Projekt z Analizy Obrazów – Dokumentacja

Damian Raczyński
Sebastian Konik
Mariusz Biegański

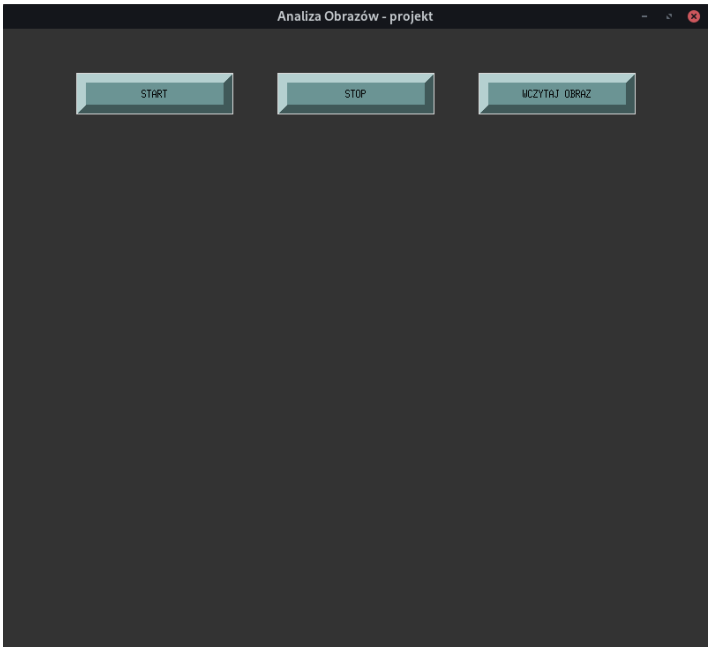
1. Opis projektu

Celem projektu było napisanie aplikacji, która wykrywa czy dana osoba lub grupa osób ma założone maseczki na twarzy. Osiągnięte zostało to dzięki wytrenowanej sieci neuronowej, przyjmującej obraz z kamery albo plik z komputera.

2. Działanie projektu

Po uruchomieniu programu, użytkownik dostaje okno z trzema przyciskami

- START – włączenie kamery,
- STOP – wyłączenie kamery,
- WCZYTAJ OBRAZ – dodanie obrazku formatu *.jpg, .jpeg* lub *.png* do programu.



Rysunek 1 Okno programu po uruchomieniu.

Program odpowiednio wyświetla wszelkie informacje odnośnie problemów z kamerą, czy przeglądarkę plików po wciśnięciu przycisku WCZYTAJ OBRAZ.

Wykrywanie twarzy zostaje wykonane automatycznie – jeżeli dana osoba ma założoną maskę, wówczas wokół twarzy pokazuje się zielony kwadrat. Jeżeli jednak dana osoba nie ma założonej maski, wokół twarzy pokazany zostaje czerwony kwadrat.

Program jest w stanie wykryć wiele twarzy na jednym obrazku – w takich sytuacjach wokół twarzy każdej wykrytej osoby zostaje narysowany odpowiedni kwadrat.

2.1. Uruchomienie projektu – system Linux

W celu uruchomienia programu na systemie Linux, należy pobrać plik wykonywalny Projekt, po czym wykonać następujące polecenia w katalogu w którym znajduje się pobrany plik:

```
chmod u+x ./Projekt
```

