POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Projekt z Przetwarzania Informacji Wizyjnej

Opracowanie metody detekcji uśmiechu w barwnych obrazach cyfrowych.

Autorzy: Patryk Kühne, Paweł Pendziałek

Rok IV, semestr VII, sekcja Ti2 - 5:

Kierujący pracą: prof. dr hab. Bogdan Smołka

Gliwice, listopad 2020

Spis treści

1. Wstęp

Uśmiech jest bardzo ważnym niewerbalnym sposobem wyrażania swoich emocji oraz intencji i odgrywa znaczącą rolę w codziennym życiu społecznym. Jako że, interpretacja emocji jest kluczowa dla interakcji ludzi, warto, aby umiejętność tą posiadały również maszyny. Ton głosu oraz mowa ciała to elementy, które pozwalają na przekazywanie emocji, ale bardzo ważny jest również wyraz twarzy, który można badać za pomocą systemów wizyjnych. Detekcja i interpretacja różnych emocji przekazywanych przez człowieka jest na pewno czymś, co miałoby szerokie spektrum zastosowań, lecz my chcielibyśmy się skupić na ten moment jedynie na detekcji uśmiechu, który może być interpretowany jako miara szczęścia lub radości.

2. Cel i zakres projektu

Celem naszego projektu jest opracowanie metody, która pozwoli nam na jak najdokładniejszą detekcję uśmiechu na obrazach cyfrowych. Zakres naszej pracy obejmuje zaimplementowanie odpowiedniej metody detekcji, rzetelne przetestowanie jej działania oraz dodatkowo stworzenie minimalistycznego interfejsu naszej aplikacji.

3. Realizacja projektu

3.1. Określenie problemu

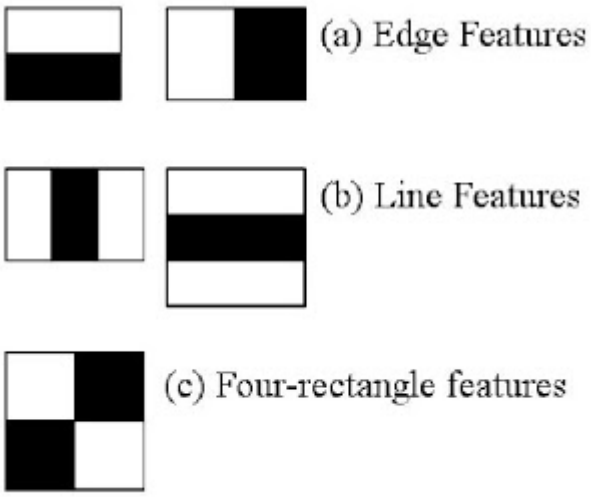
Pierwszym postawionym przez nas celem było znalezienie sposobu na zawężenie obszaru poszukiwań uśmiechu na obrazie. Warunkiem koniecznym do wykrywania uśmiechu jest obecność na obrazie osoby, której uśmiech można potencjalnie wykryć. Sama detekcja ludzi na obrazach nie jest jednak wystarczająca – ludzie mogą być odwróceni w taki sposób, że nie będzie widać ich twarzy, a zatem próba detekcja uśmiechu w takich przypadkach jest pozbawiona sensu. Zawężając jeszcze bardziej dziedzinę naszych poszukiwań, stwierdziliśmy, że odpowiednim podejściem, przed wykrywaniem uśmiechu będzie detekcja twarzy. Podejście to jest korzystne, ponieważ detekcja uśmiechu na obszarze jedynie w obrębie twarzy ułatwia poszukiwania i zapobiega przypadkowemu wykryciu uśmiechu w innych miejscach. Twarz jest elementem o większej ilości cech charakterystycznych, a zatem jest znacznie łatwiejsza do wykrycia niż sam uśmiech.

