftrightarrow |f_1^x - f_2^x| < d_x \wedge |f_1^{szerokość} - f_2^{szerokość}| < d_x$$

$$podobieństwo_y \Leftrightarrow |f_1^y - f_2^y| < d_y \wedge |f_1^{wysokość} - f_2^{wysokość}| < d_y$$

gdzie:  $f_1$  – obecnie wykryta twarz,  $f_2$  – twarz na poprzednim kadrze

Posiadając informację o tym, czy twarz jest podobna, wystarczy porównać stany ich uśmiechów – nowy uśmiech występuje tylko i wyłącznie wtedy, kiedy obecnie wykryta twarz się uśmiecha oraz:

- a) żadna podobna twarz ze wcześniejszych kadrów się nie uśmiecha
- b) nie udało się znaleźć żadnej podobnej twarzy na zapamiętanych kadrach

Takie rozwiązanie znacząco pomaga w sytuacjach, w których biblioteki mają problemy z detekcją (twarz lub usta pojawiają się i znikają na sąsiadujących klatkach) – w takim przypadku ciągle uśmiechanie się zostałoby policzone jako kilka mniejszych uśmiechów. Dzięki opisanemu wcześniej mechanizmowi zostanie znaleziona poprzednia podobna twarz w ciągu  $n$  klatek, co pozwala ominąć sztucznie utworzone nowe uśmiechy.

Niestety, powyższy algorytm okazał się być niewystarczający w przypadku, w którym biblioteka wahała się, czy wykryta twarz się uśmiecha. Kiedy uśmiech znajdował na granicy progu wykrycia, mechanizm na przemian określał stan poprzedniej twarzy jako brak uśmiechu, a na najnowszej klatce jako nowy uśmiech, co również powodowało jego wielokrotne policzenie.

Problem ten został rozwiązany poprzez wprowadzenie progu czasowego pomiędzy wykryciem nowego uśmiechu. Wystarczy zapamiętać moment rozpoczęcia uśmiechu dla każdej twarzy osobno – nowy uśmiech zostanie policzony tylko i wyłącznie wtedy, kiedy różnica pomiędzy momentami wykrycia jest większa niż 700 ms.

### 3.4.2. Prezentowanie efektów zachęcających do uśmiechu

Prezentowanie efektów wymaga dodatkowej wiedzy – sama informacja o nowym uśmiechu nie wystarczy. Aby wzbogacić możliwości pokazywania prostych efektów graficznych, zdefiniowano cztery stany uśmiechu:

- a) Brak uśmiechu – użytkownik przed chwilą się nie uśmiechał, teraz też nie



- b) Rozpoczęcie uśmiechu – użytkownik przed chwilą się nie uśmiechał, teraz się uśmiecha
- c) Trwanie uśmiechu – użytkownik przed chwilą się uśmiechał, teraz też się uśmiecha
- d) Zakończenie uśmiechu – użytkownik przed chwilą się uśmiechał, teraz się nie uśmiecha

Posiadając  $n$  klatek wstecz, wystarczy sprawdzić, czy aktualny uśmiech jest nowy lub czy istniał już wcześniej. Jeżeli wykryty uśmiech został odnaleziony na jednej z zapamiętanych klatek, wystarczy porównać ją z obecnie wyświetlaną w sposób podany powyżej. Jeżeli uśmiechu nie udało się skojarzyć, wystarczy sprawdzić jego stan:

- a) Użytkownik się uśmiecha – rozpoczęcie uśmiechu
- b) Użytkownik się nie uśmiecha – brak uśmiechu


Efekty wyświetlane podczas wyświetlania obrazu w czasie rzeczywistym można podzielić na dwie grupy: zachęcające do uśmiechu oraz nagradzające uśmiech. W przypadku tych pierwszych konieczne jest wykrycie twarzy, które w chwili obecnej się nie uśmiechają, natomiast te drugie mogą być wyświetlane zarówno w samym momencie uśmiechania się, jak i wówczas, kiedy ten proces trwa od dłuższej chwili.

Czas trwania efektów nie jest jednakowy – w zależności od typu może to być wykrycie nowego wydarzenia (np. koniec uśmiechu), upływanie określonego czasu czy też po prostu zakończenie animacji.


### 3.4.3. Lista zrealizowanych efektów




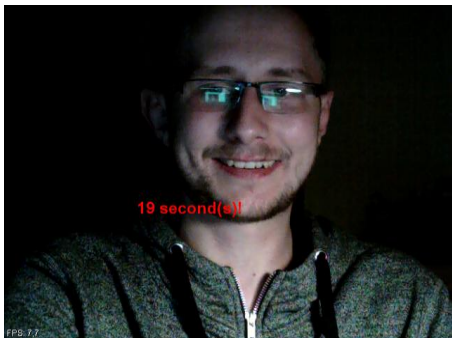
W ramach projektu zrealizowano 9 prostych efektów – listę wszystkich przedstawia *Tabela 1*. Każdy z nich jest skalowany do rozmiaru twarzy – im większa twarz (bliżej kamerki), tym większy jest efekt.

**Tabela 1.** Informacje o zrealizowanych efektach

Nazwa efektu	Przykład efektu	Opis efektu
<b>Efekty zachęcające do uśmiechu</b>		
Aureola		Jeżeli użytkownik się nie uśmiecha, nad jego głową pojawia się pulsująca aureola. Efekt znika po uśmiechnięciu się.

Nazwa efektu	Przykład efektu	Opis efektu
Doklejenie zarostu		Jeżeli użytkownik nie uśmiecha się, w dolnej części jego twarzy pojawia się długa broda lub wąsik. Efekt znika po uśmiechnięciu się.
Serce		Jeżeli użytkownik nie uśmiecha się, obok jego twarzy pojawia się małe, pulsujące serce, które z czasem rośnie. Jeżeli efekt nie przyniesie skutku, po krótkiej chwili pojawi się kolejne serduszko w innym kącie twarzy (aż do łącznej liczby 3 serduszek). Efekt znika po uśmiechnięciu się.
Doklejenie włosów		Jeżeli użytkownik się nie uśmiecha, aplikacja dokleja mu różne rodzaje włosów – afro lub grzywkę w stylu „emo”. Efekt znika natychmiast po uśmiechnięciu się.

Nazwa efektu	Przykład efektu	Opis efektu
		
<b>Efekty nagradzające uśmiech</b>		
Doklejenie ust	  	<p>W trakcie gdy użytkownik się uśmiecha, aplikacja dokleja mu różne rodzaje ust – na liście są kobiece usta z przygryzieniem, otwarte usta z zębami wampira oraz pełne usta ułożone w formie zbliżonej do callusa.</p> <p>Efekt znika natychmiast po przestaniu uśmiechania się.</p>

Nazwa efektu	Przykład efektu	Opis efektu
Obrót twarzy		W momencie rozpoczęcia uśmiechu wycinany jest fragment zawierający twarz użytkownika. Wycięty fragment obraca się i zmienia swój rozmiar aż wyleci poza ekran – wówczas efekt jest usuwany.
Siedzący motylek		W trakcie trwania uśmiechu pojawia się w lewej części ust mały niebieski motylek, który macha skrzydełkami. Efekt znika automatycznie po zaprzestaniu uśmiechania się.
Gwiazdki		W momencie gdy użytkownik uśmiecha się, pojawia się wokół jego twarzy 5 gwiazdek, które następnie poruszają się w górę i robią się przezroczyste (każda w innym tempie). W przypadku całkowitego zniknięcia gwiazdki, w losowym miejscu pojawia się następna.0 Efekt trwa, póki użytkownik się uśmiecha.
Rekordzista		Od momentu rozpoczęcia uśmiechu zostaje naliczany czas uśmiechania się. Wraz ze wzrostem czasu, efekt staje się coraz bardziej intensywny – tekst zmienia kolor i zaczyna się zwiększać. Efekt znika po zakończeniu uśmiechania się.

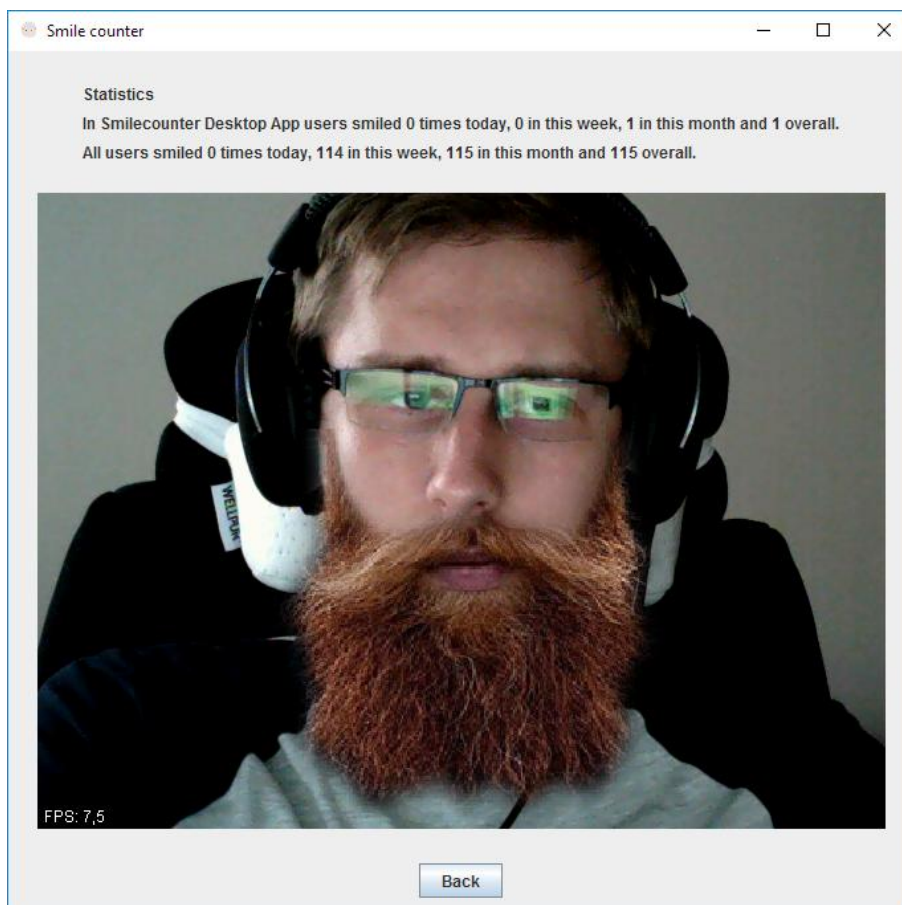
### 3.5. Aplikacja desktopowa

Wersja desktopowa aplikacji została zrealizowana przy pomocy Javy, aby umożliwić uruchomienie jej na możliwie jak największej liczbie systemów operacyjnych. Ponieważ wymaga ona bibliotek zewnętrznych, zostają one dostarczone w paczce w zależności od wybranego systemu. Sama aplikacja jest kompilowana do pliku .jar oraz .exe (tylko na systemy z rodziny Windows).

Podczas uruchamiania aplikacji wyświetlany jest krótki splash-screen (ekran ładowania), podczas którego aplikacja nawiązuje połączenie z bazą danych oraz ładuje biblioteki dynamiczne. Po zakończeniu tego procesu prezentowane jest proste menu główne, z którego użytkownik może przejść do menu ustawień lub ekranu podglądu obrazu z kamery.

#### 3.5.1. Ekran podglądu obrazu z kamery

Ekran podglądu obrazu z kamery jest najważniejszym ekranem aplikacji desktopowej. Wyłącznie podczas pracy na tym ekranie wykorzystywane są usługi wykrywające uśmiech na twarzy użytkownika. Formatka składa się z dwóch głównych części, co zostało zawarte na Rys. 14.



Rys. 14. Ekran podglądu obrazu z kamery w aplikacji desktopowej

Pierwszą częścią ekranu są statystyki wykrytych uśmiechów. Pokazane są tutaj uśmiechy z dzisiaj, zeszłego tygodnia, miesiąca oraz ogólne nie tylko dla wszystkich

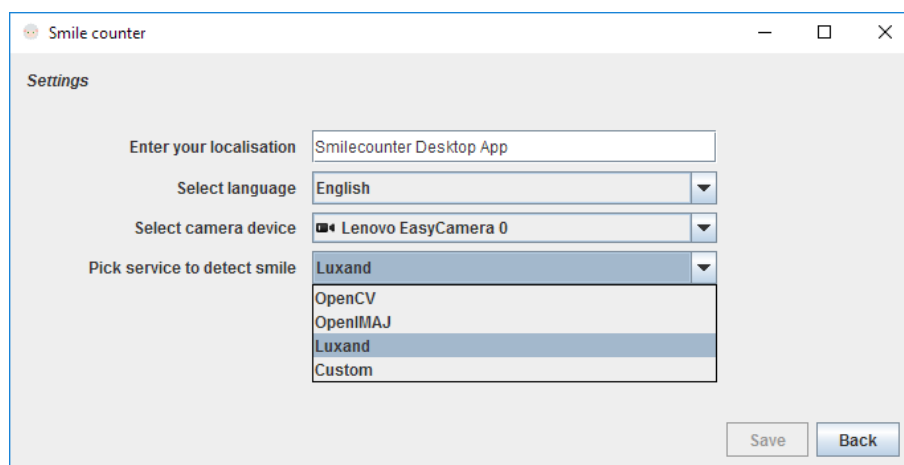


użytkowników, ale także dla lokalizacji, w jakiej została ustawiona aplikacja Smilecounter. Liczniki w statystykach są regularnie odświeżane, dzięki czemu można w łatwy sposób zweryfikować, czy aplikacja prawidłowo rozpoznała uśmiech.

Pozostałą część stanowi obraz z kamery użytkownika. To tutaj prezentowane są efekty zachęcające i wynagradzające uśmiech. Dodatkowo, jeżeli zostało wprowadzone takie ustawienie, prezentowane są wykryte fragmenty twarzy wraz z informacjami o tym, w jakim stanie się ona aktualnie znajduje. Dla uśmiechów zdefiniowano 3 kolory: czerwony, kiedy użytkownik się nie uśmiecha, żółty, kiedy uśmiech jest na pograniczu progu uśmiechania się oraz zielony, kiedy aplikacja uważa, że użytkownik się uśmiecha. W przypadku twarzy w kolorach ramki zawarte są dodatkowe informacje: jeżeli twarz się nie uśmiecha, wyświetlana jest szara ramka, jeżeli użytkownik zaczął się uśmiechać, ramka zmienia kolor na białą. Twarze z wykrytym uśmiechem mają różowy kolor ramki, natomiast w przypadku zakończenia uśmiechu wykorzystano kolor błękitny.

