



**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	Kierownik Katedry/Zakładu
<i>podpis</i>	<i>podpis</i>
dr inż. Agnieszka Landowska	

Data oddania pracy do dziekanatu:



## 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 było utworzenie aplikacji automatycznie rozpoznającej uśmiech na podstawie obrazu z kamery. 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 jest skierowana przede wszystkim do dzieci, w tym dzieci z autyzmem i ma na celu zachęcenie ich do uśmiechu.

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

Aplikacja na komputery stacjonarne, oprócz zliczania 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 was to design and implement an application that automatically recognizes smiles. 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 for children, including children with autism, and aims at encouraging them to smile.

During the project two versions of the application were created. The first was 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 .....	6
1.1. Cele pracy .....	6
1.2. Motywacja podjęcia tematu .....	7
1.3. Zastosowanie pracy magisterskiej .....	7
1.4. Układ pracy.....	7
2. Analiza problemu.....	9
2.1. Definicja uśmiechu .....	9
2.2. Sposoby detekcji uśmiechu.....	10
2.3. Klasyfikatory i kaskady klasyfikatorów .....	12
2.4. Przegląd istniejących bibliotek do rozpoznawania twarzy i emocji .....	20
3. Koncepcja rozwiązania.....	23
3.1. Przyjęta metoda detekcji uśmiechu.....	23
3.2. Testy bibliotek .....	25
3.3. Wnioski z przeprowadzonych testów .....	37
3.4. Trenowanie własnego klasyfikatora do detekcji uśmiechu .....	38
3.5. Architektura systemu .....	40
4. Projekt aplikacji .....	42
4.1. Sposoby implementacji usług do detekcji uśmiechów .....	42
4.2. Wykrywanie uśmiechu w czasie rzeczywistym .....	44
4.3. Prezentowanie efektów zachęcających do uśmiechu .....	46
5. Implementacja .....	51
5.1. Aplikacja na komputery stacjonarne .....	51
5.2. Aplikacja internetowa .....	55
5.3. Kod źródłowy aplikacji .....	65
6. Ewaluacja gotowej aplikacji.....	66
6.1. Analiza statystyk oraz wyników ankiety z aplikacji internetowej .....	66
6.2. Testy z dziećmi typowo rozwijającymi się .....	72
6.3. Testy z dziećmi z autyzmem .....	74
6.4. Wdrożenie .....	76
Podsumowanie .....	77
Wykaz literatury .....	79
Wykaz tabel .....	81
Wykaz rysunków.....	82

## **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 wygląd twarzy 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 głównie algorytmom detekcji twarzy i jej charakterystyk – umożliwia użytkownikom nagrywanie filmów i wykonywanie 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 np. do zabezpieczania komputera przed dostępem niepowołanych osób. Innym przykładem zastosowania tego typu funkcjonalności jest detekcja uśmiechu podczas wykonywania zdjęć aparatem cyfrowym.

### **1.1. Cele pracy**

Głównym celem niniejszej pracy jest utworzenie aplikacji na komputery stacjonarne, 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 pokazana w formie statystyk na ekranie z prezentowaniem obrazu z kamery. Wartość ta powinna być na bieżąco aktualizowana. Program powinien także zachętać użytkownika do wykonania uśmiechu, poprzez prezentowanie prostych efektów na obrazie z kamery.

Prócz wspomnianych wcześniej celów podstawowych, aplikacja na komputery stacjonarne powinna dodatkowo udostępniać możliwość przypisania 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).

Zakres pracy magisterskiej został rozszerzony o utworzenie także aplikacji internetowej, której głównym zadaniem jest testowanie wykrywania uśmiechów przy pomocy zaimplementowanych usług.

W trakcie tworzenia aplikacji przewidziano możliwość wyboru biblioteki – implementacja powinna pozwolić na zmianę i wypróbowanie dowolnej 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.

Efektem pracy magisterskiej są dwie aplikacje posiadające wspólną bazę danych: aplikacja internetowa oraz na komputery stacjonarne. Obie powinny działać przy pomocy współdzielonych usług oferujących detekcję uśmiechu, co pozwoli na równoległą pracę przy obu wersjach programu.

### **1.2. Motywacja podjęcia tematu**

Główną motywacją podjęcia tematu jest zainteresowanie autora pracy magisterskiej dziedziną przetwarzania obrazu oraz 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 z obrazu 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 rozwinać go o dodatkowe cechy.

### **1.3. 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 uśmiechnięcia się. Ponieważ aplikacja jest tworzona z myślą o dzieciach, które mają problemy z okazywaniem emocji, poprzez zliczanie uśmiechów można śledzić postępy w ich rozwoju w różnych lokalizacjach.

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. Dodatkowo, dzięki zrealizowanemu sposobowi testowania usług przy pomocy aplikacji internetowej, można także zapisywać obrazy zawierające błędnie wykryty uśmiech, co pomoże w ulepszeniu detekcji.

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.

### **1.4. Układ pracy**

W trakcie tworzenia niniejszej pracy założono, że kolejność rozdziałów będzie odpowiadać etapom postępu nad rozwojem aplikacji.

W rozdziale 2. *Analiza problemu* (strona 9.) zdefiniowano czym jest uśmiech oraz jakie są główne sposoby jego detekcji. Szczególną uwagę poświęcono klasyfikatorom ze względu na ich popularność oraz szybkość działania.

Rozdział 3. *Koncepcja rozwiązania* (strona 23.) zawiera informacje na temat rozpoznanych bibliotek, które można wykorzystać do detekcji uśmiechu. Można w nim znaleźć wyniki testów każdej z nich oraz dane na temat tego, która została wybrana jako domyślna.

W rozdziale 4. *Projekt aplikacji* (strona 42.) zawarto informacje o sposobie implementacji usług opartych o rozpoznanie wcześniejszej biblioteki. Znajdują się w nim też problemy związane z detekcją uśmiechu w czasie rzeczywistym oraz informacje o sposobach, w jakie zostały one rozwiązane. Rozdział zamyka lista zrealizowanych efektów nagradzających oraz zachęcających do uśmiechania się wraz z informacjami o sposobie ich implementacji.

Rozdział 5. *Implementacja* (strona 51.) pokazuje, w jaki sposób została zaimplementowana aplikacja na komputery stacjonarne oraz aplikacja webowa, wraz z informacjami na temat ich konfiguracji.

W rozdziale 6. *Ewaluacja gotowej aplikacji* (strona 66.) pokazano, w jaki sposób przebiegała ewaluacja zaimplementowanych aplikacji. Znajdują się w nim wyniki testów z wersji internetowej programu oraz opisany został proces testów z dziećmi typowo rozwijającymi się oraz chorymi na autyzm. Można w nim także znaleźć informacje na temat wdrożenia aplikacji na Politechnice Gdańskiej.

## 2. ANALIZA PROBLEMU

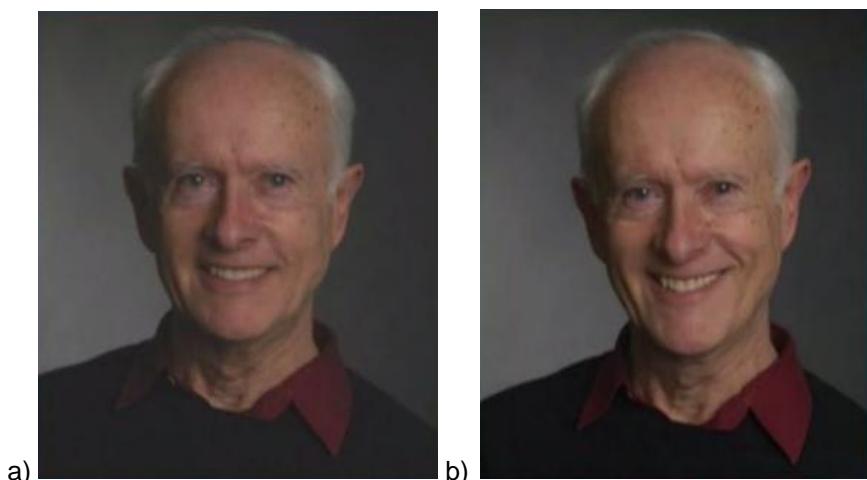
### 2.1. Definicja uśmiechu

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 kąciki ku górze [5]. Uśmiech najczęściej wiąże się z emocją zdefiniowaną jako radość – ponieważ jednak odgrywa on znacząną 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 [4]. Zauważył on, że podczas szczerego uśmiechania się powieki oczu rozszerzają się, a w ich kącikach pojawiają się zmarszczki.

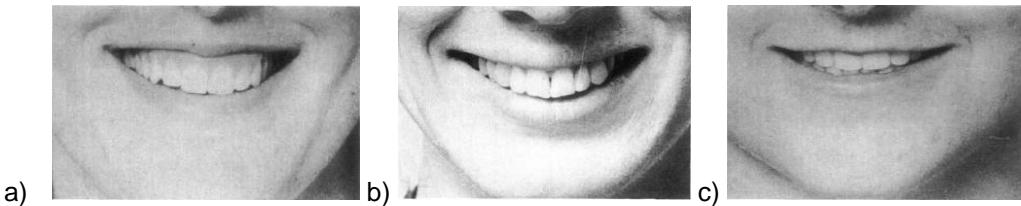
Rys. 1. Pokazuje różnicę pomiędzy wspomnianym wcześniej uśmiechem Duchenne a tzw. uśmiechem Pan Am (tzw. „sztucznym”, zwanym także „*botox smile*”). Zyskał on swoją nazwę od amerykańskich linii lotniczych *Pan American World Airways*, których stewardessy stały się stereotypem tego uśmiechu [3].



Rys. 1. Różnice pomiędzy uśmiechem Pan Am oraz Duchenne [3]

a) Uśmiech Pan Am, b) Uśmiech Duchenne

Kolejną zmienną różnicającą poszczególne uśmiechy jest ich stopień. Im większa emocja towarzyszy człowiekowi, tym jego uśmiech zazwyczaj staje się szerszy. Prócz uśmiechu zamkniętego, w którym uniesione są jedynie kąciki ust, istnieje także uśmiech otwarty, którego stopnie można podzielić na trzy grupy ([6]), co zostało przedstawione na przykładzie Rys. 2.



Rys. 2. Przedstawienie stopni uśmiechu otwartego [6]

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 kącików 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, a jego cechy różnią się pomiędzy poszczególnymi rasami ludzi. Chociaż każdy z nas posiada własne cechy uśmiechu, dentysi najczęściej definiują trzy ich główne typy [11]. Pierwszy z nich, zwany *commissure smile*, jest najbardziej typowym wzorcem (posiada go aż 67% populacji) i polega na tym, że kąciki ust są podciągnięte ku górze, a usta się rozciągają w kształcie łuku. Taki uśmiech posiadają m.in. Frank Sinatra czy Jennifer Aniston. Drugim rodzajem uśmiechu jest tzw. *cuspid smile*, który posiada ok. 31% populacji i w którym usta zazwyczaj podczas uśmiechania są ułożone na kształt diamentu (np. Elvis czy Tom Cruise). Ostatnia grupa, którą posiada ok. 2% populacji, nazwany został *complex smile*, w którym usta przypominają dwa równolegle do siebie położone litery V (kształt *chevron*). Taki uśmiech miała m.in. Marylin Monroe.

W ramach pracy magisterskiej zdecydowano się wykrywać uśmiechy bez względu na to, czy uśmiech jest szczerzy, 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.

## 2.2. Sposoby detekcji uśmiechu

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

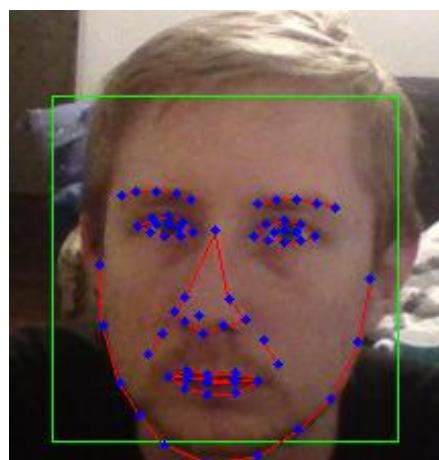
Do detekcji uśmiechu wystarczy odnalezienie twarzy użytkownika – na niej znajdują się wszelkie informacje pozwalające określić, czy osoba się uśmiecha, czy nie. Głównymi metodami służącymi do odpowiedzi na to pytanie są: zbadanie emocji użytkownika (radość zazwyczaj definiowana jest przy pomocy uśmiechu)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

### 2.2.1. 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 postaci macierzy. Typowymi emocjami uwzględnianymi przez takie biblioteki są: smutek, gniew, strach, zaskoczenie, wstręt oraz radość. Do wykrywania uśmiechu najbardziej przydatną emocją jest właśnie 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. Na Rys. 3. pokazano przykład wyświetlania 70 punktów przy pomocy biblioteki FaceSDK firmy Luxand.



Rys. 3. Punkty kluczowe twarzy wyświetlane przy pomocy Luxand FaceSDK [opr. wł.]

Ponieważ zadaniem niniejszej pracy magisterskiej jest detekcja uśmiechu (a nie emocji radości), do prawidłowej 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 na znaczące 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.

#### *2.2.2. 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 klasyfikatorów) 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, ponieważ przy jego pomocy proces wyszukiwania uśmiechu jest stosunkowo szybki, w porównaniu do metody wskazywania położenia punktów kluczowych twarzy. Kolejną zaletą jest fakt, że klasyfikatory mogą być wykorzystywane do wyszukiwania dowolnych rodzajów obiektów na obrazie.

Dokładność wskazywanych uśmiechów w głównej mierze zależy od klasyfikatorów (od liczby zebranych próbek pozytywnych i negatywnych, ich jakości, zróżnicowaniu, stosunku itd.). Część bibliotek pozwala także na wprowadzenie dodatkowych ustawień (np. stosunek wysokości do szerokości w wykrywanym uśmiechu). Warto jednak zauważyć, że w większości przypadków dokładność będzie mniejsza niż przy wykorzystaniu rozpoznawania punktów kluczowych 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.

### **2.3. Klasyfikatory i kaskady klasyfikatorów**

Klasyfikatory 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. Są one 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, rzadziej spotykany jest klasyfikator typu HOG (*histogram of oriented gradients*). Różnica pomiędzy nimi polega głównie na sposobie przedstawiania informacji, co prowadzi do różnic w dokładności i szybkości działania algorytmu.

### 2.3.1. Boosting i jego wykorzystanie w klasyfikatorach

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 [9, 12], a zatem służy do zwiększenia skuteczności dowolnego algorytmu uczenia.

Polega ona 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, 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 – czyli próbka pozytywna lub negatywna), natomiast wartość decyzyjna  $y_i$  określa, czy na danym obrazie znajduje się uśmiech. Podstawowym założeniem działania algorytmu AdaBoost jest stwierdzenie, że dla dowolnego rozkładu, jeżeli otrzymamy wystarczająco dużą liczbę próbek pozytywnych i negatywnych, można w wielomianowym czasie otrzymać klasyfikator, który z dużym prawdopodobieństwem odpowie na pytania lepiej niż podejście losowania odpowiedzi [13]. 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 „czy wskazany rejon posiada kolor zębów” oraz „czy rejon otoczony jest ciemniejszym kolorem (wargą)?”. Owe mniejsze problemy nazywane są *słabyimi 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. 4 [9].

