

**Projekt Indywidualny na kierunku Automatyka i Robotyka**  
**Politechnika Warszawska**  
**Wydział Elektryczny**



Magda Anna Koprowska, email: [284991@pw.edu.pl](mailto:284991@pw.edu.pl), Nadia Kerrouche, email:  
01123125@pw.edu.pl  
Semestr VI studiów I stopnia, Automatyka i Robotyka Stosowana  
Wersja i data wykonania sprawozdania: **wersja 3, 30.06.2019**  
Prowadzący: dr inż. Witold Czajewski

## **Spis treści**

<b>1.</b>	<b>WPROWADZENIE .....</b>	<b>2</b>
<b>2.</b>	<b>MOŻLIWE ROZWIĄZANIA .....</b>	<b>3</b>
2.1.	METODY WYKRYWANIA TWARZY .....	4
2.1.1.	<i>Wykrywanie twarzy i jej elementów przy użyciu klasyfikatora kaskadowego „Haar Feature-based Cascade Classifier”.</i> .....	4
2.1.2	<i>Wykrywanie twarzy i jej elementów przy użyciu „Deep Learning based Face Detector”</i> .....	6
2.2	METODY ROZPOZNAWANIA TWARZY .....	7
2.2.1	<i>Rozpoznawanie twarzy metodą EigenFaceRecognizer (PCA)</i> .....	7
2.2.2	<i>Rozpoznawanie twarzy metodą FisherFaceRecognizer (LDA)</i> .....	7
2.2.3	<i>Rozpoznawanie twarzy metodą LBPHFaceRecognizer</i> .....	8
2.3	WYBÓR ROZWIĄZANIA .....	8
<b>3</b>	<b>ZAŁOŻENIA PROJEKTU .....</b>	<b>9</b>
<b>4</b>	<b>REALIZACJA PROJEKTU.....</b>	<b>11</b>
4.2	SZCZEGÓLOWY OPIS REALIZACJI.....	11
4.3	OPIS IMPLEMENTACYJNY .....	27
4.4	<i>Opis uruchomieniowy</i> .....	32
4.4.1	<i>Instalacja środowiska</i> .....	32
4.4.2	<i>Instalacja biblioteki OpenCV.</i> .....	34
4.4.3	<i>Instalacja środowiska w systemie macOS (Mojave 10.14.5).</i> .....	36
4.4.4	<i>Instalacja biblioteki openCV.</i> .....	38
4.4.5	<i>Konfiguracja kamery w urządzeniu.</i> .....	45
<b>5</b>	<b>PODSUMOWANIE I WNIOSKI .....</b>	<b>47</b>
<b>6</b>	<b>MOŻLIWOŚCI ROZBUDOWY .....</b>	<b>49</b>
	<b>BIBLIOGRAFIA .....</b>	<b>50</b>
	<b>DODATEK .....</b>	<b>2</b>
6.2	PCH.CPP .....	2
6.3	PROJEKT.CPP.....	2
6.4	ŽRÓDŁO.CPP .....	2

## 1. Wprowadzenie

OpenCV (Open Source Computer Vision Library) to biblioteka oprogramowania komputerowego do wizji i uczenia maszynowego typu opensource. Biblioteka OpenCV została zbudowana w celu zapewnienia wspólnej infrastruktury dla aplikacji wizyjnych i przyspieszenia wykorzystania percepcji maszyn w produktach komercyjnych.

Biblioteka ma ponad 2500 zoptymalizowanych algorytmów, w tym kompleksowy zestaw klasycznych i najnowocześniejszych algorytmów komputerowego widzenia i uczenia maszynowego. Algorytmy te mogą być używane do wykrywania i rozpoznawania twarzy, identyfikacji obiektów, klasyfikowania działań ludzkich w filmach, śledzenia ruchów kamery, śledzenia poruszających się obiektów, wydobywania modeli 3D obiektów, tworzenia chmur punktów 3D z kamer stereo, łączenia obrazów w celu uzyskania wysokiej rozdzielczości obraz całej sceny, znajdowania podobnych obrazów w zbiorze, usuwania efektu czerwonych oczu z obrazów wykonanych przy użyciu lampy błyskowej, obserwowania ruchu oczu, czy rozpoznawania krajobrazów. OpenCV ma ponad 47 tysięcy użytkowników a szacowana liczba pobrań przekracza 18 milionów. Biblioteka jest szeroko stosowana w firmach, grupach badawczych i organach rządowych.

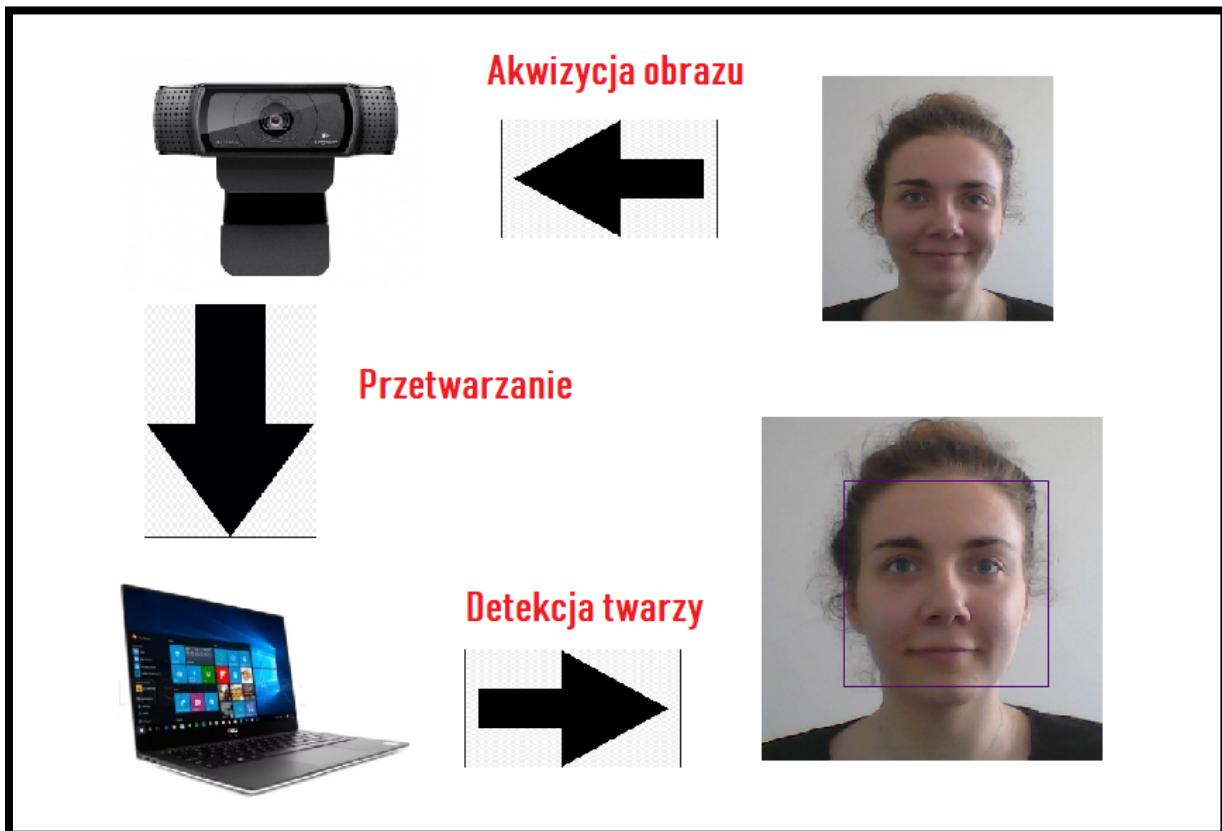
Celem projektu jest zapoznanie się z możliwościami oferowanymi przez bibliotekę OpenCV w zakresie rozpoznawania obrazu oraz zrozumienie tematyki wykorzystywania sieci neuronowych do rozpoznawania twarzy przy użyciu samodzielnie wytrenowanych klasyfikatorów i detektorów.

Projekt został wykonany z użyciem biblioteki OpenCV w wersji 4.0.1 w języku C++ w środowisku Microsoft Visual Studio.

## 2. Możliwe rozwiązania

Biblioteka OpenCV oferuje szereg klasyfikatorów do wykrywania twarzy czy innych obiektów, które są stworzone przez twórców biblioteki i dostarczone wraz z nią oraz klasyfikatorów, które są dostępne w sieci jako dodatkowe moduły.

Funkcjonowanie każdego programu z wykorzystanym algorytmem opiera się na tej samej sekwencji działań, przedstawionej poniżej:



Rysunek 1. Diagram funkcjonowania programu.

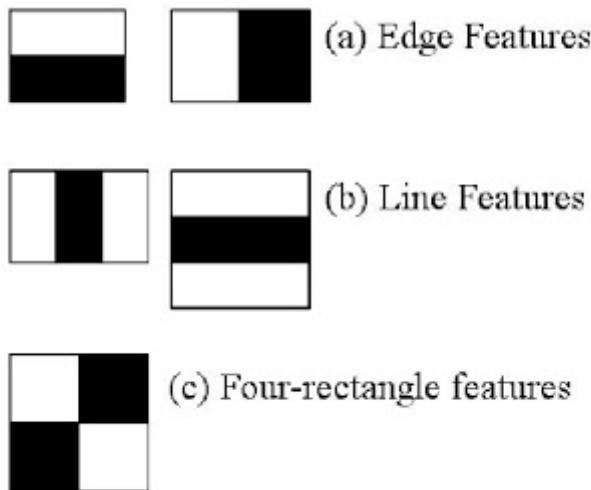
## 2.1. Metody wykrywania twarzy.

### 2.1.1. Wykrywanie twarzy i jej elementów przy użyciu klasyfikatora kaskadowego „Haar Feature-based Cascade Classifier”.

Wykrywanie obiektów za pomocą klasyfikatorów kaskadowych opartych na cechach Haara jest skuteczną metodą wykrywania obiektów zaproponowaną przez Paula Violer i Michaela Jonesa [1]. Algorytmy detekcji twarzy opierają się na wykrywaniu pewnych prostych wzorców geometrycznych o wysokim kontraście (ciemne i jasne plamy).

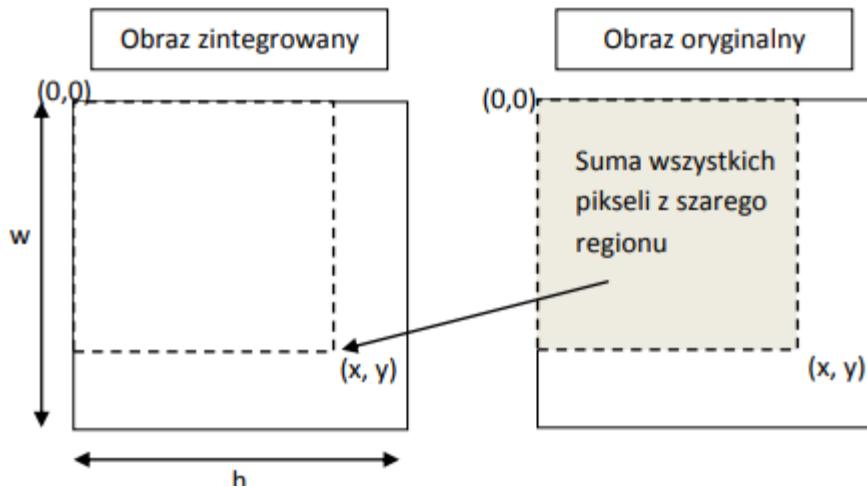
Klasyfikator jest uczony na podstawie przykładowych obiektów o znormalizowanych wymiarach - zwanych przykładami pozytywnymi - oraz przykładowych obrazów niezawierających poszukiwanych obiektów - zwanych przykładami negatywnymi. Klasyfikator, po zaprezentowaniu tysięcy obrazów pozytywnych i negatywnych odpowiednio buduje swoje drzewa decyzyjne wykrywające poszczególne wzorce podstawowe. Dobrze wytrenowany klasyfikator pozwala na wykrycie obecności nauczonego obiektu w obrazie.

Następnie z obrazu zostają wyodrębnione jego cechy. Każda cecha jest pojedynczą wartością uzyskaną przez odjęcie sumy pikseli pod białym prostokątem od sumy pikseli pod czarnym prostokątem. Są to cechy krawędziowe, liniowe i centralne.



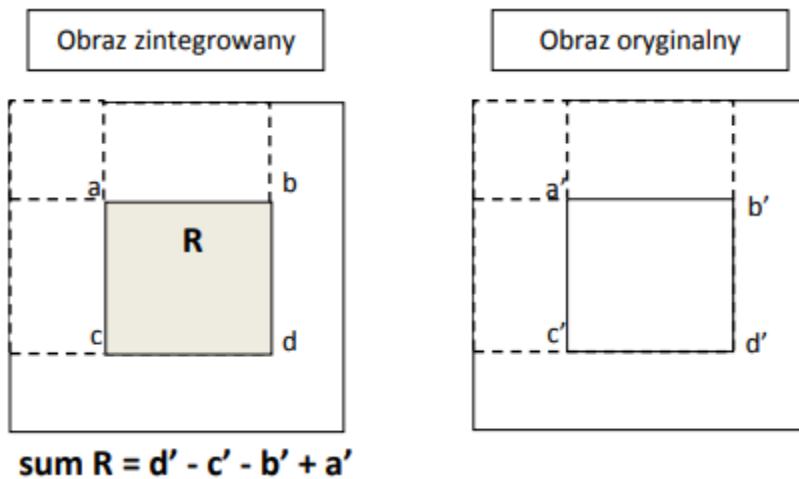
Rysunek 2. Wyodrębnianie cech przez algorytm Haara.

Obliczanie różnic prostokątnych obszarów na obrazie jest bardzo czasochłonne, dlatego algorytm detekcji obiektów wykorzystuje tzw. obraz zintegrowany (ang. integral image). Jest on reprezentacją obrazu, która w punkcie o współrzędnych  $(x, y)$  przechowuje wartość sumy wartości pikseli znajdujących się na lewo i powyżej punktu  $(x, y)$ .



Rysunek 3. Obliczanie obrazu zintegrowanego z oryginalnego. Źródło: [3]

Wykorzystując obraz zintegrowany sumę każdego obszaru można policzyć za pomocą trzech operacji.



Rysunek 4. Obliczanie sumy pikseli regionu  $R$  obrazu źródłowego za pomocą czterech punktów obrazu zintegrowanego. Źródło: [3]

Stworzenie własnego klasyfikatora Haara do wykorzystania go przy detekcji twarzy wymaga zgromadzenia obszernej bazy pozytywnych i negatywnych zdjęć, co pociągałoby za sobą spory nakład czasowy oraz obliczeniowy. Dlatego w projekcie został wykorzystany klasyfikator stworzony przez twórców biblioteki.

### **2.1.2 Wykrywanie twarzy i jej elementów przy użyciu „Deep Learning based Face Detector”**

Algorytm jest oparty na detektorze Single Shot-Multibox i wykorzystuje architekturę ResNet-10 jako szkielet. Model został przeszkolony za pomocą obrazów dostępnych w sieci. OpenCV zapewnia dwa modele tego detektora twarzy.

- Wersja zmiennoprzecinkowa 16 oryginalnej implementacji caffe
- 8-bitowa skwantowana wersja wykorzystująca Tensorflow

Single Shot-Multibox wykrywa obiekty na obrazach przy użyciu pojedynczej głębokiej sieci neuronowej. SSD dyskretyzuje przestrzeń wyjściową ramek ograniczających do zestawu domyślnych ramek w różnych proporcjach i skalach według lokalizacji mapy obiektu. W czasie przewidywania sieć generuje wyniki dla obecności każdej kategorii obiektów w każdym polu domyślnym i generuje korekty w polu, aby lepiej dopasować kształt obiektu. Ponadto sieć łączy przewidywania z wielu map obiektów z różnymi rozdzielczościami, aby w naturalny sposób obsługiwać obiekty o różnych rozmiarach.

Model SSD jest prosty w porównaniu z metodami, które wymagają propozycji obiektów, ponieważ całkowicie eliminuje generowanie propozycji, a następnie etap ponownego próbkowania pikseli lub funkcji i obejmuje wszystkie obliczenia w jednej sieci.

## 2.2 Metody rozpoznawania twarzy.

### 2.2.1 Rozpoznawanie twarzy metodą EigenFaceRecognizer (PCA)

Metoda ta polega na projekcji wielowymiarowego wektora (obrazu) na nowy układ współrzędnych, który tworzą pewne charakterystyczne obrazy, obrazy – Eigenfaces. Główną ideą jest to, aby przedstawić obraz jako liniową kombinację bazowych obrazów, które współczynniki określają dany obraz.