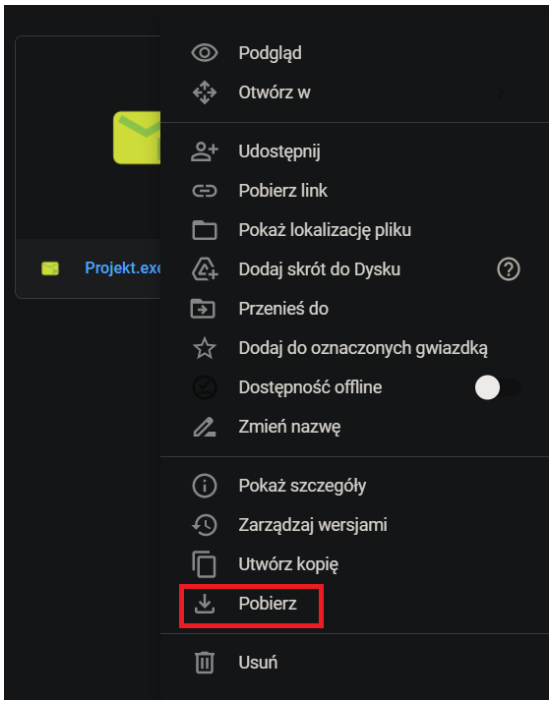
```
./Projekt
```

Po chwili użytkownik powinien zobaczyć wyżej przedstawione okno oraz być w stanie wykorzystać wszystkie funkcje programu.

2.2. Uruchomienie projektu – system Windows

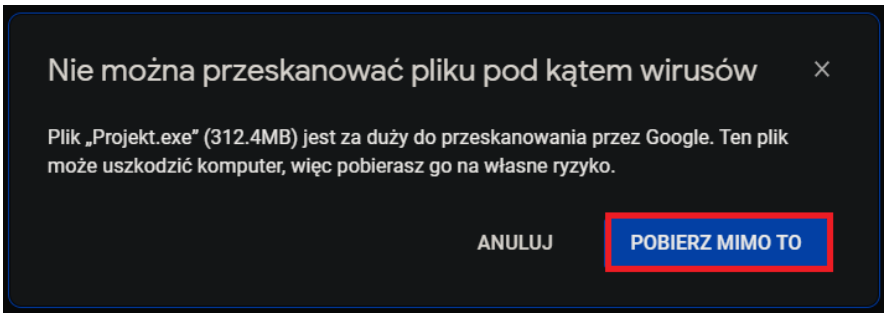
W celu uruchomienia programu na systemie Windows, należy pobrać plik wykonywalny Projekt.exe.

Prawym przyciskiem myszy klikamy na plik. Otworzy nam się okienko, potem klikamy pobierz, jak zaznaczono na rysunku czerwonym prostokątem.



Rysunek 2 Dysk Google - pobranie dokumentu.

Podczas pobierania pliku z dysku Google należy zezwolić na pobranie



Rysunek 3 Dysk Google - zezwolenie na pobieranie.

Następnie, po podwójnym kliknięciu w pobrany plik, użytkownik powinien po chwili zobaczyć okno programu i być w stanie korzystać ze wszystkich funkcjonalności programu.

Jeśli Windows zablokuje pobrany plik, postępujemy zgodnie z kolejnymi obrazkami, klikając tak jak zaznaczono kolorem czerwonym na zdjęciach.

Rysunek 4 Okna filtru Windows Defender.