**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  $K_{silny}$ :

$$K_{silny}(x) = sign(\sum_{t=1}^T \alpha_t K_t(x))$$


```

Rys. 4. Pseudokod algorytmu AdaBoost

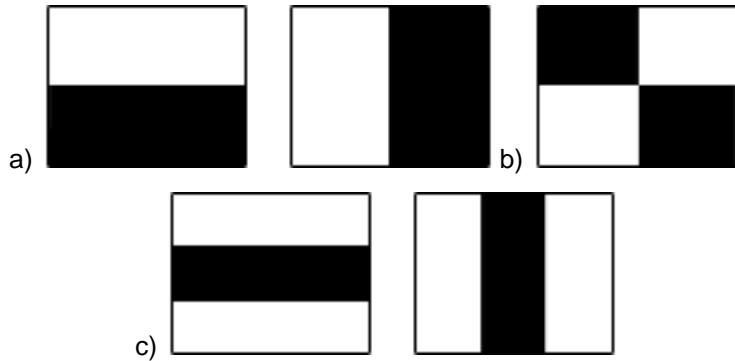
Każdy krok algorytmu tworzy nowy słaby klasyfikator, który stara się lepiej rozwiązać problemy poprzedniego. W pierwszym etapie każda instancja problemu otrzymuje równą wagę ( $\forall_{1 < i < n+1} w_x(i) = \frac{1}{n}$ ), a we wszystkich kolejnych iteracjach wagi są zwiększane instancjom, na które klasyfikatory nie były w stanie prawidłowo odpowiedzieć. Dzięki temu każdy kolejny krok algorytmu będzie starał się rozwiązać trudniejsze problemy. Każdemu z utworzonych klasyfikatorów jest ustawiony współczynnik  $\alpha_t$  na podstawie jego błędu  $e_t$ , obliczonego z uwzględnieniem wag wszystkich instancji – czyli im mniej się dany klasyfikator myli, tym ważniejszy będzie podczas próby udzielenia ostatecznej odpowiedzi.

Na zakończenie algorytmu zostaje zwrócony klasyfikator silny, który odpowiada na pytanie korzystając z utworzonych wcześniej klasyfikatorów słabych. Można przedstawić to w taki sposób, 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 [13].

### 2.3.2. Klasyfikatory Haara

Podstawową metodą porównywania obrazów przy użyciu tego typu 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. 5.

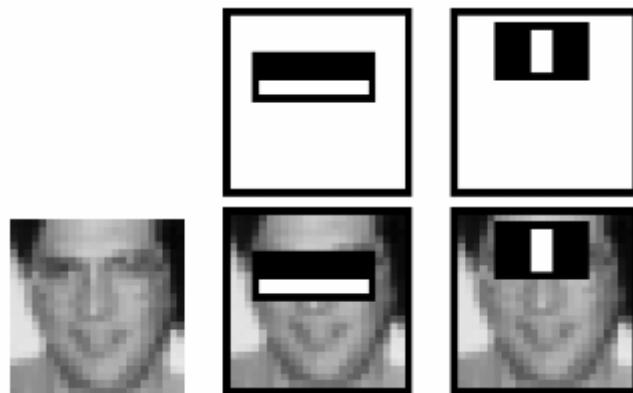


Rys. 5. Cechy obrazu badane przez klasyfikator [opr. wł.]

a) krawędzie b) cztery prostokąty c) linie

Jednostki cech Haara składają się z kilku prostokątów podzielonych na dwie 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 [17].

W ten sposób otrzymuje się prostą informację typu „dolina część jest jaśniejsza niż góra”. Tak uzyskane dane są następnie wykorzystywane do skategoryzowania przetwarzanego 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 policzki. Przykładowe wykorzystanie cech Haara na zdjęciu twarzy pokazuje Rys. 6.



Rys. 6. Przykład nałożenia cech Haara na zdjęciu [8]

Oprócz sposobu ułożenia sąsiadujących prostokątów, kolejnymi ważnymi parametrami jednostek cech Haara są ich rozmiar oraz położenie. Zwyczajne przeszukiwanie obrazu poprzez rozpoczęcie z jednostką  $2 \times 2$  pikseli i przesuwaniem jej po całym obrazie byłoby bardzo nieefektywne – już dla obrazu  $24 \times 24$  pikseli istnieje ponad 160 000 możliwości. Aby 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 ważnym 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 tym celu wprowadzono 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 przetwarzania 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 przetwarzanie 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].

W ramach badań prowadzonych nad klasyfikatorami wciąż ulepszane jest działanie kaskad klasyfikatorów. Jednym z rozwijanych podejść jest łączenie ze sobą dwóch różnych klasyfikatorów [8]. Informacje zwrócone przez każdy klasyfikator można pogrupować zgodnie z tablicą pomyłek, jaką pokazuje *Tabela 1*.

**Tabela 1. Tablica pomyłek wykorzystywana podczas detekcji uśmiechów**

		Czy uśmiech znajdował się we wskazanym fragmencie?	
		Tak	Nie
Czy klasyfikator wykrył twarz we wskazanym fragmencie?	Tak	<i>True positive</i> Uśmiech został prawidłowo wykryty	<i>False positive</i> Uśmiech został błędnie wykryty (błędy detekcji – ang. <i>false alarms</i> )
	Nie	<i>False negative</i> Uśmiech nie został wykryty	<i>True negative</i> Klasyfikator prawidłowo stwierdził, że we wskazanym fragmencie nie ma uśmiechu

Załóżmy, że mamy dwa klasyfikatory. Pierwszy jest nieco wolniejszy, ale za to potrafi bardzo dokładnie wykrywać uśmiechy (ma niewielki błąd dla przypadku *true positive*). Drugi klasyfikator jest bardzo szybki, prawie nigdy nie omija uśmiechu, ale za to w połowie przypadków wykrywa błędne detekcje (np. na ścianie). Wówczas można wykorzystać obydwa klasyfikatory w ramach jednej kaskady: drugi klasyfikator jako pierwszy bada przetwarzany fragment obrazu, a jeżeli stwierdzi, że może na nim znajdować się twarz, uruchamiany jest pierwszy z nich, aby znacząco zwiększyć pewność prawdziwej detekcji.

### 2.3.3. Klasyfikatory LBP

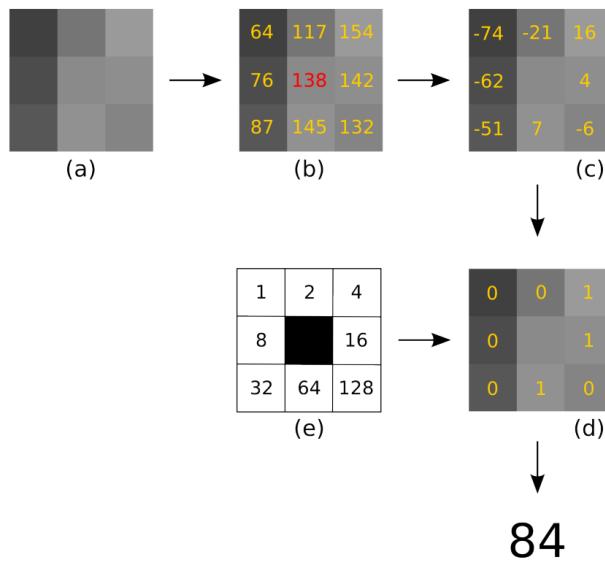
Innym popularnym sposobem implementacji klasyfikatora jest LBP – *Local Binary Pattern*. Służy on do badania zależności pomiędzy intensywnością pikseli na tekstuze [7]. W przypadku LBP wykorzystano fakt, że wspólne cechy (takie jak krawędzie, linie czy punkty) mogą być przedstawione przy pomocy wartości 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 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:

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

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. 7.

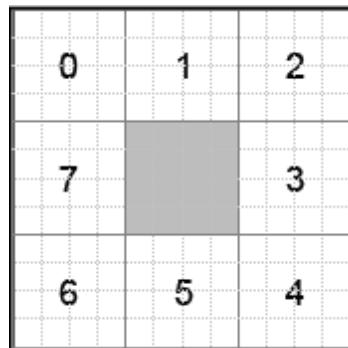


Rys. 7. Przykład działania klasyfikatora LBP [7]

Pierwszy etap przetwarzania fragmentu (a) pokazuje, jak wygląda on przed procesowaniem LBP. Każdy z nich zawiera jeden piksel środkowy, we wcześniejszym wzorze oznaczony jako  $p_{sr}$  oraz osiem pikseli sąsiadujących  $p_i$ . Kolejność sąsiadów na liście zazwyczaj jest ustalana jako zgodna z ruchem wskazówek zegara, począwszy od lewego górnego rogu. W kolejnym kroku (b) na każdym z pikseli umieszczono poziom jego intensywności. Etap (c) pokazuje naniesione różnice pomiędzy pikselem środka a jego

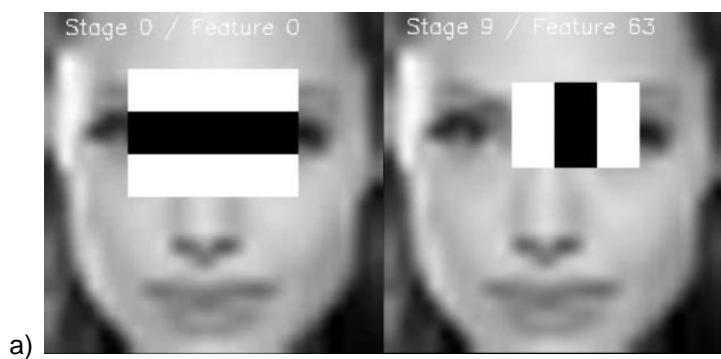
sąsiadami. Następnie (d) pokazuje, które z sąsiadujących pikseli otrzymały wartość 1, czyli są intensywniejsze od środka. Macierz oznaczona jako (e) prezentuje wagi każdego piksela zgodnie z jego kolejnością ( $2^i$ ). Ostatnim etapem jest dodanie do siebie wag pikseli, które w poprzednim kroku otrzymały wartość 1. Zastosowanie wielu takich bloków i złożenie informacji o ich sąsiedztwie pozwala na określenie stopnia podobieństwa dwóch fragmentów.

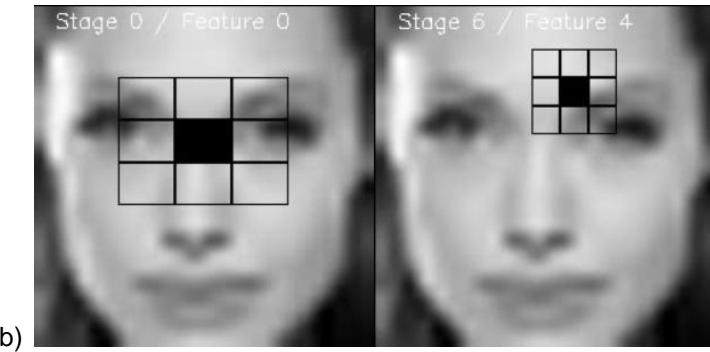
Zwyczajna wersja klasyfikatora jest zbyt lokalna i nie wystarczy do detekcji twarzy oraz uśmiechów. Rozszerzeniem LBP jest wieloblokowy LBP (*Multi-block LBP – MBLBP*), który zamiast operować na pojedynczych pikselach (środek i sąsiedzi), operuje na różnych grupach pikseli (najczęściej 3x3), a wartości intensywności w każdej grupie uśrednia [10] – zostało to pokazane na Rys. 8.



Rys. 8. Przykład działania MBLBP [opr. wł.]

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 [15]. Dodatkowo, podobnie jak w przypadku klasyfikatorów Haara, tutaj również wykorzystywany jest mechanizm kaskad.





Rys. 9. Różnice w określaniu cech twarzy przez klasyfikatory [15]

a) klasyfikator Haara, b) klasyfikator LBP

Rys. 9. pokazuje na przykładzie zdjęcia twarzy różnice w stosowaniu jednostek klasyfikatora do określania cech obiektów.

#### 2.3.4. 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 odnaleźć.

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 sąsiadnych 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 błędnych detekcji (ang. *false alarms*) podczas wykrywania 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) [16].

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 próbками negatywnymi (krok 2) oraz pozytywnymi (krok 3).

Jako wynik powyższych operacji otrzymuje się gotowy klasyfikator w postaci pliku *.xml*, który następnie można wykorzystać bibliotece.

## **2.4. Przegląd istniejących bibliotek do rozpoznawania twarzy i emocji**

Koncepcja rozpoznawania emocji użytkownika na podstawie obrazu 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 znajomość tego języka przez autora pracy magisterskiej oraz postawione wymagania o wieloplatformowość. Ze względu na konieczność przetwarzania w czasie rzeczywistym i 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).

### **2.4.1. 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 Intela) 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. *wrapper* 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, Lunux, MacOS, Android, iOs oraz BlackBerry. Od końca roku 2013 biblioteka posiada natywne API dla języka Java ([24]), wcześniej ten cel spełniała biblioteka JavaCV.

Działanie biblioteki opiera się o zastosowanie klasyfikatorów Haara oraz LBP, które wykorzystuje do wszystkich operacji detekcji fragmentów obrazu. Ze względu na jej popularność, w sieci można znaleźć wiele gotowych klasyfikatorów. Twórcy biblioteki udostępnili zbiór kilku gotowych klasyfikatorów [18], jednak znaleźć można również ciekawe ich rodzaje na stronach tworzonych przez społeczność [23]. Chociaż 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 przetwarzaniem wideo w czasie rzeczywistym.

#### 2.4.2. 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. używając technologię *Maven* [26], nie trzeba też załączać żadnych bibliotek dynamicznych. Przez swoją prostotę w wykorzystaniu biblioteka nieco traci na wydajności – 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 jego kanałów, wykrywanie krawędzi oraz metody do rysowania na obrazie, zarządzanie jego histogramem czy też porównywanie kilku obrazów ze sobą (wraz z ustaleniem punktów kluczowych odpowiadających na drugim obrazie).

Najważniejszą funkcjonalnością biblioteki jest detekcja zadanych elementów przy pomocy klasyfikatorów Haara. Chociaż OpenIMAJ posiada działający dosyć sprawnie własny klasyfikator twarzy, można także załadować kaskadę z pliku – obsługiwany jest 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).

#### 2.4.3. FaceSDK

FaceSDK jest jedną z komercyjnych bibliotek udostępnionych przez firmę Luxand [27], która zarejestrowanym użytkownikom daje możliwość otrzymania 30-dniowego, bezpłatnego klucza do korzystania z biblioteki (bez niego jakiekolwiek korzystanie z metod wystawionych przez twórców jest niemożliwe). FaceSDK jest wieloplatformowe – 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 firmy Luxand 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. KONCEPCJA ROZWIĄZANIA**

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 było ograniczenie się do detekcji przede wszystkim uśmiechów otwartych ze względu na trudności z ustaleniem uniwersalnego progu pewności dla uśmiechów zamkniętych – należało jednak spróbować wykrywać oba typy uśmiechów. Aplikacja powinna umożliwić wykrywanie wielu uśmiechów na raz, odpowiednio radząc sobie z ich zliczaniem.

W ramach pracy magisterskiej należało 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. Należało też dobrać odpowiedni rodzaj klasyfikatorów do prawidłowej pracy usług.

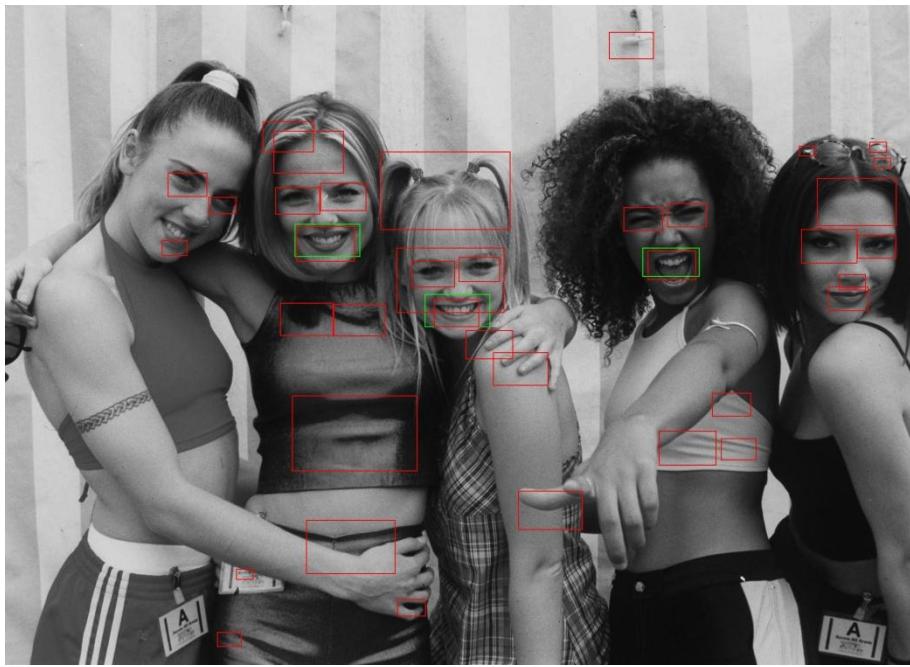
Przed przystąpieniem do implementacji postanowiono zrealizować testy bibliotek na różnych zbiorach danych. Dzięki wynikom takich testów można było ustalić, który sposób zostanie wykorzystany w aplikacji do detekcji uśmiechów. W testach tych należało także sprawdzić, czy wykorzystywane mechanizmy zadziałają poprawnie dla różnych ras ludzi.