### 3.5.2. Ekran ustawień

Ekran ustawień aplikacji desktopowej zawiera najczęściej zmieniające się ustawienia z punktu widzenia użytkownika. Listę zaimplementowanych opcji pokazuje Rys. 15.



**Rys. 15. Ekran ustawień w aplikacji desktopowej**

Najważniejszym polem do edycji jest combobox z możliwością zmiany podłączonej do komputera kamery oraz usługi, która ma być wykorzystywana podczas detekcji uśmiechów. Dodatkowymi udostępnionymi ustawieniami jest nazwa lokalizacji oraz możliwość zmiany języka.

### 3.5.3. Konfiguracja i sposób uruchomienia aplikacji

Główny sposób konfiguracji odbywa się poprzez dołączenie pliku *smilecounter.config*, zawartego w katalogu z aplikacją. Zawiera on szereg ustawień, wpływających na działanie programu. Listę wszystkich takich parametrów zawiera *Tabela 2*.

**Tabela 2. Lista parametrów konfiguracyjnych dla aplikacji desktopowej**

Parametr	Opis	Wartość domyślna
application.name	Nazwa aplikacji, wyświetlana jako nagłówek programu.	Smile counter
application.availableLanguages	Lista dostępnych języków wykorzystywanych w programie, oddzielonych przecinkiem. Aby język działał prawidłowo, aplikacja musi posiadać pliki z tłumaczeniami wewnątrz siebie. Lista wspieranych języków to: <i>pl</i> oraz <i>en</i> .	pl,en
application.defaultLanguage	Domyślny język dla aplikacji.	en
database.simpleType	Określa, czy aplikacja nie ma łączyć się do bazy danych (prosta implementacja bazy w pamięci)	false
database.storeFaces	Określa, czy aplikacja powinna zapisywać zdjęcia podczas detekcji uśmiechów.	false
database.connectionUrl	Adres do bazy danych MongoDB	mongodb://159.203.96.56:27017/smilecounter
database.refreshDataInterval	Interwał dla odświeżania danych z bazy danych (liczniki uśmiechów na ekranie z kamerą).	10
affective.default.service	Domyślna usługa do detekcji uśmiechu. Lista dostępnych usług to: <i>LUXAND</i> , <i>OPEN_CV</i> , <i>OPENIMAJ</i> oraz <i>CUSTOM</i> .	CUSTOM
affective.luxand.key	Klucz do licencji FaceSDK.	-
affective.libs.path	Ścieżka do niestandardowej lokalizacji z bibliotekami dynamicznymi. Jeżeli nie została podana, aplikacja spróbuje wczytać je z katalogu <i>libs</i> z lokalizacji aplikacji.	-
affective.smileConfidence.threshold	Próg pewności uśmiechu, dla którego zostanie on wykryty jako	0.45

Parametr	Opis	Wartość domyślna
	nowy uśmiech. Wykorzystywany jedynie w przypadku usług opierających się o FaceSDK.	
affective.showDetectedFragments	Określa, czy aplikacja powinna rysować prostokąty zawierające rozpoznane elementy na obrazie (usta oraz twarz). Kolory ramek różnią się w zależności od stanu twarzy: brak uśmiechu – twarz w kolorze szarym, usta czerwone, rozpoczęcie uśmiechu - twarz różowa, uśmiechanie się – twarz biała, usta zielone lub żółte (w zależności od stopnia uśmiechu), zakończenie uśmiechu – twarz niebieska.	false
settings.locationName	Określa domyślną wartość nazwy lokalizacji, wykorzystywanej przy zapisywaniu uśmiechu.	Smilecounter Desktop App

Uruchomienie aplikacji dla usług innych niż OpenIMAJ wymaga dostarczenia bibliotek dynamicznych - domyślnie ładowane są one z katalogu *libs* w katalogu z aplikacją. Opcjonalny jest plik konfiguracyjny *smilecounter.config* (jego parametry zostały opisane wcześniej). Aby uruchomić aplikację, wystarczy wystartować *SmileCounter.exe*. W celu debugowania (zapisania logów aplikacji) udostępniono także aplikację w formacie *.jar*, którą można uruchomić przy pomocy skryptu *run\_jar\_smilecounter.bat*, tworzącego plik *logi.txt* w katalogu z aplikacją.

### 3.6. Aplikacja webowa

Wersja webowa aplikacji została podzielona na dwa niezależne moduły: frontend oraz backend. Część frontendowa została napisana przy pomocy frameworka AngularJS i umożliwia uruchomienie jej w postaci demo – nie wymaga wówczas dostępu do części backendowej i prezentuje statyczne dane, przygotowane na etapie implementacji. Backend został napisany przy użyciu Javy i udostępnia REST-owe endpointy oferujące dostęp do API usług wykrywających uśmiech. Została ona dodatkowo wzbogacona o narzędzie *Swagger*, które wspomaga dokumentację endpointów i oferuje przejrzysty i prosty interfejs do wyświetlania oraz testowania listy endpointów. Zarówno frontend, jak i backend są udostępniane w postaci pliku *.war*, a ich domyślnym kontenerem jest *WildFly 10*.



### 3.6.1. Strona główna

Zadaniem głównego ekranu aplikacji webowej jest poinformowanie użytkownika o tym, czym jest Smilecounter oraz jakie akcje może wykonać w następnej kolejności. Link do strony głównej znajduje się w formie ikonki domu w navbarze, który pojawia się na wszystkich pozostałych ekranach w górnej części strony. Na tej podstronie użytkownik ma możliwość zmiany języka wykorzystywanego w całej aplikacji – może wybrać pomiędzy językiem angielskim (domyślny) lub językiem polskim. Wybór jest zapisywany w ciasteczkach przeglądarki – nie trzeba przełączać języka za każdym razem po wejściu na stronę.



Rys. 16. Ekran główny aplikacji webowej

Jak widać na Rys. 16, najważniejszym elementem strony głównej jest informacja o tym, że Smilecounter służy do zliczania uśmiechów użytkowników. Na tym etapie użytkownik może rozpocząć testowanie usług do wykrywania uśmiechu, pobrać aplikację desktopową

lub obejrzeć listę zdjęć z uśmiechami osób, które wyraziły zgodę na ich zapisanie. Znajduje się tu też licznik uśmiechów, który pokazuje zawsze aktualne dane i na bieżąco się odświeża.

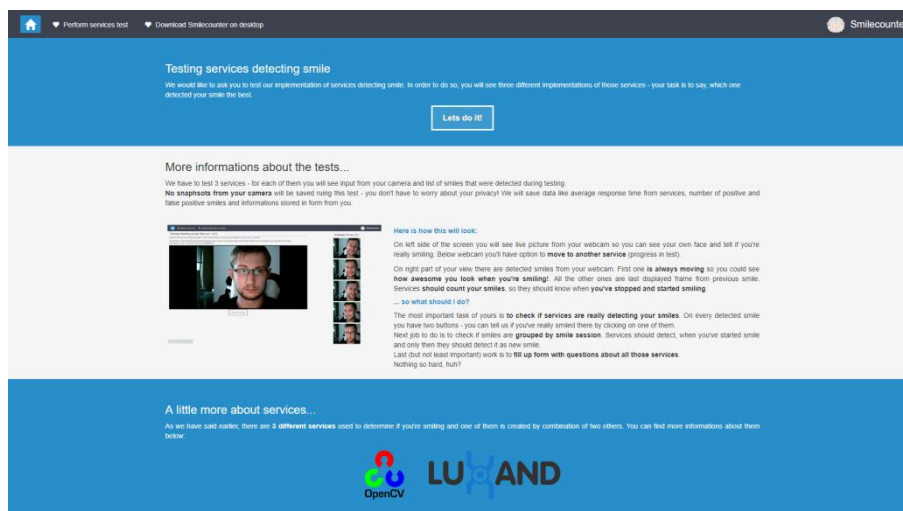
Kolejnym fragmentem strony głównej jest prezentowanie statystyk zebranych podczas zliczania uśmiechów. Znajdują się tu trzy wykresy: pierwszy, liniowy, pokazuje rozkład liczby wykrytych uśmiechów dla każdego dnia ostatniego miesiąca. Kolejny wykres o typie *donut* pokazuje stosunek liczby uśmiechów bez zapisanych uśmiechów do liczby uśmiechów, w których użytkownik zezwolił na zapisanie zdjęcia. Ostatni, a zarazem najciekawszy wykres o typie *radar* pokazuje wyniki przeprowadzonych testów w aplikacji webowej. Dla każdej z trzech testowanych usług prezentowane są następujące statystyki: czas spędzony na testowaniu usługi, liczba wykrytych uśmiechów, liczba uśmiechów oznaczonych przez użytkownika jako prawdziwe, liczba uśmiechów oznaczonych przez użytkownika jako pomyłka usługi oraz czas, jaki usługa potrzebowała na odesłanie odpowiedzi. Każda statystyka została uśredniona względem liczby testów. Najlepszy wynik w danej kategorii jest wyświetlany w nawiasie przy opisie, a wartości na wykresie są zrealizowane w formie procentów do wartości maksymalnej.

Ostatnią sekcją są statystyki informujące o częstotliwości (ostatni dzień, tydzień oraz miesiąc) uśmiechów, lokalizacjach z największą liczbą uśmiechów oraz listą zdjęć użytkowników, którzy wyrazili zgodę na ich zapisanie, posortowane względem daty malejąco.

### 3.6.2. Testowanie usług wykrywających uśmiech

Podczas testowania usług użytkownik ma okazję zapoznać się z trzema implementacjami usług wykrywających uśmiechy: OpenCV, Luxand oraz połączenie obu wspomnianych. Kolejność testowanych usług jest ustalana losowo za każdym razem po wejściu na formatkę testowania.

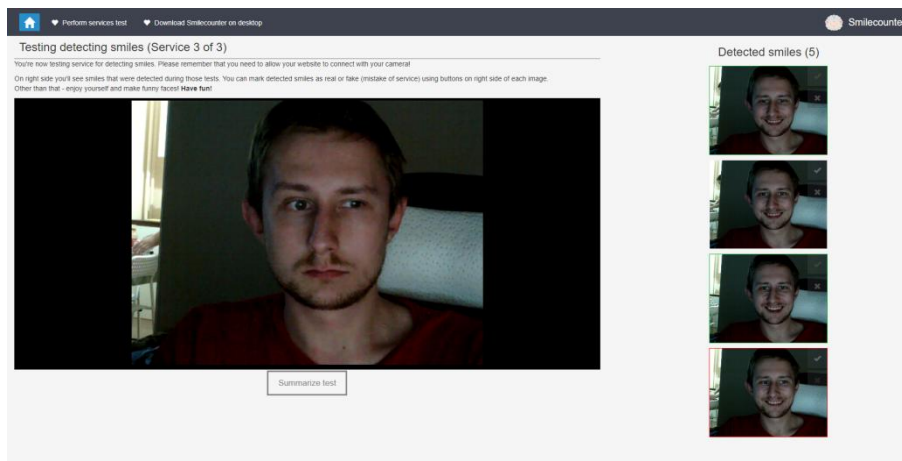
Proces testów składa się z czterech głównych kroków: wprowadzenia do testów, testowania, ekranu z formularzem i wynikami testów oraz ekranu z możliwością zapisania wskazanych zdjęć do bazy danych.



Rys. 17. Ekran wstępny testowania usług wykrywających uśmiech

Rys. 17. prezentuje pierwszy etap testowania usług wykrywających uśmiech. Znajduje się na nim przycisk umożliwiający rozpoczęcie procesu testowania. Poniżej znajduje się krótki opis zabawy wraz z przydatnymi informacjami oraz odpowiedziami na typowe pytania, jakie może zadać użytkownik.

W dolnej części podstrony wyświetlone są informacje o narzędziach, przy pomocy których zrealizowano wykrywanie uśmiechu – OpenCV oraz Luxand. Klikając na logo biblioteki, użytkownik zostaje przekierowany na jej stronę główną.



**Rys. 18. Główny ekran testowania usług wykrywających uśmiech**

Następny, najważniejszy etap procesu testowania usług wykrywających uśmiech został pokazany na Rys. 18. W lewej części ekranu pojawia się aktualny obraz z kamery internetowej oraz przycisk umożliwiający przejście do następnego kroku testów.

W prawej części interfejsu wyświetlana jest lista wykrytych uśmiechów. Usługi rozpoznają trwający uśmiech na opisanych wcześniej zasadach, zatem nowa pozycja pojawia się tylko w przypadku wystąpienia nowego uśmiechu (nie pokazuje się każda klatka ciągłego uśmiechania się). Pierwszy element na liście wyświetlany jest w postaci prostej animacji, złożonej z maksymalnie 7 klatek ostatniego lub aktualnie trwającego uśmiechu. Ze względu na wydajność aplikacji, pozostałe pozycje wyświetlane są jako pojedyncze klatki poprzednich uśmiechów.

Dodatkową akcją, jaką może podjąć użytkownik na tym ekranie jest ocena jakości wykrytego uśmiechu. Po najechnaniu na miniaturkę jednego z nich wyświetlają się dwa przyciski, pozwalające na stwierdzenie, czy usługa dobrze wykryła uśmiech, czy jest to błąd w działaniu usługi. „Prawdziwe” uśmiechy posiadają zieloną ramkę, podczas gdy „fałszywe” – czerwoną. W przypadku gdy użytkownik nie oceni zdjęcia, żadna ramka nie jest wyświetlana.

W celu uniknięcia przesycenia łącza użytkownika lub przeciążenia serwera, stosowany jest algorytm dopasowywania liczby przesyłanych klatek. Domyślnie jest to wartość 600 ms, ponieważ tyle wynosi uzyskany doświadczalnie czas średni odpowiedzi z usługi. W ramach pracy aplikacji założono, że można jednocześnie otworzyć maksymalnie 10 kanałów komunikacyjnych z częścią backendową. Za każdym razem, kiedy limit ten jest osiągnięty,

opóźnienie w przesyłaniu klatek zwiększa się o 50 ms, zmniejszane jest natomiast o 25 ms za każdym razem, kiedy aplikacja otrzymuje odpowiedź z serwera. Maksymalną wartością czasu opóźnienia w przesyłaniu klatek jest 1 s.