3.2. Zaproponowane rozwiązanie

Po określeniu problemu oraz zdefiniowaniu sposobu podejścia zaczęliśmy rozglądać się za sposobami jego rozwiązania. Analizując dostępne metody, zdecydowaliśmy się na wykorzystanie następujących technologii:

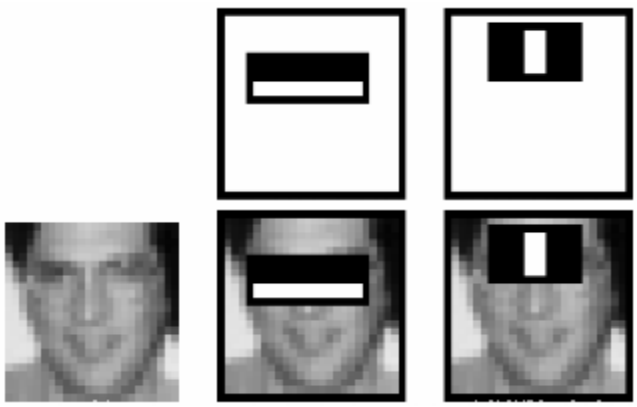
- Python – wysokopoziomowy język programowania, który oferuję rozbudowany pakiet bibliotek i modułów. Składnia tego języka pozwala na szybkie prototypowanie oraz tworzenie aplikacji o przejrzystym kodzie źródłowym.
- OpenCV – biblioteka z otwartym kodem źródłowym stworzona w celu przetwarzania obrazów. Stawia się jej wymaganie przetwarzania obrazów w czasie rzeczywistym, a zatem jest ona napisana głównie w języku C, co zapewnia jej wysoką wydajność. Często stawianym zarzutem przeciwko językowi programowania Python jest to, że jest on znacznie wolniejszy niż kod napisany w językach niższego poziomu. Zastosowanie biblioteki OpenCV, która jak wcześniej wspomniałem, zaimplementowana jest głównie w języku C, pozwala na osiągnięcie wysokiej wydajności. Python jedynie wywołuje funkcje biblioteki, które następnie są przetwarzane w takim sam sposób, jak gdyby zostały wywołane z programu napisanego w innym języku.

- Klasyfikator kaskadowy Haar'a – jest to dostępny w bibliotece OpenCV klasyfikator używający metody wykrywania obiektów zaproponowanej przez Paula Violę i Michaela Jonesa[1]. Istotą działania tego klasyfikatora są odpowiednie cechy obiektu, w naszym przypadku uśmiechu oraz pomocniczo twarzy. Cechy te zawierają informacje o zmianie kontrastu pomiędzy grupami pikseli. Daje nam to informację o tym, gdzie na obrazie znajdują się jaśniejsze a gdzie ciemniejsze obszary.



[2]

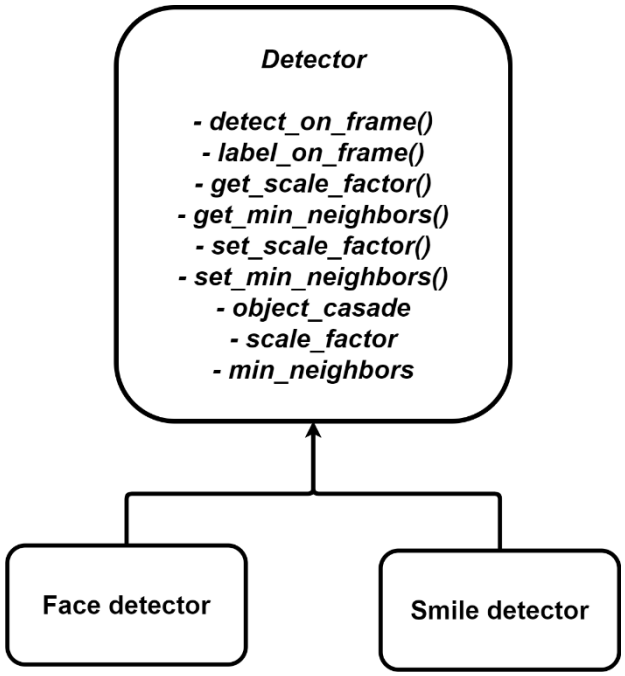
Dla przypadku wykrywania twarzy interesują nas takie elementy charakterystyczne jak oczy, nos czy usta.