Po zaimplementowaniu usług detekcji uśmiechów oraz aplikacji internetowej, przeprowadzono testy przy jej pomocy. Dzięki temu w testach wzięło udział więcej osób (użytkownicy częściej korzystali z aplikacji internetowej), co pozwoliło na przetestowanie działania mechanizmów detekcji w różnych warunkach.

#### **3.1. Przyjęta metoda detekcji uśmiechu**

Detekcja uśmiechów na obrazie w czasie rzeczywistym wymaga przetwarzania każdej klatki 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 błędnych detekcji (np. detekcja uśmiechów na ścianie) – częstym problemem jest uznawanie oczu jako ust, co przedstawiono na przykładzie Rys. 10. Przedstawia on pięć uśmiechniętych kobiet z popularnego zespołu *Spice Girls*.



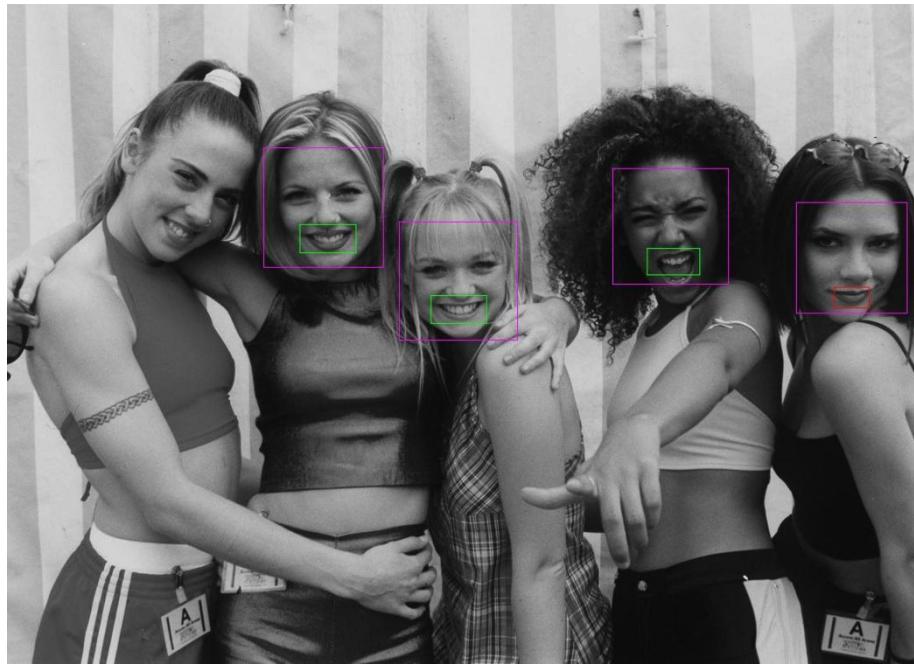
Rys. 10. Przykład działania klasyfikatora uśmiechu przy użyciu OpenCV

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

Jak można zauważyć, o ile klasyfikator uśmiechu działa w poprawie (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 [25]. 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. 11. Można na nim zobaczyć, że usługa poprawnie odnalazła pozycję ust dla każdej znalezionej twarzy (zaznaczonej na fioletowo).



Rys. 11. Przykład detekcji uśmiechu dla połączonych klasyfikatorów

Od razu można zauważać 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 jakiekolwiek 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, który działa głównie dla uśmiechów otwartych.

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

### 3.2. Testy bibliotek

Dla zweryfikowania wydajności i skuteczności detekcji uśmiechu przez poszczególne biblioteki, poddano je testom mierzącym czas przetwarzania obrazu oraz liczbę wykrytych uśmiechów. W tym celu napisany został moduł pozwalający pobranie wszystkich zdjęć zawartych we wskazanym katalogu, a następnie uruchomienie po kolejnych 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 był od momentu przekazania obrazu usłudze do momentu uzyskania odpowiedzi.

Wyniki tych testów pozwoliły na określenie wydajności oraz dokładności różnych zastosowanych rozwiązań, jednak dotyczyły jedynie wykrywania uśmiechów na pojedynczych klatkach. Oznacza to, że nie sprawdzały one pojawiania się efektów oraz wykrywania momentów rozpoczęcia i zakończenia uśmiechów – weryfikacja tych funkcjonalności odbyła się w późniejszych etapach.

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.

### 3.2.1. Własnoręcznie przygotowany zbiór obrazów

Pierwszym sposobem testowania było przygotowanie specjalnego zbioru obrazów, aby wstępnie 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. Rozdzielcość obrazów 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 2. 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 występowała w przypadku pań.

Tabela 2. Własnoręcznie utworzony zbiór testowy

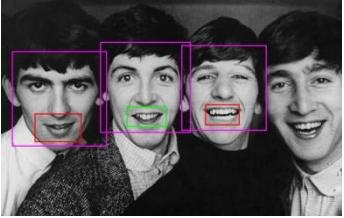
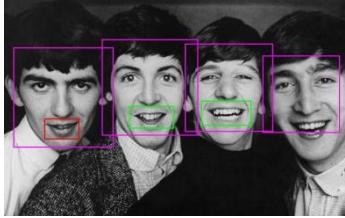
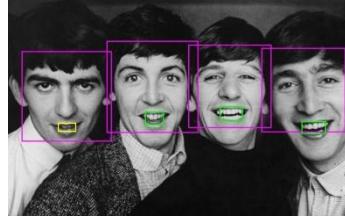
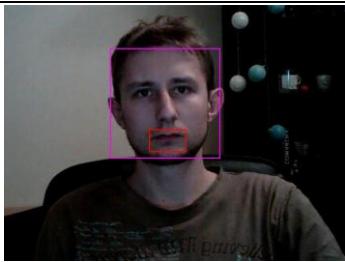
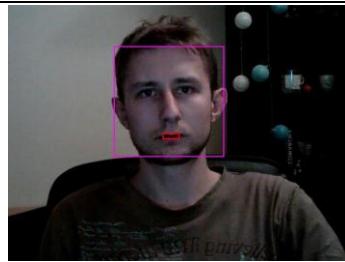
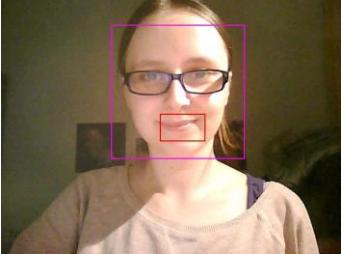
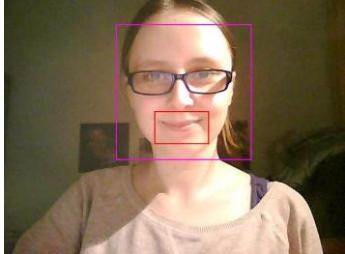
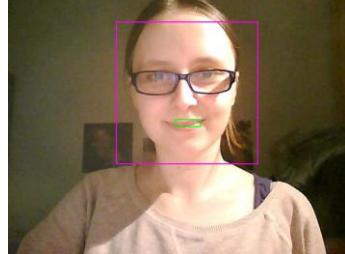
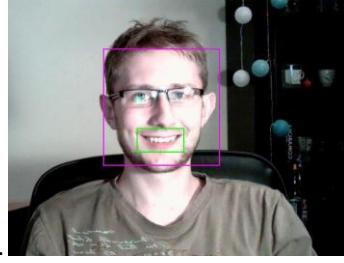
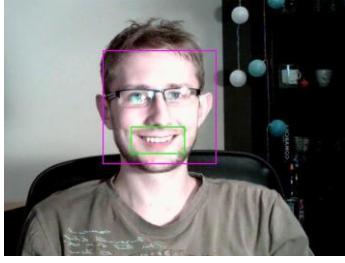
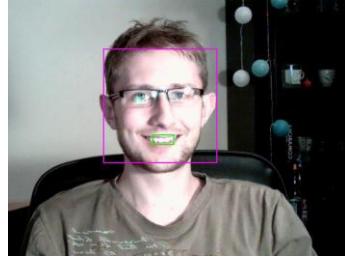
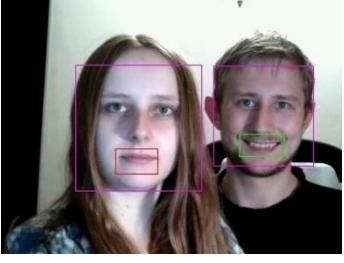
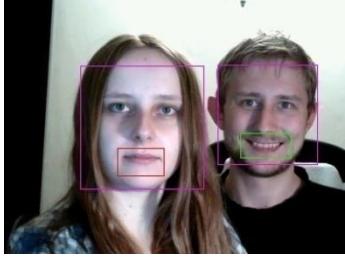
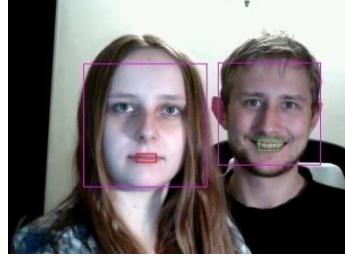
Lp.	Tytuł zdjęcia	Zdjęcie	Opis zdjęcia
1.	<b>The Beatles</b>		Czterech uśmiechniętych mężczyzn
2.	<b>Spice Girls</b>		Pięć uśmiechniętych kobiet z różnym stopniem uśmiechu (otwarte oraz zamknięte)

Lp.	Tytuł zdjęcia	Zdjęcie	Opis zdjęcia
3.	<b>Mężczyzna 1</b>		Mężczyzna, słabe oświetlenie, brak uśmiechu
4.	<b>Kobieta 1</b>		Kobieta, typowe oświetlenie, okulary, uśmiech zamknięty
5.	<b>Mężczyzna 2</b>		Mężczyzna, szeroki uśmiech, okulary, dobre oświetlenie
6.	<b>Kobieta i mężczyzna</b>		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. Do zaznaczenia stanu wykrytych ust zastosowano trzy różne kolory: czerwona ramka oznacza odnalezione usta bez uśmiechu, żółta ramka pokazuje uśmiechy w stanie zbliżonym do ustalonego progu, natomiast ramka zielona pokazuje wykryty uśmiech.

Wyniki takiego procesu zawiera *Tabela 3.*, w której pokazano detekcję twarzy i ust dla wszystkich trzech rozpoznanych bibliotek do wykrywania uśmiechu.

**Tabela 3. Wyniki graficzne testów na własnoręcznie utworzonym zbiorze**

OpenCV	OpenIMAJ	FaceSDK (Luxand)
		
		
		
		
		
		

Na załączonych zdjęciach można zauważyć, że najlepiej z wykrywaniem uśmiechów radzi sobie usługa FaceSDK firmy Luxand – 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.

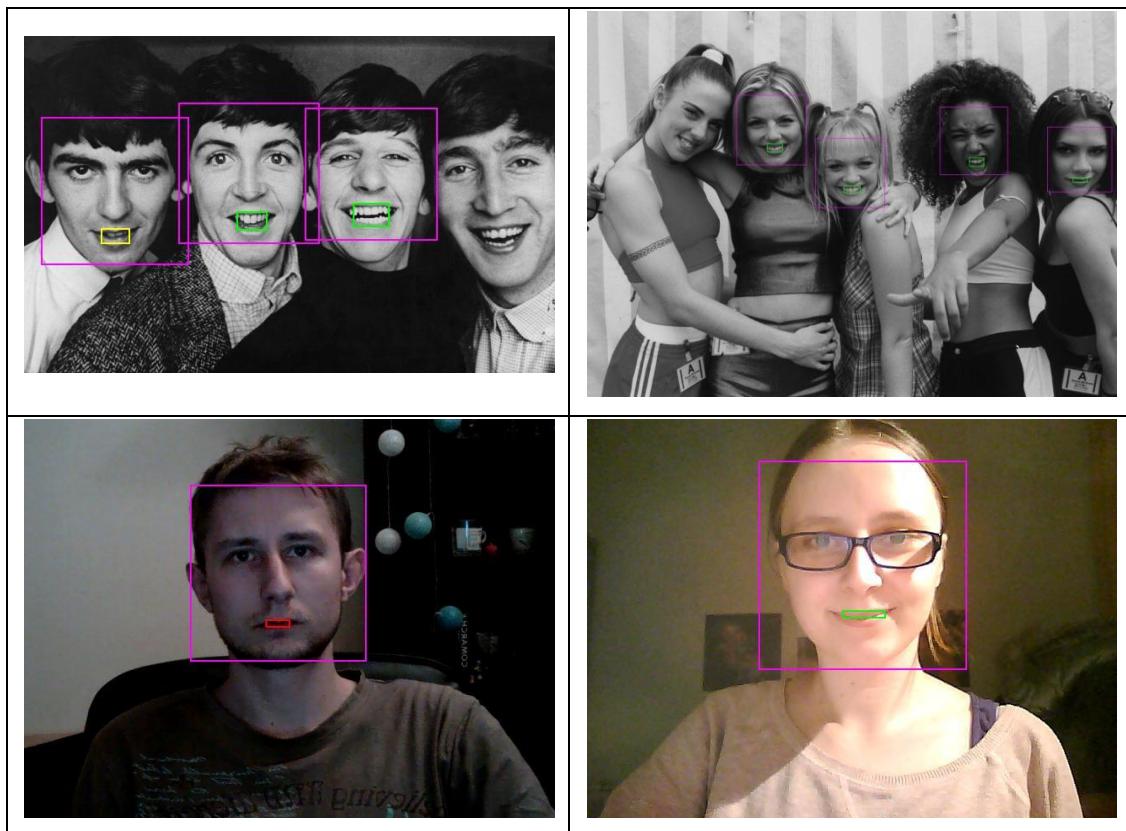
Klasyfikator uśmiechu wykorzystywany w usługach OpenCV oraz OpenIMAJ działa znacznie gorzej – w większości przypadków wykrywał jedynie uśmiechy otwarte. Co ciekawe, pomimo wykorzystania tego samego klasyfikatora, obie usługi zwróciły inną odpowiedź w przypadku jednego z członków zespołu *The Beatles*.

Kolejną obserwacją jest fakt, że usługa OpenIMAJ zwróciła dodatkowo dwie błędne detekcje, obydwie na zdjęciu o największej rozdzielczości (zdjęcie zespołu *Spice Girls*). Pierwsza z nich znajduje się na identyfikatorze dziewczyny po lewej, natomiast druga na ramieniu przedostatniej z dziewczyn.

W przypadku usługi OpenCV można zauważyc, 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).

Podczas późniejszej fazy testów (na zbiorach baz ludzkich twarzy) zauważono, że OpenCV wykrywa twarze szybciej niż FaceSDK, w dodatku zwraca mniej błędnych detekcji, natomiast biblioteka firmy Luxand bardzo dobrze radzi sobie z detekcją uśmiechu. W związku z taką obserwacją zdecydowano się na utworzenie czwartej usługi, której szczegóły implementacji znajdują się w rozdziale 4.1.4 (strona 44). Tabela 4. pokazuje wyniki testów dla tej usługi na własnoręcznie przygotowanym zbiorze.

Tabela 4. Wyniki testów usługi OpenCV + FaceSDK dla utworzonego zbioru zdjęć



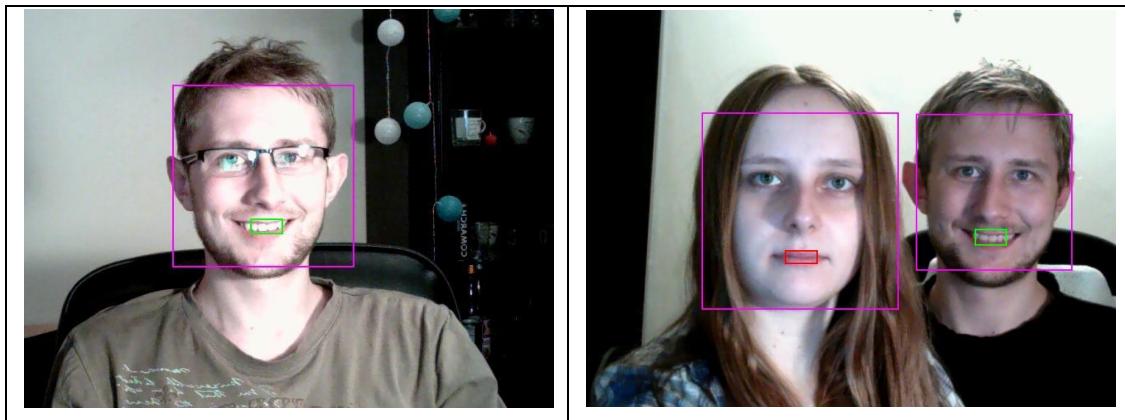


Tabela 4 pokazuje, że zastosowanie usługi łączącej OpenCV oraz FaceSDK jest także połączeniem ich skuteczności – dla wszystkich odnalezionych twarzy prawidłowo został określony uśmiech.

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

Podsumowanie wykonanych testów w formie prostych statystyk zawiera Tabela 5.

**Tabela 5. Statystyki usług z testów własnoręcznie utworzonego zbioru**

Nazwa usługi	Faktyczna liczba twarzy na zdjęciach	Faktyczna liczba uśmiechów na zdjęciach	Liczba wykrytych twarzy	Średni czas przetwarzania (bez dużego zdjęcia)	Liczba wykrytych uśmiechów (ust)
OpenCV	14	12	12	470 ms (278 ms)	6 (12)
OpenIMAJ	14	12	16	1 418 ms (585 ms)	7 (12)
FaceSDK	14	12	13	310 ms (257 ms)	11 (13)
OpenCV + FaceSDK	14	12	12	574 ms (223 ms)	10 (12)

Oprócz wspomnianych wcześniej informacji o dokładności wykorzystanych usług, można tutaj także znaleźć dane na temat średniego czasu działania usługi. Szczególnie ciekawy jest przypadek dla usługi FaceSDK – ponieważ skaluje ona wszystkie zdjęcia do jednego rozmiaru przed przystąpieniem do procesu detekcji uśmiechów, jej czas działania jest znaczco mniejszy niż w przypadku pozostałych usług. Po wykluszeniu ze średniej czasu przetwarzania 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.

### 3.2.2. Testy na istniejących bazach twarzy ludzi

Kolejnym etapem testów usług wykrywających uśmiech było wykorzystanie gotowych baz danych zawierających zdjęcie 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* [29], 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 6*.

**Tabela 6.** Wyniki testów zbioru Cohn-Kanade

Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarzą (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
<b>OpenCV</b>						
<b>Pozytywne</b>	194 ms	<b>227</b>	149 (207)	0	279x279 px	111x58 px
<b>Negatywne</b>	287 ms	<b>1 248</b>	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	<b>227</b>	223 ( <b>227</b> )	0	279x279 px	61x33 px
<b>Negatywne</b>	208 ms	<b>1 248</b>	138 ( <b>1 248</b> )	0	280x280 px	48x34 px

Pierwszą obserwacją, którą można zauważyc w prezentowanych wynikach jest fakt, że w przypadku usług korzystających z klasyfikatorów Haara przykłady pozytywne przetwarzanie 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 zauważać, 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 (błędy w detekcji).

Kolejną usługą jest OpenCV – o ile zgodnie ze wspomnianym wcześniej algorytmem przykłady pozytywne są przetwarzane 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 dobrym 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 przykładów pozytywnych, jak i negatywnych. Można łatwo zauważać, 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 błędnych detekcji.



Rys. 12. Przykład błędnej detekcji dla biblioteki Luxand

Pomimo faktu, że FaceSDK działa w sposób najbardziej dokładny, zdarza się jej zwrócić błędную informację o większej liczbie twarzy na pojedynczym zdjęciu (jednym z przykładów takiego działania jest Rys. 12.). 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 niewykrycia ust został zaprezentowany na Rys. 13.



Rys. 13. 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. 14. Porównanie rozmiarów twarzy i ust zwróconych przez biblioteki

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

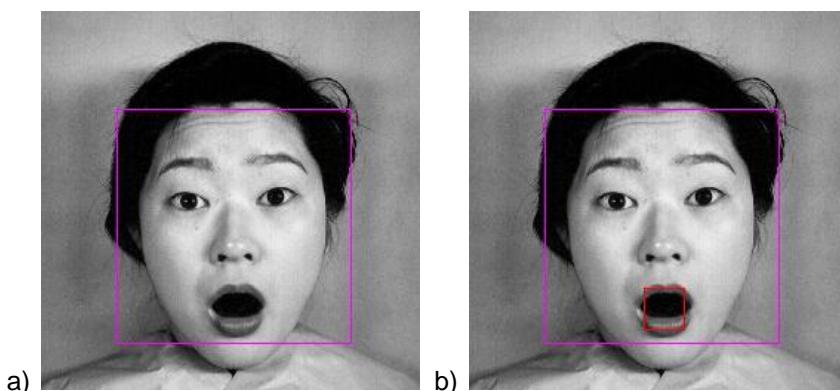
Średnie wymiary twarzy dla wszystkich testowanych usług są do siebie bardzo zbliżone i zostały pokazane na przykładzie Rys. 14. Sytuacja wygląda jednak inaczej w przypadku średnich rozmiarów ust – usługi oparte o klasyfikator zwracają fragment zawierający całe usta (a często nawet większy), podczas gdy 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*) [30], zawierający 213 zdjęć o rozdzielczości 256x256 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 7*.

**Tabela 7. Wyniki testów zbioru JAFFE**

Typ zdjęcia	Średni czas	Liczba twarzy	Liczba uśmiechów (ust)	Liczba zdjęć z więcej niż jedną twarzą (pomyłki)	Średnie wymiary twarzy	Średnie wymiary ust
<b>OpenIMAJ</b>						
<b>Pozytywne</b>	119 ms	<b>31</b>	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	<b>31</b>	6 (25)	0	160x160 px	53x30 px
<b>Negatywne</b>	78 ms	<b>182</b>	1 (84)	0	157x157 px	31x18 px
<b>Luxand (FaceSDK)</b>						
<b>Pozytywne</b>	168 ms	<b>31</b>	29 (31)	0	161x161 px	31x18 px
<b>Negatywne</b>	160 ms	<b>182</b>	21 (182)	0	160x160 px	27x20 px
<b>OpenCV + FaceSDK</b>						
<b>Pozytywne</b>	67 ms	<b>31</b>	29 (31)	0	160x160 px	31x17 px
<b>Negatywne</b>	67 ms	<b>182</b>	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 jedną błędную detekcję. 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 dla 51% twarzy zostały odnalezione usta.

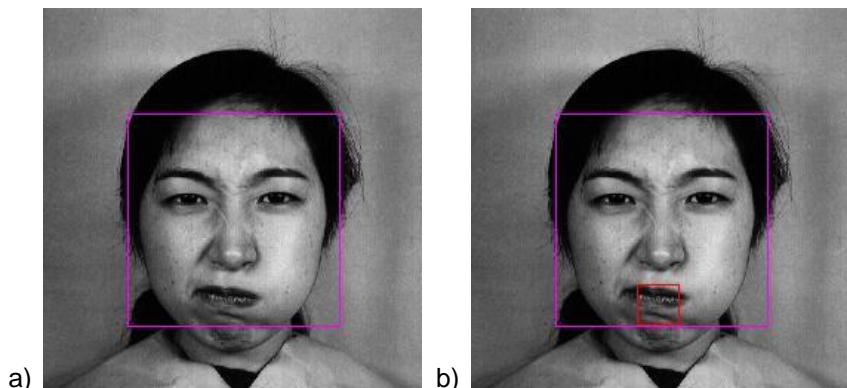


**Rys. 15. Porównanie detekcji w przypadku otwartych ust**

a) OpenCV, b) OpenCV + FaceSDK

Jednym z przypadków, dla których OpenCV nie odnajduje ust, są usta otwarte (bez uśmiechu), co pokazuje Rys. 15. 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. 16.