Obraz można widzieć jako punkt w wielowymiarowej przestrzeni. Mając pewien zbiór obrazów można wyznaczyć macierz korelacji między tymi wektorami i obliczyć wektory własne tej macierzy, które są bazowymi obrazami tzw. Eigenfaces.

Etapy działania:

1. Przygotować zbiór uczący składający się z obrazów o tych samych wymiarach/rozdzielczości (szerokość i wysokość), w których oczy i usta osób znajdują się na podobnej wysokości.
2. Poddać obrazy operacji wyrównywania histogramów (histogram equalization) w celu redukcji wpływu różnicy oświetlenia.
3. Obliczyć wartości pikseli dla średniego obrazu i odjąć ten obraz od każdego obrazu ze zbioru uczącego.
4. Utworzyć macierz zawierającą wszystkie obrazy (każdy obraz to jeden wektor).
5. Obliczyć wektory własne powyższej macierzy.
6. Obliczyć PCA na podstawie wektorów własnych (jeśli decydujemy się na pominięcie niektórych wektorów to należy odrzucić te z najmniejszymi wartościami własnymi).
7. Wykonać projekcję z wykorzystaniem PCA dla każdego obrazu ze zbioru uczącego.
8. Projekcje nowych zdjęć porównywać z projekcjami obiektów już rozpoznanych i wyszukiwać wyników najbliższych.

```
//create algorithm eigenface recognizer
Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();
//train data
model->train(images, labels);

model->save("E:/FDB/yaml/eigenface.yml");

cout << "Training finished...." << endl;
```

### 2.2.2 Rozpoznawanie twarzy metodą FisherFaceRecognizer (LDA)

Algorytm Fisherfaces (**Algorytm analizy dyskryminacyjnej**) opiera się na liniowej analizie dyskryminacyjnej (LDA) i polega na wyznaczeniu wektora cech pozwalającego na rozdzielenie obiektów przynależnych do różnych klas. W przypadku problemu rozpoznawania twarzy, przez klasy obiektów rozumiane są zbiory obrazów przedstawiające tego samego użytkownika. Problem sprowadza się do wyznaczenia wektora, który stanowi przybliżoną granicę między dwiema klasami obiektów, jednak można go uogólnić do problemu wieloklasowego.

```

    Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();

    model->train(images, labels);

    int height = images[0].rows;

    model->save("E:/FDB/yaml/fisherface.yml");

```

### 2.2.3 Rozpoznawanie twarzy metodą LBPHFaceRecognizer

W odróżnieniu od holistycznego podejścia w dwóch poprzednich przypadkach, algorytm LBPH wykorzystuje lokalne cechy przetwarzanych obiektów. Dla każdego piksela wyznaczany jest ciąg binarny na podstawie porównania wartości, z każdym z sąsiadów. W przypadku, gdy jego wartość jest większa wtedy przyjmuje wartość 1, a 0 w przypadku przeciwnym. Stąd dla każdego piksela wyznaczana jest wartość  $p$ -znakowego binarnego ciągu, nazywanego lokalnym binarnym wzorcem. Wyznaczanie można przeprowadzić w otoczeniu o dowolnym promieniu.

```

//lbph face recognier model
Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();

//training images with relevant labels
model->train(images, labels);

//save the data in yaml file
model->save("E:/FDB/yaml/LBPHface.yml");

cout << "training finished...." << endl;

```

## 2.3 Wybór rozwiązania

Wybrane przez nas rozwiązanie to zaimplementowanie wykrywania twarzy – przy pomocy klasyfikatora kaskadowego Haara oraz modelu bazującym na sieci neuronowej typu Tensorflow i porównanie ich skuteczności.

Do rozpoznawania twarzy wybrałyśmy trzy metody z modelami do rozpoznawania twarzy - EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer – ocenimy ich skuteczność względem siebie, poprawność rozpoznawania twarzy oraz czasu uczenia modelu dla różnych rozdzielczości zdjęć w bazach uczących, dla rozmiaru wyjściowego 92x112 pikseli oraz dla odpowiednio dwukrotnie większej rozdzielczości zdjęć 184x224 pikseli i dwukrotnie mniejszych rozmiarów 46x56 pikseli.

### **3 Założenia projektu**

Założeniem projektu było stworzenie algorytmu do rozpoznawania twarzy z kamery na podstawie wytrenowanych przez nas sieci neuronowych oraz opracowanie zestawienia możliwości poszczególnych metod detekcji oraz rozpoznawania twarzy.

Etapy realizacji projektu:

1. Program wykrywający twarze oraz oczy przy pomocy udostępnionego przez twórców biblioteki OpenCV klasyfikatora Haara.
  - 1.1 Pobranie i zainstalowanie biblioteki OpenCV.
  - 1.2 Stworzenie programu obsługującego obraz z kamery.
  - 1.3 Zaimplementowanie klasyfikatora Haara w programie.
2. Program wykrywający twarze przy pomocy Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow.
  - 2.1 Pobranie plików potrzebnych do załadowania detektora twarzy.
  - 2.2 Zaimplementowanie detektora w programie.
3. Program do wykrywania – przy pomocy klasyfikatora Haara lub Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow oraz rozpoznawania twarzy - przy pomocy algorytmów EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer nauczonych wykorzystując ręczne ładowanie niewielkiej (70 elementów – 7 osób) bazy obrazów twarzy, w której znalazły się również twarze członków zespołu.
  - 3.1 Pobranie z internetu zbioru obrazów twarzy.
  - 3.2 Dodanie do zbioru 20 zdjęć twarzy członków zespołu.
  - 3.2 Stworzenie pliku .csv zawierającego nazwy obrazów twarzy oraz przypisane im etykiety.
  - 3.3 Nauczenie modeli detektorów elementów bazy twarzy.
  - 3.4 Zaimplementowanie detektorów twarzy w programie.
4. Program do wykrywania – przy pomocy klasyfikatora Haara lub Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow oraz rozpoznawania twarzy - przy pomocy algorytmów EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer nauczonych wykorzystując automatyczne ładowanie niewielkiej (50 elementów) bazy obrazów twarzy.
  - 4.1 Zaimplementowanie automatycznego załadowywania elementów bazy obrazów twarzy przy pomocy funkcji readcsv.

5. Program do wykrywania – przy pomocy klasyfikatora Haara lub Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow oraz rozpoznawania twarzy - przy pomocy algorytmów EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer nauczonych wykorzystując automatyczne ładowanie bazy obrazów twarzy (900 elementów – 14 osób).
  - 5.1 Pobranie z internetu zbioru obrazów osób.
  - 5.2 Stworzenie funkcji do automatycznego wycinania i zapisywania twarzy ze zdjęć.
  - 5.3 Stworzenie nowej bazy obrazów twarzy wykorzystując wyekstraktowane zdjęcia twarzy.
  - 5.4 Nauczenie modelu nowej bazy twarzy.
  - 5.5 Eksperymentalne dobieranie wartości progu dla wykrywania twarzy oraz detekcji twarzy, aby algorytmy działały jak najdokładniej.
6. Opracowanie zestawienia możliwości poszczególnych metod detekcji oraz rozpoznawania twarzy.

Lista i podział zadań:

- Tworzenie programu- Magda Koprowska, Nadia Kerrouche
- Stworzenie bazy uczącej- Magda Koprowska
- Sprawozdanie- Magda Koprowska, Nadia Kerrouche

Harmonogram:

- 19.06.2019 – start projektu
- 19.06.2019 - 25.06.2019 – tworzenie programu
- 25.06.2019 – 28.06.2019 – tworzenie zestawienia
- 30.06.2019 – oddanie projektu

Narzędzia służące do współpracy podczas realizacji projektu:

Do realizacji projektu nie były wykorzystywane żadne narzędzia do współpracy członków zespołu.

Opis ryzyk i planów awaryjnych:

Wykonanie testów dla różnych rozdzielczości obrazów i sprawdzenie jak szybko sieć się uczy, w razie, gdy sieć nie będzie rozpoznawała twarzy w poprawny sposób dla żadnej z wybranych metod. Dodatkowym testem będzie stworzenie macierzy, w której umieszczone zostaną etykiety do jakich dana twarz została przypisana(rozpoznana).

## 4 Realizacja projektu

### 4.2 Szczegółowy opis realizacji

1. Program wykrywający twarze oraz oczy przy pomocy udostępnionego przez twórców biblioteki OpenCV klasyfikatora Haara.

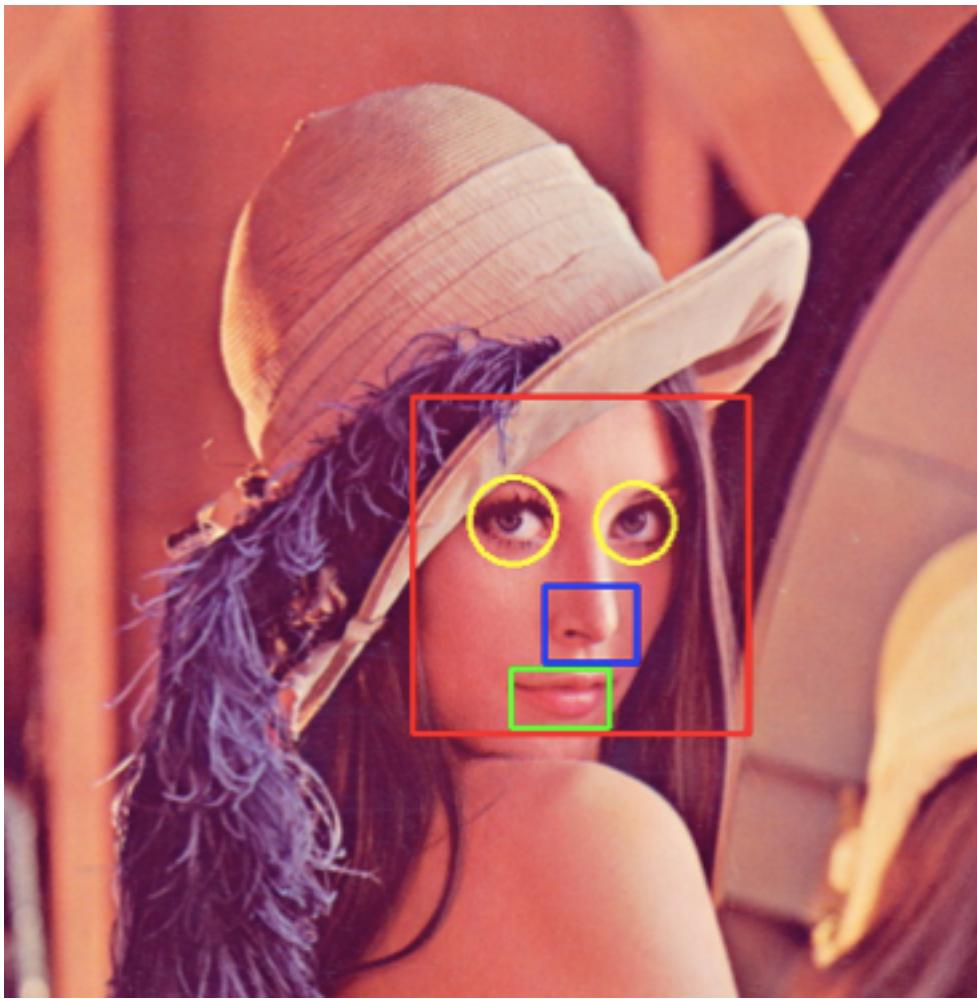
```
using namespace cv;
using namespace std;

string face_cascade_name = "haarcascade_frontalface_alt.xml";
string eye_cascade_name="haarcascade_eye.xml";
int main(){
    CascadeClassifier face_cascade, eye_cascade;
    if(!face_cascade.load(face_cascade_name)) {
        printf("Error loading cascade file for face");
        return 1;
    }
    if(!eye_cascade.load(eye_cascade_name)) {
        printf("Error loading cascade file for eye");
        return 1;
    }
    VideoCapture capture(0);

    if(!capture.isOpened())
    {
        printf("error to initialize camera");
        return 1;
    }
    Mat cap_img,gray_img;
    vector<Rect> faces, eyes;

    while(1)
    {
        capture >> cap_img;
        waitKey(10);
        cvtColor(cap_img, gray_img, COLOR_BGR2GRAY);
        cv::equalizeHist(gray_img,gray_img);
        face_cascade.detectMultiScale(gray_img, faces, 1.1, 10, CASCADE_SCALE_IMAGE | CASCADE_DO_CANNY_PRUNING, Size(0,0), Size(300,300));
        for(int i=0; i < faces.size();i++)
        {
            Point pt1(faces[i].x+faces[i].width, faces[i].y+faces[i].height);
            Point pt2(faces[i].x,faces[i].y);
            Mat faceROI = gray_img(faces[i]);
            eye_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 | CASCADE_SCALE_IMAGE,
Size(30,30));
            for(size_t j=0; j< eyes.size(); j++)
            {
                Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y +
eyes[j].y + eyes[j].height*0.5 );
                int radius= cvRound((eyes[j].width+eyes[j].height)*0.25);
                circle(cap_img, center, radius, Scalar(255,0,0), 2, 8, 0);
            }
            rectangle(cap_img, pt1, pt2, Scalar(0,255,0), 2, 8, 0);
        }
        imshow("Result", cap_img);
        waitKey(3);
        char c = waitKey(3);
        if(c == 27)
            break;
    }
    return 0;
}
```

Rysunek 5. Kod źródłowy programu.



Rysunek 6. wykrycie twarzy, oczu, ust oraz nosa.

Jak można zauważyć na przykładzie powyższego obrazka oraz kodu, klasyfikatory Haara pozwalają nam nie tylko na detekcję twarzy, ale za pomocą jednej linijki kodu jesteśmy w stanie na wykrytej twarzy, wykryć także oczy czy też usta, zależnie od tego co chcemy otrzymać.

- Załadowanie i uruchomienie klasyfikatora

Pierwszym krokiem jest załadowanie gotowego klasyfikatora Haara do wykrywania twarzy. Aby tego dokonać należy przenieść plik haarcascade\_frontalface\_alt.xml znajdujący się standardowo pod adresem: C:\opencv\sources\data\haarcascades do folderu, w którym znajduje się program (w moim przypadku: C:\Users\ja\source\repos\Projekt\Projekt).

Następnie należy go załadować i uruchomić, co można zrealizować za pomocą dwóch linijek.

```
string face_cascade_name = "haarcascade_frontalface_alt.xml";
```

Rysunek 7. Załadowanie klasyfikatora Haara do detekcji frontu twarzy.

```
CascadeClassifier face_cascade;
```

Rysunek 8. Uruchomienie klasyfikatora Haara do detekcji frontu twarzy.

2. Program wykrywający twarze przy pomocy Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow.

- Pobranie plików potrzebnych do załadowania detektora twarzy.

Na początku należy ściągnąć pliki do załadowania detektorów z GitHuba.

Następnie przenieść je do folderu, w którym znajduje się program.

- Zaimplementowanie detektora w programie.

```
Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector.pbtxt", "opencv_face_detector_uint8.pb");
```

Rysunek 9. Stworzenie i załadowanie modelu Tensorflow - Deep Learning Face Detector.

3. Program do wykrywania – przy pomocy klasyfikatora *Haara* lub *Deep Learning based Face Detector* dla modelu wykorzystującego *TensorFlow* oraz rozpoznawania twarzy - przy pomocy algorytmów *EigenFaceRecognizer*, *FisherFaceRecognizer* oraz *LBPHFaceRecognizer* nauczonych wykorzystując ręczne ładowanie niewielkiej (70 elementów – 7 osób) bazy obrazów twarzy, w której znalazły się również twarze członków zespołu.

- Pobranie z internetu zbioru obrazów twarzy. [5]
- Dodanie do zbioru 20 zdjęć twarzy członków zespołu.



Rysunek 10. Przykładowe zdjęcia twarzy członków zespołu.

- Stworzenie pliku .csv zawierającego nazwy obrazów twarzy oraz przypisane im etykiety.