[2]

3.3. Wykonanie

Naszą aplikację postanowiliśmy stworzyć w sposób obiektowy. Najważniejszym elementem są klasy wykrywające na obrazie twarz oraz uśmiech.

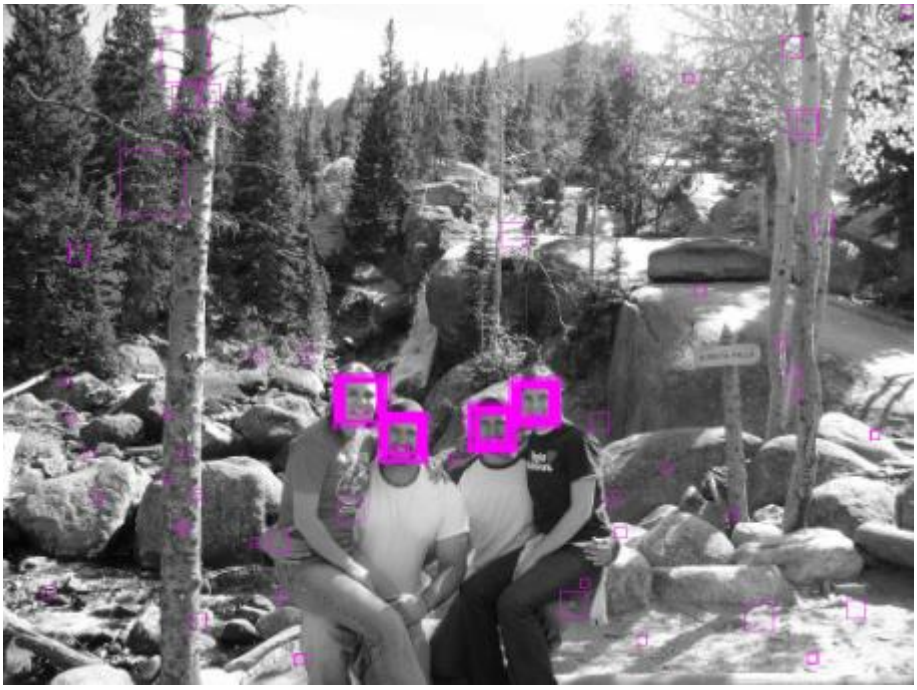


Klasa *Detector* jest klasą abstrakcyjną z metodami oraz atrybutami takimi jak przedstawiono powyżej. Istnienie obiektu klasy *Detector* nie ma sensu, gdyż nie ma ona zdefiniowanego obiektu, który ma wykrywać. Klasy *Face detector* oraz *Smile detector* to klasy dziedziczące z klasy *Detector*. Klasa bazowa ma zaimplementowane działanie funkcji *detect_on_frame()* oraz *label_on_frame()*, które odpowiednio wykrywają obiekt na obrazie oraz zaznaczają miejsce, w którym on występuje. Klasy dziedziczące muszą więc jedynie podczas inicjalizacji zdefiniować atrybut *object_cascade* za pomocą odpowiedniego klasyfikatora kaskadowego obiektu, który będą wykrywać oraz zdefiniować parametry *scale_factor* oraz *min_neighbors*.