Rys. 16. 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 (384x384 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.

### 3.2.3. 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 8.

Tabela 8. 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>					
Cohn-Kanade pozytywne	134	227 (0)	1 (205)	182x182	68x41
Cohn-Kanade negatywne	156	1251 (3)	6 (931)	185x185	55x33
JAFFE pozytywne	36	31 (0)	0 (30)	112x112	47x28

<b>JAFFE negatywne</b>	32	182 (0)	0 (141)	112x112	37x22
<b>OpenCV + FaceSDK</b>					
<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

Tabela 8 pokazuje, że zgodnie z oczekiwaniemi 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 rażąca 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. 17.



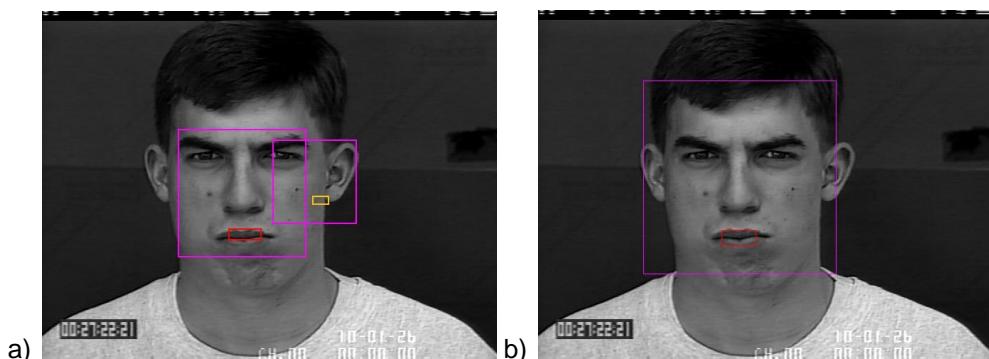
Rys. 17. Różnice w detekcji pomiędzy klasyfikatorami Haara i LBP

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

Na Rys. 17. 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 przetwarzanym 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 błędnych detekcji dla zbioru Cohn-Kanade. Jeden z nich został pokazany na Rys. 18.



Rys. 18. Przykład wystąpienia błędnej detekcji przy użyciu klasyfikatora LBP

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

Wszystkie wystąpienia błędnych detekcji 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.

### 3.3. Wnioski z przeprowadzonych testów

Dla wszystkich zaimplementowanych usług udało się zrealizować detekcję uśmiechu. Zgodnie z założeniami o możliwości przełączania się pomiędzy bibliotekami do detekcji, zdecydowano się udostępnić użytkownikowi wszystkie cztery implementacje.

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 załada się w poprawny sposób.

Zgodnie z przeprowadzonymi testami najlepszą implementacją wydaje się być usługa łącząca możliwości FaceSDK z OpenCV, ze względu na połączenie dokładności działania i nieco lepszej wydajności od konkurencyjnych usług. W związku z taką obserwacją, zostanie ona ustawiona jako domyślna dla każdego użytkownika. 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 przetwarzania 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 błędnych detekcji w aplikacji.

Klasyfikatory LBP wydają się mieć większe zastosowanie w przypadku mniej wydajnych urządzeń niż komputery stacjonarne, na które została napisana aplikacja 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.

#### **3.4. Trenowanie własnego klasyfikatora do detekcji uśmiechu**

W ramach pracy magisterskiej zdecydowano się spróbować wygenerować własny klasyfikator uśmiechu. Jako zbiór próbek pozytywnych ustalono fragmenty zdjęć zawierające jedynie uśmiech o różnym stopniu otwarcia – od postaci zamkniętej aż do postaci szeroko otwartej. Zbiór ten zawierał łącznie 379 obrazów, a jego fragment został pokazany na Rys. 19.



Rys. 19. 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. 20.



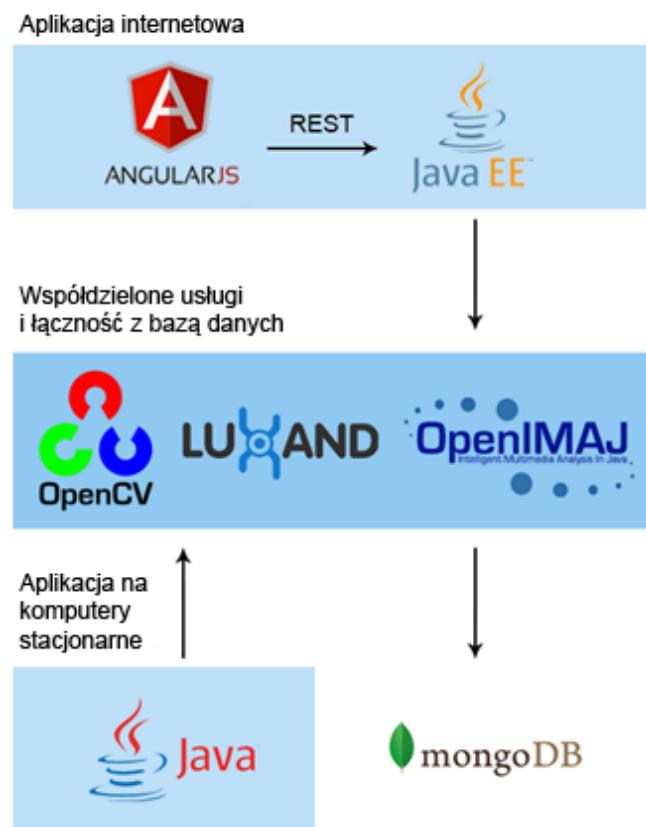
Rys. 20. Fragment zebranego zbioru próbek negatywnych

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 błędą 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.

### 3.5. Architektura systemu

Zgodnie ze zdefiniowanymi założeniami i wynikami testów aplikację zdecydowano się podzielić trzy główne moduły, co przedstawiono na Rys. 21.



Rys. 21. Koncepcja architektury systemu

Najważniejszy z nich, będący rdzeniem całej aplikacji, stanowi logika wykrywania uśmiechów oraz połączenie 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 *nosql (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 dziewięciokrotne przyspieszenie w stosunku do bazy Oracle [2].

Aplikacja na komputery stacjonarne realizowana jest w technologii Java 8, a jej głównym celem jest wyświetlanie obrazu z kamery i zliczanie uśmiechów w czasie rzeczywistym. Elementem wyróżniającym tę wersję aplikacji jest funkcjonalność prezentowania efektów zachęcających użytkownika do uśmiechu – dzięki wbudowanej warstwie usług jest znacznie wydajniejsza niż jej internetowy odpowiednik, co pozwala na rozbudowanie funkcjonalności.

Aplikacja internetowa składa się z dwóch części: frontend oraz backend. Pierwsza z nich została zbudowana w oparciu o framework *Angular 1.6* i pozwala na wykonanie testów bibliotek do wykrywania uśmiechu. Dodatkowo umożliwia ona pobranie aplikacji na komputery stacjonarne w zależności od systemu operacyjnego użytkownika oraz wyświetla zebrane statystyki w postaci przejrzystych wykresów (np. prezentujących wyniki z testów usług). Część backend zrealizowana jest w technologii *Java EE* i udostępnia usługi REST wykorzystywane w części frontend.

## 4. PROJEKT APLIKACJI

### 4.1. 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.

#### 4.1.1. 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 [19] oraz LBP [22], klasyfikatora ust w stanie zwykłym [20] oraz klasyfikatora uśmiechu [21]. 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” czy „skaluj obraz”, 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 relatywne do wymiarów zdjęcia w taki sposób, aby prawidłowo wykrywać wszystkie twarze. Detekcja uśmiechów dla twarzy mniejszych niż  $30\text{ px}$  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 przetwarzania pojedynczego zdjęcia skrócił się o ok.  $50\text{-}100\text{ ms}$ ) – wyklucza ono bowiem wyszukiwanie wszystkich fragmentów mniejszych niż  $30\text{ px}$ .

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  $\frac{1}{8}x\frac{1}{10}$  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 pewności wykrycia uśmiechu, w przypadku wykrycia ust parametr ten ustawiono na wartość  $1$ , natomiast w przypadku wykrycia ust jego wartość wynosi  $0$ .

#### 4.1.2. 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 zbliżone.

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. Jedyną 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  $(30,30)$  dla klasyfikatora twarzy, z tych samych względów, co w przypadku usługi OpenCV.

#### 4.1.3. 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 bez dodatkowych kroków informuje 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, na który wskazuje 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 [28]. 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.

#### 4.1.4. 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 błędne detekcje – 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.

## 4.2. Wykrywanie uśmiechu w czasie rzeczywistym

Algorytmy wykorzystywane w pracy magisterskiej realizują detekcję uśmiechu, ale nie jego rozpoznawanie. Dzięki takiemu podejściu można znaczaco 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.

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 rozwiązań kojarzenia uśmiechów może być losowość algorytmów – niedokładność lub nawet niewykrycie uśmiechu czy twarzy 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 zapamiętywanych 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, ponieważ uśmiech powoduje zmianę szerokości i wysokości ust. Zgodnie z taką obserwacją nie można sprawdzać podobieństwa na podstawie wykrytych ust - lepszym rozwiązaniem wydaje się być porównanie twarzy, ponieważ nie tylko dane zwarcane przez biblioteki są dosyć stabilne, ale także jej wymiary nie ulegają drastycznej zmianie podczas wykonywania uśmiechu. W związku z takim założeniem, logiczny 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 rozmazany obraz, który i tak nie mógłby być zinterpretowany. Wystarczy zatem proste sprawdzanie, czy pozycja i rozmiar twarzy jest podobny. Algorytm sprawdzający podobieństwo wygląda zatem następująco:

$$d_x = \frac{(f_1^{\text{szersz}} + f_2^{\text{szersz}})}{8}, d_y = \frac{(f_1^{\text{wysokosc}} + f_2^{\text{wysokosc}})}{8}$$

$$\text{podobieństwo}_x \Leftrightarrow |f_1^x - f_2^x| < d_x \wedge |f_1^{\text{szersz}} - f_2^{\text{szersz}}| < d_x$$

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

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 (wykryto nową twarz)

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 wahala 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 rozwiązyany 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.

#### **4.3. 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 dla twarzy:

- 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), upłynięcie określonego czasu czy też po prostu zakończenie animacji.

#### 4.3.1. Lista zrealizowanych efektów

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

**Tabela 9.** Informacje o zrealizowanych efektach

Nazwa efektu	Przykład efektu	Opis efektu
<b>Efekty zachęcające do uśmiechu</b>		
Aureola <i>(halo)</i>	 FPS: 7,6	Jeżeli użytkownik się nie uśmiecha, nad jego głową pojawia się pulsująca aureola. Efekt znika po uśmiechnięciu się.
Doklejenie zarostu <i>(beard, mustache)</i>	 FPS: 7,5  FPS: 7,7	Jeżeli użytkownik nie uśmiecha się, w dolnej części jego twarzy pojawia się długa broda lub wąsik. Efekt znika po uśmiechnięciu się.

Nazwa efektu	Przykład efektu	Opis efektu
Serce (heart)		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 (afro, emo)		Jeżeli użytkownik się nie uśmiecha, aplikacja dokleja mu różne rodzaje włosów – afro lub grzywkę w stylu „emo”. Efekt znika natychmiast po uśmiechnięciu się.
Efekty nagradzające uśmiech		
Doklejenie ust (mouths)		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 całusa. Efekt znika natychmiast po przestaniu uśmiechania się.

Nazwa efektu	Przykład efektu	Opis efektu
		
Obrót twarzy (face-rotation)		<p>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.</p>
Siedzący motylek (butterfly)		<p>W trakcie trwania uśmiechu pojawia się w lewej części ust mały niebieski motylek, który macha skrzydełkami. Efekt znika automatycznie po zaprzestaniu uśmiechania się.</p>
Gwiazdki (star)		<p>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</p>

Nazwa efektu	Przykład efektu	Opis efektu
		<p>następna.</p> <p>Efekt trwa, póki użytkownik się uśmiecha.</p>
Rekordzista <i>(champion)</i>		<p>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.</p> <p>Efekt znika po zakończeniu uśmiechania się.</p>

## **5. IMPLEMENTACJA**

W ramach projektu powstały dwie aplikacje – wersja na komputery stacjonarne oraz aplikacja internetowa.

### **5.1. Aplikacja na komputery stacjonarne**

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

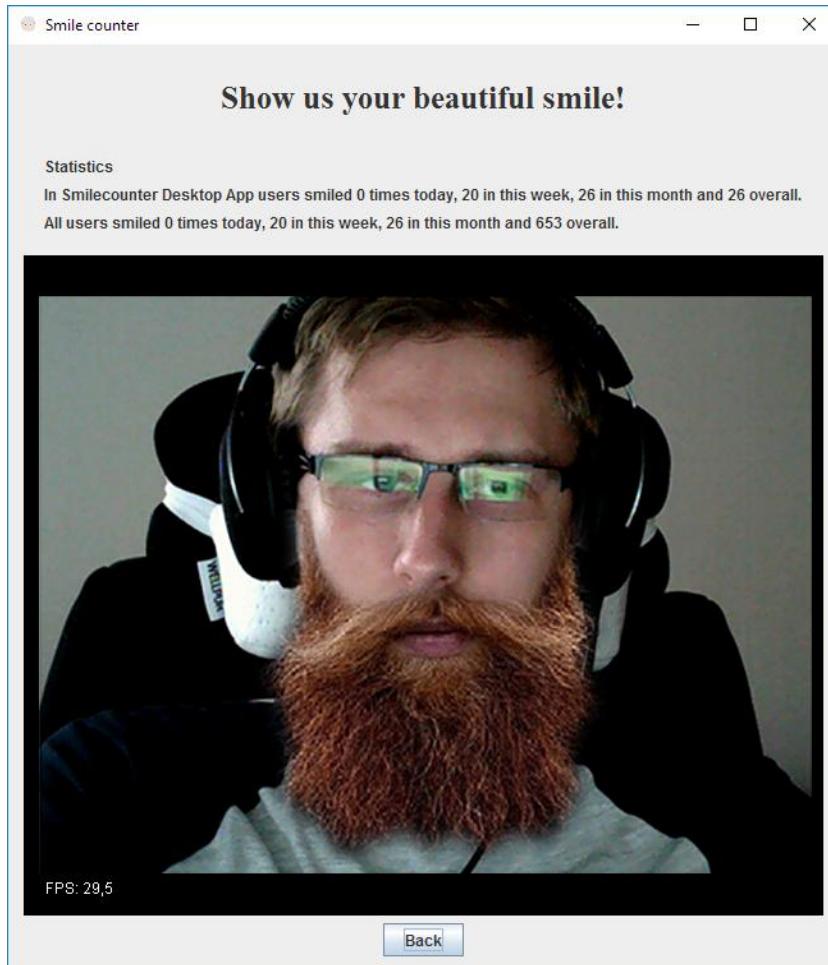
Podczas uruchamiania aplikacji wyświetlany jest krótki ekran ładowania (ang. *splash-screen*), podczas którego aplikacja nawiązuje połaczenie z bazą danych oraz ładuje biblioteki dynamiczne. Po zakończeniu tego procesu prezentowane jest proste menu główne, z którego użytkownik może przejść do menu ustawień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