218	Nadia_005.jpg	2
219	Nadia_006.jpg	2
220	Nadia_007.jpg	2
221	Nadia_008.jpg	2
222	Nadia_009.jpg	2
223	Nadia_010.jpg	2
224	Nadia_011.jpg	2
225	Nadia_012.jpg	2
226	Nadia_013.jpg	2
227	Nadia_014.jpg	2
228	Nadia_015.jpg	2
229	Nadia_016.jpg	2
230	Nadia_017.jpg	2
231	Nadia_018.jpg	2
232	Nadia_019.jpg	2
233	Nadia_020.jpg	2
234	Nadia_021.jpg	2
235	Nadia_022.jpg	2
236	Nadia_023.jpg	2

Rysunek 11. Fragment pliku *face\_recognition\_video* będącego bazą twarzy otwartego w programie MS Excel.

- Nauczenie modeli detektorów elementów bazy twarzy.

```
//stworzenie detektora
Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
//lub
Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
//lub
Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

/*załadowanie obrazów i ich etykiet z pliku .csv do modelu przy pomocy funkcji
readcsv */
vector<Mat> images;
vector<int> labels;
string filename = "face_recognition_video.csv";

try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku \"" << filename << "\". Powod: " <<
e.msg << endl;

    exit(1);
}
//uczenie modelu
model->train(images, labels);
```

- Zaimplementowanie detektorów twarzy w programie.

```
//stworzenie detektora
Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
lub
Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
lub
Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

/* załadowanie obrazów i ich etykiet z pliku .csv do modelu przy pomocy funkcji
readcsv */

vector<Mat> images;
vector<int> labels;

string filename = "face_recognition_video.csv";

try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku \"" << filename << "\". Powod:
" << e.msg << endl;

    exit(1);
}

//uczenie modelu
model->train(images, labels);
```

4. Program do wykrywania – przy pomocy klasyfikatora Haara lub Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow oraz rozpoznawania twarzy - przy pomocy modeli EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHRecognizer nauczonych wykorzystując automatyczne ładowanie niewielkiej (50 elementów) bazy obrazów twarzy.
  - Zaimplementowanie automatycznego załadowywania elementów bazy obrazów twarzy przy pomocy funkcji readcsv.

```

void read_csv(const string & filename, vector<Mat> & images, vector<int> &
labels)
{
    ifstream file(filename.c_str(), ifstream::in);
    if (!file)
    {
        cout << "Otwarcie pliku " << filename << " nie powiodlo sie.";
        return;
    }
    string line, path, classlabel;
    while (getline(file, line))
    {
        stringstream liness(line);
        getline(liness, path, ';');
        getline(liness, classlabel);
        //classlabel.erase(0, 1);
        //path.erase(path.length() - 1, path.length());
        if (!path.empty() && !classlabel.empty())
        {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
            cout << path << "=" << atoi(classlabel.c_str()) << endl;
        }
    }
}

```

Rysunek 9. Kod funkcji readcsv.

```

vector<Mat> images;
vector<int> labels;

string filename = "face_recognition_video.csv"; //nazwa pliku csv

try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku \\" << filename << "\\. Powod:
" << e.msg << endl;
    exit(1);
}

```

*Implementacja funkcji readcsv w programie*

332	George_Bush_twarz_0043.jpg	3
333	George_Bush_twarz_0041.jpg	3
334	George_Bush_twarz_0039.jpg	3
335	George_Bush_twarz_0037.jpg	3
336	George_Bush_twarz_0035.jpg	3
337	George_Bush_twarz_0033.jpg	3
338	George_Bush_twarz_0031.jpg	3
339	George_Bush_twarz_0029.jpg	3
340	George_Bush_twarz_0027.jpg	3
341	George_Bush_twarz_0025.jpg	3
342	George_Bush_twarz_0023.jpg	3
343	George_Bush_twarz_0021.jpg	3
344	George_Bush_twarz_0019.jpg	3
345	George_Bush_twarz_0017.jpg	3
346	George_Bush_twarz_0015.jpg	3
347	George_Bush_twarz_0013.jpg	3
348	George_Bush_twarz_0011.jpg	3

Rysunek 12. Wycinek bazy twarzy – pliku *face\_recognition\_video.csv* otwartego w programie MS Excel

5. Program do wykrywania – przy pomocy klasyfikatora Haara lub Deep Learning based Face Detector dla modelu wykorzystującego TensorFlow oraz rozpoznawania twarzy - przy pomocy algorytmów EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer nauczonych wykorzystując automatyczne ładowanie bazy obrazów twarzy (900 elementów – 14 osób).

- Pobranie z Internetu zbioru obrazów osób. [6]

Link do zbioru: <http://vis-www.cs.umass.edu/lfw/#deepfunnel-anchor>

- Stworzenie funkcji do automatycznego wycinania i zapisywania twarzy ze zdjęć.

Funkcja realizująca to zadanie jest przedstawiona na poniższym fragmencie kodu.

```

int rozpoznawanie_twarzy_obrazy_ladowanie() {

    // stworzenie i załadowanie parametrów detektora twarzy
    Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
                                              "opencv_face_detector.pbtxt");

    Mat testSample = imread("Magdusia_042.jpg", 0);
    int img_width = testSample.cols;
    int img_height = testSample.rows;
    int ilosc_zdjec = 48;
    int nr_fotki = 24;

    for(int i = 0; i < ilosc_zdjec; i++)
    {
        ++nr_fotki;
        Mat obraz = imread(format("Luiz_Inacio_Lula_da_Silva_%04d.jpg", nr_fotki), 1);

        if (obraz.cols == 0) {
            cout << "Error reading file " << endl;
            return -1;
        }

        Mat obraz_przeskalowany;

        try {
            resize(obraz, obraz_przeskalowany, Size(img_width, img_height), 1.0, 1.0, 1);
        }
        catch (cv::Exception& e) {
            cerr << e.msg << endl;

            exit(1);
        }

        //przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej
        //twarze

        Mat blob = cv::dnn::blobFromImage(obraz_przeskalowany, 1.0, Size(img_width, img_height),
                                           Scalar(104.0, 177.0, 123.0), true, false);

        // uruchomienie głębszej sieci neuronowej wykrywającej twarze
        net.setInput(blob, "data");
        Mat detekcje = net.forward("detection_out");
        // przeformatowanie wyniku: tyle wierszy ile twarzy,
        // w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt -
        // twarz, więc tam zawsze jest 1),
        // pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków
        // prostokąta zawierającego obiekt (x1, y1, x2, y2)
        Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F,
                               detekcje.ptr<float>());

        vector<Rect> twarze;
        for (int i = 0; i < macierz_detekcji.rows; i++)
        {
            if (macierz_detekcji.at<float>(i, 2) > 0.6) //pewność wykrycia (w skali 0-1)
            {
                // odczyt punktów tworzących prostokąt wokół twarzy
                int x1 = (macierz_detekcji.at<float>(i, 3) * obraz.cols);
                int y1 = (macierz_detekcji.at<float>(i, 4) * obraz.rows);
                int x2 = (macierz_detekcji.at<float>(i, 5) * obraz.cols);
                int y2 = (macierz_detekcji.at<float>(i, 6) * obraz.rows);

                // zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze (czyli
                // właśnie prostokąty otaczające twarze)
                twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));
                //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni
                //element wektora twarze: twarze.back()

                Mat przeskalowana_twarz;
                Mat przeskalowana_twarz_szara;

                //funkcja rysująca prostokąt na ekranie
                rectangle(obraz, twarze.back(), CV_RGB(0, 255, 100), 1, LINE_8, 0);
                //resize(twarze, przeskalowana_twarz, Size(img_width, img_height), 1.0, 1.0, 1);

                resize(obraz(twarze.back()), przeskalowana_twarz, Size(img_width, img_height), 1.0, 1.0, 1);

                imshow("przeskalowana_twarz", przeskalowana_twarz);
                cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara, COLOR_BGR2GRAY);
                //imwrite(format("Arnold_Schwarzenegger_twarz_%04d.jpg", nr_fotki++), przeskalowana_twarz);
                //imwrite(format("David_Beckham_twarz_%04d.jpg", nr_fotki++), przeskalowana_twarz);
                imwrite(format("Luiz_Inacio_Lula_da_Silva_twarz_%04d.jpg", nr_fotki), przeskalowana_twarz);
            }
        }
        imshow("test", obraz);
    }
    return 0;
}

```

- Stworzenie nowej bazy obrazów twarzy wykorzystując wyekstraktowane zdjęcia twarzy.
- Nauczenie modelu nowej bazy twarzy.

```
model->train(images, labels);
```

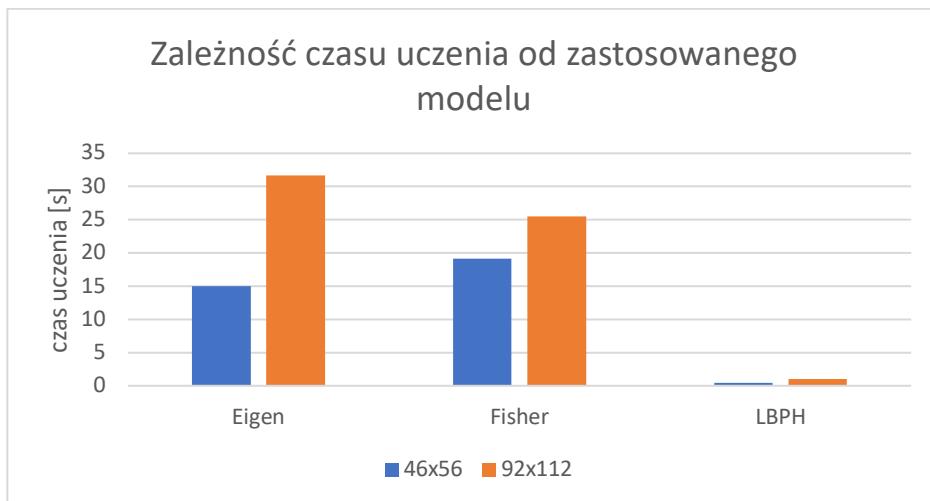
- Eksperymentalne dobieranie wartości progu dla wykrywania twarzy oraz detekcji twarzy, aby algorytmy działały jak najdokładniej.

```
model->setThreshold(150);
double current_threshold2 = model->getThreshold();
```

6. Opracowanie zestawienia możliwości poszczególnych metod detekcji oraz rozpoznawania twarzy.

- Czas uczenia modelu  
(dla procesora Intel® Core™ i5-6200 CPU 2,4 GHz przy podłączonym zasilaniu).
  - EigenFaceRezognizer
    - Dla rozdzielczości 92x112 pikseli [600 elementów]:  
~31,67 s (średnia arytmetyczna trzech prób)
    - Dla rozdzielczości 46x57 pikseli [600 elementów]:  
~14,97s (średnia arytmetyczna trzech prób),
    - Dla rozdzielczości 200x200 pikseli:  
Nie udało się przeprowadzić testów dla tej rozdzielczości.
  - FisherFaceRezognizer
    - Dla rozdzielczości 92x112 pikseli [600 elementów]:  
~25,5 s (średnia arytmetyczna trzech prób),
    - Dla rozdzielczości 46x57 pikseli [600 elementów]:  
~19,14 s (średnia arytmetyczna trzech prób),
    - Dla rozdzielczości 200x200 pikseli:  
Nie udało się przeprowadzić testów dla tej rozdzielczości.
  - LBPHFaceRezognizer
    - Dla rozdzielczości 92x112 pikseli:  
~1,02s (średnia arytmetyczna trzech prób),
    - Dla rozdzielczości 46x57 pikseli [600 elementów]:  
~0,4s (średnia arytmetyczna trzech prób),
    - Dla rozdzielczości 200x200 pikseli:  
Nie udało się przeprowadzić testów dla tej rozdzielczości.

Zebrane wyniki przedstawiłyśmy na poniższym wykresie:



Kolejnym testem było sprawdzenie czasu uczenia sieci (Tensorflow) dla trzech modeli rozpoznawania twarzy w przypadku malej bazy danych (54 elementy) i dużej bazy danych (900 elementów).

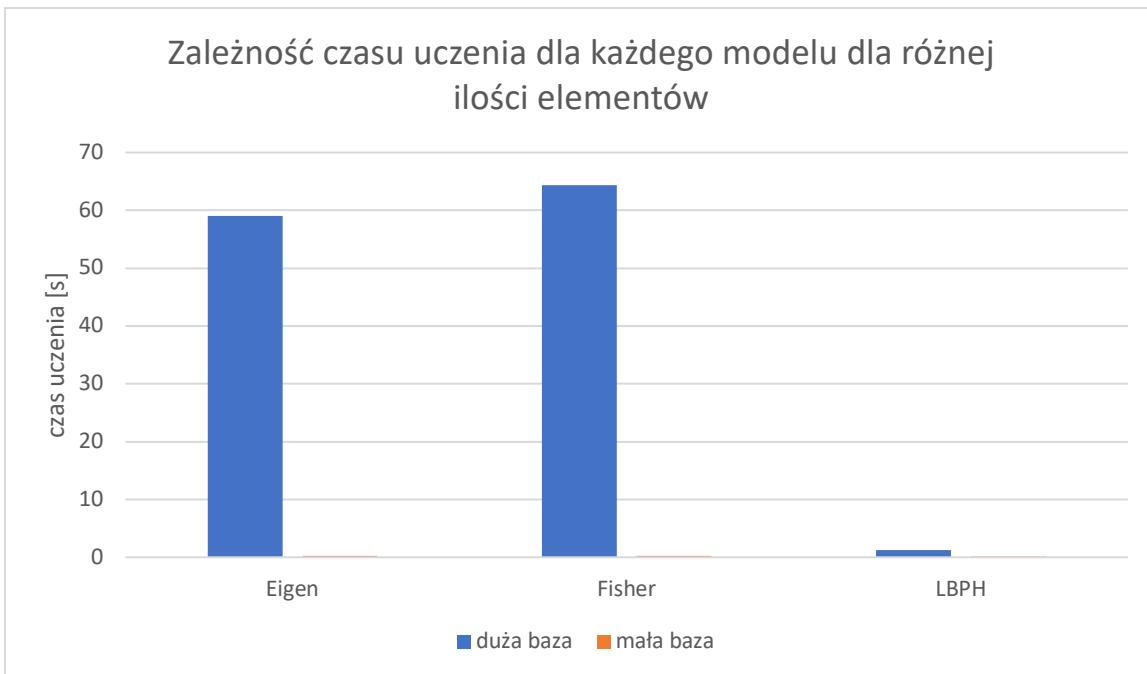
Dla małej bazy:

- EigenFaceRezognizer  
~0,193s (średnia arytmetyczna z trzech prób).
- FisherFaceRezognizer  
~0,201s (średnia arytmetyczna z trzech prób).
- LBPHFaceRezognizer  
~0,146s (średnia arytmetyczna z trzech prób).

Dla dużej bazy:

- EigenFaceRezognizer  
~59,07s
- FisherFaceRezognizer  
~64,34s
- LBPHFaceRezognizer  
~1,24s

Wyniki zostały przedstawione na poniższym wykresie:



Trzecim testem było przetestowanie poprawności działania funkcji rozpoznającej twarz z wgranego zdjęcia, które nie znalazło się w bazie uczącej. Szesnaście prób wykonanych dla czternastu osób w bazie oraz jednej twarzy, która nie znajdowała się w bazie, a także obraz, na którym nie znajduje się żadna twarz – tzw. nie-twarz.

Sprawdzane zostały wszystkie trzy metody tj. EigenFaceRecognizer, FisherFaceRecognizer oraz LBPHFaceRecognizer dla dwóch rozdzielczości obrazów w bazie uczącej 46x56 pikseli oraz 92x112 pikseli oraz ustawionym thresholdzie równym 200.

Ze względu na chęć ograniczenia złożoności testu, zmniejszyliśmy liczbę próbek z czternastu do pięciu.



Rysunek 14. Twarz nie znajdująca się w bazie.