Parametr *scale_factor* jest wykorzystywany do stworzenia tzw. piramidy skali. Klasyfikator kaskadowy ma możliwość wykrycia obiektów o stałym, niewielkim rozmiarze, w przypadku twarzy jest to 24 na 24 piksele. Niemniej jednak skalując obraz wejściowy, zmieniamy również rozmiar obiektów na nim występujących. Odpowiednio zmniejszając obraz, jesteśmy w ten sposób w stanie wykryć na obrazie większe obiekty (twarze oraz uśmiechy). Parametr *scale_factor* definiuje, z jakim krokiem skalujemy nasz obraz. Skalując obraz z mniejszym krokiem, zwiększamy szansę na dopasowanie wielkości

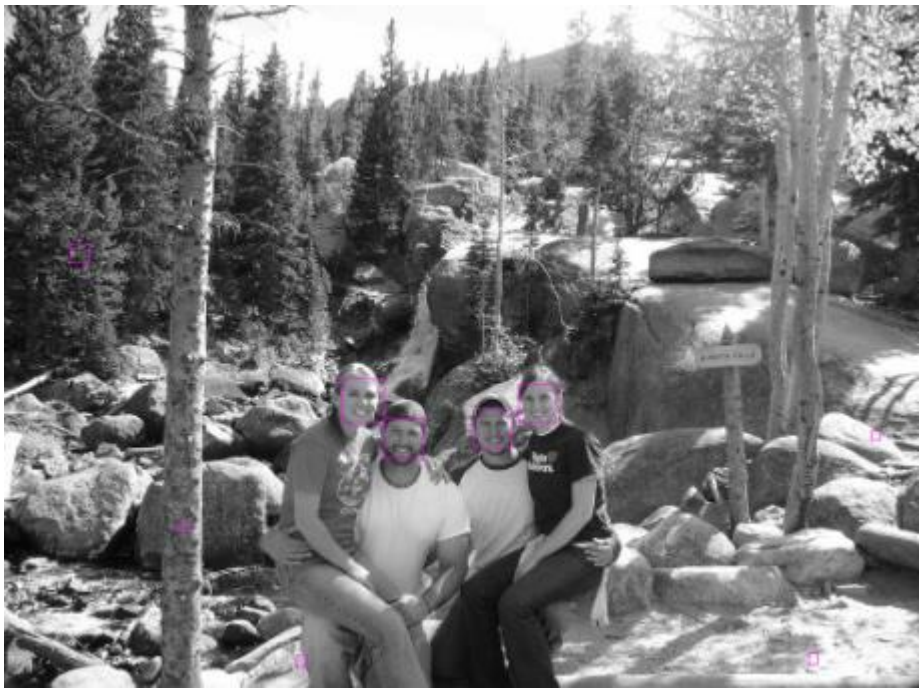
obiektu do rozmiaru modelu klasyfikatora, a zatem wykrycie obiektu jest bardziej prawdopodobne. Mały krok skalowania powoduje wzrost ilości iteracji, a zatem również wolniejsze działanie algorytmu.

Parametr *min_neighbors* zapobiega wykrywaniu obiektów fałszywie pozytywnych. Bardzo często po wykonaniu skalowania obrazu, obiekty na nim są wykrywane ponownie, co w przypadku wyzerowania parametru *min_neigbours* skutkuje następującym wynikiem:



[3]

Widać tutaj, że twarze osób zostały wielokrotnie wykryte, lecz widać również wiele pojedynczych elementów uznanych niepoprawnie za twarz. Parametr *min_neighbors* definiuje jak wiele razy dany obiekt powinien zostac wykryty, aby mógł zostać uznany za prawidłowy. Zmiana tego parametru z 0 na 1 prowadzi od następujących wyników:



[3]

Jak widać, twarze czterech osób nadal są wykrywane poprawnie, lecz ilość wyników nieprawidłowych jest zdecydowanie mniejsza. Stosując wartość parametru równą 3, otrzymamy satysfakcjonujący wynik.