Jeżeli prędkość Internetu, wydajność komputera lub przeciążenie serwera uniemożliwiają w prawidłowym przesyłaniu danych na serwer (czyli wartość opóźnienia w przesyłaniu klatek osiągnie swoją maksymalną wartość), poniżej obrazu z kamery prezentowany jest komunikat o zbyt małej liczbie przesyłanych klatek na sekundę – taka praca aplikacji uniemożliwia wykrywanie ciągłości uśmiechów i często prowadzi do zdegenerowanych wyników, ze względu na często duże różnice pomiędzy przesłanymi klatkami.

Perform services test
 Download Smilecounter on desktop
 Smilecounter

### Testing summary

Service 1	
Time spent using service	14.445 sec
Average service response time	126 ms
Detected smiles	4
Smiles set as real smiles	0
Smiles not confirmed	0
Smiles set as fake smiles	4

Service 2	
Time spent using service	0.191 sec
Average service response time	0 ms
Detected smiles	0
Smiles set as real smiles	0
Smiles not confirmed	0

Service 3	
Time spent using service	40.206 sec
Average service response time	212 ms
Detected smiles	0
Smiles set as real smiles	2
Smiles not confirmed	1
Smiles set as fake smiles	1

### Form

Please answer below questions about each service you have tested (it will take only few minutes). It will help us make better product in the future.

#### Service 1

Did the smile detection deal with most of your smiles?

☐ Yes  
☐ Rather yes  
☐ Rather not  
☐ Certainly not

What types of smiles did the smile detection detect?

☐ Wide open smiles (visible gums and teeth)  
☐ Open smiles (visible teeth)  
☐ Closed smiles (teeth and gums not visible)  
☐ It wasn't detecting any smiles of mine

Please summarize smile detection with this service (are you pleased with the results?):

#### Service 2

Did the smile detection deal with most of your smiles?

☐ Yes  
☐ Rather yes  
☐ Rather not  
☐ Certainly not

What types of smiles did the smile detection detect?

☐ Wide open smiles (visible gums and teeth)  
☐ Open smiles (visible teeth)  
☐ Closed smiles (teeth and gums not visible)  
☐ It wasn't detecting any smiles of mine

Please summarize smile detection with this service (are you pleased with the results?):

#### Service 3

Did the smile detection deal with most of your smiles?

☐ Yes  
☐ Rather yes  
☐ Rather not  
☐ Certainly not

What types of smiles did the smile detection detect?

☐ Wide open smiles (visible gums and teeth)  
☐ Open smiles (visible teeth)  
☐ Closed smiles (teeth and gums not visible)  
☐ It wasn't detecting any smiles of mine

Please summarize smile detection with this service (are you pleased with the results?):

### General questions

Gender

☐ Man  
☐ Woman

Age

Age

Do you think services grouped smiles correctly?

☐ Yes  
☐ Rather yes  
☐ Rather not  
☐ Certainly not

What do you think about smile detection? Will it be useful in the future?

☐ Yes, I think it will be used in many applications  
☐ I didn't think about it until now  
☐ No, I don't think information about user's smiles are useful for applications

Send

**Rys. 19. Ekran podsumowujący testy usług**

Kolejnym etapem testowania usług jest ekran podsumowania testu, zaprezentowanym na Rys. 19., jest ekran podsumowujący testy usług.

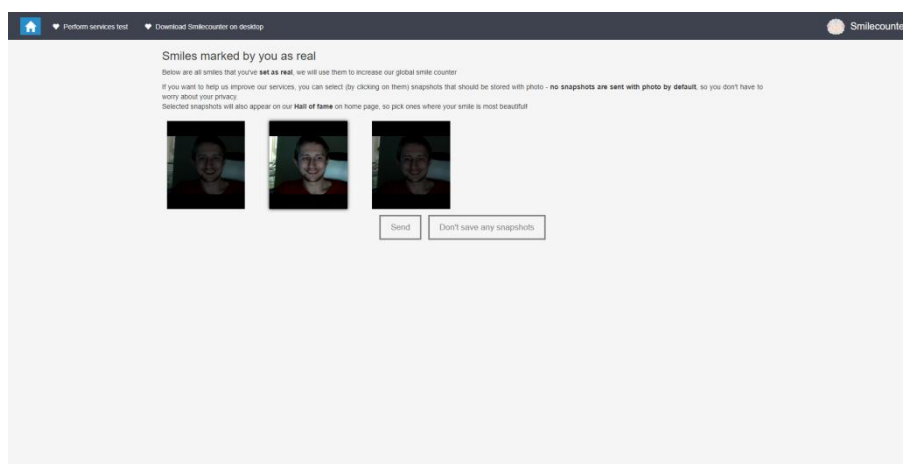
Pierwszą częścią formatki są techniczne dane związane z testowaniem każdej z usług: czas spędzony na formatce, średni czas odpowiedzi z usługi, liczba wykrytych przez usługę uśmiechów, liczba uśmiechów oznaczonych przez użytkownika jako prawdziwe, liczba uśmiechów oznaczonych przez użytkownika jako błędne oraz liczba uśmiechów, których użytkownik nie ocenił.

Poniżej części statystycznej prezentowany jest formularz, w którym żadne pole nie jest obowiązkowe – użytkownik może po prostu wykonać test usług, dzięki czemu zapisane zostaną dane statystyczne. Osoby chętne pomóc w testach usług mogą natomiast odpowiedzieć na szereg pytań dotyczących każdej z usług (pytania są powielone, użytkownik nie wie, która usługa wykorzystywała jaką bibliotekę ze względu na losowość): czy usługa poradziła sobie z wykryciem większości uśmiechów, jakie rodzaje uśmiechów zostały wykryte (szeroko otwarte, otwarte, zamknięte, żadne) oraz pytanie otwarte, w którym użytkownik może wprowadzić dodatkowe uwagi odnośnie działania usługi. W dalszej części można znaleźć także pytanie o grupowanie ciągłych uśmiechów (wykrywanie startu i zakończenia uśmiechu).

Poniżej sekcji z pytaniami dotyczącymi usługi znajduje się grupa pytań ogólnych, pomocnych w stworzeniu profilu użytkownika. Wyświetlane są w niej pytania o płeć użytkownika, jego wiek oraz zdanie odnośnie przyszłości usług przetwarzających emocje. W przypadku, gdy użytkownik twierdzi, że usługi tego typu będą wykorzystywane w wielu aplikacjach, pojawia się dodatkowe pytanie, w którym użytkownik proszony jest o wskazanie, w jakich aplikacjach chciałby, by takie usługi się znalazły.

Po kliknięciu na przycisk Wyślij u dołu ekranu wyniki testów są zapisywane w bazie danych.

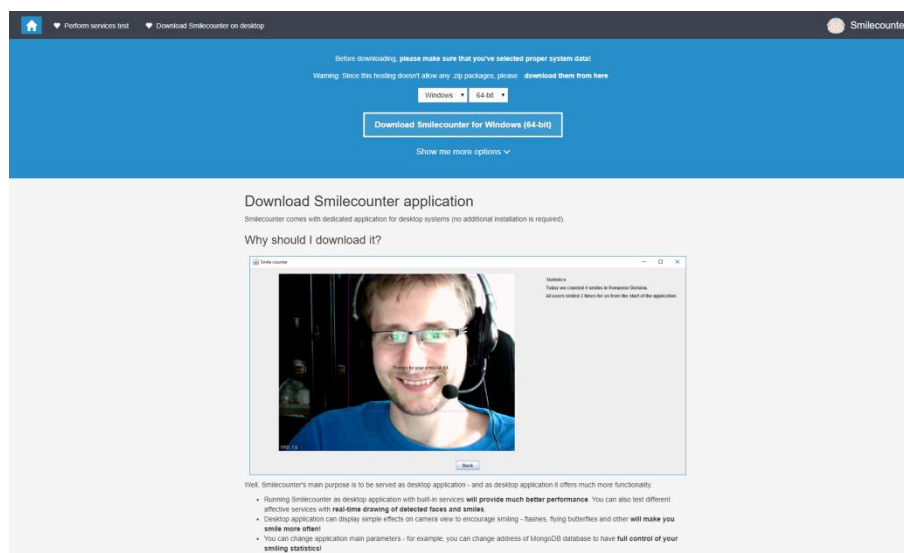
Jeżeli użytkownik podczas testowania usług oznaczył jakikolwiek uśmiech jako prawdziwy, po wysłaniu wyników testu pojawia się dodatkowa formatka, w której można wybrać zdjęcia do wyświetlenia na stronie głównej, co prezentuje Rys. 20.



Rys. 20. Ekran zapisywania uśmiechów wykrytych podczas testu usług

### 3.6.3. Ekran pobierania aplikacji desktopowej

Jednym z ekranów dostępnych w aplikacji webowej jest formatka do pobierania wersji desktopowej programu. Ponieważ Smilecounter wykorzystuje biblioteki dynamiczne, wersje do pobrania będą różniły się w zależności od rodziny oraz architektury systemu operacyjnego użytkownika.



**Rys. 21. Ekran pobierania aplikacji desktopowej**

Na Rys. 21. można zobaczyć, że główną częścią formatki jest określenie wersji aplikacji do pobrania. Domyślnie Smilecounter dobiera parametry do systemu operacyjnego użytkownika, jednak ma on możliwość dostosowania ustawień do własnych potrzeb. Oprócz pobrania pełnej paczki z aplikacją, istnieje opcja pobrania jedynie pliku .exe oraz .jar dla użytkowników, którzy już raz pobrali Smilecounter i chcieliby jedynie zaktualizować jego wersję do nowszej.

W dolnej części ekranu znajdują się dodatkowo informacje o tym, czym różni się aplikacja desktopowa od webowej i co użytkownik może zyskać pobierając tę wersję programu na swój komputer.

### 3.6.4. Konfiguracja i sposób uruchomienia aplikacji

Konfiguracja aplikacji webowej została podzielona na dwie części, ze względu na podział modularny backend oraz frontend. W przypadku serwera konfiguracja jest bardzo podobna do aplikacji desktopowej – odbywa się to poprzez plik *smilecounter.config*, jest on jednak dużo bardziej uproszczony. Zawierają się w nim tylko trzy parametry: *affective.luxand.key*, *database.connectionUrl* oraz *affective.libs.path*, a ich znaczenie opisuje dokładniej wspomniana wcześniej Tabela 2.

Ponieważ aplikacja powinna być uruchomiona na serwerze *WildFly 10*, wskazanie położenia pliku *smilecounter.config* odbywa się poprzez modyfikację konfiguracji serwera (plik *standalone.xml*) – należy dodać do *subsystemu* z zawartością zmiennych globalnych pozycję typu String:



```
<simple name="java:global/pathToProperties"
value="/sciezka/do/pliku" type="java.lang.String"/>
```

Konfiguracja części frontendowej odbywa się poprzez edycję pliku *application.config.json* wewnątrz paczki, w katalogu */config*. Zawiera on szereg parametrów, które prezentuje *Tabela 3*.

**Tabela 3. Lista parametrów konfiguracyjnych dla części frontendowej**

Parametr	Opis	Wartość domyślna
html5Mode	Określa, czy aplikacja ma wyświetlać linki w formacie HTML5 (opcja AngularJS).	false
languages	Lista dostępnych języków w aplikacji, przedstawiona w formie tablicy.	["en", "pl"]
defaultAlertsDisplayTimeInSec	Czas wyświetlania komunikatów. W przypadku wartości 0, komunikaty nigdy nie znikną.	0
demoVersion	Określa, czy aplikacja ma działać w wersji demo (nie łączyć się z usługami).	false
debugVersion	Określa, czy aplikacja ma działać w wersji debug (ma prezentować dodatkowe informacje związane z Angularem).	true
restApi	Adres do usług REST-owych po stronie serwera. Może być linkiem.	/smilecounter/rest
supportedSystems	Lista wspieranych systemów podczas pobierania aplikacji, przedstawiona w formie tablicy. Wymaga umieszczenia plików z odpowiednim suffixem w katalogu <i>files</i> .	["Windows"]
refreshTime	Domyślny czas odświeżania danych na stronie głównej.	10
servicesToTest	Lista usług (przedstawiona w formie tablicy), które będą	["LUXAND", "OPEN_CV", "CUSTOM"]

Parametr	Opis	Wartość domyślna
	testowane. Kolejność testowania usług jest losowana po wejściu na formatkę.	
snapshotSendInterval	Początkowy czas (podany w ms), określający częstotliwość wysyłania klatki z kamerki na serwer.	600
snapshotSendSize	Rozmiar klatki z kamerki internetowej wysyłanej na serwer.	{"width" : 320, "height" : 240}
previousFramesInSnapshotDetecting	Liczba poprzednich klatek trzymanyh w pamięci (przydatna do grupowania uśmiechów).	1
maxFramesRequest	Maksymalna liczba żądań na formacie testowania usług wysyłanych jednocześnie. Pozwala na zmniejszenie obciążenia komputera i serwera.	10

Warto także wspomnieć, że aplikacja webowa została w pełni skonfigurowana przy pomocy Dockera – w tym celu należy wejść w moduł *Docker* wewnątrz plików źródłowych aplikacji, dodać najnowszą wersję aplikacji do *backend/artifacts*, a następnie wydać polecenie *docker-compose up -d*. Wersja dockerowa aplikacji opiera się o dwa kontenery. Pierwszym z nich jest *backend*, zawierający *backend.war* oraz *frontend.war* oraz zainstalowane najnowsze wersje bibliotek dynamicznych. Drugim kontenerem jest *database*, zawierający gotową bazę MongoDB. Konfiguracja obu kontenerów zawiera się w pliku *docker-compose.yml*.