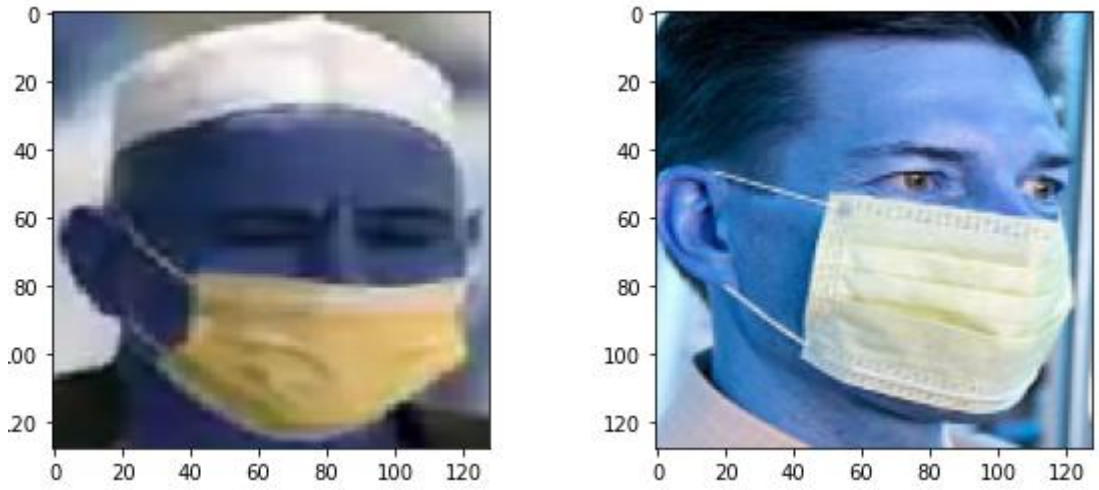
3. Opis szczegółowy

Program został w pełni napisany przy użyciu języka Python. Proces tworzenia aplikacji został podzielony na dwie części:

3.1. Trenowanie sieci neuronowej

Model musiał zostać wytrenowany na dość dużym zbiorze danych, aby osiągnąć wynik, który pozwolił na wystarczająco dobre działanie aplikacji. Użyliśmy do tego gotowego zbioru danych. Zdjęcia które ten zbiór zawiera przedstawiają osobę lub wiele osób naraz w masce lub bez. Z tego względu nie mogliśmy po prostu wrzucić całego zdjęcia do trenowania, ponieważ wynik byłby przypadkowy. Zbiór danych miał już określone etykiety na twarzach z maskami lub bez nich. Skorzystaliśmy z nich i ze zdjęć wycięliśmy same twarze z wynikiem 1 lub 0 – czy na twarzy jest maseczka. Użyliśmy sieci typu Convolutional Neural Network.

Przykładowe zdjęcia, przekształcone do rozmiaru 128x128, użyte do trenowania sieci:



Rysunek 5 Przykładowe twarze wykorzystane do trenowania sieci neuronowej.

Model w aplikacji ocenia tylko czy na twarzy jest maska lub jej nie ma, więc potrzebne jest także użycie rozwiązania wykrywającego twarze, które będą danymi wejściowymi modelu.

Przy zastosowaniu tego podejścia, udało nam się uzyskać pewność detekcji maski na poziomie 98%.

```
Epoch 35/40
144/144 [==============================] - 39s 269ms/step - loss: 0.0382 - accuracy: 0.9861 - val_loss: 0.1505 - val_accuracy: 0.9565
Epoch 36/40
144/144 [==============================] - 39s 270ms/step - loss: 0.0367 - accuracy: 0.9872 - val_loss: 0.1868 - val_accuracy: 0.9504
Epoch 37/40
144/144 [==============================] - 39s 270ms/step - loss: 0.0476 - accuracy: 0.9815 - val_loss: 0.1713 - val_accuracy: 0.9513
Epoch 38/40
144/144 [==============================] - 39s 269ms/step - loss: 0.0410 - accuracy: 0.9850 - val_loss: 0.1507 - val_accuracy: 0.9574
Epoch 39/40
144/144 [==============================] - 39s 274ms/step - loss: 0.0300 - accuracy: 0.9887 - val_loss: 0.1764 - val_accuracy: 0.9522
Epoch 40/40
144/144 [==============================] - 39s 270ms/step - loss: 0.0460 - accuracy: 0.9837 - val_loss: 0.1424 - val_accuracy: 0.9530
```

Rysunek 6 Trenowanie modelu - informacje dla ostatnich epok.

3.2. Interfejs graficzny z obsługą wytrenowanego modelu

Interfejs graficzny został utworzony z wykorzystaniem biblioteki Tkinter. Aplikacja wczytuje utworzony modelu oraz klasyfikator kaskadowy oparty na cechach Haara[1]. Następnie tworzy przyciski i przypisuje im odpowiedni funkcje.

- Przycisk START – funkcja start()

Funkcja posiada zmienną good, która może zostać nadpisana przez funkcję video. Zmienna ta mówi o tym, czy kamera została wykryta oraz czy kamera nie została odłączona w trakcie wyświetlania. Jeżeli tak, to zmienna zostaje ustawiona na wartość zero, informując przy tym pozostałe funkcje poprzez zmienną cameraOn, że kamera została wyłączona.