Aplikacja działa w dwóch trybach: offline oraz online. W przypadku pierwszego, w pliku %USERDIR%/.smilecounter/database.dat przechowywana jest liczba uśmiechów wykryta danego dnia, co pozwala na prezentowanie odpowiednich statystyk. Wersja online łączy się z bazą danych i informacje o uśmiechach są do niej zapisywane.

#### **5.1.1. Ekran podglądu obrazu z kamery**

Ekran podglądu obrazu z kamery jest najważniejszym ekranem aplikacji na komputery stacjonarne. Wyłącznie podczas pracy na tym ekranie wykorzystywane są usługi wykrywające uśmiech na twarzy użytkownika. W zależności od wprowadzonych ustawień, na obrazie z kamery rysowane są dodatkowo informacje o wykrytych fragmentach twarzy oraz efekty nagradzające i zachęcające do uśmiechu. Formatka składa się z dwóch głównych części, co pokazano na Rys. 22.

Wykryte fragmenty twarzy informują użytkownika 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 biały. Twarze z wykrytym uśmiechem mają różowy kolor ramki, natomiast w przypadku zakończenia uśmiechu wykorzystano kolor błękitny.



Rys. 22. Ekran podglądu obrazu z kamery w aplikacji na komputery stacjonarne

Pierwszą częścią ekranu jest licznik uśmiechów dla aktualnie trwającej sesji – resetuje się on za każdym razem po wejściu na ekran. Jeżeli użytkownik jeszcze się nie uśmiechnął, prezentowany jest napis zachęcający do uśmiechu. Licznik odświeża się od razu po uśmiechnięciu, dzięki czemu można łatwo zweryfikować, czy uśmiech został prawidłowo policzony.

Pod licznikiem sesyjnym umieszczone zostały statystyki wykrytych uśmiechów. Pokazana jest tutaj ich liczba z dzisiaj, zeszłego tygodnia, miesiąca oraz od początku działania aplikacji. Statystyki dotyczą nie tylko wszystkich użytkowników, ale także dla lokalizacji, w jakiej została ustalona aktualnie działająca aplikacja Smilecounter. Liczniki w statystykach są regularnie odświeżane (domyślnie co 10 sekund).

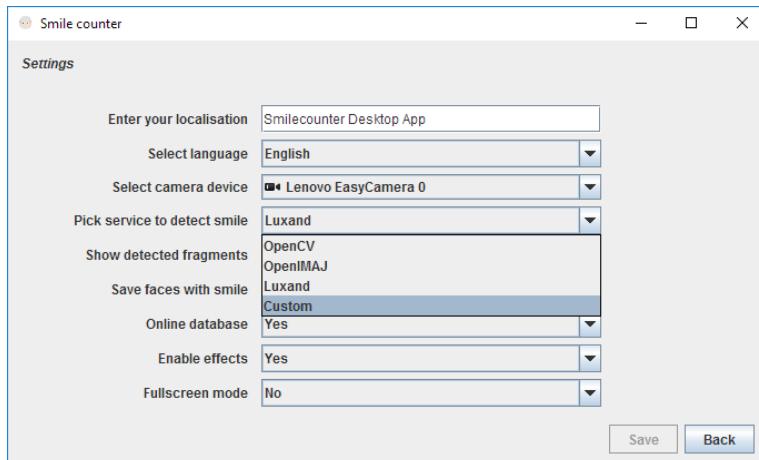
Pozostałą część stanowi obraz z kamery użytkownika. To tutaj w czasie rzeczywistym prezentowane są wykryte fragmenty twarzy oraz efekty nagradzające i zachęcające do uśmiechu. Obraz z kamery rozciąga się do okna aplikacji, dzięki czemu w trybie pełnoekranowym jest on dużo bardziej szczegółowy.

#### 5.1.2. Ekran ustawień

Ekran ustawień aplikacji na komputery stacjonarne zawiera najczęściej zmieniające się ustawienia z punktu widzenia użytkownika. Ich zmiana nadpisuje konfigurację zawartą w pliku

*smilecounter.config* w trakcie aktualnie trwającej sesji (nie zapisują się po zamknięciu aplikacji).

Listę zaimplementowanych opcji pokazuje Rys. 23.



Rys. 23. Ekran ustawień w aplikacji na komputery stacjonarne

Najważniejszym polem do edycji jest combobox z możliwością zmiany podłączonej do komputera kamery oraz usługi, która ma być wykorzystywana podczas detekcji uśmiechów. Z poziomu tego ekranu można także zdecydować, czy aplikacja ma pokazywać wykryte fragmenty twarzy i efekty zachęcające do uśmiechu, czy ma podłączyć się do bazy danych MongoDB oraz czy ma w niej zapisywać twarze w momencie uśmiechu. Dodatkowymi udostępnionymi ustawieniami jest nazwa lokalizacji, możliwość zmiany języka oraz przełączanie się pomiędzy trybem pełnoekranowym i okienkowym.

#### 5.1.3. 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 10.

Tabela 10. Lista parametrów konfiguracyjnych dla aplikacji na komputery stacjonarne

Parametr	Opis	Wartość domyślna
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

Parametr	Opis	Wartość domyślna
database.storeFaces	O 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 (liczniuki uśmiechów na ekranie z kamerą).	10
affective.default.service	Domyślana 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 nowy uśmiech. Wykorzystywany jedynie w przypadku usług opierających się o FaceSDK.	0.45
affective.showDetectedFragments	O 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

Parametr	Opis	Wartość domyślna
settings.locationName	Okręsła domyślną wartość nazwy lokalizacji, wykorzystywanej przy zapisywaniu uśmiechu.	Smilecounter Desktop App
settings.fullscreenMode	Okręsła, czy aplikacja powinna pracować w trybie pełnoekranowym.	False
effects.effectsList	Pozwala na określenie listy wyświetlanych efektów. Efekty powinny być rozdzielone przecinkiem, a ich nazwy prezentuje Tabela 9.	butterfly,beard,heart,afro, mustache,emo,star, mouths,halo,face- rotation,champion
effects.enabled	Okręsła, czy aplikacja powinna prezentować efekty nagradzające i zachęcające do uśmiechu.	true

Uruchomienie aplikacji dla usług innych niż OpenIMAJ wymaga dostarczenia bibliotek dynamicznych - domyślnie ładowane są one z katalogu *libs* w katalogu z aplikacją. Plik konfiguracyjny *smilecounter.config* jest opcjonalny (jego parametry zostały opisane wcześniej) – jeżeli plik nie istnieje, zostaną użyte ustawienia domyślne.

Aby uruchomić aplikację, wystarczy wystartować *SmileCounter.exe*. W celu diagnozy ewentualnych błędów (zapisania logów aplikacji) udostępniono także aplikację w formacie *.jar*, którą można uruchomić przy pomocy skryptu *run\_jar\_smilecounter.bat*. Różni się ona od podstawowej wersji *.exe* tym, że w trakcie działania aplikacji zapisuje dodatkowo logi w pliku *logi.txt*.

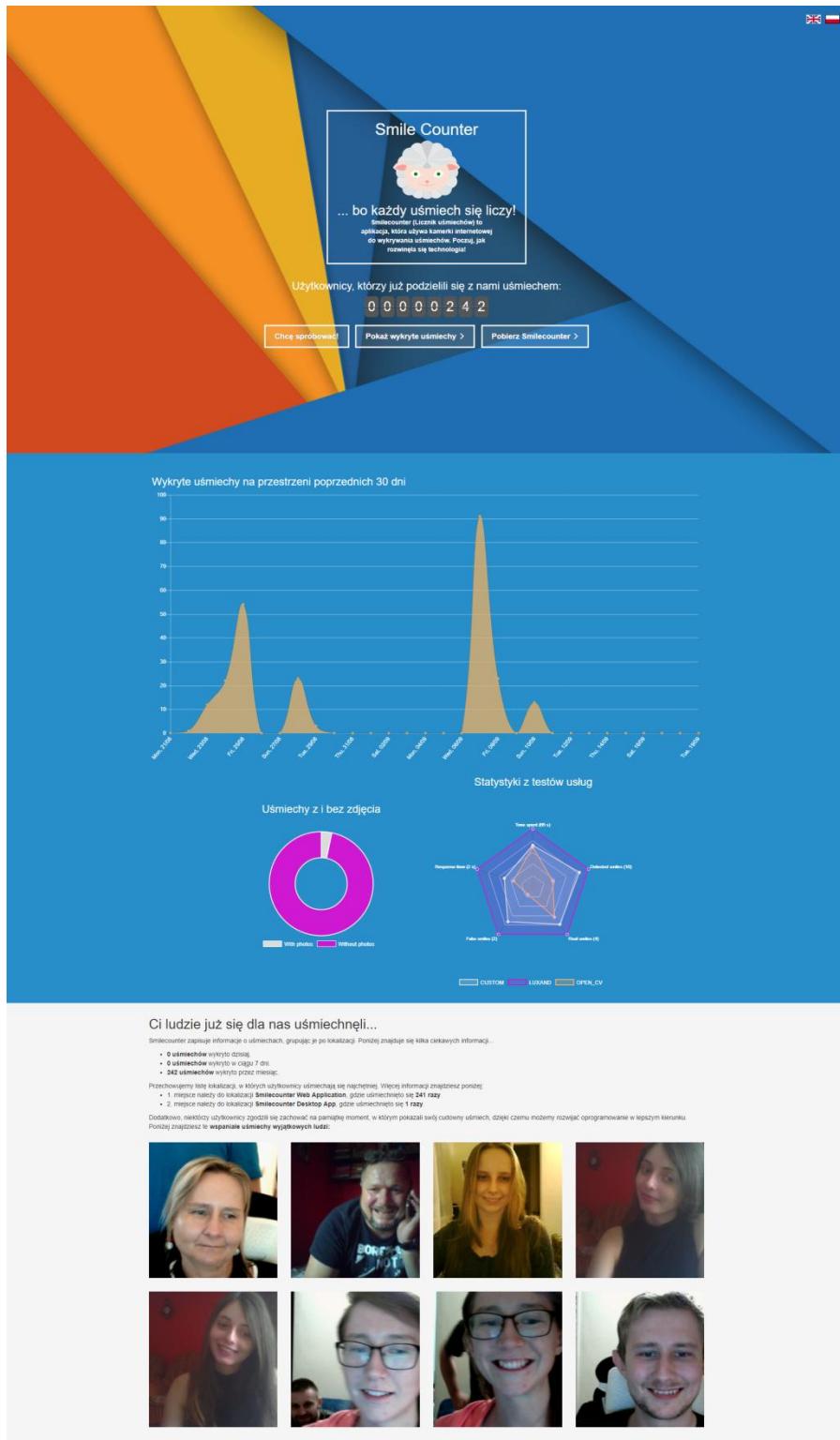
## 5.2. Aplikacja internetowa

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

### 5.2.1. Strona główna

Zadaniem głównego ekranu aplikacji internetowej 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 pasku nawigacyjnym, 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. 24. Ekran główny aplikacji internetowej

Jak widać na Rys. 24, najważniejszym elementem strony głównej jest informacja o tym, że Smilecounter służy do zliczania uśmiechów użytkowników. Na tym etapie użytkownik może rozpoczęć testowanie usług do wykrywania uśmiechu, pobrać aplikację na komputer stacjonarny 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.

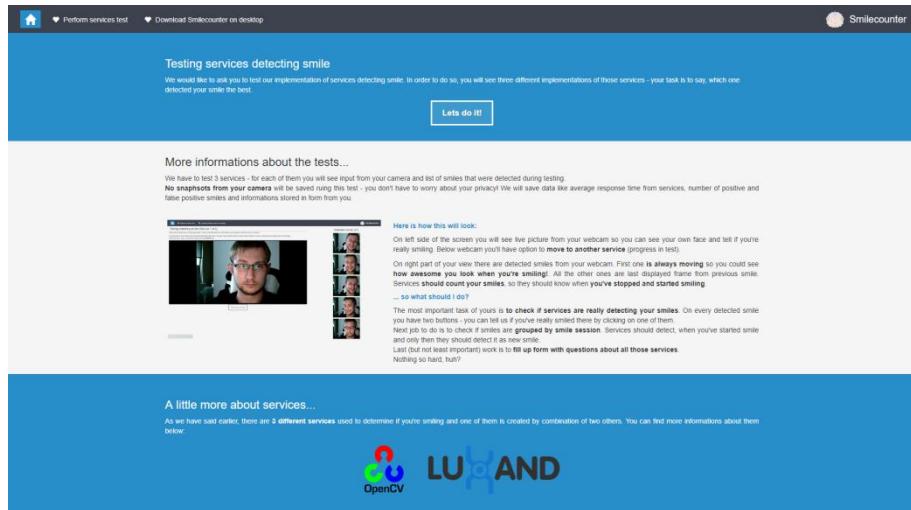
Kolejnym fragmentem strony głównej jest prezentowanie statystyk zebranych podczas zliczania uśmiechów. Znajdują się tu trzy wykresy: pierwszy, liniowy, pokazuje rozkład liczby wykrytych uśmiechów dla każdego dnia ostatniego miesiąca. Kolejny wykres o typie *donut* pokazuje stosunek liczby użytkowników, którzy wyrazili zgodę na zapisanie zdjęcia do liczby wykrytych uśmiechów ogółem. Ostatni, a zarazem najciekawszy wykres o typie *radar* pokazuje wyniki przeprowadzonych testów w aplikacji internetowej. 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.

#### *5.2.2. Ekran testowania usług wykrywających uśmiech w aplikacji internetowej*

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.

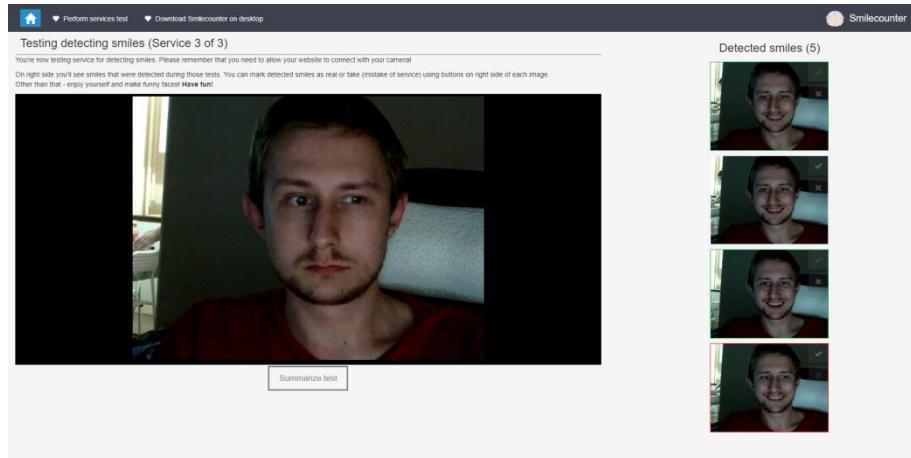
Proces testów składa się z pięciu głównych kroków: wprowadzenia do testów, testowania, ekranu z formularzem, opcjonalnego ekranu z możliwością zapisania wskazanych zdjęć do bazy danych oraz ekranu z wynikami testów.



Rys. 25. Ekran wstępny testowania usług wykrywających uśmiech

Rys. 25. 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 wyświetlane 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. 26. Główny ekran testowania usług wykrywających uśmiech

Następny, najważniejszy etap procesu testowania usług wykrywających uśmiech został pokazany na Rys. 26. W lewej części ekranu pojawia się aktualny obraz z kamerki 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 najechaniu 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 serwerem. Za każdym razem, kiedy limit ten jest osiągany, opóźnienie w przesyłaniu klatek zwiększa się o 50 ms, zmniejszany 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.

Kolejnym etapem testowania usług jest ekran z formularzem, w którym żadne pole nie jest obowiązkowe. Aplikacja podczas testów zbiera przede wszystkim dane techniczne z odpowiedzi otrzymanych z usług do wykrywania uśmiechów, a formularz traktowany był jako dodatkowa możliwość na zebranie informacji na temat poprawności programu.

Ekran ten został pokazany na Rys. 27.

**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

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/hear about it until now  
 No, I don't think information about user's smiles are useful for applications

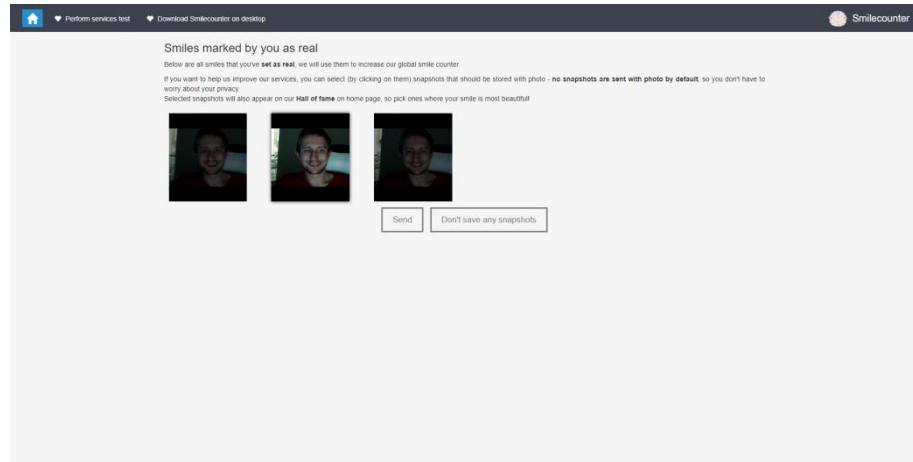
**Rys. 27. Ekran z formularzem dotyczącym działania usług**

Osoby chętne pomóc w testach usług mogły odpowiedzieć na szereg pytań dotyczących każdej z usługi (pytania są powielone dla każdej implementacji, 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 pytaniem 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

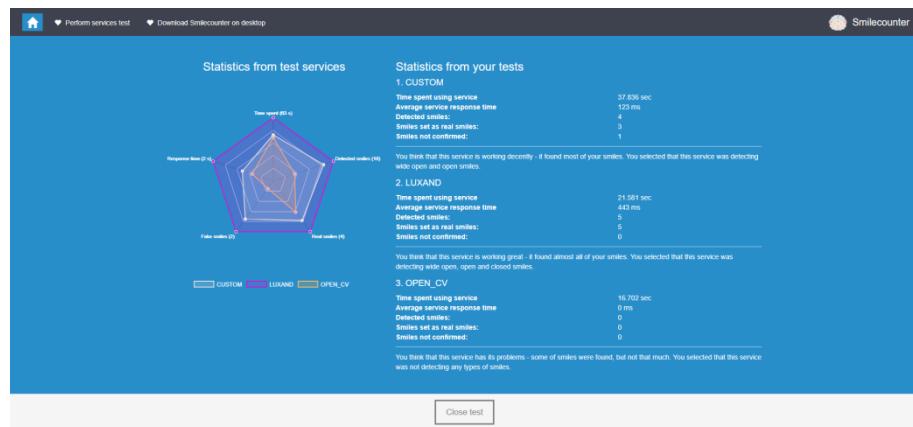
aplikacjach, pojawia się dodatkowe pytanie, w którym użytkownik proszony jest o wskazanie, w jakich aplikacjach chciałby, by takie usługi się znalazły.

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. 28.



Rys. 28. Ekran zapisywania uśmiechów wykrytych podczas testu usług

Ostatnim etapem testów jest ekran prezentujący podsumowanie wyników procesu. Został on pokazany na Rys. 29.

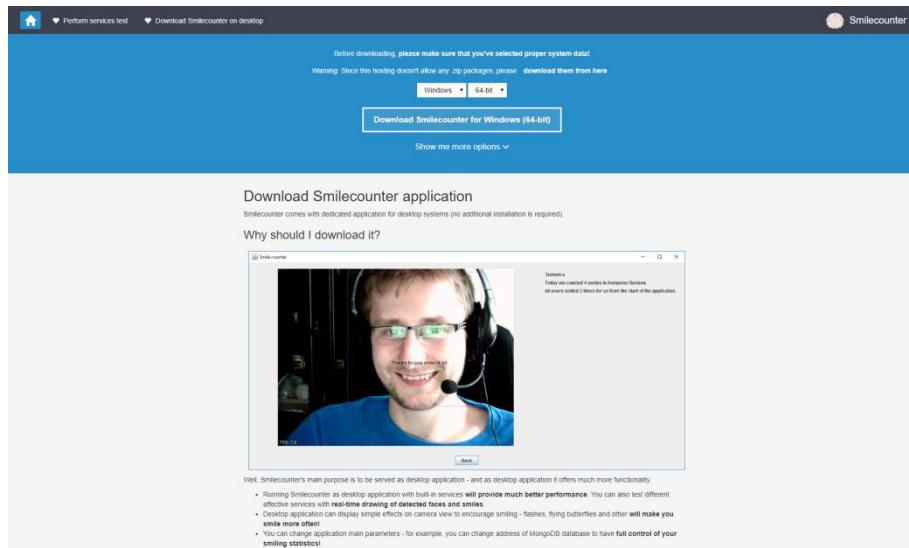


Rys. 29. Ekran prezentujący wyniki testów

W lewej części ekranu prezentowany jest wykres typu *radar* analogiczny do wyświetlonego na stronie głównej – zawiera on statystyki ze wszystkich przeprowadzonych do tej pory testów. Po prawej stronie pokazane zostały informacje, jakie udało się uzyskać z aktualnie przeprowadzonych testów: 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. Dodatkowo prezentowane są odpowiedzi, jakich udzielił użytkownik w pytaniach formularza na temat każdej z usług.

### 5.2.3. Ekran pobierania aplikacji na komputery stacjonarne

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



Rys. 30. Ekran pobierania aplikacji na komputery stacjonarne

Na Rys. 30. 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 na komputery stacjonarne od internetowej oraz co użytkownik może zyskać pobierając tę wersję programu na swój komputer.

### 5.2.4. Konfiguracja i sposób uruchomienia aplikacji

Konfiguracja aplikacji internetowej została podzielona na dwie części, ze względu na podział modularny backend oraz frontend. W przypadku serwera konfiguracja jest bardzo podobna do aplikacji na komputery stacjonarne – 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 10.

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 11*.

**Tabela 11. 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 Angulariem).	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 odpowiednią nazwą w katalogu <code>files</code> .	["Windows"]
refreshTime	Domyślny czas odświeżania danych na stronie głównej.	10
servicesToTest	Lista usług (przedstawiona w formie tablicy), które będą testowane. Kolejność testowania	["LUXAND", "OPEN_CV", "CUSTOM"]

Parametr	Opis	Wartość domyślna
	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 trzymanych w pamięci (przydatna do grupowania uśmiechów).	1
maxFramesRequest	Maksymalna liczba żądań na formatce testowania usług wysyłanych jednocześnie. Pozwala na zmniejszenie obciążenia komputera i serwera.	10