## 4. WERYFIKACJA ROZWIĄZANIA

W celu zapewnienia wysokiej jakości aplikacji *Smilecounter* zdecydowano się poddać ją procesowi dogłębnej, kilkupoziomowej weryfikacji. Składała się ona z dwóch głównych etapów: testów bibliotek oraz usług wykrywających uśmiech wystawionych przy pomocy aplikacji webowej, a następnie wdrożenia aplikacji w ośrodku dla dzieci z autyzmem.

### 4.1. Testy bibliotek

W celu zweryfikowania wydajności i skuteczności w detekcji uśmiechu przez poszczególne biblioteki, poddano je testom mierzącym czas procesowania obrazu oraz liczbę wykrytych uśmiechów. W tym celu napisany został moduł pozwalający pobranie wszystkich zdjęć zawartych we wskazanym katalogu, a następnie uruchomienie po kolei wszystkich usług na zadanym zbiorze. Program liczył takie statystyki jak średni czas detekcji, liczba znalezionych twarzy, liczba znalezionych uśmiechów, liczba znalezionych ust (niezależnie od stanu) oraz liczba wykrytych zdjęć, na których jest więcej niż jedna osoba. Czas trwania detekcji mierzony jest od momentu przekazania obrazu usłudze do momentu uzyskania odpowiedzi.

Wyniki tych testów pozwalają na określenie wydajności oraz dokładności różnych zastosowanych rozwiązań, jednak dotyczą jedynie wykrywania uśmiechów na pojedynczych klatkach. Oznacza to, że nie weryfikują one pojawiania się efektów oraz wykrywania momentów rozpoczęcia i zakończenia uśmiechów – weryfikacja tych funkcjonalności odbywa się w późniejszych rozdziałach.

Wszystkie testy zostały przeprowadzone na maszynie z 12 GB pamięci RAM, procesorem Intel i7-4700MQ oraz na dysku HDD WDC WD10JPCX. Zainstalowanym systemem operacyjnym na stacji był Windows 10 Professional 64-bit.

Dla usług wykorzystujących bibliotekę FaceSDK ustawiono próg pewności uśmiechu na 0.45 – pozwala on na wykrywanie nie tylko uśmiechów otwartych, ale także części uśmiechów zamkniętych.






#### 4.1.1. Własnoręcznie przygotowany zbiór obrazów

Pierwszym sposobem testowania było przygotowanie specjalnego zbioru obrazów, aby sprawdzić uniwersalność bibliotek. W celu pokrycia w miarę szeroko różnorodności, wykorzystano zdjęcia z różną liczbą osób. Starano się także tak je dobrać, aby znajdowały się na nich osoby w różnym wieku, płci i innych atrybutach (np. zarost, okulary). Dodatkowo, zdjęcia zrealizowano w dobrym i złym oświetleniu. Rozdzielczość obrazów także była zróżnicowana – najmniejsze zdjęcie było w formacie 630x400 px, zaś największe w formacie 2060x1496 px.

Test ten miał na celu zbadanie dokładności wykorzystywanych usług (czyli detekcji twarzy i uśmiechów), a nie wydajności. *Tabela 4.* pokazuje wszystkie zdjęcia użyte w zbiorze - łącznie dodano do niego 6 zdjęć. Znajdowało się na nich 14 osób, w tym 12 uśmiechniętych.

Połowę z nich stanowili mężczyźni, z których tylko jeden się nie uśmiechał – analogiczna sytuacja występuje w przypadku pań.

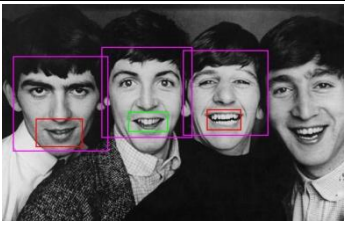
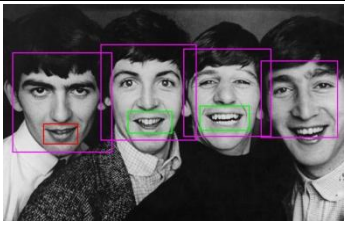
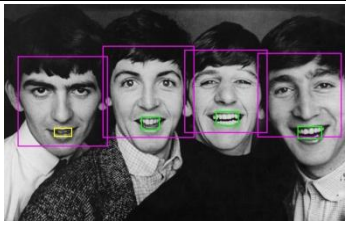

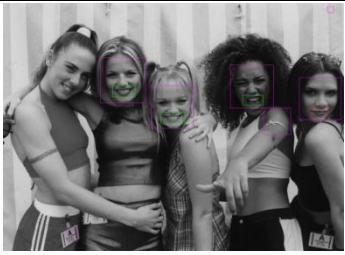

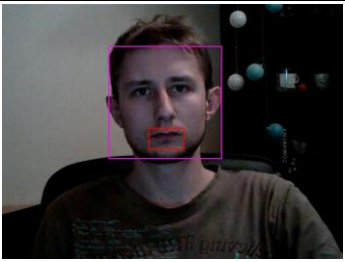
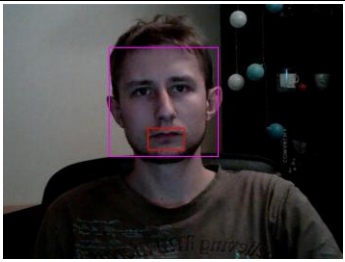
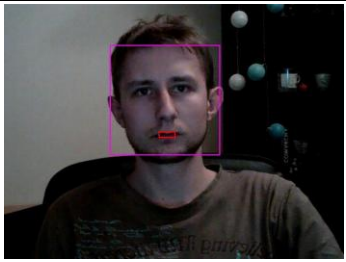
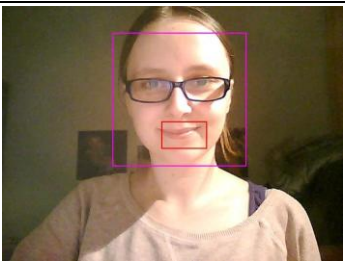
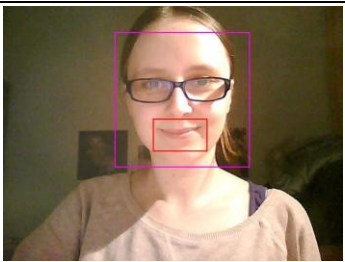
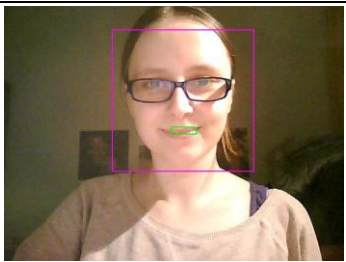
**Tabela 4. Własnoręcznie utworzony zbiór testowy**

Lp.	Tytuł zdjęcia	Zdjęcie	Opis zdjęcia
1.	The Beatles		Czterech uśmiechniętych mężczyzn
2.	Spice Girls		Pięć uśmiechniętych kobiet z różnym stopniem uśmiechu (otwarte oraz zamknięte)
3.	Mężczyzna 1		Mężczyzna, słabe oświetlenie, brak uśmiechu
4.	Kobieta 1		Kobieta, typowe oświetlenie, okulary, uśmiech zamknięty
5.	Mężczyzna 2		Mężczyzna, szeroki uśmiech, okulary, dobre oświetlenie

Lp.	Tytuł zdjęcia	Zdjęcie	Opis zdjęcia
6.	Kobieta i mężczyzna		Kobieta bez uśmiechu i bez okularów, mężczyzna z uśmiechem otwartym

Każde zdjęcie w zbiorze zostało przekazane do zaimplementowanych usług. Wyniki zostały zapisane do obrazu, na którym zaznaczono wykryte elementy odpowiednimi kolorami. Ramka fioletowa oznacza odnalezioną twarz, czerwona – odnalezione usta (bez uśmiechu), natomiast ramka zielona pokazuje wykryty uśmiech. Wyniki takiego procesu zawiera *Tabela 5*.

**Tabela 5. Wyniki graficzne testów na własnoręcznie utworzonym zbiorze**

OpenCV	OpenIMAJ	FaceSDK (Luxand)
		
		
		
		

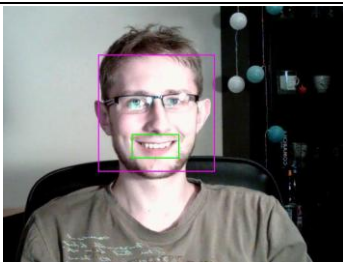
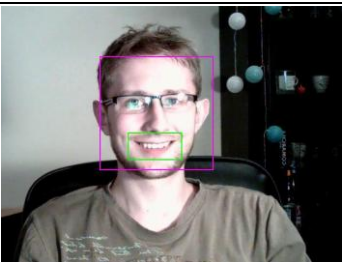
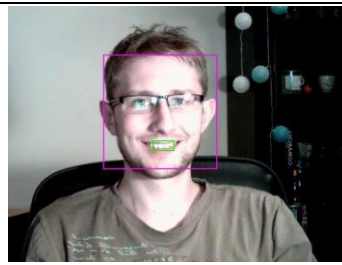
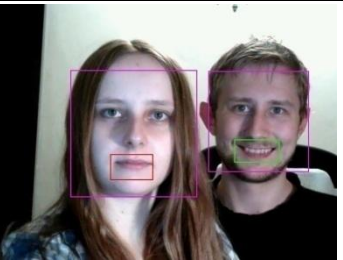
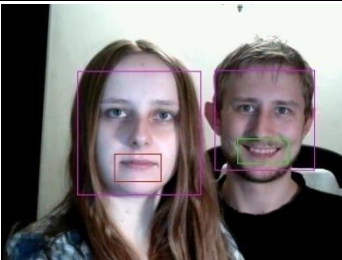
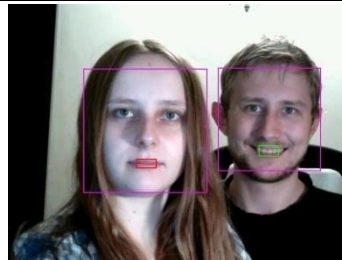
OpenCV	OpenIMAJ	FaceSDK (Luxand)
		
		

Tabela pokazuje zestawienie działania jedynie podstawowych usług na utworzonym zbiorze – dla usługi wykrywającej twarze przy pomocy biblioteki OpenCV oraz uśmiechu poprzez wykorzystanie możliwości SDK wyniki testów prezentuje *Tabela 6*, która zostanie omówiona później.

Na załączonych zdjęciach można zauważyć, że najlepiej z wykrywaniem uśmiechów radzi sobie usługa FaceSDK od Luxanda – nie wykryła ona prawidłowo uśmiechu tylko dla jednego przypadku, w którym twarz jest mocno obrócona – jednak żadna z pozostałych usług także nie była w stanie wskazać poprawnej odpowiedzi na tym zdjęciu. Widać zatem, że usługi do detekcji uśmiechu nie radzą sobie dobrze w przypadku, kiedy twarz jest mocno obrócona i pod kątem w kierunku do kamery.

Klasyfikator uśmiechu wykorzystywany w usługach OpenCV oraz OpenIMAJ działa znacznie gorzej – w większości przypadków wykrywał jedynie uśmiechy otwarte. Co ciekawe, pomimo wykorzystania tego samego klasyfikatora obie usługi zwróciły inną odpowiedź w przypadku jednego z członków zespołu *The Beatles*.

Kolejną obserwacją jest fakt, że usługa OpenIMAJ zwróciła dodatkowo dwa *false alarmy*, obydwa na zdjęciu o największej rozdzielczości (zdjęcie zespołu *Spice Girls*). Pierwszy z nich znajduje się na identyfikatorze dziewczyny po lewej, natomiast drugi na ramieniu przedostatniej z dziewczyn.

W przypadku usługi OpenCV można zauważyć, że jako jedyna nie wykryła twarzy dla ostatniego członka zespołu *The Beatles* – dzieje się tak prawdopodobnie dlatego, że zastosowany klasyfikator nie mógł sobie poradzić z sytuacją, w której niewielka część twarzy jest ucięta (nie mieści się w kadrze zdjęcia).



Tabela 6. Wyniki testów usługi OpenCV + FaceSDK dla utworzonego zbioru zdjęć

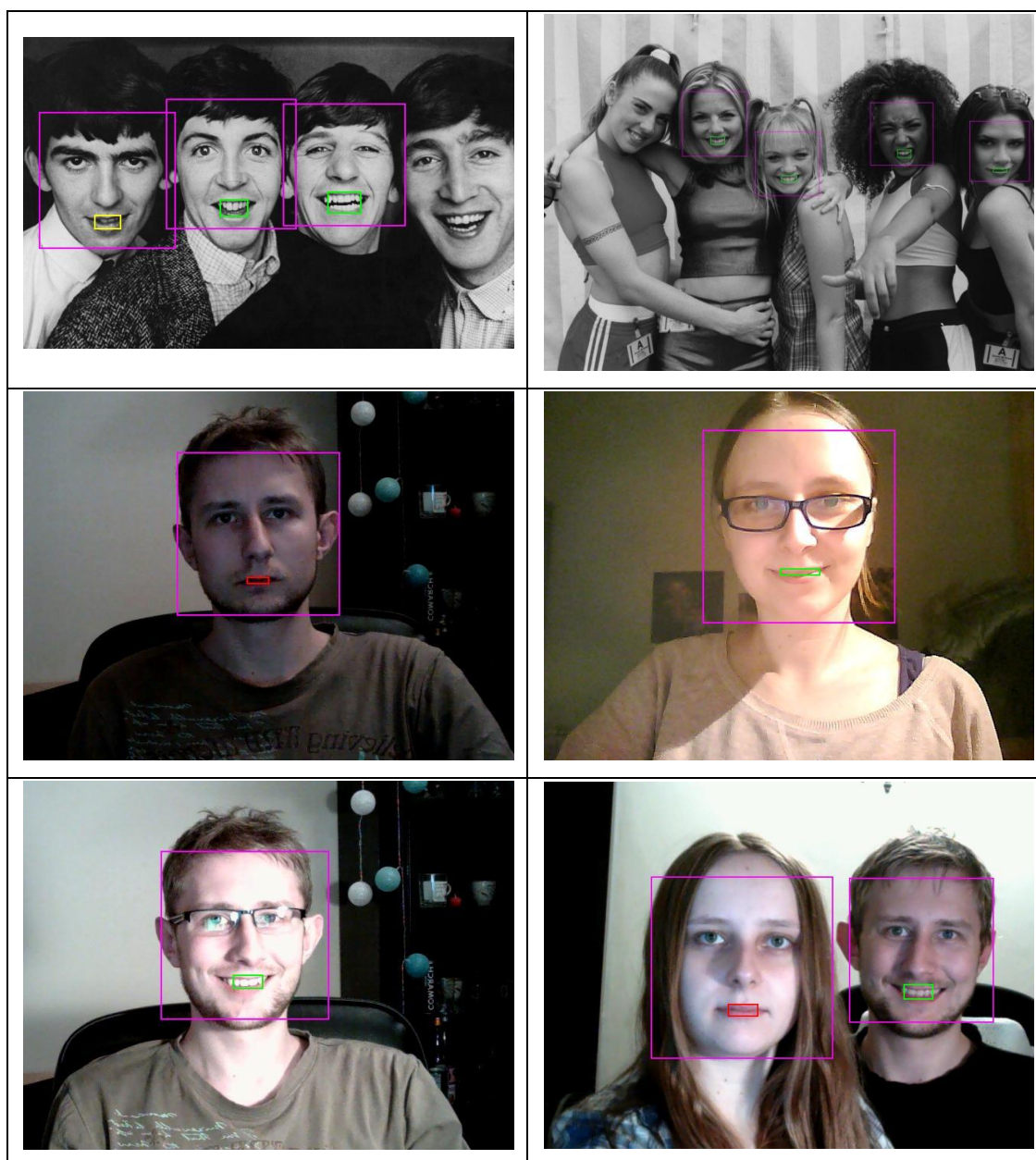


Tabela 6 pokazuje, że zastosowanie usługi łączącej OpenCV oraz FaceSDK jest także połączeniem ich skuteczności – dla wszystkich odnalezionych twarzy prawidłowo został określony uśmiech.