[1] https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html

- Funkcja video(root)

Funkcja włącza kamerę, jeśli kamera nie jest dostępna to wyświetla odpowiedni komunikat na ekranie. Następnie przygotowuje miejsce do wyświetlenia kamerki w programie. Na koniec wywołuje funkcję show_frames().

- Funkcja show_frames()

Funkcja pobiera obraz z kamery, sprawdza czy kamerka jest podłączona, w przypadku błędu wyświetla odpowiedni komunikat, ustawia zmienną cameraOn i kończy działanie. Następnie przekształca obraz z kamery by był jak najlepiej interpretowany przez sieć neuronową. Sieć neuronowa wykrywa twarze i przewiduje czy są na nich maseczki. Na zdjęcie jest nakładany odpowiedniego koloru prostokąt wskazujący gdzie została wykryta twarz oraz czy jest w maseczce czy nie. Jeśli jest w maseczce to prostokąt jest koloru zielonego, a jeśli nie to jest koloru czerwonego. Na koniec funkcja wyświetla przeanalizowany obraz z kamery i wywowłuje samą siebie.

- Przycisk STOP – funkcja stop()

Funkcja sprawdza, czy w danym momencie jest wyświetlany widok z kamery. Jeżeli tak – zatrzymuje go i ukrywa widok z kamery. Na koniec ustawia zmienną cameraOn tak, że informuje pozostałe funkcje o pomyślnym wyłączeniu kamery.

- Przycisk WCZYTAJ OBRAZ – funkcja obrazek()

Funkcja otwiera okno do wyboru zdjęcia z dysku. Wczytuje obrazek, przekształca go by był dobrze rozpoznawalny przez sieć neuronową, następnie przesyła do sieci neuronowej i otrzymuje współrzędne prostokątu objemującego wykrytą twarz. Kolor prostokątu jest wybierany w zależności od przewidywanego wyniku przez sieć neuronową. Na koniec nakłada prostokąt na obrazek, zmniejsza tak by mieścił się w oknie programu i wyświetla go.

3.3. Budowanie projektu

Aby zbudować projekt we własnym zakresie, należy przygotować Python'a (3.9.7), po czym zainstalować parę bibliotek przy użyciu, na przykład, polecenia `pip install` *`biblioteka`* :

- opencv-python - 4.5.5.62
- tensorflow - 2.5.2
- numpy - 1.19.5
- Pillow - 9.0.0

Wyżej przedstawione wersje Pythona oraz bibliotek, to wersje wykorzystane przez nas podczas budowania projektu z linków powyżej.

Aby uruchomić program, należy wykonać poniższą komendę w głównym katalogu projektu:

```
python ./Projekt.py
```

Po chwili powinno pojawić się okno programu, pozwalające użytkownikowi na wykorzystanie pełnej funkcjonalności projektu.

W celu utworzenia pliku wykonywalnego dla aktualnego systemu operacyjnego, należy zainstalować dodatkową bibliotekę – `pyinstaller (4.8)`. Po instalacji, można utworzyć plik wykonywalny wykorzystując komendę:

- Dla Linuksa:

```
pyinstaller -F --hidden-import keras.api._v2 --hidden-import
keras.engine.base_layer_v1 --add-data haarcascade_frontalface_default.xml:.
--add-data MODEL:MODEL Projekt.py
```

- Dla Windowsa:

```
pyinstaller -w -F --hidden-import keras.api._v2 --hidden-import
keras.engine.base_layer_v1 --add-data haarcascade_frontalface_default.xml;.
--add-data MODEL;MODEL Projekt.py
```

Po upływie kilku lub kilkunastu minut, w katalogu dist pojawi się plik *Projekt* lub *Projekt.exe* dla odpowiednio Linuksa lub Windowsa. Te pliki są identyczne jak te, które zostały podane w linkach powyżej.