- EigenFace Recognizer
  - Rozdzielcość 92x112 pikseli

Numer osoby z bazy

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	10	1	0	2	0	0	1	0	0	0	0	0	0	0
4	0	0	0	9	2	0	0	0	1	0	1	1	0	0	0	0
5	0	0	0	0	13	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	1	11	0	1	0	0	0	0	0	1	0	0
7	0	0	0	0	1	0	13	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	11	0	0	0	1	0	1	0	0
9	0	0	0	0	2	0	0	0	11	0	0	0	1	0	0	0
10	0	0	0	0	2	0	0	0	0	12	0	0	0	0	0	0
11	0	0	0	0	0	1	0	0	0	0	10	2	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	13	0	1	0	0
13	0	0	0	0	0	0	0	1	1	0	0	0	12	0	0	0
14	0	0	0	0	2	0	0	0	0	2	0	0	0	10	0	0
15 **	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16 ***	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14

\*na wgrywanym zdjęciu

\*\* - oznacza twarz spoza bazy

\*\*\* - oznacza nie-twarz

$$TP = 163, FP = 36, FN = 14, TN = 14$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{163}{177} = 92\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{163}{199} = 81\%$$

$$F1=2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{0,81 \cdot 0,92}{0,81 + 0,92} = 86\%$$

o Rozdzielcość 46x56 pikseli

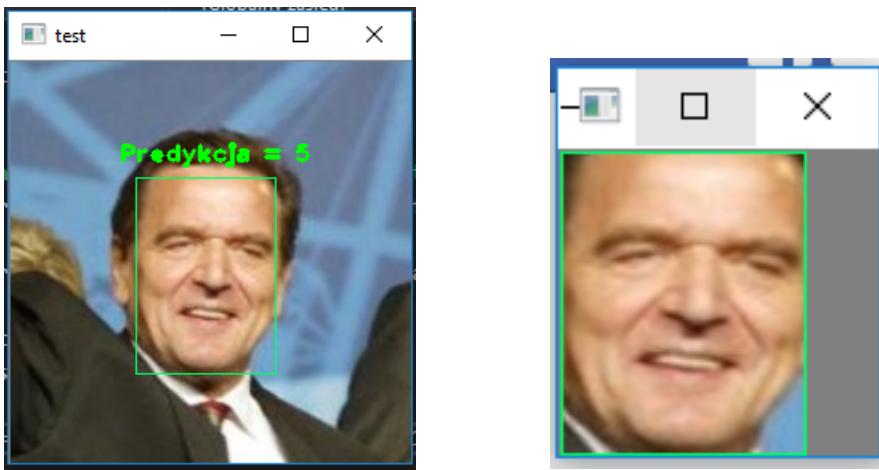
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	3	0	0	0	1	0	0	0	0	1	0	0	0
5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	4	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	3	0	0	1	0	0	0	1	0	0
8	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	4	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
11	0	0	0	0	1	0	0	0	0	0	3	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
14	0	1	0	0	0	0	0	1	0	0	0	0	0	3	0	0
15	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

$$TP = 59, FP = 11, FN = 5, TN = 5$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{59}{64} = 92,19\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{59}{70} = 84,29\%$$

$$F1=2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{0,9219 \cdot 0,8429}{0,9219 + 0,8429} = 2 \cdot 0,44 = 88\%$$



Rysunek 15. Przykładowa poprawna rozpoznanie twarz.

Wynikiem testu jest macierz, w której kolumny pokazują wynik każdej próby, a wiersze kolejne osoby z bazy. Łatwo zauważyc, że na przekątnej macierzy znajdują się największe liczby, co pokazuje, ile razy twarz została poprawnie rozpoznana, bądź przypisana błędnie. Jak widać dwie pierwsze twarze zostały rozpoznanie bezbłędnie, za każdym razem, ponieważ w bazie znajdowało się odpowiednio 319 oraz 105 zdjęć. Ponadto zdjęcia były kolejnymi klatkami z filmu przy niezbyt dobrym oświetleniu w odróżnieniu od innych twarzy z bazy. Dla twarzy, których ilość zdjęć w bazie uczącej jest bliska pięćdziesięciu i są one wykonane w dobrym świetle, sieć dokonuje złego przypisania średnio od jednego do dwóch razy. Największe rozbieżności widoczne są dla zdjęć, których w bazie jest stosunkowo mało, w porównaniu do reszty (od 18 do 28), jednak sieć i tak najczęściej razy rozpoznała twarz poprawnie. Dla mniejszej rozdzielczości macierz pomyłek prezentuje analogiczną zależność. Metoda ta jednak, pomimo ustawionego progu nie radzi sobie z przypisaniem wartości -1 dla twarzy, których nie ma w bazie. Zauważylamy, że przypisuje jej etykietę osoby, której zdjęć w bazie jest najwięcej (319) i są one wykonane w różnych warunkach oświetleniowych.

- FisherFaceRecognizer
  - Rozdzielcość 92x112 pikseli

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	4	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
11	0	0	0	0	1	0	0	0	0	0	4	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

$$TP = 66, FP = 4, FN = 0, TN = 10$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{66}{66} = 100\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{66}{70} = 94,29\%$$

$$F1 = 2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{1 \cdot 0,9429}{1 + 0,9429} = 97\%$$

- Rozdzielcość 46x56 pikseli

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
11	1	0	0	0	1	0	0	0	0	0	3	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

$$TP = 64, FP = 6, FN = 0, TN = 10$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{64}{64} = 100\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{64}{70} = 91,43\%$$

$$F1 = 2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{1 \cdot 0,9143}{1 + 0,9143} = 95,52\%$$

Możemy stwierdzić, że metoda FisherFaceRecognizer jest najskuteczniejsza, ze względu na największą sumę liczb na diagonali macierzy pomyłek. W porównaniu z innymi metodami ilość błędnych przypisań jest najmniejsza, przez co możemy wysunąć wniosek, iż metoda ta radzi sobie najlepiej spośród trzech będących obiektem testów. Dla mniejszej rozdzielcości obrazów uczących, liczba pomyłek nieznacznie wzrosła w porównaniu do większej rozdzielcości.

- LBPHFaceRecognizer

- Rozdzielcość 92x112 pikseli

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	3	0	1	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	3	1	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	3	1	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	4	0	0	0	1	0	0	0	0
9	1	0	0	0	0	0	0	0	4	0	0	0	1	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0	3	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
13	0	0	0	0	1	0	0	0	0	0	0	0	4	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

$$TP = 59, FP = 11, FN = 0, TN = 10$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{59}{59} = 100\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{59}{70} = 84,29\%$$

$$F1=2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{1 \cdot 0,8429}{1+0,8429} = 91,48\%$$

- Rozdzielcość 46x56 pikseli

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	3	0	1	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	3	1	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	3	1	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	4	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	4	0	0	0	1	0	0	0
10	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0	3	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
13	0	0	0	0	1	0	0	0	0	0	0	0	4	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

$$TP = 59, FP = 11, FN = 0, TN = 10$$

$$\text{Czułość (TPR)} = \frac{TP}{TP+FN} = \frac{59}{59} = 100\%$$

$$\text{Precyzja (PPV)} = \frac{TP}{TP+FP} = \frac{59}{70} = 84,29\%$$

$$F1 = 2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} = 2 \cdot \frac{1 \cdot 0,8429}{1 + 0,8429} = 91,48\%$$

Metoda ta radzi sobie nieznacznie gorzej niż FisherFaceRecognizer, lecz uwzględniając jej bardzo krótki czas uczenia (ok. 60-krotnie mniejszy niż dla FisherFaceRecognizer), wyrasta na metodę bardzo atrakcyjną.

### 4.3 Opis implementacyjny

- Funkcje
  - *rozpoznawanie\_twarzy\_kamera();*

Funkcja wykorzystująca obraz z kamery do rozpoznawania twarzy – przy pomocy detektora twarzy na bazie sieci neuronowej oraz rozpoznawania twarzy przy pomocy EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer.

```

int plabel;
double confidence;

model->predict(przeskalowana_twarz_szara, plabel, confidence);

cout << "etykieta " << plabel << endl << "odleglosc " << confidence << endl;

string box_text = format("Predykcja = %d", plabel);

int pos_x = max(twarze[i].x - 10, 0);
int pos_y = max(twarze[i].y - 10, 0);

putText(obraz_z_kamery, box_text, Point(pos_x, pos_y), FONT_HERSHEY_PLAIN,
1.0, CV_RGB(0, 255, 0), 2.0);

}

}

imshow("test", obraz_z_kamery);
}
}
}
```

- *rozpoznawanie\_twarzy\_kamera\_ladowanie();*

Funkcja wykorzystująca obraz z kamery do rozpoznawania twarzy – przy pomocy detektora twarzy na bazie sieci neuronowej oraz rozpoznawania twarzy przy pomocy

EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer i zapisującą wychwycone twarze jako pliki .jpg do folderu programu.

```
resize(obraz_z_kamery(twarze.back()), przeskalowana_twarz, Size(img_width,
img_height), 1.0, 1.0, 1);
imwrite(format("Magda_twarz_%03d.jpg", nr_fotki++),
przeskalowana_twarz);
cout << "zdjęcie: " << format("Magda_%03d.jpg", nr_fotki) << endl;

imshow("przeskalowana twarz", przeskalowana_twarz);
cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara,
COLOR_BGR2GRAY);

//imwrite(format("Magdusia_%03d.jpg", nr_fotki++),
przeskalowana_twarz);
//imwrite(format("Nadia_%03d.jpg", nr_fotki++), przeskalowana_twarz);

//int plabel = model->predict(przeskalowana_twarz);
int plabel = model->predict(przeskalowana_twarz_szara);

cout << plabel << endl;

string box_text = format("Predykcja = %d", plabel);
```

- *rozpoznawanie\_twarzy\_obrazy();*

Funkcja wczytująca obraz oraz znajdująca na nim twarz przy pomocy detektora twarzy na bazie sieci neuronowej oraz rozpoznająca twarz przy pomocy EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer.

```
model->train(images, labels);

//zapisywanie modelu do pliku yaml
string save_filename = "model.yaml";
model->save(save_filename);

string obraz_testowy_nazwa = "angelina_jolie.jpg";

Mat testSample = imread("magda_720_1280_bezokularow_wycięty_010.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

Mat obraz_testowy = imread(obraz_testowy_nazwa, 0);
Mat obraz_testowy_przeskalowany;

resize(obraz_testowy, obraz_testowy_przeskalowany, Size(img_width, img_height),
1.0, 1.0, 1);

if (!obraz_testowy.data)
{
    cout << "Nie odnaleziono " << obraz_testowy_nazwa << ".";
    return -2;
}

//rozpoznawanie twarzy i wyrzucanie numeru
int plabel = model->predict(obraz_testowy_przeskalowany);
cout << plabel;
```

- *rozpoznawanie\_twarzy\_obrazy\_ladowanie();*

Funkcja wczytująca obraz oraz znajdującą na nim twarz przy pomocy detektora twarzy na bazie sieci neuronowej oraz rozpoznającą twarz przy pomocy EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer i zapisującą wychwycone twarze jako pliki jpg do folderu programu.

```
for(int i = 0; i < ilosc_zdjec; i++)
{
    ++nr_fotki;
    Mat obraz = imread(format("Luiz_Inacio_Lula_da_Silva_%04d.jpg", nr_fotki), 1);

    if (obraz.cols == 0) {
        cout << "Error reading file " << endl;
        return -1;
    }
}
```

```
Mat blob = cv::blobFromImage(obraz_przeskalowany, 1.0, Size(img_width,
img_height), Scalar(104.0, 177.0, 123.0), true, false);

// uruchomienie głębszej sieci neuronowej wykrywającej twarze
net.setInput(blob, "data");
Mat detekcje = net.forward("detection_out");
```

- *rozpoznawanie\_twarzy\_video\_haar();*

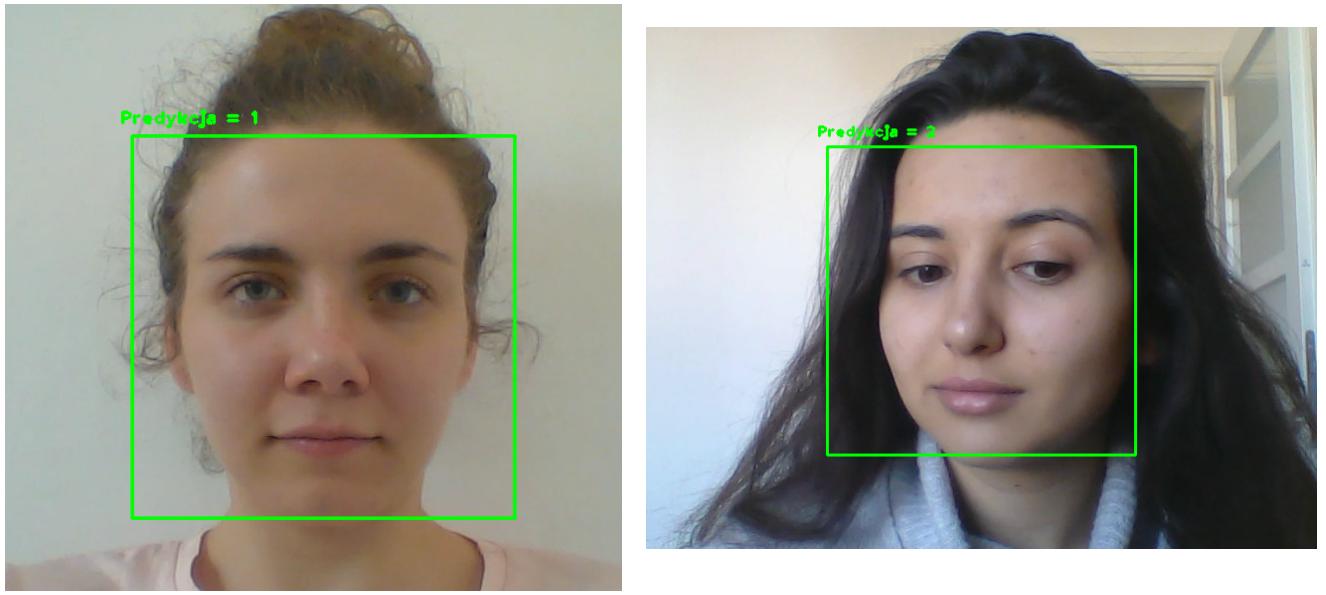
Funkcja wykorzystująca obraz z filmu do rozpoznawania twarzy – przy pomocy detektora twarzy na bazie klasyfikatora kaskadowego Haara oraz rozpoznawania twarzy przy pomocy EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer.

```
cvtColor(obraz_z_kamery, img_gray, COLOR_BGR2GRAY);           //Konwersja obrazu z kamery
do odcieni szarosci
    face_cascade.detectMultiScale(img_gray, faces, 1.1, 2, 0 |
CASCADE_SCALE_IMAGE, Size(img_width, img_height));
```

```
point pt1(faces[i].x + faces[i].width, faces[i].y + faces[i].height);      //kwadrat na
twarz
    Point pt2(faces[i].x, faces[i].y);
    rectangle(obraz_z_kamery, pt1, pt2, Scalar(0, 255, 0), 2, 8, 0);

    int plabel = model->predict(resized_gray);

    string box_text = format("Predykcja = %d", plabel);
```



Rysunek 13. Twarze rozpoznane metodą rozpoznawanie\_twarzy\_video\_haar().

- *rozpoznawanie\_twarzy\_video\_tensorflow();*

Funkcja wykorzystująca obraz z filmu do rozpoznawania twarzy – przy pomocy detektora twarzy na bazie sieci neuronowej oraz rozpoznawania twarzy przy pomocy EigenFaceRecognizer, FisherFaceRecognizer lub LBPHFaceRecognizer.

```
//przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej
//wykrywającej twarze
Mat blob = cv::dnn::blobFromImage(obraz_z_kamery, 1.0, Size(300, 300),
Scalar(104.0, 177.0, 123.0), true, false);
```

```
int plabel = model->predict(przeskalowana_twarz);
cout << plabel << endl;

string box_text = format("Predykcja = %d", plabel);
```

- *readcsv();*

Funkcja służąca do załadowania bazy obrazów twarzy poprzez odczytanie ich z pliku .csv.

```

/* załadowanie obrazów i ich etykiet z pliku .csv do modelu przy pomocy funkcji
readcsv */

    vector<Mat> images;
    vector<int> labels;

    string filename = "face_recognition_video.csv";

    try {
        read_csv(filename, images, labels);
    }
    catch (cv::Exception& e) {
        cerr << "Problem z otwarciem pliku \"" << filename << "\". Powod:
" << e.msg << endl;

        exit(1);
    }

```

- Źródła danych

- Baza obrazów twarzy – face\_recognition\_video.csv

Jest to plik .csv, w którym przechowywane są nazwy obrazów oraz przypisane im etykiety.

- model.yaml

Plik przechowujący wytrenowany model.

- opencv\_face\_detector\_uint8.pb
- opencv\_face\_detector.pbtxt
- haarcascade\_frontalface\_alt.xml

gotowy, wytrenowany już klasyfikator, rozpoznający twarz dostarczony wraz z biblioteką OpenCV.