Niestety, ponieważ usługa OpenCV nie jest w stanie sobie poradzić z detekcją twarzy członka zespołu *The Beatles*, wspólna implementacja działa gorzej niż zastosowanie jedynie biblioteki Luxandu.

Podsumowanie wykonanych testów w formie prostych statystyk zawiera Tabela 7.

**Tabela 7. Statystyki usług z testów własnoręcznie utworzonego zbioru**

Nazwa usługi	Liczba wykrytych twarzy	Średni czas procesowania (bez dużego zdjęcia)	Liczba wykrytych uśmiechów (ust)
<b>OpenCV</b>	12	470 ms (278 ms)	6 (12)
<b>OpenIMAJ</b>	16	1 418 ms (585 ms)	7 (12)
<b>FaceSDK</b>	13	310 ms (257 ms)	11 (13)
<b>OpenCV + FaceSDK</b>	12	574 ms (223 ms)	10 (12)

Oprócz wspomnianych wcześniej informacji o dokładności wykorzystanych usług, można tutaj także znaleźć dane na temat średniego czasu procesowania usługi. Szczególnie ciekawy jest tutaj przypadek dla usługi FaceSDK – ponieważ skaluje ona wszystkie zdjęcia do jednego rozmiaru przed przystąpieniem do procesu detekcji uśmiechów, jej czas jest znacząco mniejszy niż w przypadku pozostałych usług. Po odjęciu od średniej czasu procesowania zdjęcia zespołu *Spice Girls* sytuacja ulega zmianie – wówczas najszybciej działającą usługą jest połączenie możliwości OpenCV z biblioteką FaceSDK. Ponieważ aplikacja będzie działała na klatkach z kamery internetowej w czasie rzeczywistym, ważniejszym wydaje się wynik nie uwzględniający tak dużego formatu obrazów.

#### 4.1.2. Testy na istniejących bazach twarzy ludzi

Kolejnym etapem testów usług wykrywających uśmiech było wykorzystanie gotowych baz danych zawierających zdjęcia ludzkich twarzy, wyrażających zróżnicowane emocje - na wszystkich obrazach znajdowała się pojedyncza osoba. Zawartość każdej z baz podzielono na dwie grupy: zdjęcia, które zawierały uśmiechy dowolnego stopnia (zarówno zamknięte, jak i otwarte) oraz zdjęcia, które zawierały wszystkie pozostałe emocje.

Pierwszym zbiorem ludzkich twarzy, który poddano testowi, był *Cohn-Kanade* [17], który posiadał łącznie 1 475 zdjęć, w tym 1 248 negatywnych oraz 227 pozytywnych. Baza twarzy zawierała osobniki w średnim wieku, obydwu płci oraz różnych narodowości. 1 683 zdjęć było w formacie 640x490 px i w skali szarości, natomiast 168 zdjęć były zapisane w kolorze i w formatach 720x480 px lub 640x480 px. Wyniki testów dla zbioru Cohn-Kanade prezentuje *Tabela 8*.

**Tabela 8. Wyniki testów zbioru Cohn-Kanade**

Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarz (pomyłki)	Średnie wymiary twarzy	Średnie wymiary ust
<b>OpenIMAJ</b>						
<b>Pozytywne</b>	507 ms	249	187 (227)	21	277x277 px	139x71 px
<b>Negatywne</b>	611 ms	1 344	43 (1 163)	92	280x280 px	98x58 px



Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarz (pomyłki)	Średnie wymiary twarzy	Średnie wymiary ust
<b>OpenCV</b>						
<b>Pozytywne</b>	194 ms	227	149 (207)	0	279x279 px	111x58 px
<b>Negatywne</b>	287 ms	1 248	30 (895)	0	280x280 px	73x43 px
<b>Luxand (FaceSDK)</b>						
<b>Pozytywne</b>	228 ms	231	225 (231)	4	285x285 px	61x33 px
<b>Negatywne</b>	243 ms	1 267	135 (1 267)	19	283x283 px	48x34 px
<b>OpenCV + FaceSDK</b>						
<b>Pozytywne</b>	209 ms	227	223 (227)	0	279x279 px	61x33 px
<b>Negatywne</b>	208 ms	1 248	138 (1 248)	0	280x280 px	48x34 px

Pierwszą obserwacją, którą można zauważyć w prezentowanych wynikach jest fakt, że w przypadku usług korzystających z klasyfikatorów Haara zdjęcia pozytywne procesowane były nieco szybciej, niż negatywne – dzieje się tak z dwóch powodów. Główną przyczyną jest działanie algorytmu – jeżeli usługi nie znajdą uśmiechu, przeszukują dolny obszar twarzy ponownie w celu znalezienia zwykłych ust. Drugi powód jest bardziej oczywisty - wszystkie zdjęcia pozytywne w zbiorze okazały się być w mniejszym formacie, 640x490 px, i w skali szarości, podczas gdy w zbiorze zdjęć negatywnych niektóre zdjęcia były kolorowe i ich szerokość wynosiła 720 px.

Testy OpenIMAJ, zgodnie z założeniami, zwracały wyniki z największymi opóźnieniami. Dzieje się tak, ponieważ całość została napisana w Javie, podczas gdy pozostałe biblioteki działają dzięki załączonym bibliotekom dynamicznym, napisanych w języku C. Warto także zauważyć, że dokładność biblioteki również nie jest najlepsza – pomimo faktu, że wszystkie zdjęcia zawierały pojedyncze twarze, aż na 113 z nich usługa zwróciła informacje o kilku twarzach (*false alarm*).

Kolejną usługą jest OpenCV – o ile zgodnie ze wspomnianym wcześniej algorytmem zdjęcia pozytywne są procesowane najszybciej z pozostałych usług, dołożenie kolejnego klasyfikatora do znalezienia ust bez stanu uśmiechu sprawia, że jest ona wolniejsza nawet od FaceSDK – czyli od biblioteki opierającej swoje algorytmy na punktach kluczowych twarzy.

Wyniki testów pokazały także, że połączenie OpenCV i FaceSDK pod względem wydajności okazało się być trafionym pomysłem – dla przedstawionego zbioru zdjęć odpowiedź z usługi średnio wynosi ok. 200 ms niezależnie od stanu ust.

Kolorem zielonym oznaczono przypadki, w których usługi znalazły dokładnie taką samą liczbę twarzy, ile było zdjęć – zarówno dla obrazów pozytywnych, jak i negatywnych. Można łatwo zauważyć, że wszystkie takie przypadki dotyczą klasyfikatora twarzy wykorzystywanego

przez usługę OpenCV – działa on na tyle dobrze, że na żadnym z zadanych zdjęć usługa nie wykryła *false alarmu*.



**Rys. 22. Przykład *false alarmu* dla biblioteki Luxand**

Pomimo faktu, że FaceSDK działa w sposób najbardziej dokładny, zdarza się jej zwrócić błędną informację o większej liczbie twarzy na pojedynczym zdjęciu (jednym z przykładów takiego działania jest Rys. 22). Co szczególnie ciekawe, biblioteka nie tylko wykryła twarz w miejscu, w której jej nie ma – znalazła w tym fragmencie także „usta” w stanie zbliżonym do uśmiechu.

O ile klasyfikator twarzy dla OpenCV działa zaskakująco sprawnie, klasyfikator wykrywania ust oraz uśmiechów nie sprawia już takiego wrażenia – jedynie dla 75% twarzy udało się znaleźć usta ich dolnej części. Przykład takiego działania został zaprezentowany na Rys. 23.



**Rys. 23. Przykład niewykrycia ust przez bibliotekę OpenCV**

Zgodnie z powyższymi obserwacjami, biblioteka łącząca możliwości OpenCV oraz FaceSDK powinna zwracać dokładniejsze wyniki. Wykonane testy zdają się to potwierdzać – w przypadku czwartej usługi na wszystkich zdjęciach znaleziono tylko jedną twarz, a dla każdej z nich wykryto usta w dolnej części twarzy. Pomimo zwiększenia dokładności wykrywania twarzy usługa zwraca odpowiedź w czasie szybszym niż FaceSDK.



**Rys. 24. Porównanie rozmiarów twarzy i ust zwróconych przez biblioteki**

a) OpenCV + FaceSDK, b) OpenCV c) OpenIMAJ

Średnie wymiary twarzy dla wszystkich testowanych usług są do siebie bardzo zbliżone i zostały pokazane na przykładzie Rys. 24. Sytuacja wygląda jednak inaczej w przypadku średnich rozmiarów ust – usługi oparte o klasyfikator zwracają fragment zawierający całe usta (a często nawet większy), podczas gdy usługi wykorzystujące FaceSDK zwracają informacje jedynie o centralnej części ust (czyli pomiędzy założonymi punktami kluczowymi). Wydaje się zatem, że prezentowanie efektów związanych z ustami będzie dokładniejsze przypadku wykorzystania FaceSDK w celu wskazania pozycji ust.

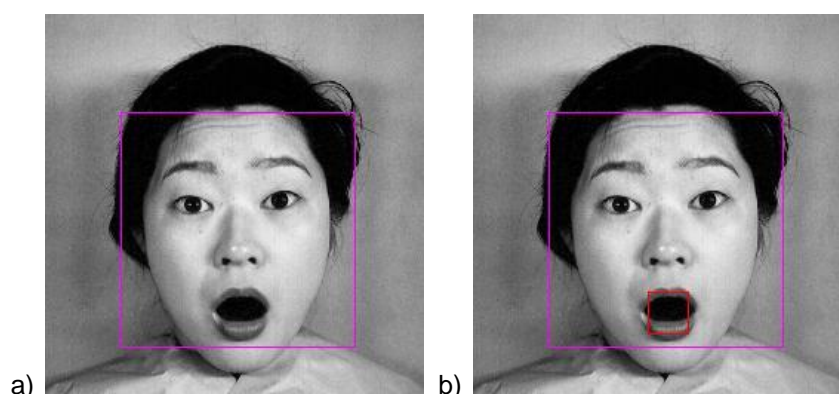
Kolejnym zbiorem, który został wykorzystany w ramach testów, był JAFFE (*The Japanese Female Facial Expression*) [18], zawierający 213 zdjęć o rozdzielczości 256 px na 256 px. Znajdowały się na nich wyłącznie kobiety w średnim wieku oraz narodowości japońskiej. Bazę podzielono na 182 zdjęcia negatywne i 31 pozytywnych. Ponieważ domyślnym formatem wszystkich obrazów był .TIFF, przekonwertowano je na wysokiej jakości pliki o formacie .JPEG. Wyniki testów dla tego zbioru prezentuje *Tabela 9*.

**Tabela 9. Wyniki testów zbioru JAFFE**

Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarz (pomyłki)	Średnie wymiary twarzy	Średnie wymiary ust
<b>OpenIMAJ</b>						
<b>Pozytywne</b>	119 ms	31	20 (31)	0	157x157 px	74x39 px
<b>Negatywne</b>	145 ms	183	2 (171)	1	157x157 px	53x32 px
<b>OpenCV</b>						
<b>Pozytywne</b>	84 ms	31	6 (25)	0	160x160 px	53x30 px
<b>Negatywne</b>	78 ms	182	1 (84)	0	157x157 px	31x18 px
<b>Luxand (FaceSDK)</b>						
<b>Pozytywne</b>	168 ms	31	29 (31)	0	161x161 px	31x18 px

Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarz (pomyłki)	Średnie wymiary twarzy	Średnie wymiary ust
Negatywne	160 ms	182	21 (182)	0	160x160 px	27x20 px
OpenCV + FaceSDK						
Pozytywne	67 ms	31	29 (31)	0	160x160 px	31x17 px
Negatywne	67 ms	182	17 (182)	0	157x157 px	27x27 px

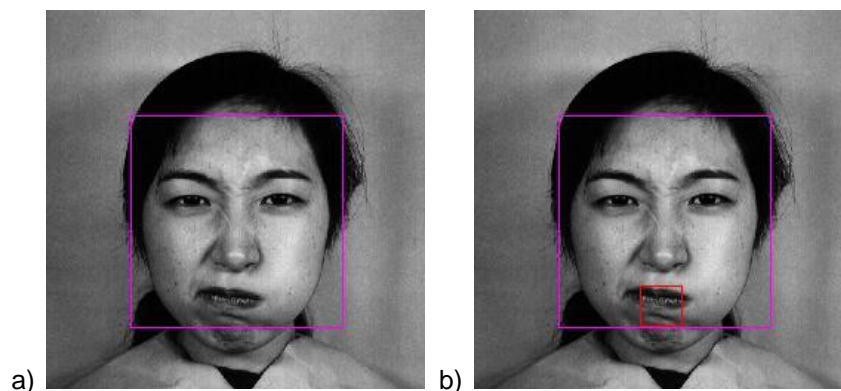
Podobnie jak w poprzednim teście, kolorem zielonym oznaczono przypadki, w którym została wykryta prawidłowa liczba ust i twarzy. Można od razu zauważyć, że prawie wszystkie usługi prawidłowo wykryły liczbę twarzy zarówno na zdjęciach pozytywnych, jak i negatywnych – wyjątkiem jest usługa OpenIMAJ, która dodatkowo wykryła jeden *false alarm*. Potwierdziły się także spostrzeżenia z poprzedniego testu dotyczące prawidłowego wykrywania uśmiechów – usługi oparte o FaceSDK prawidłowo wykryły wszystkie uśmiechy, choć niektóre z nich znajdowały się na granicy progu pewności. Znacznie gorzej wypadła usługa OpenCV, która wykryła jedynie 19 % uśmiechów na zdjęciach pozytywnych, a jedynie 51 % twarzy.