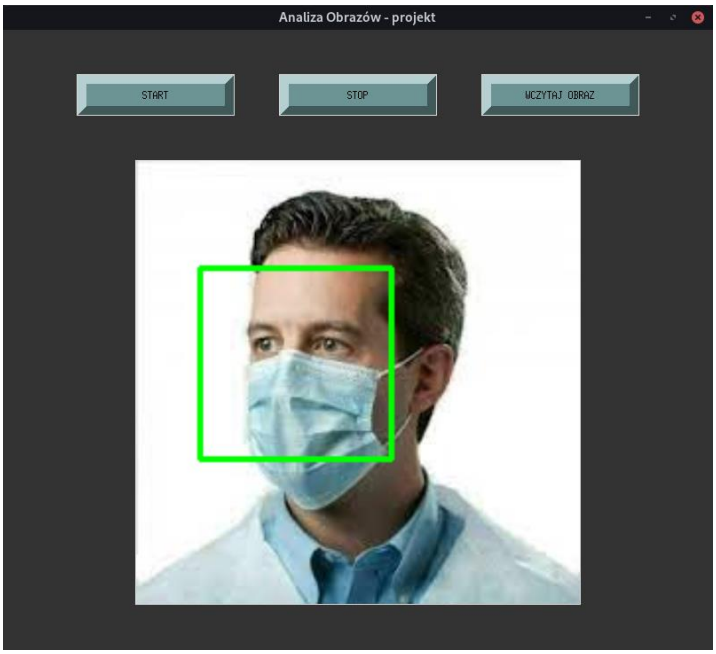
4. Podział prac

- Damian Raczyński
  - Plik wykonywalny – system Linux
  - Obsługa błędów w programie
  - Dokumentacja
- Sebastian Konik
  - Trenowanie modelu
  - Użycie modelu w aplikacji
  - Dokumentacja
- Mariusz Biegański
  - Plik wykonywalny – system Windows
  - GUI

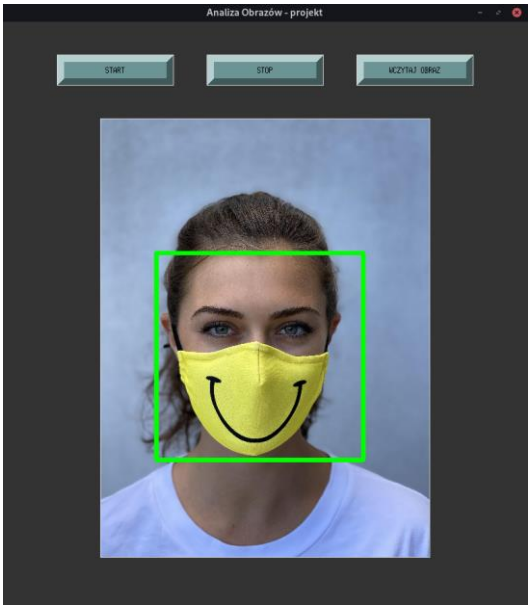
5. Co nie działa?

- Program wykorzystuje wyłącznie pierwszą dostępną kamerę na danym urządzeniu.
- Brak skalowania rozmiaru okna do wprowadzanych danych.
- Potencjalne ucięcie tego, co widać na kamerze w celu wyświetlenia rezultatu w odpowiednim rozmiarze.
- Problem z wykrywaniem maski dla osób noszących okulary oraz kiepskich warunków na obrazie, jak np.: słabe oświetlenie, rozmyte twarze w tle które mogą być dla nas istotne.
- Pliki wykonywalne ważą dużo, długo się wczytują.
- Długie wczytywanie kamery dla plików wykonywalnych.
- Stosunkowo spory spadek częstotliwości wyświetlania klatek na kamerze przy pomyślnym wykryciu twarzy.
- Wykrywanie twarzy w niektórych maseczkach.
- Program czasami wykrywa twarze tam, gdzie w rzeczywistości ich nie ma.