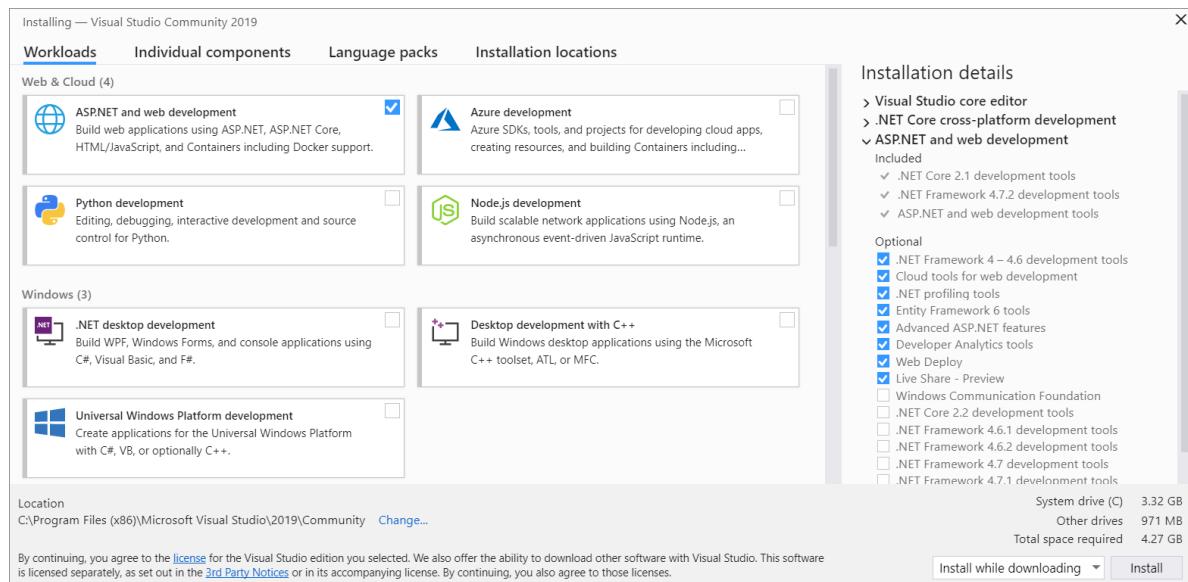
## 4.4 Opis uruchomieniowy

### 4.4.1 Instalacja środowiska

Pierwszym etapem projektu jest zainstalowanie środowiska Microsoft Visual Studio, w którym będzie można stworzyć program z wykorzystaniem biblioteki OpenCV.

W tym celu należy udać się na stronę: <https://visualstudio.microsoft.com/pl/downloads/> i pobrać interesującą nas wersję oprogramowania.

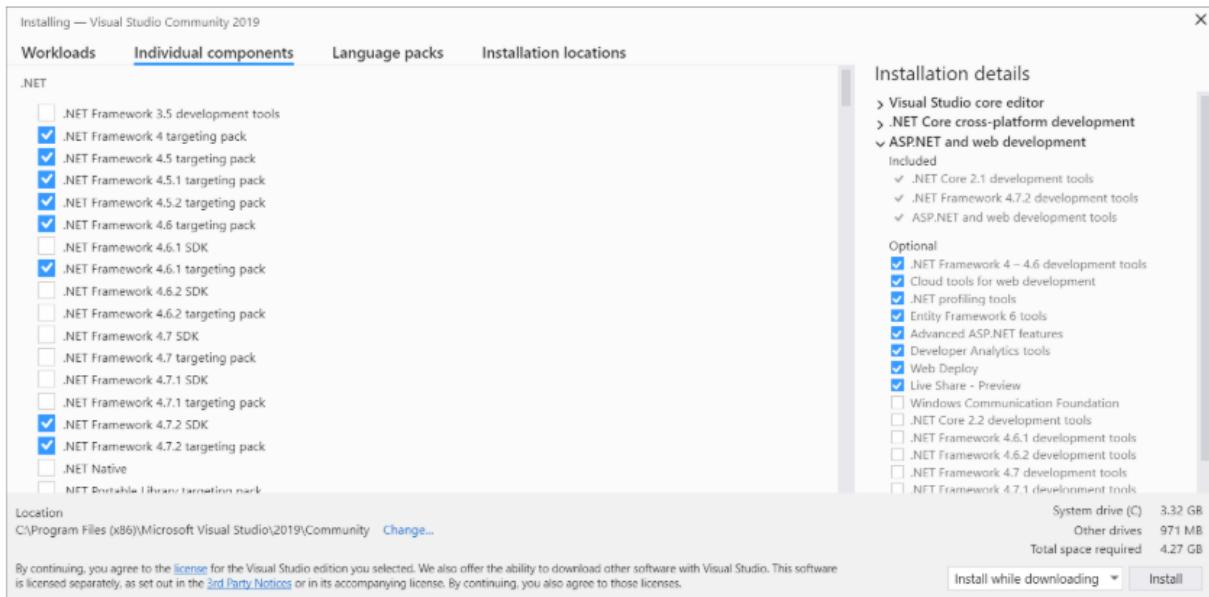
- Po zainstalowaniu instalatora należy wybrać **Workloads**, które będą nas interesowały.



Rysunek 14. Okienko instalacyjne Microsoft Visual Studio - Workloads. Źródło: docs.microsoft.com

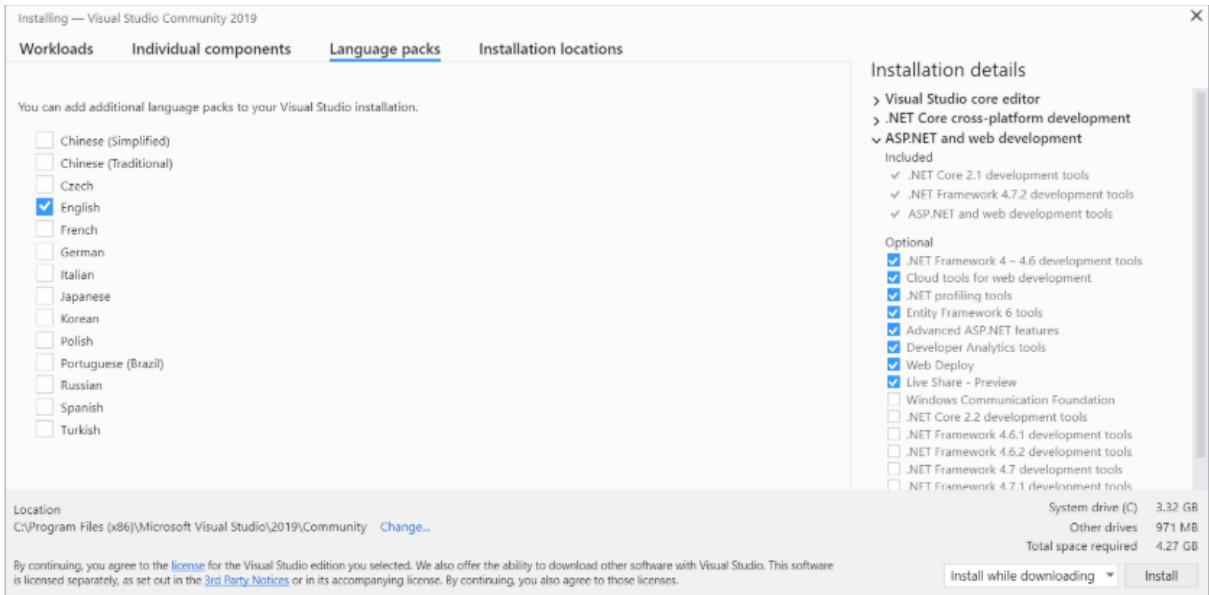
Projekt został wykonany przy wykorzystaniu języka C++, dlatego należy kliknąć opcję **Desktop development with C++**.

- W przypadku zakładki **Individual Components** najlepszym rozwiązaniem jest pozostawienie ustawień podstawowych.

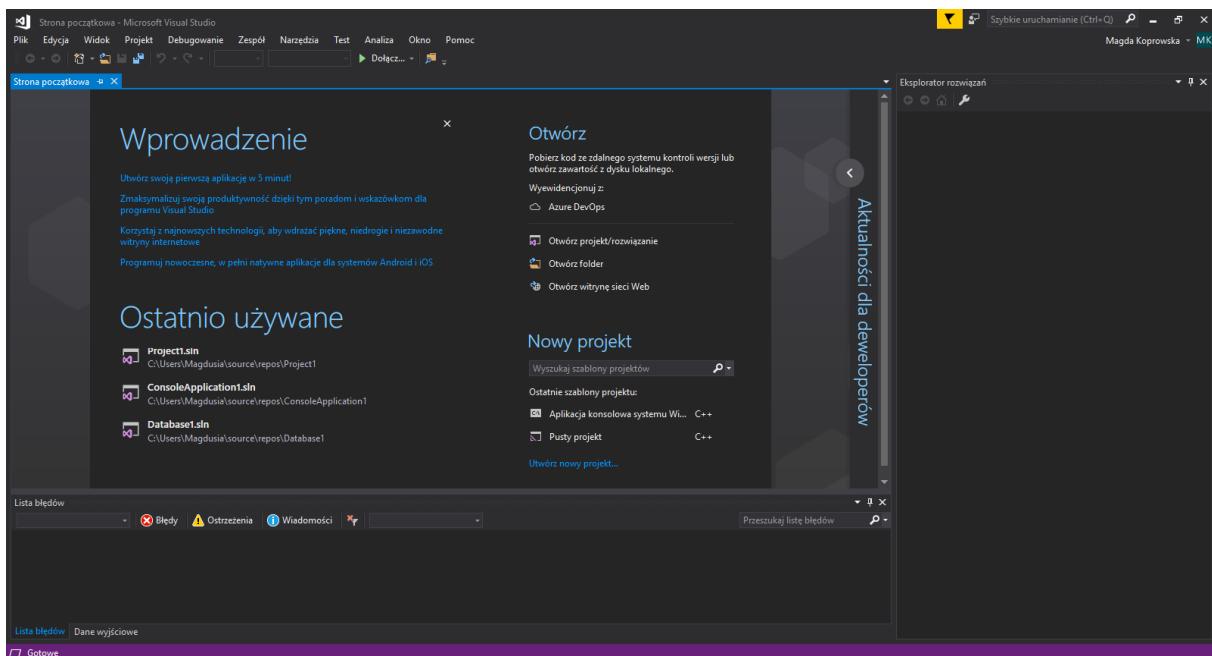


Rysunek 15. Okienko instalacyjne Microsoft Visual Studio – Individual Components. Źródło: docs.microsoft.com

- W zakładce **Language Packs** wybieramy język oprogramowania.



Rysunek 2. Okienko instalacyjne Microsoft Visual Studio – Language Packs. Źródło: docs.microsoft.com

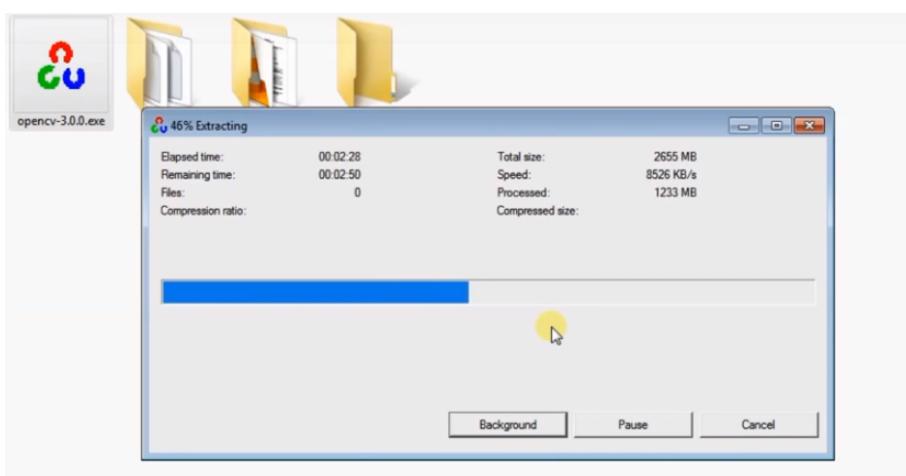


Rysunek 16. Okno startowe Microsoft Visual Studio.

Po zainstalowaniu Microsoft Visual Studio należy je uruchomić i wybrać **nowy pusty projekt C++**.

#### 4.4.2 Instalacja biblioteki OpenCV.

1. Pierwszym etapem projektu jest zainstalowane biblioteki OpenCV, z której będziemy korzystać. W tym celu należyściągnąć paczkę z biblioteką ze strony: <https://opencv.org/releases/>.
2. Następnie należy wypakować bibliotekę.



Rysunek 17. Proces wypakowywania biblioteki OpenCV.

3. Kolejny krok to przeniesienie wypakowanego folderu biblioteki na dysk C.
4. Następnie należy dodać bibliotekę do zmiennych środowiskowych.  
W tym celu należy pokonać ścieżkę Panel Sterowania>System i zabezpieczenia>System>Zaawansowane Ustawienia Systemu>Zmienne środowiskowe>Ścieżka>Zmień>Nowy>C:\opencv\build\x64\vc15\bin
5. Abytrzymać możliwość debugowania projektu należy wejść w ustawienia pliku źródłowego projektu i następnie C++>General>Additional Include Directories i wpisać w pole ścieżkę C:\opencv\build\include oraz Linker>General>Additional Library Directories C:\opencv\build\x64\vc15\lib

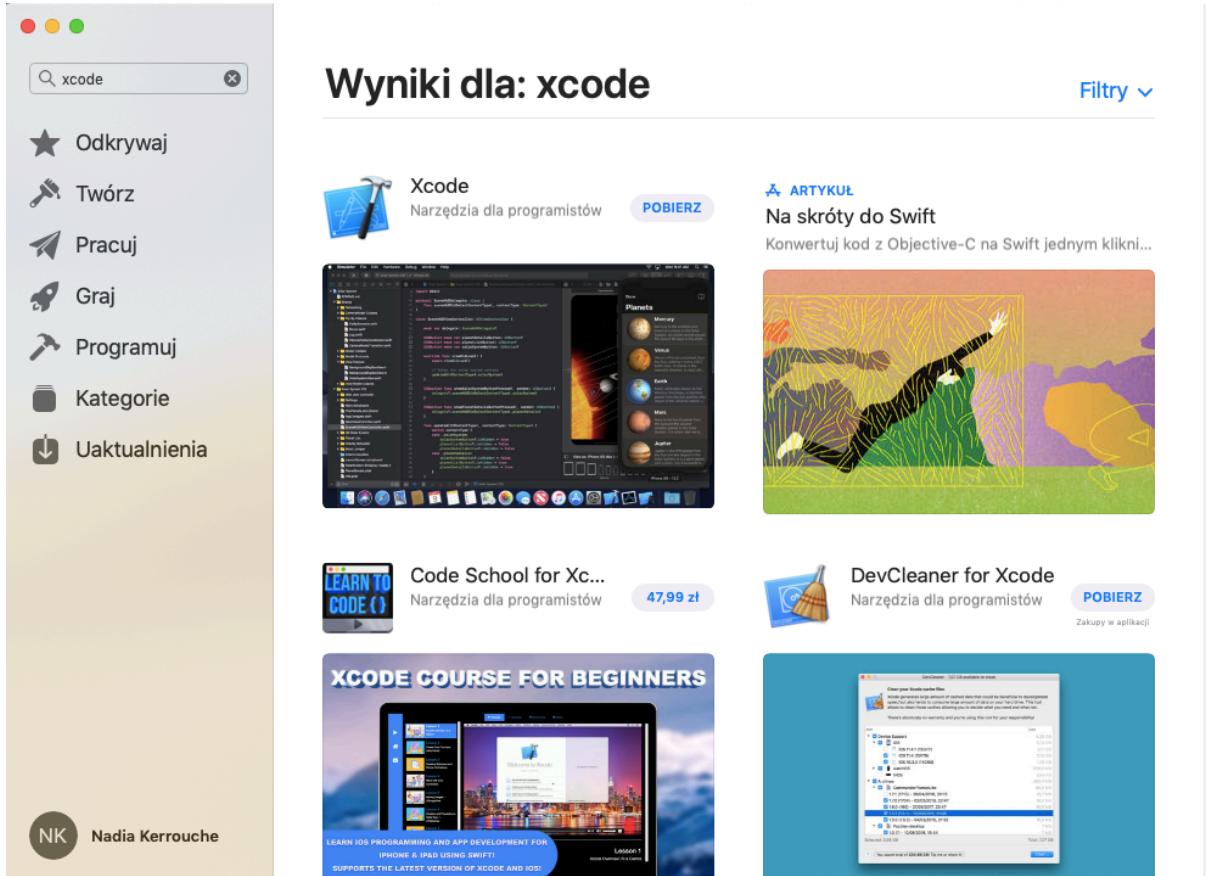
Wymagania sprzętowe i programowe:

<b>Obsługiwane systemy operacyjne</b>	Program Visual Studio 2017 można zainstalować i uruchamiać w następujących systemach operacyjnych: <ul style="list-style-type: none"> <li>• Windows 10 w wersji 1507 lub nowszej: Home, Professional, Education i Enterprise (wersje LTSC i S nie są obsługiwane)</li> <li>• Windows Server 2016: Standard i Datacenter</li> <li>• Windows 8.1 (z aktualizacją 2919355): Core, Professional i Enterprise</li> <li>• Windows Server 2012 R2 (z aktualizacją 2919355): Essentials, Standard, Datacenter</li> <li>• Windows 7 z dodatkiem SP1 (z najnowszymi aktualizacjami): Home Premium, Professional, Enterprise, Ultimate</li> </ul>
<b>Sprzęt</b>	<ul style="list-style-type: none"> <li>• Procesor 1,8 GHz lub szybszy (zalecany dwurdzeniowy lub lepszy).</li> <li>• 2 GB pamięci RAM; zalecane 4 GB pamięci RAM (co najmniej 2,5 GB w przypadku uruchamiania na maszynie wirtualnej).</li> <li>• Miejsce na dysku twardym: maksymalnie 130 GB dostępnego miejsca (w zależności od instalowanych funkcji); typowe instalacje wymagają 20–50 GB wolnego miejsca.</li> <li>• Szybkość dysku twardego: aby zwiększyć wydajność, system Windows i program Visual Studio należy zainstalować na dysku półprzewodnikowym (SSD).</li> <li>• Karta wideo obsługująca rozdzielcość ekranu co najmniej 720p (1280 x 720); program Visual Studio będzie działać najlepiej przy rozdzielcości WXGA (1366 x 768) lub wyższej.</li> </ul>

Rysunek 18.Tabela wymagań sprzętowych programu Microsoft Visual Studio 2017. Źródło: docs.microsoft.com.