Rys. 25. Porównanie detekcji w przypadku otwartych ust

a) OpenCV, b) OpenCV + FaceSDK

Jednym z przypadków, dla których OpenCV nie odnajduje ust, są usta otwarte (bez uśmiechu), co pokazuje Rys. 25. Łatwo też można zauważyć, że klasyfikator uśmiechu nie radzi sobie dobrze w przypadkach, kiedy usta są wygięte w nienaturalny sposób (np. w formie grymasu), co można zobaczyć na Rys. 26.



**Rys. 26. Porównanie detekcji w przypadku grymasu**

a) OpenCV, b) OpenCV + FaceSDK

Warto zauważyć, że tym razem usługa FaceSDK okazała się być najwolniejsza. Dzieje się tak dlatego, że zdjęcia są mniejszego rozmiaru niż ustawiony jako format domyślny ( $384 \times 384$  px) – oznacza to, że dla mniejszych zdjęć usługa najpierw je powiększy, a dopiero potem na nich operuje. Najszybciej działającą usługą była zatem ponownie opcja łącząca możliwości OpenCV i FaceSDK – w przypadku znajdowania cech twarzy w wyznaczonym rejonie biblioteka Luxandu nie zmienia rozmiaru wejściowego obrazu.

#### 4.1.3. Porównanie klasyfikatora twarzy Haara z LBP

Dodatkowym testem była weryfikacja, którego rodzaju klasyfikator sprawdzi się lepiej w domenie wykrywania uśmiechów. W tym celu usługi oparte o implementację OpenCV zostały zasilone dodatkowo o klasyfikator LBP, służący do wykrywania twarzy z przodu. Obydwie usługi zostały przetestowane zarówno na zbiorze Cohn-Kanade, jak i JAFFE – wyniki testów prezentuje *Tabela 10*.

**Tabela 10. Zastosowanie klasyfikatora LBP dla obu zbiorów**

Zbiór	Średni czas [ms]	Liczba twarzy (wielokrotnych)	Liczba uśmiechów (ust)	Średnie wymiary twarzy [px]	Średnie wymiary ust [px]
<b>OpenCV</b>					
<b>Cohn-Kanade pozytywne</b>	134	227 (0)	1 (205)	182x182	68x41
<b>Cohn-Kanade negatywne</b>	156	1251 (3)	6 (931)	185x185	55x33
<b>JAFFE pozytywne</b>	36	31 (0)	0 (30)	112x112	47x28
<b>JAFFE negatywne</b>	32	182 (0)	0 (141)	112x112	37x22

OpenCV + FaceSDK					
<b>Cohn-Kanade pozytywne</b>	137	227 (0)	197 (227)	182x182	50x33
<b>Cohn-Kanade negatywne</b>	165	1251 (3)	194 (1251)	185x185	46x32
<b>JAFFE pozytywne</b>	42	31 (0)	25 (31)	112x112	30x21
<b>JAFFE negatywne</b>	38	182 (0)	20 (182)	112x112	27x20

Jak można odczytać z powyższej tabeli, zgodnie z oczekiwaniami klasyfikator LBP zwracał odpowiedzi w krótszych czasach niż klasyfikator Haara – dla zbioru Cohn-Kanade jest to zysk rzędu ok. 70 ms, natomiast dla zbioru JAFFE o 50 ms (czyli prawie dwa razy szybciej).

Niestety, przyspieszenie pracy algorytmów zostało okupione spadkiem w dokładności detekcji. Najbardziej rażąco różnicą jest wpływ klasyfikatora twarzy LBP na klasyfikator uśmiechu: dla zbioru JAFFE żaden uśmiech nie został wykryty na pozytywnych zdjęciach, natomiast dla zbioru Cohn-Kanade został wykryty jedynie jeden uśmiech. Dzieje się tak głównie ze względu na zmianę średnich wymiarów twarzy, jakie zwraca klasyfikator LBP w stosunku do klasyfikatora Haara – różnice zostały pokazane na Rys. 27.



**Rys. 27. Różnice w detekcji pomiędzy klasyfikatorami Haara i LBP**

a) Klasyfikator LBP, b) Klasyfikator Haara c) FaceSDK

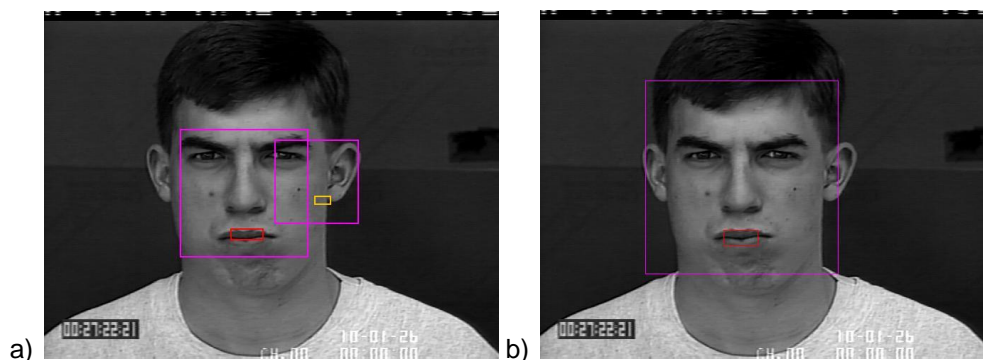
Na załączonym rysunku od razu widać różnicę – klasyfikator Haara zdaje się operować na zewnętrznych rozmiarach twarzy (od czoła po podbródek), podczas gdy klasyfikator LBP



zwraca raczej wewnętrzną pozycję twarzy (bez czoła i podbródka). Jak widać na zdjęciu procesowanym przez klasyfikator Haara, uśmiech wykryty przez klasyfikator uśmiechu nie zmieścił się w ramach twarzy wykrytej przy pomocy LBP, co sprawiło, że algorytm go odrzucił. Warto także zauważyć, że biblioteka FaceSDK działa w sposób podobny do klasyfikatora Haara – jednorodny sposób detekcji twarzy pozwala na znacznie łatwiejszą implementację wyświetlania efektów niezależnie od wybranej usługi. Doklejanie np. fragmentów włosów lub brody przy wykorzystaniu klasyfikatorów LBP musiałoby zatem posiadać własną logikę pozycjonowania i skalowania efektów.

W przypadku implementacji łączącej biblioteki OpenCV oraz FaceSDK sytuacja wygląda nieco lepiej. Przy wykorzystaniu tej usługi udało się poprawnie wykryć 86% uśmiechów, co jest spadkiem o 12% w porównaniu z zastosowaniem klasyfikatora Haara.

Kolejnym problemem związanym z wykorzystaniem klasyfikatorów LBP jest utrata dokładności w detekcji prawidłowych twarzy – pojawiło się tutaj kilka *false alarmów* dla zbioru Cohn-Kanade. Jeden z nich został pokazany na Rys. 28.



**Rys. 28. Przykład wystąpienia *false alarmu* przy użyciu klasyfikatora LBP**

a) OpenCV+FaceSDK z użyciem klasyfikatora LBP b) użycie klasyfikatora Haara

Wszystkie wystąpienia *false alarmów* w przypadku usługi LBP dotyczyły sytuacji podobnej do pokazanej na rysunku – wykrył on lewe oko i ucho jako osobną twarz. Choć takich przypadków nie było wiele (jedynie trzy), duże podobieństwo w lokalizacji wykrytej dodatkowej twarzy sugeruje, że jest to problem powtarzalny – jeżeli użytkownik ustawiłby się do kamery w podobny sposób jak na zdjęciu, prawdopodobnie zostałby policzony jako dwie fizyczne osoby.

#### 4.1.4. Analiza statystyk oraz wyników ankiety z aplikacji webowej

Kolejnym etapem testowania usług pozwalających na detekcję uśmiechu był moduł wystawiony wewnątrz aplikacji webowej. Dostęp do niego mieli wszyscy użytkownicy posiadający kamerkę internetową.

Głównym celem wdrożenia aplikacji webowej dla użytkowników było pozyskanie w łatwy sposób osób chętnych do przetestowania usług wykrywających uśmiech. Dzięki braku potrzeby jakiegokolwiek instalacji można udało się stworzyć moduł testujący usługi i zbierający statystyki z pracy każdej usługi oraz opinie użytkowników, zebrane przy pomocy prostego formularza.

Dzięki obserwacji żywych użytkowników można było nie tylko przetestować działanie usług na różnych zbiorach twarzy i kamerek w czasie rzeczywistym, ale także uzyskać potwierdzone informacje o tym, czy wykryte uśmiechy były prawidłowe oraz czy usługi potrafiły w prawidłowy sposób wykryć początek i koniec uśmiechu.

Po wykonaniu testów wypełnienie formularza było opcjonalne – podstawowe dane dotyczyły statystyk pozyskanych podczas działania usług. Dane te dotyczyły liczby wykrytych uśmiechów, liczby uśmiechów oznaczonych jako poprawne lub błędne, czas odpowiedzi usług oraz czas spędzony na testowaniu usługi.

Aplikację webową postawiono na serwerze DigitalOcean.com. Parametry serwera wynosiły: 4 GB RAM, 60 GB HDD, 4 TB transferu oraz procesor 2-rdzeniowy (wersja dropletu za 40\$). Wykorzystanym systemem operacyjnym był Ubuntu 17.06, a całość aplikacji została uruchomiona na Dockerze (kontenery z WildFly 10 oraz MongoDB, na którym zapisano wszystkie wyniki testów).

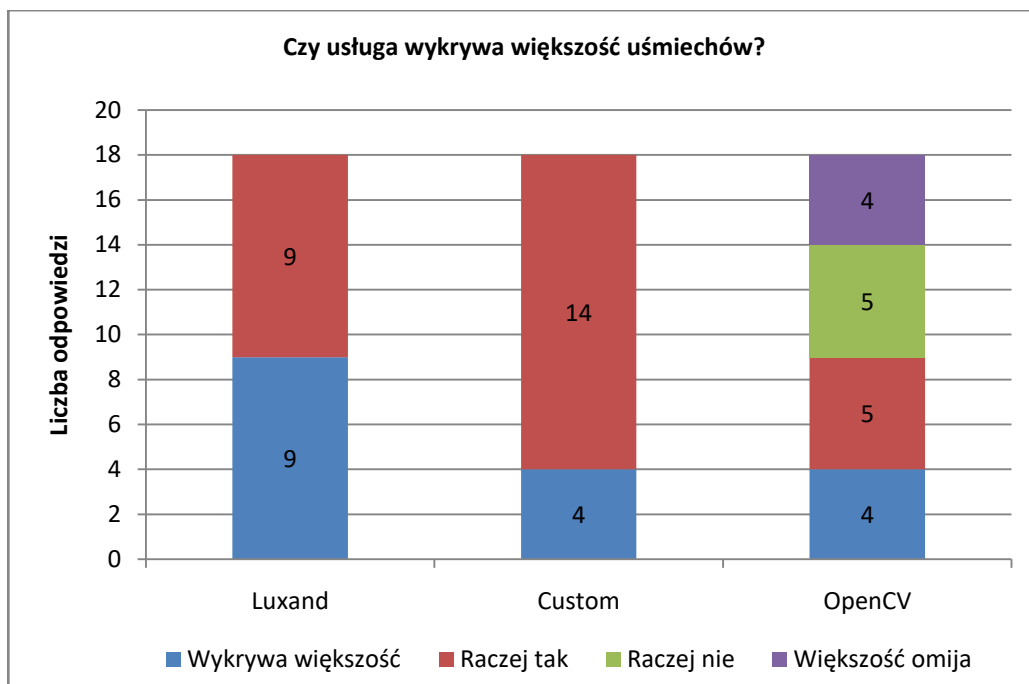
Podczas weryfikacji statystyk i opinii warto pamiętać o tym, że użytkownicy korzystali ze sprzętu różnego rodzaju – często zdarzało się, że prędkość Internetu lub wydajność komputera nie pozwalały na wysyłanie kilku klatek na sekundę. W takim wypadku prezentowany był odpowiedni alert, że praca aplikacji może nie być stabilna. Dodatkowym utrudnieniem był fakt, że każdą procesowaną klatkę trzeba w pierwszej kolejności przesłać na serwer, co powoduje zwiększenie czasu na otrzymanie odpowiedzi z usługi, czy na zdjęciu znajduje się uśmiech.

Ponieważ biblioteka OpenIMAJ zwracała odpowiedzi w zbyt wolnym czasie, została ona wykluczona z testów w aplikacji webowej – w jej przypadku przesyłano jedną klatkę na ok. 1.5 s, co przeszkadzało w sprawnym wykrywaniu ciągłości uśmiechu.

Podczas testów aplikacji webowej udało się przeprowadzić 32 testy, z których 18 użytkowników postanowiło dodatkowo wypełnić krótki formularz zawierający pytania na temat działania wszystkich trzech zaimplementowanych usług oraz kilka pytań ogólnych.

Jedną z najważniejszych informacji uzyskanych od użytkowników podczas tych testów była odpowiedź na pytanie, czy usługa wykrywa większość uśmiechów – odpowiedzi te pokazuje Rys. 29.