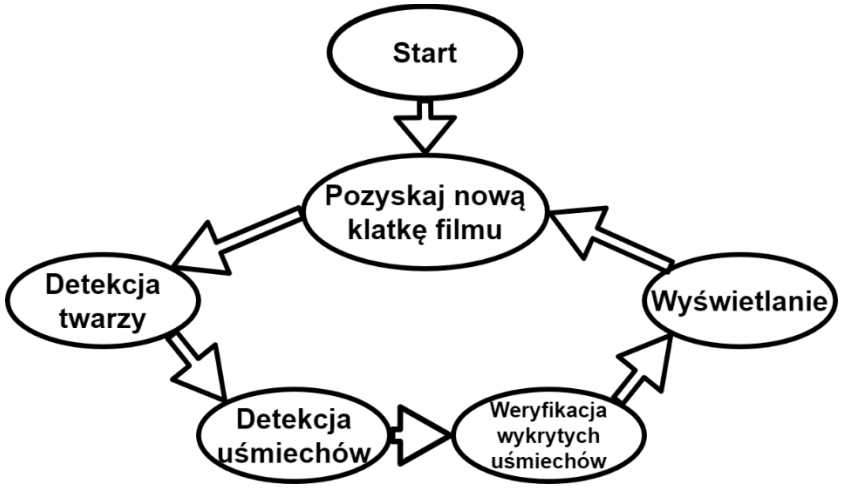
[3]

Można zatem wywnioskować, że zwiększanie parametru powoduje zmniejszenie szansy na wykrycie obiektu, lecz zwiększa prawdopodobieństwo, że obiekt będzie wykryty poprawnie.

Dodatkowo nasz program składa się z następujących zaimplementowanych przez nas klas:

App (QWidget)	Thread (QThread)	MyVideoCapture
Główna klasa naszej aplikacji, która jest odpowiedzialna za inicjalizację oraz obsługę interfejsu użytkownika, a również przypływ informacji w całej aplikacji. Klasa ta dziedziczy z klasy QWidget modułu Qt[4]	Jest to klasa, która w nowym wątku odczytuje kolejne klatki filmu. Dziedziczy ona z klasy QThread dostępnej w module Qt.Synchronizacja jest wykonana za pomocą mechanizmu sygnałów i slotów dostępnych również dzięki modułowi Qt.	Klasa pozwalająca na łatwe otwarcie oraz odczytywanie klatek z filmu. Wykorzystywana jest w klasie Thread.

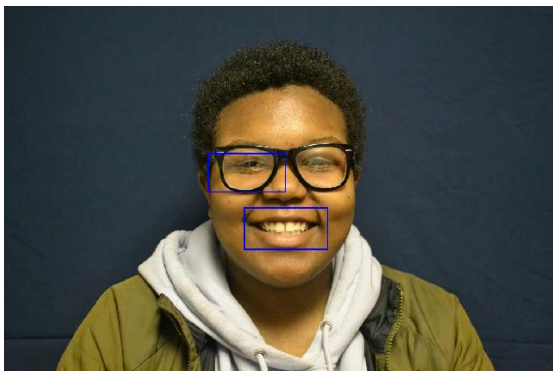
Uproszczony schemat działania naszej aplikacji prezentuje się następująco:

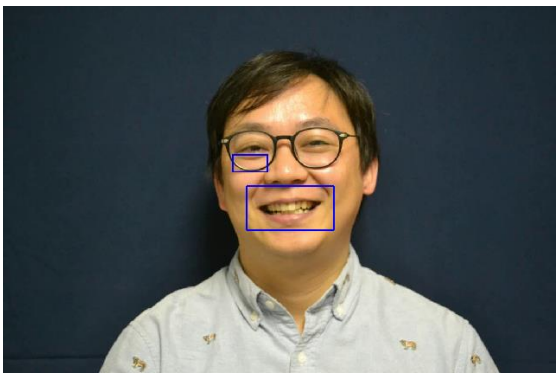


Schemat ten przedstawia jedynie kluczowe elementy działania aplikacji i pomija elementy takie jak inicjalizacja czy mechanizmy synchronizacji.

3.4.Problemy

Często pojawiającym się problem było wykrywanie uśmiechów w górnej części twarzy.