#### 5.2.5. Uruchamianie aplikacji internetowej przy pomocy Dockera

Ponieważ prawidłowa konfiguracja środowiska z uwzględnieniem instalacji potrzebnych bibliotek jest czasochłonna (głównie przez instalację OpenCV), aplikacja internetowa została w pełni skonfigurowana przy pomocy Dockera. Środowisko zostało podzielone na dwa kontenery: *smilecounter-db*, zawierający bazę danych MongoDB oraz *smilecounter-backend*, w którym znajduje się serwer *WildFly 10*. Oba zarządzane są przy pomocy *docker-compose* i można je uruchomić poprzez polecenie *docker-compose up -d*.

Kontener z bazą danych podczas startu inicjalizowany jest zrzutem bazy danych z momentu zakończenia testów – baza MongoDB jest wystawiona na porcie 27015.

Kontener z serwerem składa się z dwóch poziomów obrazów. W obrazie bazowym, oprócz zainstalowanych bibliotek OpenCV, zawarta jest także niezbędna konfiguracja HTTPS dla WildFly - nowoczesne aplikacje nie pozwalają na odczytywanie obrazu z kamery aplikacjom wystawionym poprzez protokół HTTP. Obraz bazowy został także zainstalowany na DockerHubie pod nazwą *theudeem/wildfly-smilecounter*, dzięki czemu nie trzeba go lokalnie tworzyć. Główny obraz kontenera z serwerem zawiera w sobie instalację artefaktów aplikacji internetowej – taki podział hierarchii obrazów pozwala na szybkie i łatwe odtworzenie środowiska dla nowych wersji. Aplikacje są dostępne pod portem 8443. W pliku *docker-compose.yml* poprzez zmienne środowiskowe można wprowadzić klucz dla biblioteki FaceSDK (zmienna *LUXAND\_KEY*) oraz ew. podać niestandardowy adres dla połączeń z bazą MongoDB

(zmienna *DATABASE*, domyślnie ustawiona na adres kontenera z bazą danych, *mongodb://smilecounter-db:27017/smilecounter*).

### **5.3. Kod źródłowy aplikacji**

Repozytorium aplikacji składa się z ośmiu modułów:

- a) *Create/Desktop\_App* – prosta aplikacja budowana przy pomocy *Mavena*, która pozwala na utworzenie plików *.exe* oraz *.jar* do uruchamiania aplikacji (zależy od modułów *Services* oraz *Desktop*)
- b) *Desktop* – kod źródłowy aplikacji na komputery stacjonarne. Znajdują się tutaj klasy wyświetlające ekrany przy pomocy *Java Swing*, usługi zarządzające wyświetlaniem efektów oraz wykrywające ciągłość uśmiechu. Wykrywanie uśmiechów odbywa się poprzez usługi z modułu *Services*.
- c) *Docker* – moduł pozwalający na uruchamianie środowiska dla aplikacji internetowej przy pomocy *Dockera*
- d) *Documentation* – moduł zawierający wszystkie informacje wykorzystywane w procesie tworzenia dokumentacji
- e) *Haar-trainer* – moduł odpowiadający za trenowanie własnego klasyfikatora Haara
- f) *Services* – główny moduł aplikacji. Zawiera usługi do detekcji twarzy i uśmiechów oraz łączności z bazą danych (zarówno online, jak i offline). Znajduje się tutaj też zbiór współdzielonych funkcjonalności pomiędzy pozostałymi modułami.
- g) *Test* – moduł wykonywający testy na zadanych zbiorach danych dla wszystkich zaimplementowanych usług.
- h) *Web App* – moduł zawierający aplikację internetową, podzieloną na podmoduły *frontend* oraz *backend*.

Całość projektu została napisana w IDE *IntelliJ IDEA 2017*. W przypadku uruchomienia projektu z katalogu głównego kodu źródłowego zostaną zaczytane wszystkie konfiguracje uruchamiania funkcjonalności każdego z modułów (np. wystartowanie *Dockera*, wykonanie testów czy też uruchomienie aplikacji na komputery stacjonarne).

## **6. EWALUACJA GOTOWEJ APLIKACJI**

Ewaluacja aplikacji została podzielona na trzy etapy. Pierwszym z nich były testy z dorosłymi ludźmi przy pomocy aplikacji internetowej. Po ich zakończeniu przeprowadzono testy ze zdowymi dziećmi. Ostatnim etapem były testy z dziećmi chorymi na autyzm.

### ***6.1. Analiza statystyk oraz wyników ankiety z aplikacji internetowej***

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

Główym celem wdrożenia aplikacji internetowej dla użytkowników było pozyskanie w łatwy sposób osób chętnych do przetestowania usług wykrywających uśmiech. Dzięki braku potrzeby jakiejkolwiek 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 internetową 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).

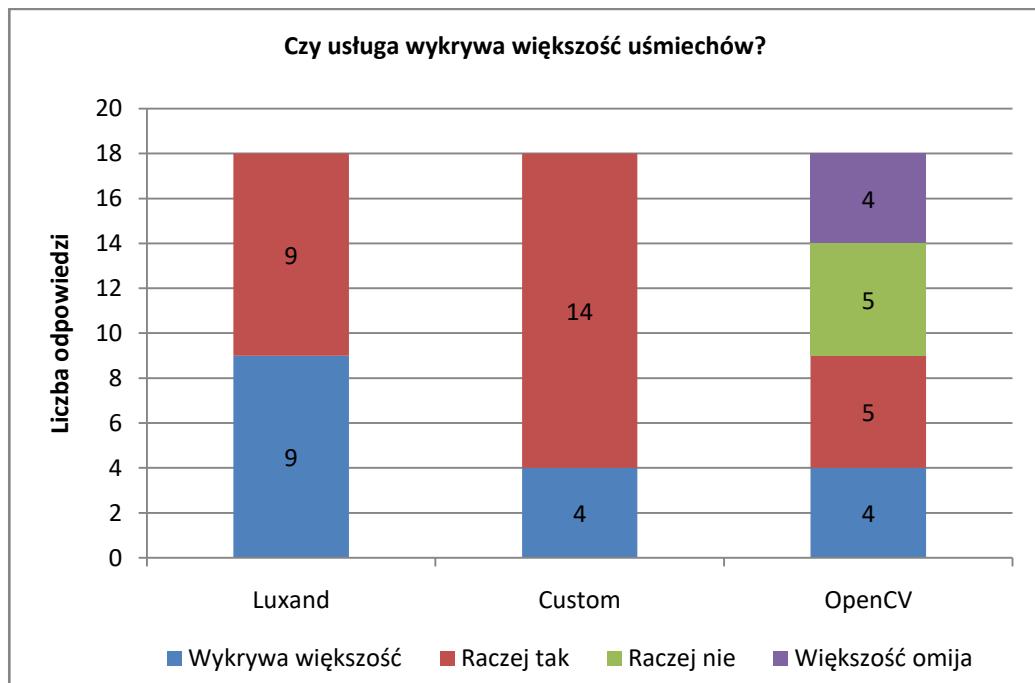
Podczas weryfikacji statystyk i opinii warto pamiętać o tym, że użytkownicy korzystali ze sprzętu różnego rodzaju – często zdarzało się, że prędkość Internetu lub wydajność komputera nie pozwalały na wysyłanie kilku klatek na sekundę. W takim wypadku prezentowany był odpowiedni alert, że praca aplikacji może nie być stabilna. Dodatkowym utrudnieniem był fakt, że każdą przetwarzaną 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 internetowej – w jej przypadku przesyłano jedną klatkę na ok. 1.5 s, co przeszkadzało w sprawnym wykrywaniu ciągłości uśmiechu.

Podczas testów aplikacji internetowej udało się przeprowadzić 32 testy, w których 18 użytkowników postanowiło dodatkowo wypełnić krótki formularz zawierający

pytania na temat działania wszystkich trzech zaimplementowanych usług oraz kilka pytań ogólnych.

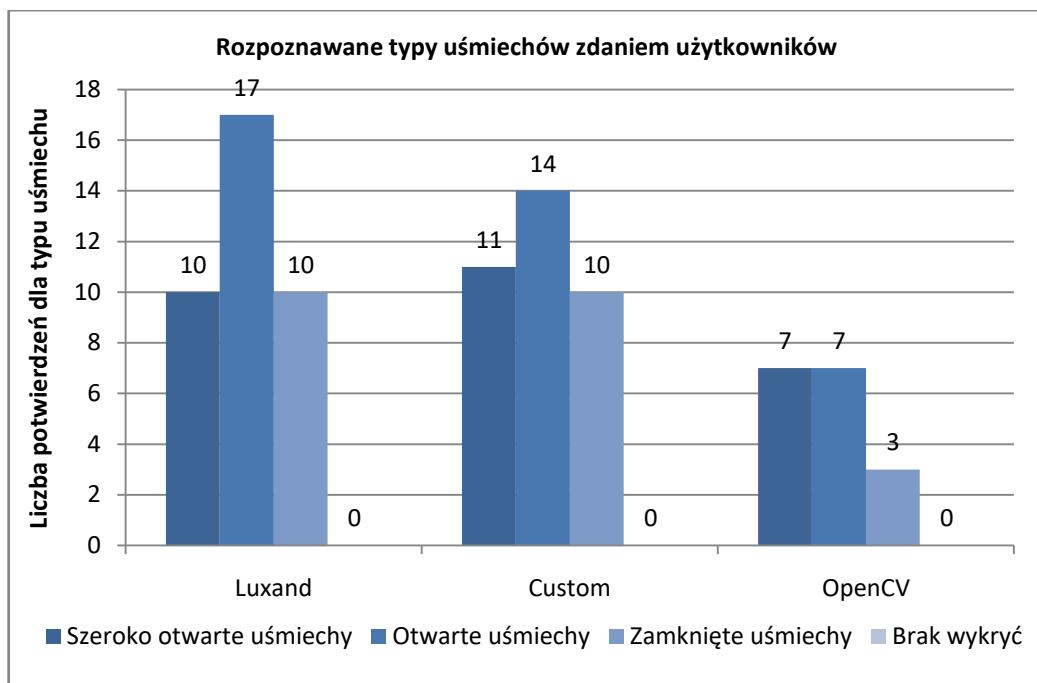
Jedną z najważniejszych informacji uzyskanych od użytkowników podczas tych testów była odpowiedź na pytanie, czy usługa wykrywa większość uśmiechów – odpowiedzi te pokazuje Rys. 31.



Rys. 31. 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.

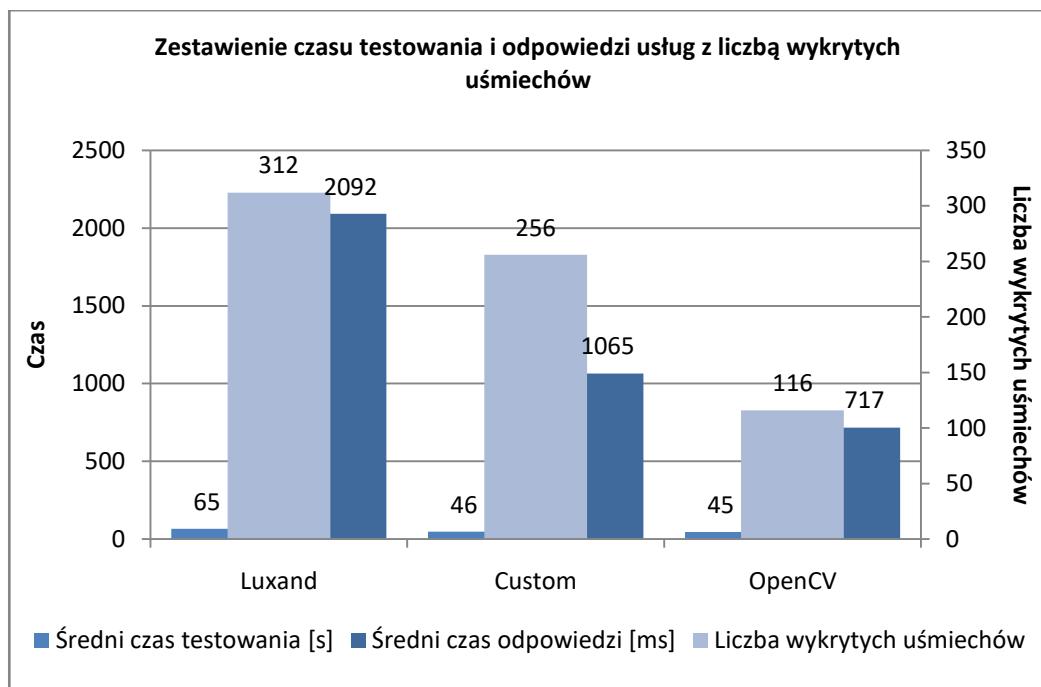
W odpowiedzi na pytanie, dlaczego OpenCV otrzymało tak niewiele głosów pozytywnych, może okazać się zestawienie wyników, jak użytkownicy oceniali typy wykrywanych uśmiechów przez każdą z usług. Pytanie zostało zaprezentowane w formie pytania wielokrotnnej odpowiedzi – użytkownicy mogli ocenić, czy usługa wykrywała szeroko otwarte uśmiechy, zwykłe otwarte uśmiechy oraz zamknięte uśmiechy. Dodatkową opcją było zaznaczenie, że usługa nie wykrywała żadnych uśmiechów. Takie dane zostały pokazane na Rys. 32.



Rys. 32. Przedstawienie typów wykrywanych uśmiechów zdaniem użytkowników

Wykres zaprezentowany na Rys. 32. 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 zestawić ze sobą czas spędzony na testowaniu usługi z liczbą uśmiechów przez nią wykrytych. Informacje te zostały pokazane na Rys. 33.

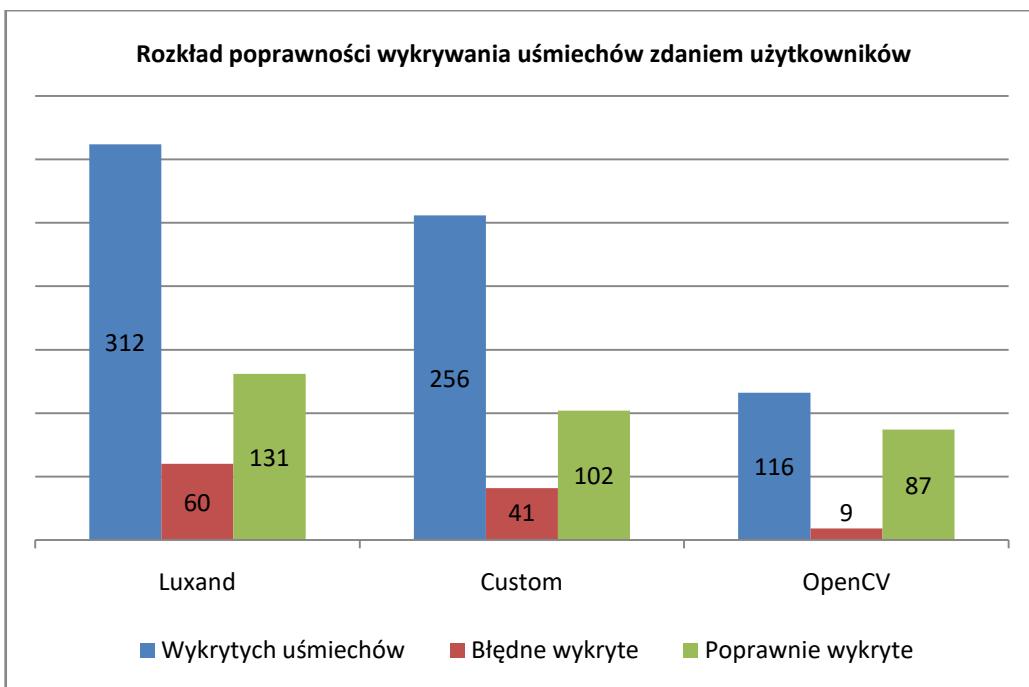


Rys. 33. Wykres prezentujący zależność wykrytych uśmiechów do czasu testowania

Na przedstawionym wykresie można zauważać, 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 zaważył 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 przetwarzania 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 usługę.

Sama liczba wykrytych uśmiechów nie informuje o tym, czy usługa wykrywała je w sposób prawidłowy – zawsze istnieje ryzyko powstawania błędnych detekcji. Aby sprawdzić częstotliwość ich występowania dla wszystkich usług, użytkownicy mieli możliwość oznaczenia uśmiechu jako poprawne i błędnie wykrytego. Zależność pomiędzy liczbą wszystkich wykrytych uśmiechów a opinią użytkowników na temat poprawności działania mechanizmu detekcji przedstawia Rys. 34.



**Rys. 34. Zestawienie liczby wykrytych uśmiechów do skuteczności ich detekcji**

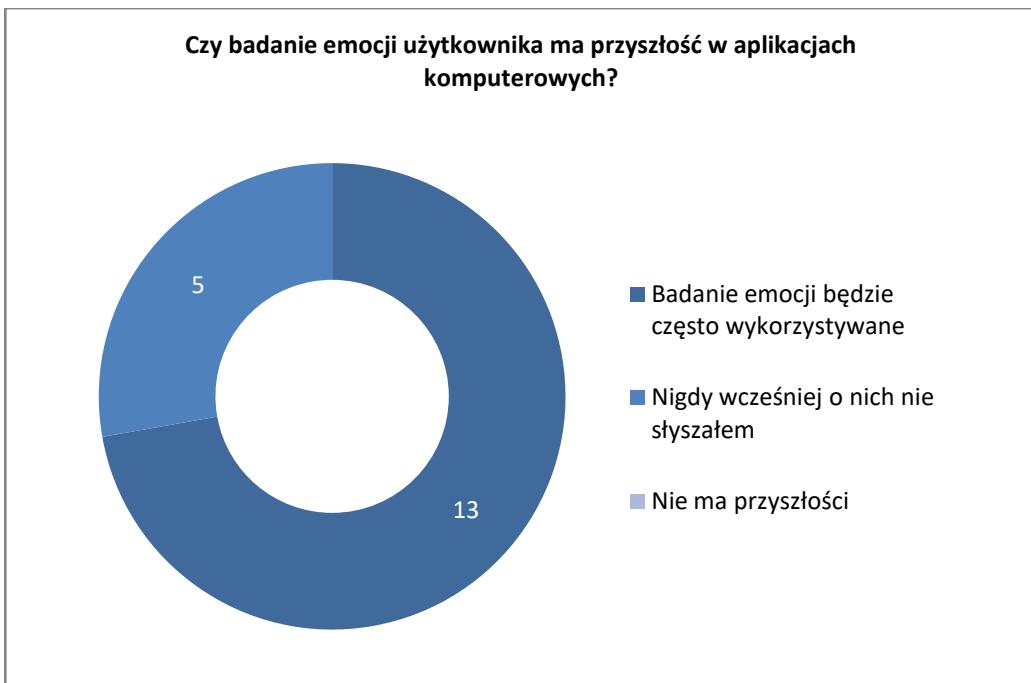
Tylko połowa użytkowników skorzystała z mechanizmów pozwalających na ocenę poprawności detekcji uśmiechów – jedynie testy usługi OpenCV zostały w 75% potwierdzone. Pośród uzyskanych wyników można zauważyć, że choć usługa ta wykrywała uśmiechy rzadziej od swoich rywali, w większości przypadków były to 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 12*. Widać tutaj, że najczęściej formularz wypełniały kobiety. Większość osób testujących usługę mieściła się w przedziale wiekowym od 20 do 25 lat.

**Tabela 12. 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 przetwarzających 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 przetwarzanie emocji nie ma żadnej przyszłości w dziedzinie aplikacji komputerowych – wyniki przedstawia *Rys. 35*.



**Rys. 35. 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 przetwarzania emocji nie mają żadnej przyszłości. Wydaje się to potwierdzać istniejący trend we wdrażaniu usług w różnych aplikacjach.

#### 6.1.1. Wnioski z przeprowadzonych testów

Testy przeprowadzone przy pomocy aplikacji internetowej potwierdzają poprawną implementację detekcji uśmiechu. Użytkownicy szczególnie zadowoleni byli z usług, które do wykrywania uśmiechu wykorzystywały bibliotekę firmy Luxand. Nawet dla architektury *klient-serwer*, z dużo większymi opóźnieniami, zdaniem użytkowników w głównej realizowały one postawione im zadania zliczania uśmiechów. Zastosowanie jedynie usługi OpenCV wydaje się

być jednak mniej pożądane – działanie klasyfikatora uśmiechu pozostawia wiele do życzenia, co odzwierciedliły opinie użytkowników.

## **6.2. Testy z dziećmi typowo rozwijającymi się**

Kolejnym etapem walidacji było przeprowadzenie testów aplikacji na komputery stacjonarne w dwuosobowej grupie dzieci. Pierwszym z nich był 6-letni, bardzo ekspresyjny chłopiec, drugą osobą była natomiast nieco starsza (9 lat) i raczej nieśmiała dziewczynka. Dla obojga dzieci zdecydowano się nagrywać twarz przy pomocy kamery oraz, wykorzystując program CamStudio, wykonać nagranie ekranu (ang. screencast) z momentu korzystania z aplikacji. Dzięki temu można było prześledzić wykrywanie uśmiechów i prezentowanie efektów, porównując je z emocjami widocznymi na twarzy dzieci. Detekcję fragmentów twarzy wykonywano jedynie przy pomocy usługi łączącej możliwości bibliotek OpenCV oraz FaceSDK.

Badanie każdego z dzieci trwało ok. 10 minut. Po przełączeniu się na polską wersję językową aplikacja nie zmieniła swojej nazwy, przez co dzieci (nie znając języka angielskiego) nie wiedziały, do czego ona służy. Następnie został im zaprezentowany ekran z podglądem obrazu z kamery, a na nim wykryte fragmenty twarzy oraz efekty zachęcające do uśmiechu. Przez pierwszy moment dzieci nie rozumiały, co należy zrobić - ponieważ zabrakło informacji zachęcających do uśmiechu, trzeba było w pierwszej kolejności wytlumaczyć uczestnikom zabawy, na czym polega aplikacja i że powinna ona wykrywać ich uśmiech.

Podczas testów w wykonaniu dziewczynki można było zauważyc, że nie była ona przekonana do aplikacji. Jej naturalny sposób uśmiechania się to uśmiechy zamknięte, podczas gdy aplikacja wykrywa tego rodzaj uśmiechów w raczej przecienny sposób – zgodnie ze wcześniejszymi informacjami, trudno ustalić jednolity próg uśmiechu zamkniętego dla różnych osób. Ponieważ większość jej uśmiechów nie została wykryta, sprawiało to dla niej wrażenie, że program nie działa poprawnie. Dodatkowym utrudnieniem był fakt, że kamera widocznie ją stresowała, przez co dziewczynka raczej nieśmiało się uśmiechała. Na koniec testów przyznała także, że liczba prezentowanych efektów była zbyt mała i zbyt często się powtarzały. Niestety, dziewczynka stwierdziła, że aplikacja jej się nie spodobała.