#### 4.4.3 Instalacja środowiska w systemie macOS (Mojave 10.14.5).

Pierwszym krokiem jest pobranie aplikacji Xcode, dostępnej w App Store. Jeśli system jest starszy niż macOS Mojave 10.14.4, wtedy należy pobrać starszą wersję aplikacji Xcode kompatybilnej z wersją systemu, która jest dostępna na stronie producenta apple.com.



Rysunek 19. Okno aplikacji App Store.

Po zainstalowaniu środowiska można rozpoczęć utworzenie naszego projektu:

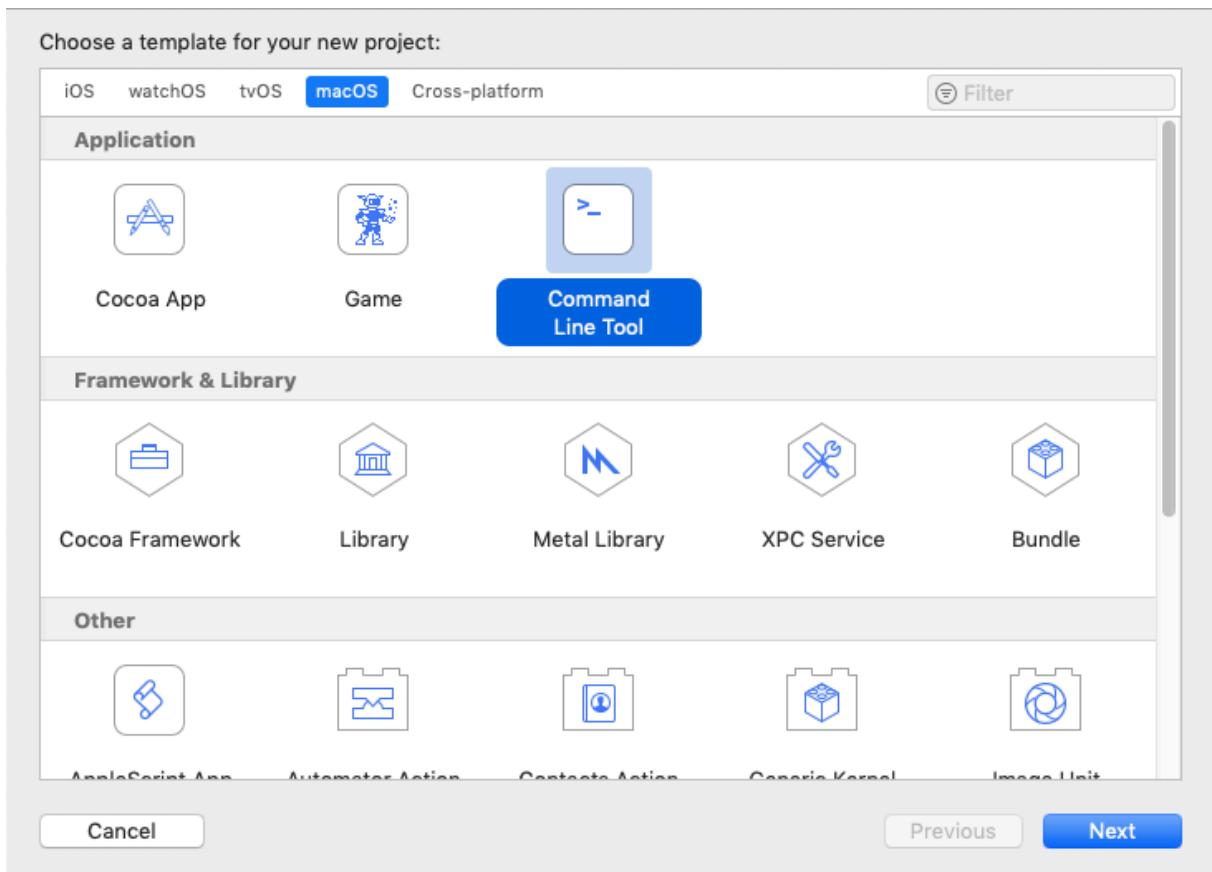


Rysunek 20. Okno startowe aplikacji Xcode.

Wybieramy opcję *Create a new Xcode project* i w kolejnym oknie *Choose a template for your new project* zaznaczamy odpowiednio:

- macOS
- Command Line Tool

Następnie należy kliknąć *next*.

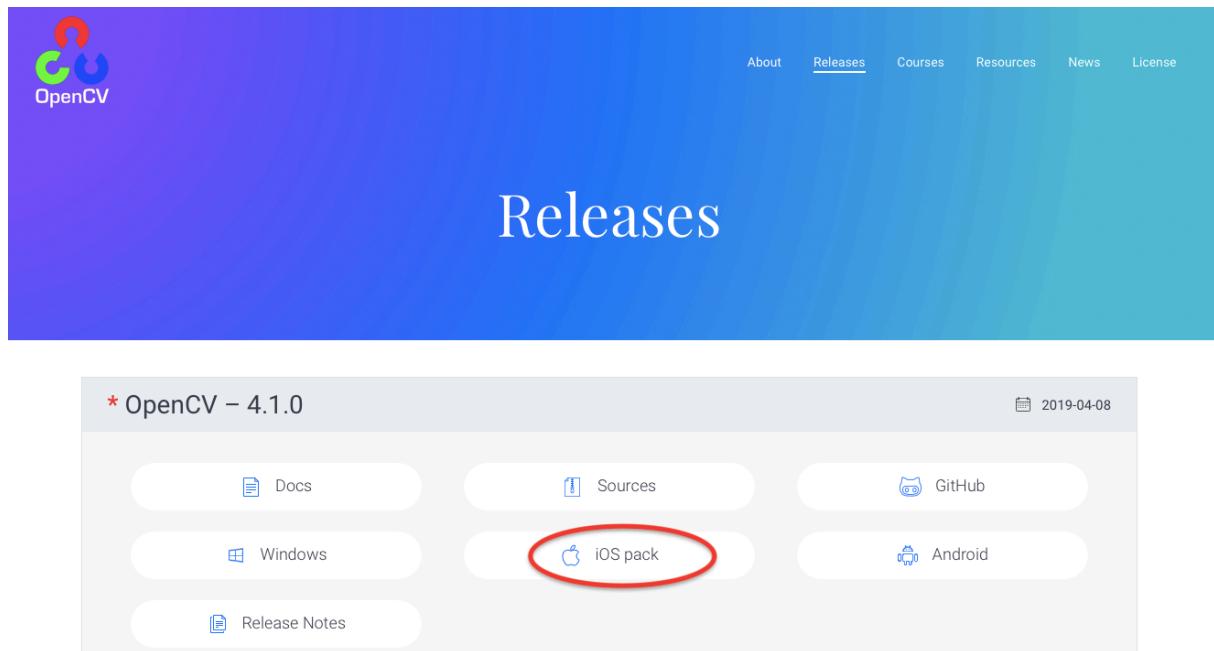


Rysunek 21. Okno programu Xcode z wyborem postaci naszej aplikacji.

Jeśli już mamy zainstalowane środowisko, możemy zacząć instalację biblioteki openCV.

#### 4.4.4 Instalacja biblioteki openCV.

W pierwszym kroku musimy pobrać wyżej wspomnianą bibliotekę ze strony <https://opencv.org/releases/> i wybieramy iOS pack.



Rysunek 22. Strona OpenCV, z której możemy pobrać iOS pack.

Do instalacji biblioteki potrzebujemy jeszcze pobrać Macports ze strony <https://www.macports.org>. Wybieramy po lewej stronie *Available Downloads* i z kolejnej strony wybieramy wersję kompatybilną z naszą wersją systemu: (w moim przypadku jest to wersja zaznaczona na poniższym rysunku).

▼ Assets 29

<a href="#">MacPorts-2.5.4-10.10-Yosemite.pkg</a>	3.41 MB
<a href="#">MacPorts-2.5.4-10.10-Yosemite.pkg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.11-ElCapitan.pkg</a>	3.41 MB
<a href="#">MacPorts-2.5.4-10.11-ElCapitan.pkg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.12-Sierra.pkg</a>	3.4 MB
<a href="#">MacPorts-2.5.4-10.12-Sierra.pkg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.13-HighSierra.pkg</a>	3.44 MB
<a href="#">MacPorts-2.5.4-10.13-HighSierra.pkg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.14-Mojave.pkg</a>	3.44 MB
<a href="#">MacPorts-2.5.4-10.14-Mojave.pkg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.4-Tiger.dmg</a>	3.95 MB
<a href="#">MacPorts-2.5.4-10.4-Tiger.dmg.asc</a>	228 Bytes
<a href="#">MacPorts-2.5.4-10.5-Leopard.dmg</a>	3.98 MB

Rysunek 23. Strona MacPorts z dostępnymi wersjami.

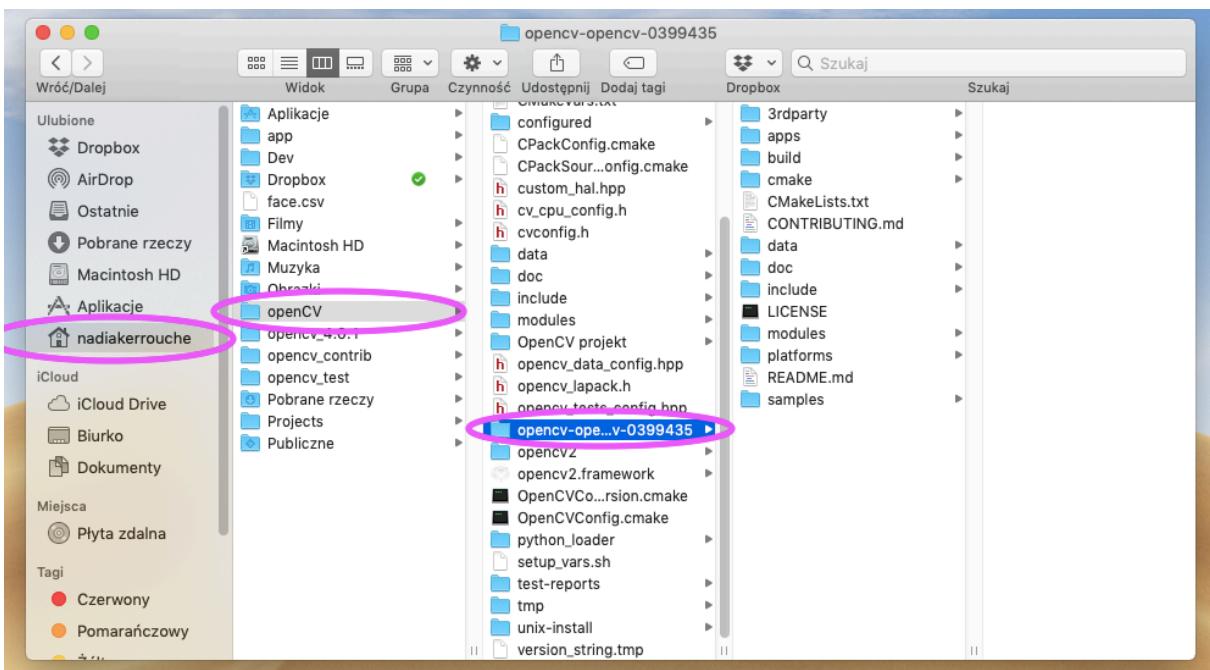
Po zainstalowaniu MacPorts umieszczamy pobraną bibliotekę openCV w wybranym przez nas miejscu, należy wybrać najodpowiedniejszą dla nas lokację, gdyż nie będziemy mogli później przenieść jej w inne miejsce bez konsekwencji, ponieważ w kolejnych krokach będziemy podawać odpowiednie ścieżki dla naszego programu.

Kolejnym krokiem jest sprawdzenie czy MacPorts zostały poprawnie zainstalowane, sprawdzamy to poprzez odpowiednie komendy w terminalu (znajdziemy go poprzez wyszukiwanie *Spotlight* bądź wchodząc w *Launchpad->Inne->terminal*).

```
Last login: Thu Jun 20 16:09:03 on ttys001
[MacBook-Air-Nadia:~ nadiakerrouche$ port
Warning: port definitions are more than two weeks old, consider updating them by
running 'port selfupdate'.
MacPorts 2.5.4
Entering shell mode... ("help" for help, "quit" to quit)
[Users/nadiakerrouche] > ]
```

Rysunek 24. Okno Terminala.

Następnie otwieramy nowe okno terminala oraz drugie okno *Findera*, aby znaleźć miejsce, gdzie docelowo umieściliśmy bibliotekę openCV.

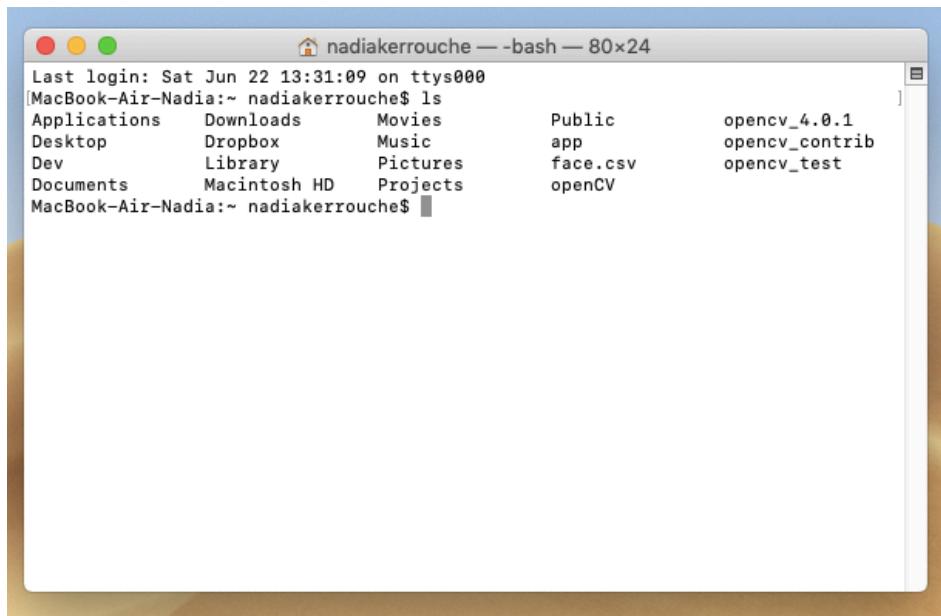


Rysunek 25. Ścieżka do biblioteki OpenCV.