6. Prezentacja działania programu

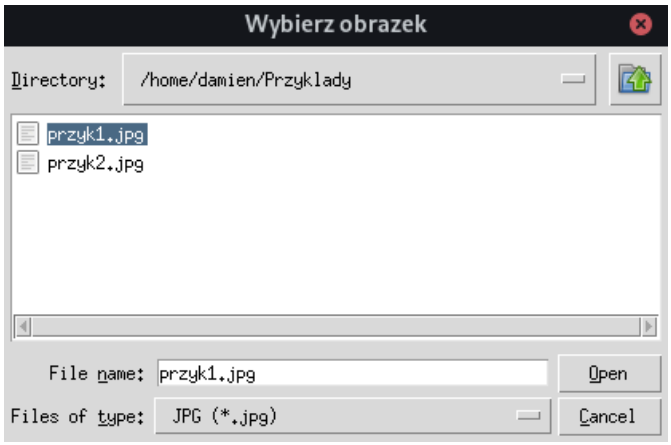


Rysunek 7 Wczytanie przykładowego obrazka (225x225) z urządzenia.

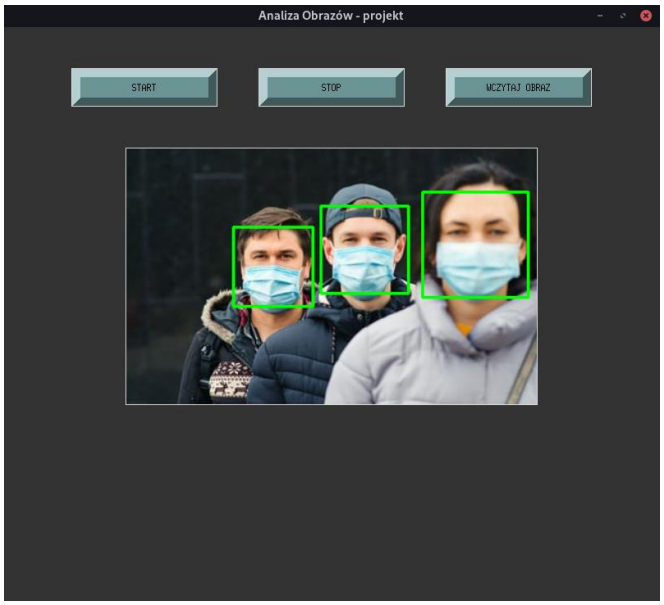


Rysunek 8 Wczytanie przykładowego obrazka (3024x4032) z urządzenia.

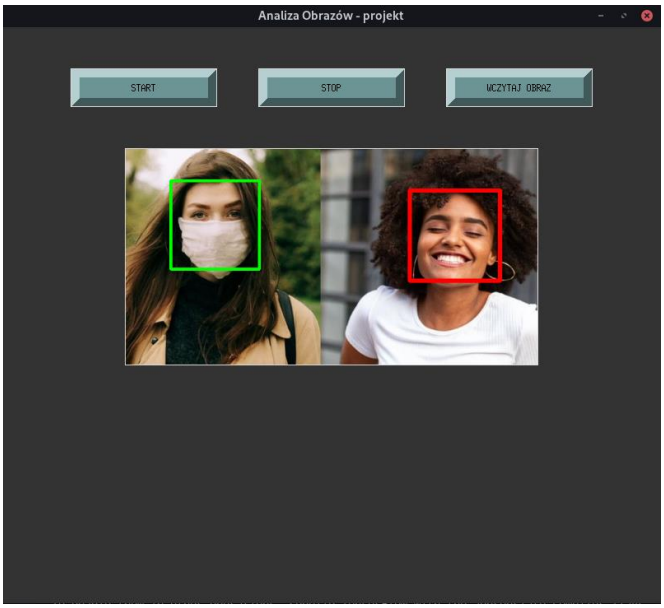
Rysunek 9 Okna ostrzegające o odłączeniu kamery oraz o niewykryciu kamery.



Rysunek 10 Okno wyboru obrazka.



Rysunek 11 Przykładowe wykrycie wielu twarzy z założonymi maseczkami (408x225)



Rysunek 12 Przykładowy obraz (679x256) z dwiema osobami - w masce oraz bez maski.