W przypadku chłopca sprawa wyglądała zupełnie inaczej. Ponieważ jest on z natury ekspresyjny, dużo łatwiej szło mu uśmiechanie się. Z początku było widać, że rozpraszało go odbicie lustrzane z kamery – przez dłuższą chwilę chłopiec dziwił się, że przechylając się w lewo, widzi siebie po prawej stronie ekranu. Dodatkowym utrudnieniem był fakt, że aplikacja zajmowała niewielki fragment monitora, a w tle znajdował się program do nagrywania, przez co z początku chłopiec zwracał na niego co chwilę uwagę.

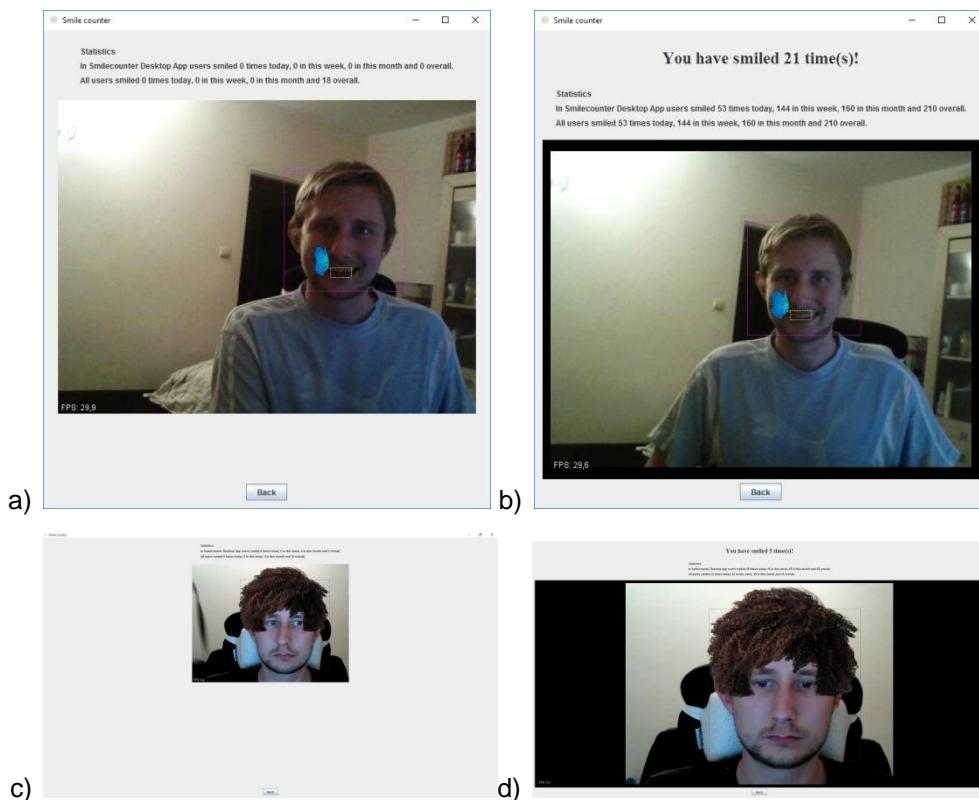
W trakcie trwania testu było widoczne, że chłopca bardzo bawiły prezentowane efekty. Próbował wchodzić z nimi w interakcję, zbliżał się i oddalał od kamery, aby powiększyć niektóre z nich. Chłopiec bardzo energicznie się poruszał, przez co algorytm badający podobieństwo twarzy na podstawie ich położenia pomiędzy sąsiednimi klatkami okazjonalnie wykrywał jego twarz jako nową. Wydaje się to jednak zrozumiałe, ponieważ w trakcie szybkich ruchów jego postać była bardzo rozmacona, co uniemożliwiałoby prawidłowe rozpoznanie twarzy. Działo się tak,

ponieważ chłopiec bawił się, uciekając od efektów oraz próbując je „zjeść”. W trakcie tych testów dało się także zauważać, że wykrywanie twarzy słabiej działa w przypadku średnio i mocno przechylonej głowy.

Chłopiec sprawiał wrażenie, że praca z aplikacją oznaczała dla niego dobrą zabawę - obrał sobie za cel osiągnięcie 100 uśmiechów, a udało się mu uśmiechnąć aż 104 razy. Zapoznanie się z liczbą wykrytych uśmiechów sprawiało mu jednak problem – podczas prób odczytania statystyk musiał pochylić się i dobrze się przyjrzeć, aby podać prawidłową wartość. Stwierdził jednak, że aplikacja i prezentowanie wszystkich efektów bardzo mu się spodobały – w pewnym momencie nawet nie chciał zakończyć testów.

Po przeanalizowaniu wyników testów zdecydowano się wprowadzić kilka zmian w aplikacji - zostały one pokazane na Rys. 36, na którym zaprezentowano porównanie wersji aplikacji przed testami z wersją aplikacji po poprawkach.

Przede wszystkim wprowadzono tryb pełnoekranowy, który pozwoli na zmniejszenie rozproszenia użytkowników aplikacji podczas zabawy, zwłaszcza, że docelowymi odbiorcami mają być dzieci. W takim trybie pole z podglądem zajmuje całe dostępne miejsce, pomijając licznik oraz statystyki, a sam obraz kamery jest w nim wycentrowany i maksymalnie rozciągnięty.



Rys. 36. Porównanie wersji aplikacji przed i po pierwszym etapie walidacji

- a) poprzednia wersja aplikacji - tryb okienkowy b) nowa wersja aplikacji - tryb okienkowy
- c) poprzednia wersja aplikacji – tryb pełnoekranowy d) nowa wersja aplikacji – tryb pełnoekranowy

Kolejną dużą zmianą było wprowadzenie sesyjnego licznika uśmiechów, który resetuje się za każdym razem po wejściu na ekran z kamerą. Ponieważ uczestnicy testu mieli

wątpliwości, czy aplikacja prawidłowo policzyła ich uśmiech, zdecydowano się znacząco powiększyć nowy komponent względem pozostałych statystyk. Dodatkowo, aby nieco ułatwić zrozumienie sposobu działania aplikacji, jeżeli licznik sesyjny wskazuje na zerową liczbę wykrytych uśmiechów, komponent wyświetla tekst zachęcający do uśmiechania się. Przetłumaczono także nazwę aplikacji w polskiej wersji językowej, dzięki czemu osoby nie znające języka angielskiego mogą od razu zrozumieć, jaki jest cel aplikacji.

### **6.3. Testy z dziećmi z autyzmem**

Ostatnim krokiem ewaluacji gotowej aplikacji były testy z dwoma chłopcami chorymi na autyzm. W celu przeprowadzenia badania udano się do *Instytutu Wspomagania Rozwoju Dziecka* w Gdańsku i, podobnie jak w poprzednich testach, zdecydowano się na nagrywanie ekranu komputera oraz obrazu z kamery pokazującej twarz dziecka. Jako usługę wykrywającą uśmiechy ustwiono połączenie możliwości bibliotek OpenCV oraz FaceSDK. Tym razem wykorzystano tryb pełnoekranowy aplikacji, dzięki czemu dzieci nie były rozpraszczone przez programy w tle. Testy były krótkie i trwały odpowiednio 4 oraz 2 minuty.

W pierwszej kolejności działanie aplikacji zostało zaprezentowane opiekunce dzieci. Po krótkiej zabawie z programem zdecydowała się wyłączyć wszystkie efekty zachęcające do uśmiechu, ponieważ dzieci chore na autyzm bawiłyby się nimi, zamiast się uśmiechać. Według niej, dzieci powiązałyby prezentowanie efektów z poruszaniem się na boki, co mogłoby utrudnić zrozumienie celu aplikacji.

Pierwszy chłopiec, który wziął udział w teście był nieco starszy i w jego przypadku istniała możliwość komunikacji werbalnej. W pierwszej chwili zaskoczył się obrazem z kamery i nie wiedział, co należy dalej zrobić. Dopóki opiekunka nie wy tłumaczyła mu, na czym polega zabawa, chłopiec siedział wpatrując się w wykryte fragmenty twarzy i zastanawiając się, czemu odpowiadają żółte i czerwone prostokąty. Kiedy pierwszy raz aplikacja wykryła jego uśmiech, wyglądał na nieco zdziwionego, jednak z każdym kolejnym efektem zdawał się coraz bardziej z nich cieszyć. Podczas testów widać było, że chłopiec uśmiechał się dosyć przeciągle, przez co aplikacja traktowała to jako pojedynczy uśmiech. Kiedy opiekunka poprosiła go o odczytanie licznika już wykrytych uśmiechów i wykonanie kilku następnych, ciągłość uśmiechów sprawiała, że chłopiec zastanawiał się, kiedy licznik się zwiększy. Najbardziej przypadł mu do gustu efekt *motylek* oraz *rekordzista*, podczas którego ucieszył się ze swojego wyniku. Łącznie aplikacja naliczyła 13 uśmiechów, a sama detekcja twarzy i uśmiechów działała bardzo sprawnie.

W przypadku drugiego chłopca sytuacja była nieco trudniejsza, ponieważ nie można było skomunikować się z nim werbalnie. Chłopiec przez większość czasu robił różne miny, wykorzystując do nich głównie język, przez co aplikacja miała częściowe problemy z odnalezieniem ust. Bawił się ze swoim odbiciem w kamerze, a kiedy aplikacja wykryła uśmiech, zdawał się być zaskoczony efektami. Wydaje się jednak, że nie udało się mu zrozumieć celu zabawy. Aplikacja wykryła jedynie 6 uśmiechów.

Po wykonaniu testów odbyła się ponownie rozmowa z opiekunką w celu ich podsumowania. Pierwsza uwaga dotyczyła efektów nagradzających uśmiech – powinny one być znacznie bardziej atrakcyjne i rzucające się w oczy, ponieważ dzieci z autyzmem mogły ich nawet nie zauważyc. Jako przykład takiego efektu opiekunka użyła twarzy zwierzątka w formie animowanej, rysowanej grafiki, podobnego do logo aplikacji. Zwierzątko takie miałoby być prezentowane na całym obrazie z kamery i powinno się uśmiechać. Potwierdziły się też częściowo jej obawy o efektach zachęcających do uśmiechu – w przypadku pierwszego chłopca nawet pomocnicze informacje o wykrytych fragmentach twarzy bardzo przykuwały uwagę. Zgodnie z jej radą, nie należy pokazywać niczego, co nagradza brak uśmiechu. Dzieci powinny być zachęcane do niego przy pomocy bodźców zewnętrznych, np. poprzez łaskotanie przez opiekuna lub powinny spróbować samodzielnie odgadnąć cel aplikacji, dzięki czemu uśmiech nie będzie wymuszony. Lista efektów powinna być także znacznie bardziej rozbudowana, żeby rzadziej się powtarzały.

Kolejnym problemem poruszonym podczas tego spotkania była detekcja ciągłości uśmiechu. Ponieważ jeden z chłopców był zdziwiony, dlaczego mimo uśmiechania się licznik nie został zwiększyony, opiekunka zaproponowała ustawienie limitu czasowego dla pojedynczego uśmiechu. Po jego przekroczeniu uśmiech zostanie uznany za nowy, a prezentowany efekt się zmieni. Ze względu na fakt, że prawidłowe wykrywanie ciągłości uśmiechu to jeden z głównych celów niniejszej pracy, funkcjonalność taka powinna być możliwa do wyłączenia.

Trudnym do rozwiązania jest problem wyjaśnienia dzieciom, na czym polega aplikacja. Z częścią dzieci z autyzmem komunikacja werbalna jest mocno utrudniona, a spora część z nich nie radzi sobie jeszcze dobrze z czytaniem. Zgodnie z uwagą opiekunki, efekty zachęcające do uśmiechu nie powinny być prezentowane. Głównym problem był także fakt, że dzieci nie zwracały uwagi na licznik sesyjny, który zachęcał do uśmiechu.

Jedną z propozycji było odtwarzanie sygnału dźwiękowego o treści „*Uśmiechnij się!*”, jednak po krótkiej konsultacji zdecydowano się zrezygnować z tego pomysłu ze względu na fakt, że trudno jest ustalić moment, w którym miałyby być odtwarzany oraz mógłby on powodować rozproszenie uwagi dziecka.

Dzieci z autyzmem, które nie potrafią czytać, a komunikacja werbalna z nimi jest utrudniona, postanowiono spróbować nauczyć celu aplikacji poprzez bodźce zewnętrzne (np. wspomniane wcześniej łaskotanie przez opiekuna) i obserwację zachowań efektów. Dla starszych dzieci, które lepiej radzą sobie z czytaniem, postanowiono zmodyfikować działanie komponentu z licznikiem sesyjnym. Jeżeli od dłuższej chwili (określonej np. parametrem) aplikacja nie wykryła uśmiechu, licznik niezależnie od liczby wykrytych uśmiechów powinien przejść w stan zachęcania, w którym prezentowałby prosty komunikat typu „*Uśmiechnij się!*”. Tekst takiego komunikatu powinien wyróżniać się kolorystycznie, między innymi poprzez mruganie lub dynamiczną zmianę kolorów. Dzięki temu dzieci zwróciłyby większą uwagę na napis i dowiedziały się, co należy zrobić.

## **6.4. Wdrożenie**

### **6.4.1. Wdrożenie aplikacji internetowej na serwerze Politechniki Gdańskiej**

Podczas wdrożenia aplikacji na serwerze Politechniki Gdańskiej wykorzystano w całości mechanizm uruchamiania środowisko przy pomocy Dockera. Zainstalowana wersja internetowa aplikacji umożliwiała także pobranie wersji na komputery stacjonarne skonfigurowaną w taki sposób, żeby automatycznie podpięta była pod odpowiednią bazę danych.

## PODSUMOWANIE

Pierwszym etapem pracy magisterskiej było rozpoznanie istniejących bibliotek, które mogłyby być wykorzystane w niniejszej pracy magisterskiej. Udało się odnaleźć cztery biblioteki: OpenCV, OpenIMAJ, FaceSDK oraz JavaCV, którą zdecydowano się porzucić ze względu na fakt, że była jedynie nakładką OpenCV dla języka Java. Trzy pozostałe postanowiono wykorzystać w implementacji usług do detekcji uśmiechu.

Utworzone w taki sposób usługi poddano testom w celu określenia ich wydajności i dokładności. W trakcie wykonywania testów odkryto, że biblioteka OpenCV w najszybszym czasie zwraca dokładne informacje o położeniu twarzy, ale przeciętnie radzi sobie z detekcją uśmiechu. Biblioteka FaceSDK miała natomiast problemy z wykrywaniem błędów detekcji (ang. *false alarms*), jednak w najdokładniejszy sposób potrafiła ustalić pozycję oraz stopień uśmiechu ust (w przedziale  $<0, 1>$ ). W związku z tym zdecydowano się na zaimplementowanie dodatkowej usługi, która wykrywa twarze przy pomocy OpenCV, a uśmiechy dzięki możliwościom FaceSDK. Taka implementacja okazała się być najszybsza i nie miała prawie wcale problemów z błędami detekcji. Wszystkie usługi implementowały wspólny interfejs, dzięki czemu w bardzo łatwy sposób można się między nimi przełączać.