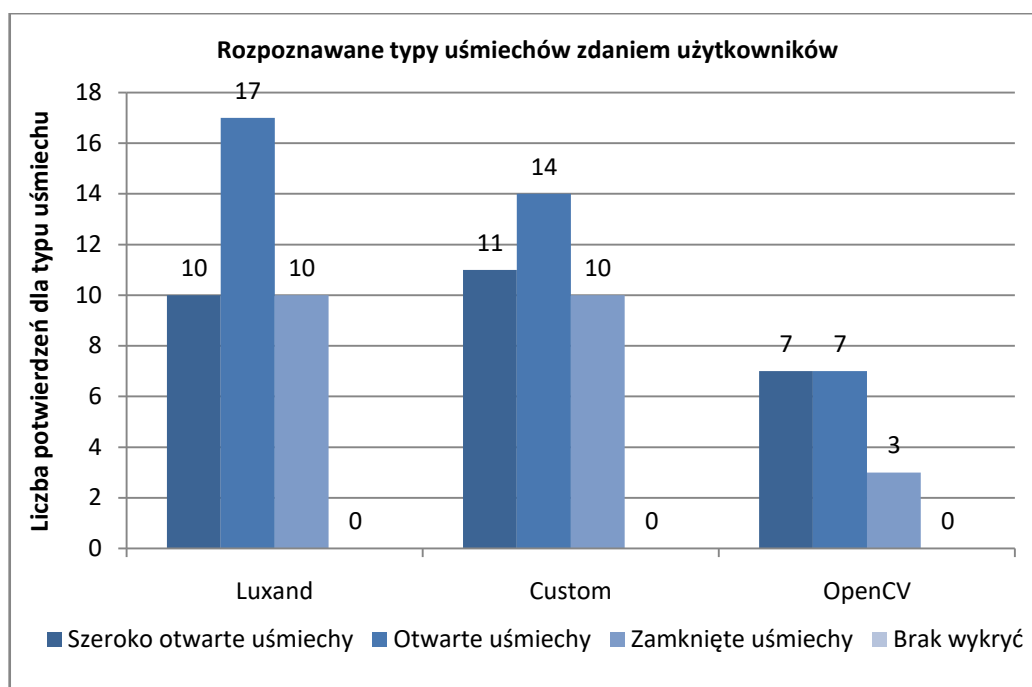




**Rys. 29. Wykres przedstawiający opinie na temat działania usług**

Najgorzej w badaniu wypadła usługa OpenCV – zdania użytkowników były bardzo podzielone. Znacznie lepiej jest w przypadku obydwu pozostałych usług: zarówno dla usługi FaceSDK, jak i utworzonej poprzez połączenie biblioteki Luxandu z OpenCV użytkownicy nie dodali ani jednej negatywnej opinii.

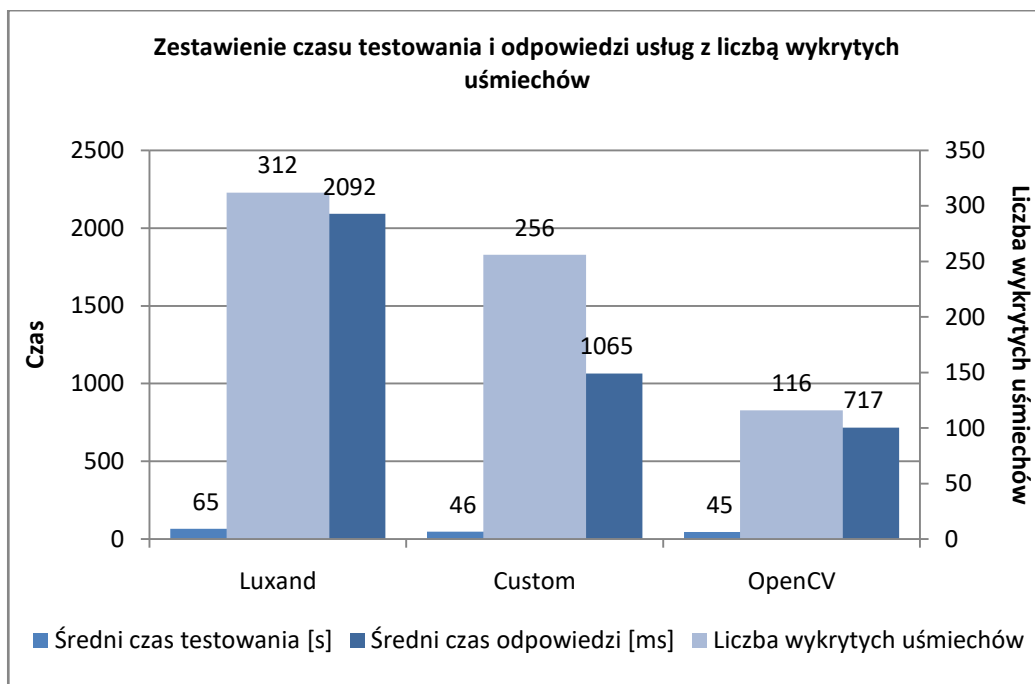
W odpowiedzi na pytanie, dlaczego OpenCV otrzymało tak niewiele głosów pozytywnych, może okazać się zestawienie wyników, jak użytkownicy oceniali typy wykrywanych uśmiechów przez każdą z usług. Pytanie zostało zaprezentowane w formie pytania wielokrotnej odpowiedzi – użytkownicy mogli ocenić, czy usługa wykrywała szeroko otwarte uśmiechy, zwykle otwarte uśmiechy oraz zamknięte uśmiechy. Dodatkową opcją było zaznaczenie, że usługa nie wykrywała żadnych uśmiechów. Takie dane zostały pokazane na Rys. 30.



**Rys. 30. Przedstawienie typów wykrywanych uśmiechów zdaniem użytkowników**

Powyższy wykres pokazuje, że zdaniem użytkowników żadna z usług nie miała całkowitych problemów z wykrywaniem uśmiechów. Usługi zdecydowanie najlepiej radziły sobie z detekcją uśmiechów otwartych, a uśmiechy zamknięte były najlepiej wykrywane przez implementacje wykorzystujące do tego bibliotekę FaceSDK. Warto także zauważyć, że klasyfikator uśmiechu dla OpenCV okazał się być przeciętny w skuteczności, ponieważ nawet uśmiechy otwarte były wykrywane rzadziej niż w przypadku konkurencyjnych usług.

W celu zweryfikowania, czy liczba wykryć uśmiechów przez usługę OpenCV jest tak niska ze względu na „opuszczanie” testów usługi przez użytkowników, zdecydowano się zestawzić ze sobą czas spędzony na testowaniu usługi z liczbą uśmiechów przez nią wykrytych. Informacje te zostały pokazane na Rys. 31.



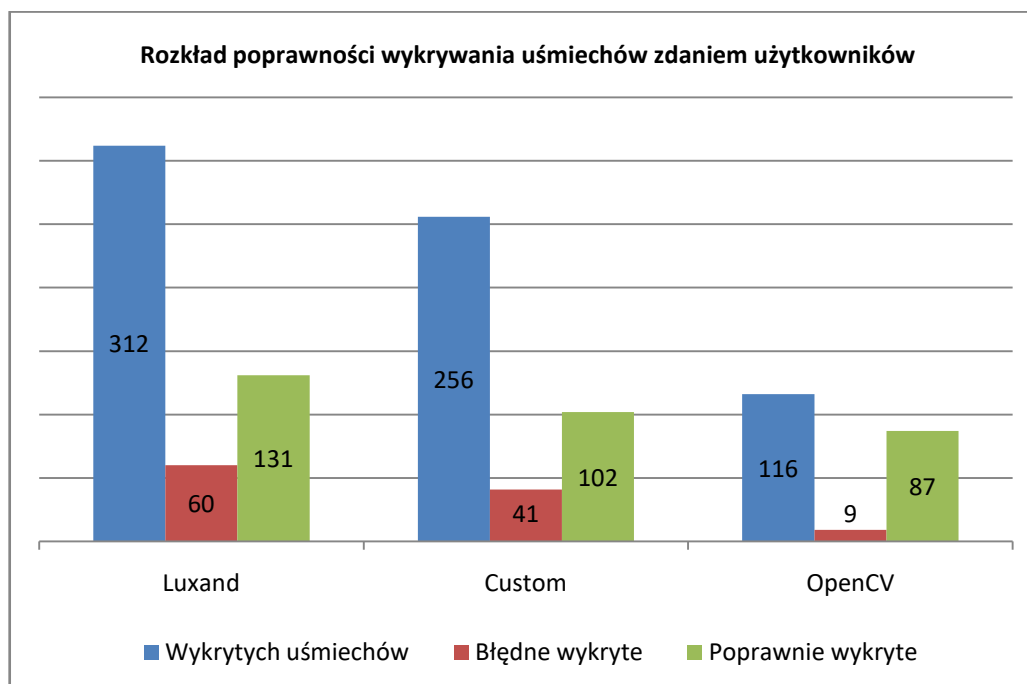
**Rys. 31. Wykres prezentujący zależność wykrytych uśmiechów do czasu testowania**

Na przedstawionym wykresie łatwo można zauważyć, że użytkownicy najwięcej czasu spędzali przy usłudze FaceSDK – średnio czas ten wynosił ok. jednej minuty. Kolejną obserwacją jest fakt, że zarówno mieszana implementacja, jak i usługa oparta o OpenCV były średnio testowane przez czas zbliżony do tej wartości – były to czasy rzędu odpowiednio 46 oraz 45 sekund. W związku z taką informacją można wykluczyć, że na niewielką liczbę wykrytych uśmiechów miał wpływ czas spędzony na testowaniu usługi OpenCV.

Kolejną statystyką możliwą do odczytania z wykresu jest średni czas odpowiedzi z serwera dla każdej z usług. Najlepiej radziła sobie z tym biblioteka OpenCV, niewiele gorzej implementacja łącząca obie funkcjonalności. Najgorszy czas przypadł usłudze Luxandu i wynosił aż 2 s. Stało się tak ze względu na niektóre testy użytkowników, w których czas odpowiedzi na pojedyncze żądanie wynosił aż 6-7 sekund. Ten sam użytkownik zawyżał średnie czasy również dla pozostałych usług – po odrzuceniu tych testów biblioteka OpenCV zwracała dane w czasie 500 ms, FaceSDK w czasie 1 122 ms, a wersja łączona w czasie 718 ms. Warto w tym miejscu przypomnieć, że czas procesowania żądania zależał w dużej mierze od wydajności komputera i prędkości przesyłania połączenia z Internetem użytkownika – obciążeniem serwera sterowano ustalaniem kolejności wykonywania testów, użytkownicy byli proszeni o trzymanie się zaplanowanego harmonogramu w taki sposób, aby tylko jedna osoba na raz testowała usługi.

Sama liczba wykrytych uśmiechów nie informuje o tym, czy usługa wykrywała je w sposób prawidłowy – zawsze istnieje ryzyko powstawania *false alarmów*. Aby sprawdzić częstotliwość ich występowania dla wszystkich usług, użytkownicy mieli możliwość oznaczenia uśmiechu jako poprawnie i błędnie wykrytego. Zależność pomiędzy liczbą wszystkich wykrytych

uśmiechów a opinią użytkowników na temat poprawności działania mechanizmu detekcji przedstawia Rys. 32.



**Rys. 32. Zestawienie liczby wykrytych uśmiechów do skuteczności ich detekcji**

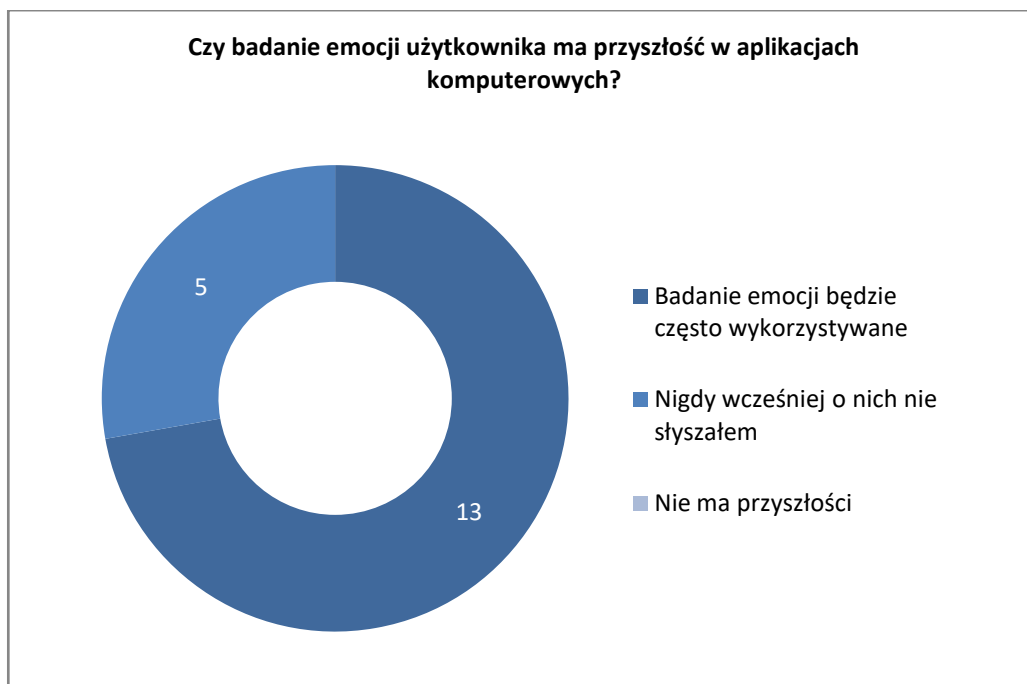
Jak widać na załączonym wykresie, tylko połowa użytkowników skorzystała z mechanizmów pozwalających na ocenę poprawności detekcji uśmiechów – jedynie testy usługi OpenCV zostały w 75% potwierdzone. Pośród uzyskanych wyników można zauważyć, że choć usługa ta wykrywała uśmiechy rzadziej od swoich rywali, w większości przypadków były to prawdziwe uśmiechy (aż 83%). W przypadku usług opartych o bibliotekę FaceSDK częstotliwość błędnego wykrywania uśmiechów była jednak dosyć wysoka i wynosiła około 19%. Wśród opinii użytkowników można uzyskać informacje, że wykrywały one często zamknięte usta – okazuje się, że próg uśmiechu ustawiony na 0.45 dla części użytkowników wskazywał usta w stanie spoczynku.