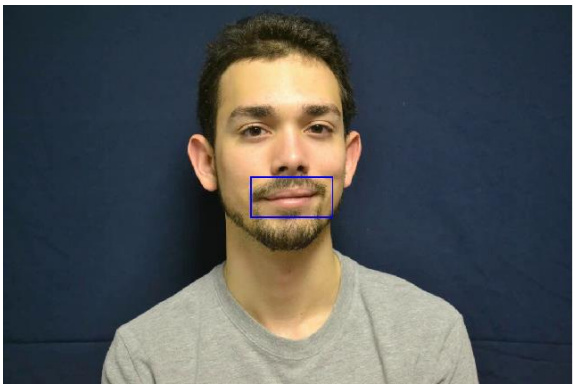
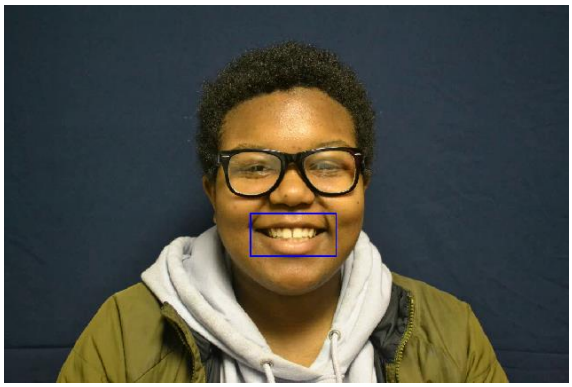
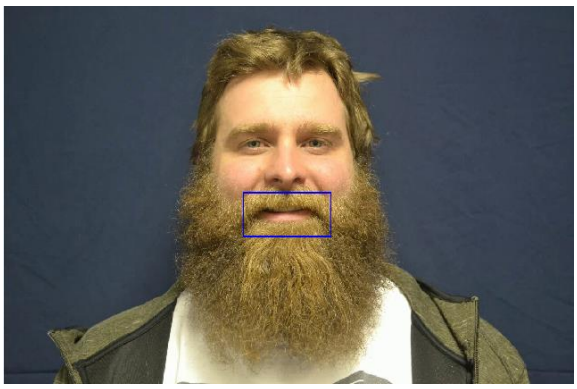
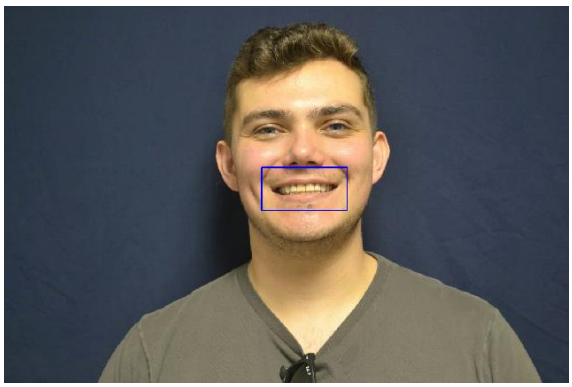
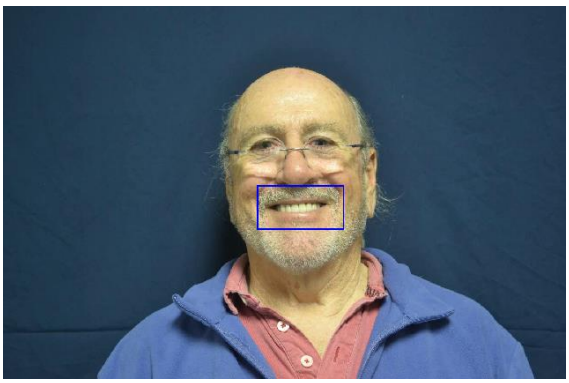
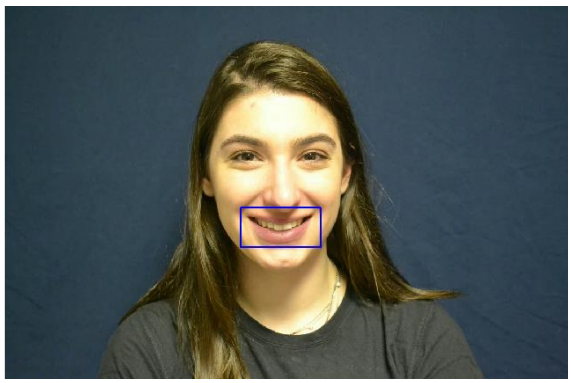
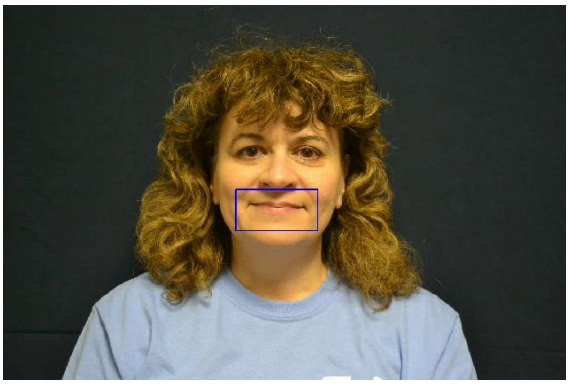
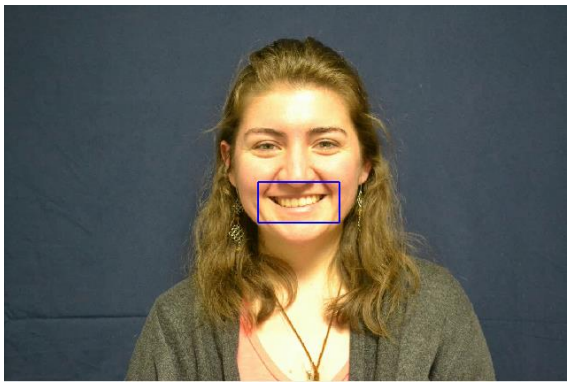
Jak widać w obu powyższych przykładach, problem ten objawiał się w przypadku, gdy osoba na zdjęciu ma założone okulary. Kształt czarnej oprawki okularów był interpretowany jako uśmiech.