Wyniki testów pozwoliły na ustawienie dodatkowej implementacji jako domyślnej. Usługa oparta na OpenIMAJ, ponieważ czas jej działania był najdłuższy, została ustawiona jako rezerwowa i wykorzystywana jest w przypadku problemów z uruchomieniem pozostałych bibliotek.

Następnym krokiem było wykorzystanie zaimplementowanych usług w przetwarzaniu obrazu z kamery w czasie rzeczywistym. W ramach tych prac wypracowano algorytm pozwalający zliczać uśmiechy trwające wiele klatek. Został on zaimplementowany dzięki zdefiniowaniu czterech stanów uśmiechu: brak uśmiechu, początek uśmiechu, trwanie uśmiechu oraz zakończenie uśmiechu.

Podczas przetwarzania obrazu w czasie rzeczywistym pojawiło się kilka problemów z detekcją uśmiechu. Zaimplementowano mechanizmy, które zwiększyły odporność na niepewności danych zwracanych przez bibliotekę – błędy detekcji, znikanie uśmiechów między klatkami oraz uśmiechy na pograniczu progu, które powodowały zliczanie każdej klatki jako nowego uśmiechu.

Zdefiniowanie czterech stanów uśmiechu pozwoliło na zaimplementowanie prostych efektów graficznych w aplikacji na komputery stacjonarne. Podzielono je na dwie grupy: efekty zachęcające do uśmiechu, które wyświetlają się, gdy użytkownik się nie uśmiecha oraz efekty nagradzające uśmiech, wyświetlające się w trakcie trwania uśmiechu.

Równolegle z aplikacją na komputery stacjonarne powstawała wersja internetowa aplikacji, której głównym celem jest możliwość testowania usług do wykrywania uśmiechu. Po zakończeniu jej implementacji zakupiono serwer i wystawiono ją do użytku publicznego, dzięki czemu udało się zebrać 32 testy od różnych użytkowników. Badały one nie tylko wydajność oraz dokładność wykorzystywanych usług, ale także pozwalały użytkownikowi

na wyrażenie swojej opinii na temat każdej z nich. Testy potwierdziły wcześniejsze założenia na temat tego, która z usług będzie najlepsza do detekcji uśmiechu.

Po zakończeniu testów na dorosłych użytkownikach zdecydowano się przejść do następnego kroku ewaluacji gotowej aplikacji. W tym celu wykonano testy na dwóch dzieciach typowo rozwijających się. Ich reakcje były bardzo odmienne, jednak wyniki testów pozwoliły na wskazanie i naprawienie błędów w interfejsie aplikacji na komputery stacjonarne.

Ostatnim etapem ewaluacji były testy z udziałem dzieci chorych na autyzm w *Instytucie Wspomagania Rozwoju Dziecka* w Gdańsku. Testy te pokazały, że wymagania zdrowych dzieci oraz dzieci z autyzmem są zupełnie odmienne. Konsultacje z opiekunką dzieci pozwoliły na wskazanie problemów w aplikacji oraz dostosowanie jej pod potrzeby instytutu.

Aplikacja internetowa, dzięki której można także na pobrać wersję na komputery stacjonarne, została postawiona przy użyciu Dockera na serwerach Politechniki Gdańskiej. Dane przez nią zbierane na temat poprawności wykrywania uśmiechów mogą być wykorzystywane do utworzenia bardziej wydajnych klasyfikatorów detekcji twarzy oraz uśmiechów – mogą one być wykorzystane w innych pracach magisterskich o podobnej tematyce.

Główne cele niniejszej pracy magisterskiej zostały zrealizowane – powstała aplikacja na komputery stacjonarne, która przechowuje dane o automatycznie wykrytych uśmiechach we współdzielonej pomiędzy wszystkimiinstancjami bazie danych. Zebrane informacje są wyświetlane w postaci statystyk na ekranie z podglądem kamery – pokazane tam są liczniki dzienne, tygodniowe, miesięczne oraz od początku uruchomienia aplikacji, zarówno dla konkretnej instancji, jak i wszystkich instancji razem. Dodatkowo, aplikacja ta prezentuje efekty nagradzające i zachęcające do uśmiechu. W ramach prac została też utworzona aplikacja internetowa, która pozwala na testowanie zaimplementowanych usług. Projekt został zrealizowany zgodnie z konceptem – obydwie wersje korzystają ze wspólnych usług do detekcji uśmiechu.

Praca była rozwijana systematycznie, a postępy były regularnie prezentowane jej opiekunowi.

Niestety, nie udało się wytrenować własnego klasyfikatora do wykrywania uśmiechu. Pomimo częściowego zebrania próbek pozytywnych i negatywnych, proces szkolenia zajął zbyt dużo czasu i zasobów, przez co pomysł utworzenia własnego klasyfikatora został porzucony.

## WYKAZ LITERATURY

1. Snapchat – Statistics & Facts, <https://www.statista.com/topics/2882/snapchat/> (data dostępu: 09.09.2017)
2. MongoDB and Oracle Compared, <https://www.mongodb.com/compare/mongodb-oracle> (data dostępu: 09.09.2017)
3. The Science of the Smile, <https://www.uvneer.com/veneervault/science-of-the-smile/> (data dostępu: 09.09.2017)
4. Duchenne, Guillaume (1990). The Mechanism of Human Facial Expression. New York: Cambridge University Press
5. 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.
6. Anthony H.L. Tjan.. Dr. Dent., Some esthetic factors in a smile, The C. V. Mosby Company. Published by Elsevier Inc. (1984)
7. Juliano E. C. Cruz, Elcio H. Shiguemori, Lamartine N. F. Guimaraes , A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery
8. Paul Viola and Michael Jones, Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade, Mitsubishi Electric Research Lab, Cambridge, MA
9. Schapire R.E. (2013) Explaining AdaBoost. In: Schölkopf B., Luo Z., Vovk V. (eds) Empirical Inference. Springer, Berlin, Heidelberg
10. Liao S., Zhu X., Lei Z., Zhang L., Li S.Z. (2007) Learning Multi-scale Block Local Binary Patterns for Face Recognition. In: Lee SW., Li S.Z. (eds) Advances in Biometrics. ICB 2007. Lecture Notes in Computer Science, vol 4642. Springer, Berlin, Heidelberg
11. Edward Philips The Classification of Smile Patterns, J Can Dent Assoc 1999; 65:252-4
12. Boosting and AdaBoost for Machine Learning,  
<https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/> (data dostępu: 09.09.2017)
13. BOOSTING (ADABOOST ALGORITHM),  
<http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Eric-Boosting304FinalRpfd.pdf> (data dostępu: 09.09.2017)
14. Haar Cascade Visualization, <https://www.youtube.com/watch?v=hPCTwxF0qf4> (data dostępu: 09.09.2017)
15. 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) (data dostępu: 09.09.2017)
16. Training Haar-cascade, <https://singhgaganpreet.wordpress.com/2012/10/14/training-haar-cascade/> (data dostępu: 09.09.2017)
17. 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)
18. Lista klasyfikatorów Haara,  
<https://github.com/opencv/opencv/tree/master/data/haarcascades> (data dostępu: 09.09.2017)
19. 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)

20. Klasyfikator Haara dla ust, <http://alereimondo.no-ip.org/OpenCV/uploads/34/Mouth25x15.1.zip> (data dostępu: 09.09.2017)
21. Klasyfikator Haara dla uśmiechu, <https://github.com/lukagabric/PyOpenCV/blob/master/Resources/smile.xml> (data dostępu: 09.09.2017)
22. 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)
23. Haar Cascades, <http://alereimondo.no-ip.org/OpenCV/34> (data dostępu: 09.09.2017)
24. OpenCV now supports desktop Java, <http://opencv.org/opencv-java-api.html> (data dostępu: 09.09.2017)
25. 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)
26. OpenIMAJ: Intelligent Multimedia Analysis, <http://openimaj.org/> (data dostępu: 09.09.2017)
27. Luxand FaceSDK Developer's Guide, <https://www.luxand.com/facesdk/documentation/> (data dostępu: 09.09.2017)
28. Luxand FaceSDK – Face Detection, <https://www.luxand.com/facesdk/documentation/facedetection.php> (data dostępu: 09.09.2017)
29. Cohn-Kanade (CK and CK+) database, <http://www.consortium.ri.cmu.edu/ckagree/> (data dostępu: 09.09.2017)
30. The Japanese Female Facial Expression (JAFFE) Database, <http://www.kasrl.org/jaffe.html> (data dostępu: 09.09.2017)

## **WYKAZ TABEL**

Tabela 1. Tablica pomyłek wykorzystywana podczas detekcji uśmiechów .....	16
Tabela 2. Własnoręcznie utworzony zbiór testowy .....	26
Tabela 3. Wyniki graficzne testów na własnoręcznie utworzonym zbiorze.....	28
Tabela 4. Wyniki testów usługi OpenCV + FaceSDK dla utworzonego zbioru zdjęć .....	29
Tabela 5. Statystyki usług z testów własnoręcznie utworzonego zbioru.....	30
Tabela 6. Wyniki testów zbioru Cohn-Kanade .....	31
Tabela 7. Wyniki testów zbioru JAFFE.....	34
Tabela 8. Zastosowanie klasyfikatora LBP dla obu zbiorów .....	35
Tabela 9. Informacje o zrealizowanych efektach .....	47
Tabela 10. Lista parametrów konfiguracyjnych dla aplikacji na komputery stacjonarne	53
Tabela 11. Lista parametrów konfiguracyjnych dla części frontendowej .....	63
Tabela 12. Statystyki dotyczące testujących.....	70

## WYKAZ RYSUNKÓW

Rys. 1. Różnice pomiędzy uśmiechem Pan Am oraz Duchenne [3] .....	9
Rys. 2. Przedstawienie stopni uśmiechu otwartego [6] .....	10
Rys. 3. Punkty kluczowe twarzy wyświetlane przy pomocy Luxand FaceSDK [opr. wł.]	11
Rys. 4. Pseudokod algorytmu AdaBoost .....	14
Rys. 5. Cechy obrazu badane przez klasyfikator [opr. wł.].....	15
Rys. 6. Przykład nałożenia cech Haara na zdjęciu [8] .....	15
Rys. 7. Przykład działania klasyfikatora LBP [7] .....	17
Rys. 8. Przykład działania MBLBP [opr. wł.].....	18
Rys. 9. Różnice w określaniu cech twarzy przez klasyfikatory [15] .....	19
Rys. 10. Przykład działania klasyfikatora uśmiechu przy użyciu OpenCV .....	24
Rys. 11. Przykład detekcji uśmiechu dla połączonych klasyfikatorów .....	25
Rys. 12. Przykład błędnej detekcji dla biblioteki Luxand .....	32
Rys. 13. Przykład niewykrycia ust przez bibliotekę OpenCV .....	33
Rys. 14. Porównanie rozmiarów twarzy i ust zwróconych przez biblioteki.....	33
Rys. 15. Porównanie detekcji w przypadku otwartych ust.....	34
Rys. 16. Porównanie detekcji w przypadku grymasu .....	35
Rys. 17. Różnice w detekcji pomiędzy klasyfikatorami Haara i LBP .....	36
Rys. 18. Przykład wystąpienia błędnej detekcji przy użyciu klasyfikatora LBP .....	37
Rys. 19. Fragment zebranego zbioru próbek pozytywnych .....	39
Rys. 20. Fragment zebranego zbioru próbek negatywnych .....	39
Rys. 21. Koncepcja architektury systemu.....	40
Rys. 22. Ekran podglądu obrazu z kamery w aplikacji na komputery stacjonarne .....	52
Rys. 23. Ekran ustawień w aplikacji na komputery stacjonarne .....	53
Rys. 24. Ekran główny aplikacji internetowej.....	56
Rys. 25. Ekran wstępny testowania usług wykrywających uśmiech .....	58
Rys. 26. Główny ekran testowania usług wykrywających uśmiech .....	58
Rys. 27. Ekran z formularzem dotyczącym działania usług .....	60
Rys. 28. Ekran zapisywania uśmiechów wykrytych podczas testu usług .....	61
Rys. 29. Ekran prezentujący wyniki testów .....	61
Rys. 30. Ekran pobierania aplikacji na komputery stacjonarne.....	62
Rys. 31. Wykres przedstawiający opinie na temat działania usług .....	67
Rys. 32. Przedstawienie typów wykrywanych uśmiechów zdaniem użytkowników .....	68
Rys. 33. Wykres prezentujący zależność wykrytych uśmiechów do czasu testowania .	69
Rys. 34. Zestawienie liczby wykrytych uśmiechów do skuteczności ich detekcji.....	70
Rys. 35. Odpowiedzi na pytanie o przyszłość usług badających emocje użytkownika..	71
Rys. 36. Porównanie wersji aplikacji przed i po pierwszym etapie walidacji.....	73