W moim przypadku jest to: `nadiakerrouche/opencv/opencv-opencv-0399435` (natomiast u każdego może to wyglądać inaczej, zależnie od tego, gdzie umieszczona została biblioteka).

Gdy już wiemy, gdzie dokładnie jest nasza biblioteka, musimy dostać się do niej poprzez komendy w terminalu:

- ls – tworzy listę folderów bądź plików dla folderu, w którym obecnie się znajdujemy:

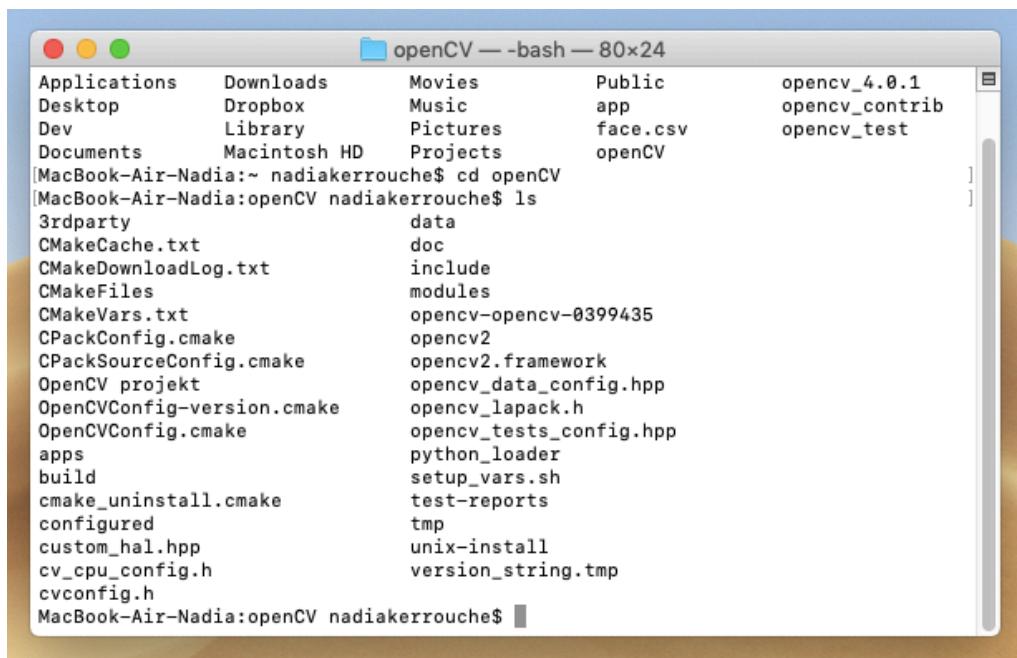


```
Last login: Sat Jun 22 13:31:09 on ttys000
[MacBook-Air-Nadia:~ nadiakerrouche$ ls
Applications    Downloads      Movies        Public      opencv_4.0.1
Desktop        Dropbox       Music         app        opencv_contrib
Dev            Library      Pictures     face.csv   opencv_test
Documents       Macintosh HD Projects    openCV
MacBook-Air-Nadia:~ nadiakerrouche$
```

Rysunek 26. Okno Terminala.

Musimy dostać się do folderu openCV, robimy do komendą:

- cd- (change directories), pozwala na zmianę folderu



```
openCV — bash — 80x24
Applications    Downloads      Movies        Public      opencv_4.0.1
Desktop        Dropbox       Music         app        opencv_contrib
Dev            Library      Pictures     face.csv   opencv_test
Documents       Macintosh HD Projects    openCV
[MacBook-Air-Nadia:~ nadiakerrouche$ cd openCV
[MacBook-Air-Nadia:openCV nadiakerrouche$ ls
3rdparty          data
CMakeCache.txt    doc
CMakeDownloadLog.txt include
CMakeFiles         modules
CMakeVars.txt     opencv-opencv-0399435
CPackConfig.cmake opencv2
CPackSourceConfig.cmake opencv2.framework
OpenCV projekt    opencv_data_config.hpp
OpenCVConfig-version.cmake opencv_lapack.h
OpenCVConfig.cmake opencv_tests_config.hpp
apps              python_loader
build             setup_vars.sh
cmake_uninstall.cmake test-reports
configured        tmp
custom_hal.hpp    unix-install
cv_cpu_config.h   version_string.tmp
cvconfig.h
MacBook-Air-Nadia:openCV nadiakerrouche$
```

Rysunek 27. Okno Terminala.

Teraz musimy zmienić kierunek na docelowy: `opencv-opencv-0399435`, ponownie za pomocą komendy `cd`. Gdy już jesteśmy w wyżej wspomnianym folderze, musimy za pomocą komendy `mkdir` (make direction) stworzyć folder `build`. W terminalu wygląda to w ten sposób:

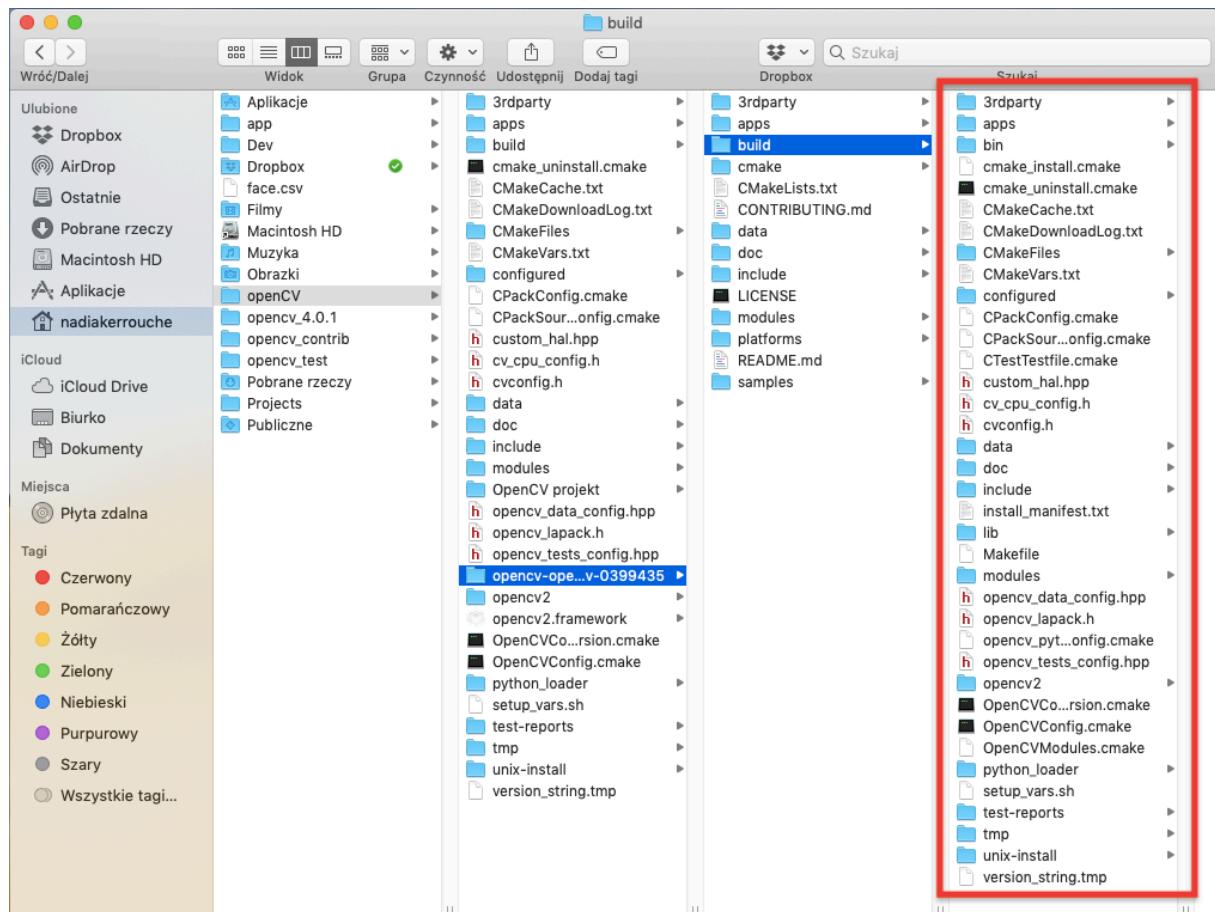
```
[MacBook-Air-Nadia:openCV nadiakerrouche$ cd opencv-opencv-0399435  
[MacBook-Air-Nadia:opencv-opencv-0399435 nadiakerrouche$ mkdir build
```

Rysunek 28. Komendy w terminalu.

```
[MacBook-Air-Nadia:opencv-opencv-0399435 nadiakerrouche$ cd build  
MacBook-Air-Nadia:build nadiakerrouche$ cmake -G "Unix Makefiles" ..
```

Rysunek 29. Komendy w terminalu.

Po wykonaniu komendy `cmake -G "Unix Makefiles" ..` w naszym folderze `build` pojawią się zaznaczone na obrazku elementy:



Rysunek 30. Zawartość folderu build.

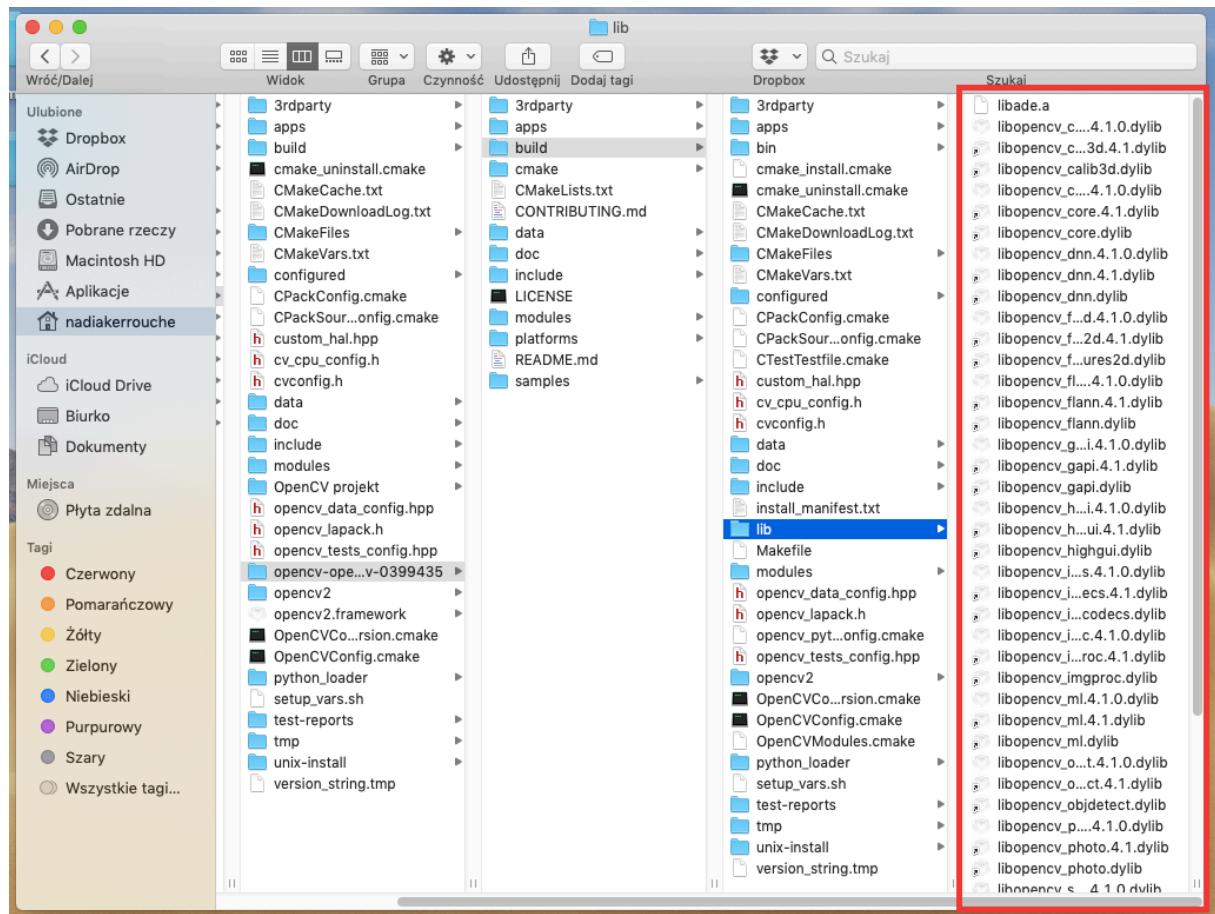
Następnie wpisujemy komendę `make -j8`, która stworzy nam więcej bibliotek openCV.

```
MacBook-Air-Nadia:~ nadiakerrouche$ make -j8
```

I w następnym kroku:

```
MacBook-Air-Nadia:~ nadiakerrouche$ sudo make install
```

W tym momencie instalacja biblioteki jest zakończona, wchodząc w folder lib, możemy zobaczyć wszystkie biblioteki jakie zostały zainstalowane.



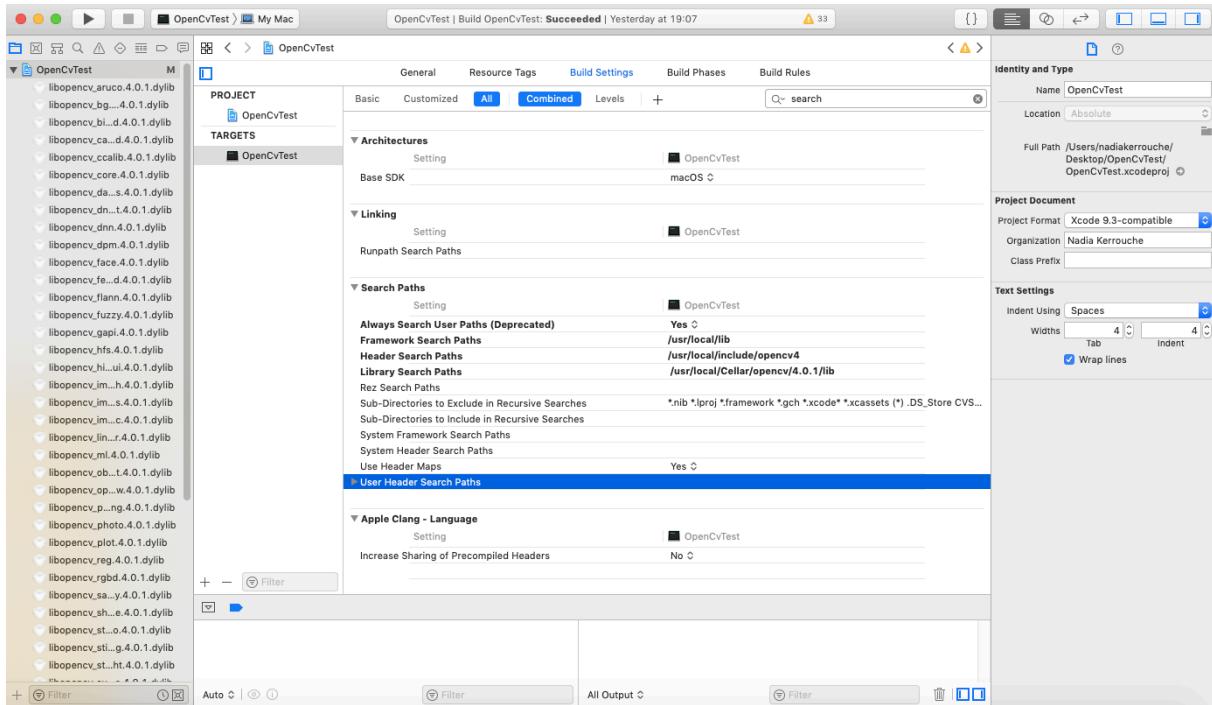
Rysunek 31. Zawartość folderu lib.

Po zainstalowaniu biblioteki openCV, możemy przejść do wcześniej stworzonego projektu w Xcode i wprowadzeniu odpowiednich ustawień.

Wybieramy kolejno: *nazwa\_naszego\_projektu->Build Settings-> All* oraz *Combined* i następnie wpisujemy w okienku **search**: **search paths** i ustawiamy *Always Search User Paths* na **Yes**.