Pierwszym podejściem do rozwiązania tego błędu była zmiana wartości parametrów *min_neigbors* oraz *scale_factor*, lecz jak się okazało, wyeliminowanie tego problemu w ten sposób wpływało negatywnie na całościowe działanie algorytmu wykrywania uśmiechów.

Postanowiliśmy zatem zaimplementować proste podejście, które analizuje położenie wykrytego uśmiechu na przestrzeni wykrytej twarzy i odrzuca te, które zostały wykryte w jej górnej części.

4. Działanie aplikacji

Testy naszej aplikacji oparliśmy na obrazach dostępnych w internetowych bazach danych[5][6]. Poniżej przedstawiamy wyniki działania:

5. Podsumowanie

Podczas pracy nad projektem zdobyliśmy cenną wiedzę oraz doświadczenie z zakresu przetwarzania informacji na obrazach cyfrowych. Udało nam się zaimplementować program, który z dobrą skutecznością wykrywa uśmiech na obrazach oraz filmach. Takie podejście może znaleźć zastosowanie m.in. w mierzeniu zadowolenia klientów w sklepach lub jako funkcja aparatu, która w momencie wykrycia uśmiechu automatycznie wykonuje fotografię. Nasza praca może też stanowić podstawę do dalszej pracy w temacie detekcji emocji na obrazach i zostać rozbudowana o kolejne funkcjonalności czym na pewno zainteresujemy się w przyszłości.

6. Literatura

[1] https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features

[2] https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[3] https://stackoverflow.com/questions/22249579/opencv-detectmultiscale-minneighbors-parameter

[4] https://pypi.org/project/PyQt5/

[5] http://tdface.ece.tufts.edu/

[6] https://www.uva-nemo.org/