Sekcja ogólna formularza zawierała między innymi pytania o wiek i płeć – zgromadzone dane pokazuje *Tabela 11*. Widać tutaj, że najchętniej formularz wypełniały kobiety. Większość osób testujących usługi mieściła się w przedziale wiekowym od 20 do 25 lat.

**Tabela 11. Statystyki dotyczące testujących**

Kobiet	Mężczyzn	Wiek < 20	Wiek < 25	Wiek >= 25
11	7	4	10	1

Jedno z pytań zawartych w formularzu dotyczyło przyszłości usług procesujących emocje zdaniem użytkownika. Użytkownik mógł zaznaczyć, że usługi takie będą często wykorzystywane w przyszłości, inną opcją było przyznanie, że pierwszy raz słyszy o takiej funkcjonalności, a ostatnia możliwość pozwalała stwierdzić, że procesowanie emocji nie ma żadnej przyszłości w dziedzinie aplikacji komputerowych – wyniki przedstawia Rys. 33.



**Rys. 33. Odpowiedzi na pytanie o przyszłość usług badających emocje użytkownika**

Jak można zauważyć, zdecydowana większość użytkowników stwierdziła, że w przyszłości wykrywanie emocji użytkownika będzie pełniło znaczącą funkcję w oprogramowaniu komputerowym. Jeżeli użytkownik zaznaczył tę opcję, w formularzu dodatkowo pojawiało się pole z prośbą o uzasadnienie, w jakiego typu aplikacjach chciałby przetestować działanie takich usług.

Wśród udzielonych odpowiedzi najczęściej wskazywano portale społecznościowe – użytkownicy chcieliby, by aplikacje wykrywały twarze użytkowników na zdjęciach i podpowiadały, kto się na nich znajduje. Inne odpowiedzi dotyczyły aplikacji do fotografowania: tutaj wskazano, że badanie emocji mogłoby wykryć, czy wszyscy użytkownicy na zdjęciu się uśmiechnęli lub zrobili odpowiednio dziwną minę i tylko wtedy wykonać zdjęcie. Ostatnim wybranym zastosowaniem usług były gry – zarówno poziom trudności jak i tematykę gry można dostosować do nastroju użytkownika (np. prezentować otoczenie pełne kolorów w przypadku, kiedy użytkownik się smuci lub uspokajać go kojącą muzyką, kiedy się denerwuje).

Warto także zauważyć, że żaden z użytkowników nie twierdzi, że usługi do procesowania emocji nie mają żadnej przyszłości. Wydaje się to potwierdzać istniejący trend we wdrażaniu usług w różnych aplikacjach.

#### **4.1.5. Wnioski z przeprowadzonych testów**

Ponieważ konfiguracja projektu zakłada możliwość wyboru biblioteki używanej do detekcji uśmiechów, wszystkie z przedstawionych zostały zaimplementowane w aplikacji desktopowej.

OpenIMAJ, która wykazała najmniejszą wydajność podczas testów, zostanie ustawiona jako biblioteka zapasowa. Jest ona jedyną biblioteką, która nie wymaga dołączania zewnętrznej

dynamicznej biblioteki, dlatego też zostanie ustawiona jako domyślna w przypadku, gdy główna biblioteka nie załaduje się w poprawny sposób.

Przeprowadzone testy, zarówno w wersji desktopowej, jak i webowej, potwierdzają teorię o prawidłowości detekcji uśmiechów. Użytkownicy wydają się być szczególnie zadowoleni z działania usług, które do wykrywania uśmiechu wykorzystują bibliotekę FaceSDK. Zastosowanie jedynie usługi OpenCV wydaje się być jednak mniej pożądane – działanie klasyfikatora uśmiechu pozostawia wiele do życzenia.

Ze względu na połączenie dokładności działania i nieco lepszej wydajności od konkurencyjnych usług, głównym sposobem detekcji uśmiechów (wybraną domyślnie) będzie usługa, która wykrywa twarze przy pomocy OpenCV, a usta wykorzystując możliwości FaceSDK. Największą wadą tego rozwiązania jest uzależnienie programu od dwóch bibliotek dynamicznych (z czego jedna z nich jest biblioteką komercyjną), bez których program nie będzie działał prawidłowo.

Pomimo uzyskania lepszych czasów średniego procesowania zdjęcia (co może okazać się kluczowe w przetwarzaniu obrazu z kamery w czasie rzeczywistym), zdecydowano się wybrać klasyfikator Haara zamiast LBP. Główną przyczyną takiej decyzji jest różnica pomiędzy zwracanymi rozmiarami twarzy, co może powodować problemy podczas prezentowania efektów zależnych od tych wymiarów. Dodatkowo, niewielkie zwiększenie tym sposobem wydajności pracy aplikacji potencjalnie zmniejsza jej dokładność – zdecydowanie lepiej jest unikać jakichkolwiek *false alarmów* w aplikacji.

Klasyfikatory LBP wydają się mieć większe zastosowanie w przypadku mniej wydajnych urządzeń niż komputery stacjonarne, na które została napisana aplikacja desktopowa Smilecounter. Jeżeli zaistnieje potrzeba uruchomienia programu na mniej wydajnych maszynach, można w łatwy sposób zaimplementować nową usługę, która oprócz wykorzystania tego typu klasyfikatorów dodatkowo będzie skalowała rozmiary twarzy w taki sposób, by format był zbliżony do pozostałych usług.

## **4.2. Ewaluacja rozwiązania**

Zgodnie z założeniami, aplikacja *Smilecounter* powinna zostać wdrożona w ośrodku dla dzieci chorych na autyzm.

### **4.2.1. Wdrożenie**

## **4.3. Walidacja**

## 5. PODSUMOWANIE

Projekt został zrealizowany zgodnie z konceptem – podzielono go na dwie wersje: desktopową i webową. Detekcja uśmiechów została wydzielona do niezależnych od wersji usług, które udostępniają proste API do przetwarzania obrazów. Obie posiadają dostęp do wspólnej bazy danych (opartej o technologię *MongoDB*) poprzez wspomniane wcześniej usługi. Dzięki takiemu podejściu można wykorzystać moduł z usługami także w innych projektach.

## WYKAZ LITERATURY

1. Snapchat – Statistics & Facts, <https://www.statista.com/topics/2882/snapchat/> (data dostępu: 09.09.2017)
2. Smile detection with OpenCV, the “nose” trick, <https://aminesehili.wordpress.com/2015/09/20/smile-detection-with-opencv-the-nose-trick/> (data dostępu: 09.09.2017)
3. Lista klasyfikatorów Haara, <https://github.com/opencv/opencv/tree/master/data/haarcascades> (data dostępu: 09.09.2017)
4. MongoDB and Oracle Compared, <https://www.mongodb.com/compare/mongodb-oracle> (data dostępu: 09.09.2017)
5. Haar Cascades, <http://alereimondo.no-ip.org/OpenCV/34> (data dostępu: 09.09.2017)
6. OpenCV now supports desktop Java, <http://opencv.org/opencv-java-api.html> (data dostępu: 09.09.2017)
7. Freitas-Magalhães, A.; Castro, E.=. Facial Expression: The Brain and The Face. Porto: University Fernando Pessoa Press. pp. 1–18. ISBN 978-989-643-034-4.
8. Duchenne, Guillaume (1990). The Mechanism of Human Facial Expression. New York: Cambridge University Press
9. Anthony H.L. Tjan.. Dr. Dent., Some esthetic factors in a smile (<http://www.sciencedirect.com/science/article/pii/S0022391384800979>)
10. The Science of the Smile, <https://www.uveneer.com/veneervault/science-of-the-smile/> (data dostępu: 09.09.2017)
11. OpenIMAJ: Intelligent Multimedia Analysis, <http://openimaj.org/> (data dostępu: 09.09.2017)
12. Training Haar-cascade, <https://singhgaganpreet.wordpress.com/2012/10/14/training-haar-cascade/> (data dostępu: 09.09.2017)
13. Face Detection using Haar Cascades, [http://docs.opencv.org/trunk/d7/d8b/tutorial\\_py\\_face\\_detection.html](http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) (data dostępu: 09.09.2017)
14. Haar Cascade Visualization, <https://www.youtube.com/watch?v=hPCTwxF0qf4> (data dostępu: 09.09.2017)
15. Explaining AdaBoost, <http://rob.schapire.net/papers/explaining-adaboost.pdf> (data dostępu: 09.09.2017)
16. Boosting and AdaBoost for Machine Learning, <https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/> (data dostępu: 09.09.2017)
17. Cohn-Kanade (CK and CK+) database, <http://www.consortium.ri.cmu.edu/ckagree/> (data dostępu: 09.09.2017)
18. The Japanese Female Facial Expression (JAFPE) Database, <http://www.kasrl.org/jaffe.html> (data dostępu: 09.09.2017)
19. Luxand FaceSDK Developer's Guide, <https://www.luxand.com/facesdk/documentation/> (data dostępu: 09.09.2017)



20. Klasyfikator Haara dla twarzy,  
[https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade\\_frontalface\\_alt.xml](https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml) (data dostępu: 09.09.2017)
21. Klasyfikator Haara dla ust, <http://alereimondo.no-ip.org/OpenCV/uploads/34/Mouth25x15.1.zip> (data dostępu: 09.09.2017)
22. Klasyfikator Haara dla uśmiechu,  
<https://github.com/lukagabric/PyOpenCV/blob/master/Resources/smile.xml> (data dostępu: 09.09.2017)
23. Luxand FaceSDK – Face Detection,  
<https://www.luxand.com/facesdk/documentation/facedetection.php> (data dostępu: 09.09.2017)
24. BOOSTING (ADABOOST ALGORITHM),  
<http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Eric-Boosting304FinalRpdf.pdf> (data dostępu: 09.09.2017)
25. Klasyfikator LBP dla twarzy,  
[https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade\\_frontalface\\_improved.xml](https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade_frontalface_improved.xml) (data dostępu: 09.09.2017)
26. A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery, [http://epacis.net/jcis/PDF\\_JCIS/JCIS11-art.0101.pdf](http://epacis.net/jcis/PDF_JCIS/JCIS11-art.0101.pdf)
27. Cascade Classifier Training,  
[http://docs.opencv.org/3.3.0/dc/d88/tutorial\\_traincascade.html](http://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html)

## WYKAZ TABEL

Tabela 1. Informacje o zrealizowanych efektach .....	33
Tabela 2. Lista parametrów konfiguracyjnych dla aplikacji desktopowej .....	39
Tabela 3. Lista parametrów konfiguracyjnych dla części frontendowej .....	49
Tabela 4. Własnoręcznie utworzony zbiór testowy .....	52
Tabela 5. Wyniki graficzne testów na własnoręcznie utworzonym zbiorze .....	53
Tabela 6. Wyniki testów usługi OpenCV + FaceSDK dla utworzonego zbioru zdjęć.....	55
Tabela 7. Statystyki usług z testów własnoręcznie utworzonego zbioru .....	56
Tabela 8. Wyniki testów zbioru Cohn-Kanade .....	56
Tabela 9. Wyniki testów zbioru JAFFE .....	59
Tabela 10. Zastosowanie klasyfikatora LBP dla obu zbiorów .....	61
Tabela 11. Statystyki dotyczące testujących .....	68

## WYKAZ RYSUNKÓW

Rys. 1. Różnice pomiędzy uśmiechem Pan Am oraz Duchenne .....	10
Rys. 2. Przedstawienie stopni uśmiechu otwartego .....	10
Rys. 3. Pseudokod algorytmu AdaBoost .....	14
Rys. 4. Cechy obrazu badane przez klasyfikator .....	15
Rys. 5. Przykład nałożenia cech Haara na zdjęciu .....	15
Rys. 6. Przykład działania klasyfikatora LBP .....	17
Rys. 7. Przykład działania MBLBP .....	18
Rys. 8. Różnice w określaniu cech twarzy przez klasyfikatory .....	18
Rys. 9. Fragment zebranego zbioru próbek pozytywnych .....	20
Rys. 10. Fragment zebranego zbioru próbek negatywnych .....	20
Rys. 11. Koncepcja architektury systemu .....	22
Rys. 12. Przykład działania klasyfikatora uśmiechu przy użyciu OpenCV .....	26
Rys. 13. Przykład detekcji uśmiechu dla połączonych klasyfikatorów .....	27
Rys. 14. Ekran podglądu obrazu z kamery w aplikacji desktopowej .....	37
Rys. 15. Ekran ustawień w aplikacji desktopowej .....	38
Rys. 16. Ekran główny aplikacji webowej .....	42
Rys. 17. Ekran wstępny testowania usług wykrywających uśmiech .....	43
Rys. 18. Główny ekran testowania usług wykrywających uśmiech .....	44
Rys. 19. Ekran podsumowujący testy usług .....	46
Rys. 20. Ekran zapisywania uśmiechów wykrytych podczas testu usług .....	47
Rys. 21. Ekran pobierania aplikacji desktopowej .....	48
Rys. 22. Przykład <i>false alarmu</i> dla biblioteki Luxand .....	58
Rys. 23. Przykład niewykrycia ust przez bibliotekę OpenCV .....	58
Rys. 24. Porównanie rozmiarów twarzy i ust zwróconych przez biblioteki .....	59
Rys. 25. Porównanie detekcji w przypadku otwartych ust .....	60
Rys. 26. Porównanie detekcji w przypadku grymasu .....	61
Rys. 27. Różnice w detekcji pomiędzy klasyfikatorami Haara i LBP .....	62
Rys. 28. Przykład wystąpienia <i>false alarmu</i> przy użyciu klasyfikatora LBP .....	63
Rys. 29. Wykres przedstawiający opinie na temat działania usług .....	65
Rys. 30. Przedstawienie typów wykrywanych uśmiechów zdaniem użytkowników .....	66
Rys. 31. Wykres prezentujący zależność wykrytych uśmiechów do czasu testowania ..	67
Rys. 32. Zestawienie liczby wykrytych uśmiechów do skuteczności ich detekcji .....	68
Rys. 33. Odpowiedzi na pytanie o przyszłość usług badających emocje użytkownika ..	69