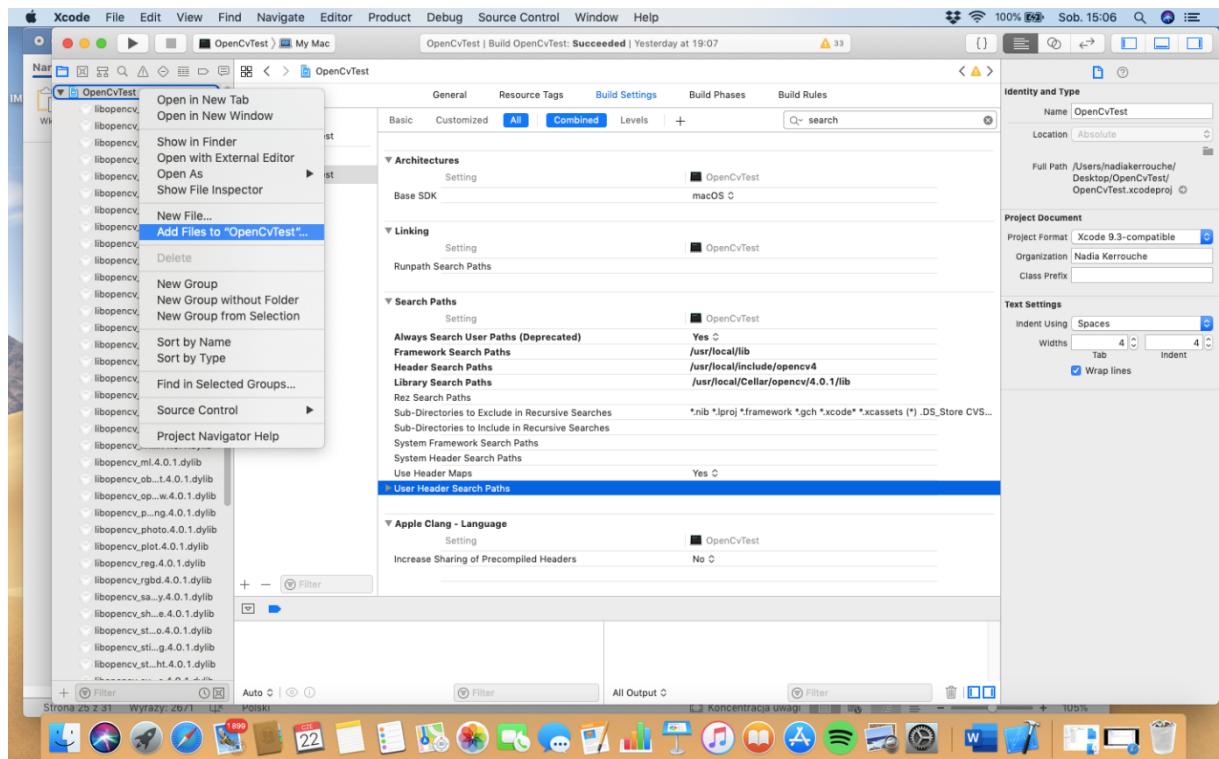
Dodajemy także ścieżki do poniższych ustawień:

- *Framework Search Path*
- *Header Search Paths*
- *Library Search Paths*



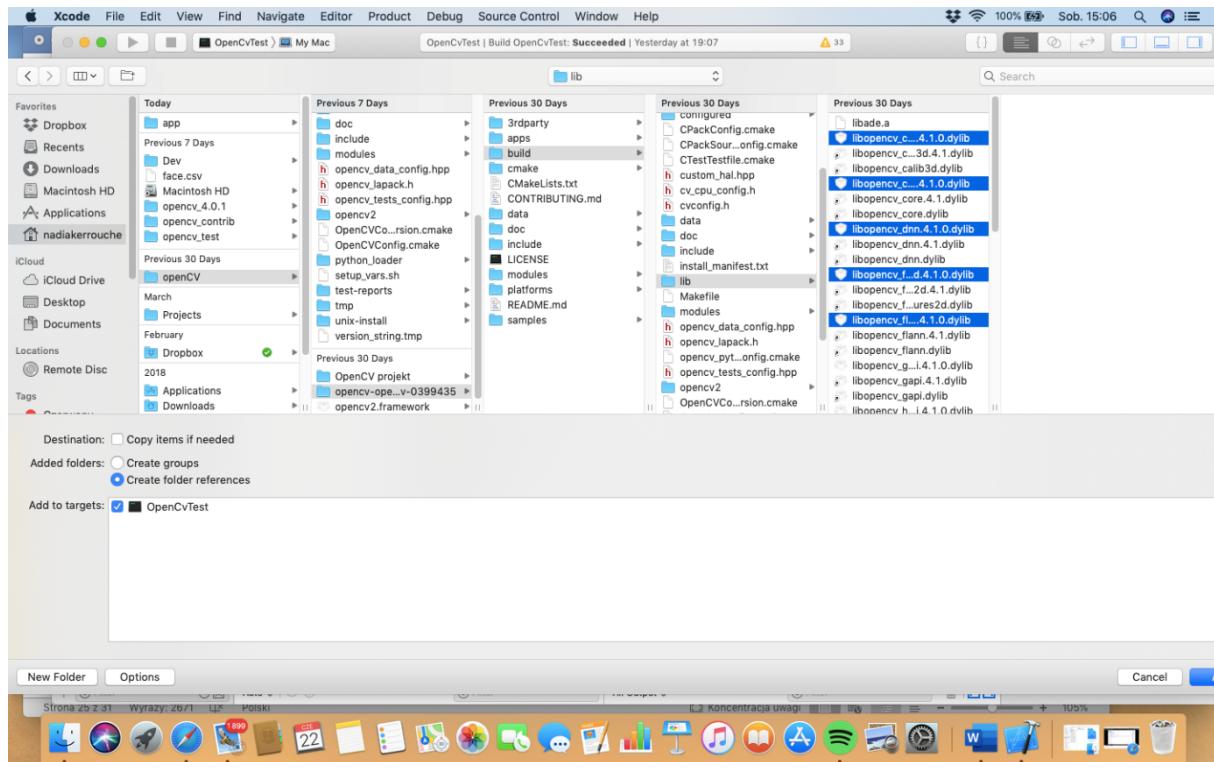
Rysunek 32. Okno ustawień w programie Xcode.

Teraz aby dodać biblioteki do naszego projektu należy kliknąć prawym przyciskiem na jego nazwę tak jak na poniższym rysunku i wybrać z listy *Add Files to "nazwa\_projektu"*...



Rysunek 33. Dodawanie bibliotek do programu.

Następnie należy przejść do folderu *lib* i wybrać z niego te biblioteki, których będziemy potrzebować w naszym projekcie, ważne jest też to, aby wybierać tylko te, które są zgodne z naszą wersją openCV, dla mnie jest to 4.1.0.dylib, zaznaczone też na poniższym rysunku.



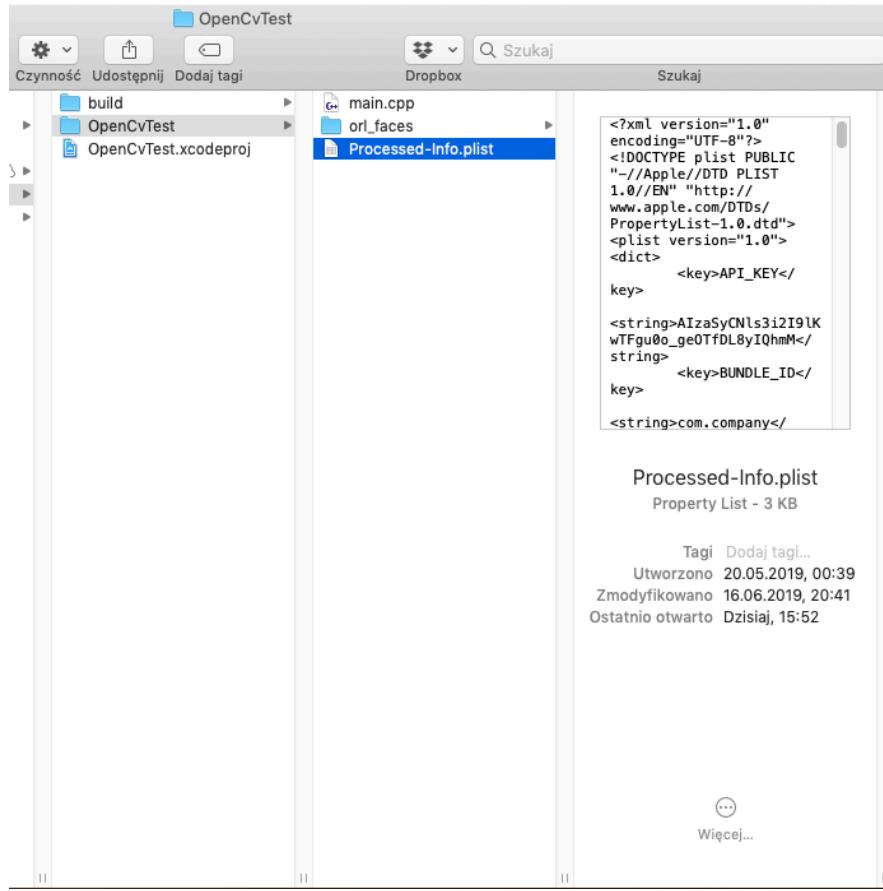
Rysunek 34. Dodawanie bibliotek do programu.

#### 4.4.5 Konfiguracja kamery w urządzeniu.

Jeżeli chcemy używać kamery w naszym urządzeniu, bez pozwolenia na dostęp do niej oraz bez zaimplementowania *info.plist*, nasz program pokaże następujący komunikat:

*OpenCvTest[7335:310504] [access] This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSCameraUsageDescription key with a string value explaining to the user how the app uses this data.*

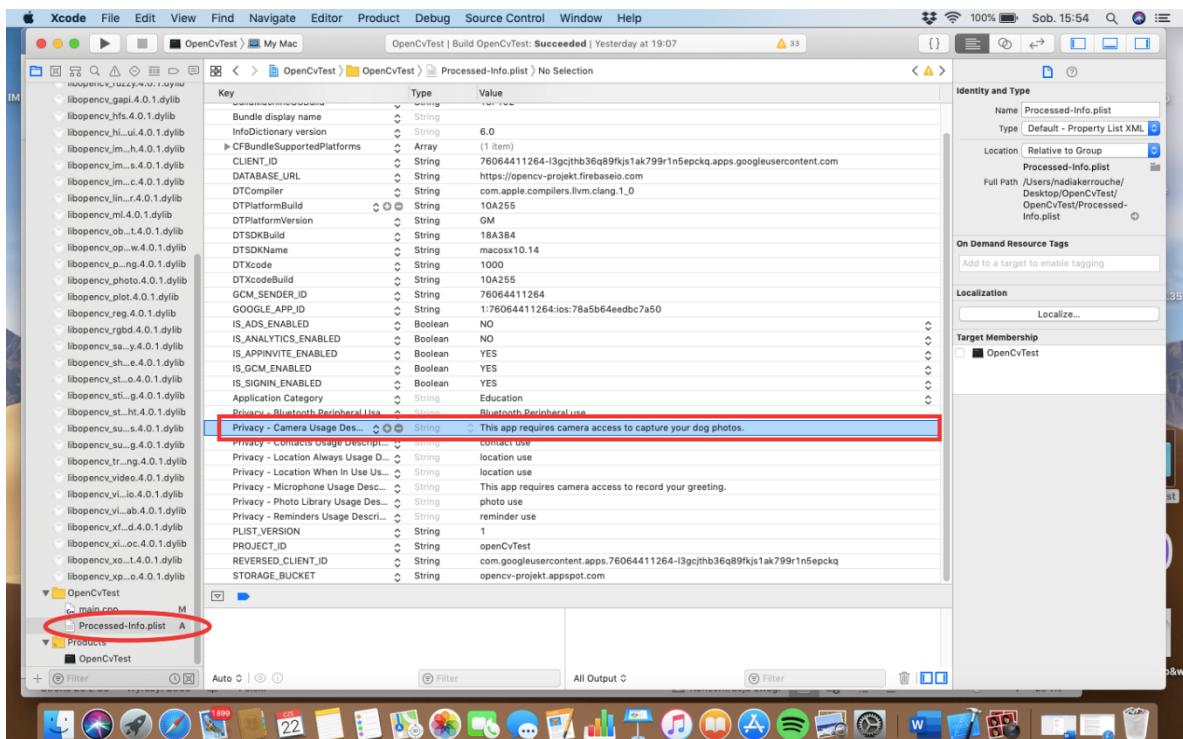
Wtedy należy znaleźć w finderze listę od aplikacji Photo Booth, skopiować ją do naszego projektu:



Rysunek 35. Miejsce na umieszczenie pliku info.plist.

I dodać do niej:

*Privacy -Camera Usage Description* wraz z opisem do czego będziemy jej używać.



Rysunek 36. Konfiguracja Info.plist.

## 5 Podsumowanie i wnioski

Projekt w miał w założeniu detekcję twarzy dwoma sposobami. Jednym z nich jest detekcja za pomocą gotowych klasyfikatorów Haara, a drugim wykorzystanie sztucznej sieci neuronowej TensorFlow. Następnym krokiem była realizacja algorytmu rozpoznawania twarzy z wykorzystaniem trzech metod: EigenFaceRecognizer, FisherFaceRecognizer i LBPHFaceRecognizer.

Narzędzia i metody:

- klasyfikator Haara,
- siec neuronowa Tensorflow,
- EigenFaceRecognizer,
- FisherFaceRecognizer,
- LBPHFaceRecognizer,

Udało się osiągnąć wyżej przedstawione założenia, aczkolwiek rozpoznawanie twarzy nie działa w pełni poprawnie i w niektórych przypadkach, podaje błędne etykiety. Można to poprawić dobierając w odpowiedni próg (threshold), czyli odległość od twarzy w zbiorze uczącym. Zauważłyśmy także, że metoda FisherFaceRecognizer działa z większą skutecznością niż dwie pozostałe i lepiej rozpoznaje twarze. Można to poprawić dobierając w odpowiedni próg

Detekcja twarzy działa poprawnie, jednak tylko z zastosowaniem klasyfikatorów Haara'a, zaś gdy wykorzystywane były sieci neuronowe (TensorFlow) program miał trudności z wykrywaniem twarzy w miejscach z gorszym oświetleniem, a także "gubił" prostokąt wykrywanej twarzy przy jej nieznacznym ruchu czy przechyleniu.

Ze względu na łatwiejszą obsługę zdecydowałyśmy się na zrealizowanie projektu w środowisku VisualStudio w systemie Windows, gdyż dostępny kompilator Xcode w systemie macOS ma ograniczone możliwości, co sprawiało trudność w początkowej realizacji projektu.

Zastosowane metody do wykrywania twarzy wybrałyśmy ze względu na łatwość implementacji oraz dostępność informacji na ich temat, jednak nie działają one dobrze przy słabych warunkach oświetleniowych.

Z przeprowadzonych testów dotyczących czasu uczenia sieci w zależności od wielkości bazy uczącej można zauważyć, że dla w przypadku dużej ilości elementów (w naszym przypadku było ich 900, każdy o rozdzielczości 112x96 pikseli) czasy uczenia modeli FisherFaceRecognizer i EigenFaceRecognizer są zbliżone i wynoszą ok 60 sekund, zaś dla modelu LBPHFaceRecognizer jest to zaledwie jedna sekunda. Czasy te przekładają się też na skuteczność rozpoznawania twarzy, gdyż model FisherFaceRecognizer działa według nas najlepiej spośród tych trzech dostępnych metod. W przypadku małej bazy uczącej (54 elementy) różnice w czasach nie były znaczące, ponieważ są to wartości rzędu części dziesiętnych, ale znowu zależność jest ta sama, uczenie modelu FisherFaceRecognizer trwało najdłużej, a LBPHFaceRecognizer najkrócej.

Kolejny test został przeprowadzony dla różnych rozdzielczości obrazów w bazie uczącej, zaś w tym przypadku ilość elementów w bazie uczącej pozostawała ta sama (600 elementów). Uczenie modelu EigenFaceRecognizer z bazy o większej rozdzielczości (92x112 pikseli) trwało ok. dwa razy dłużej niż dla rozmiaru 46x57 pikseli. Dla modelu

LBPHFaceRecognizer jest to ok. 2,5 razy dłużej, zaś dla modelu FisherFaceRecognizer stosunek czasów uczenia jest najmniejszy i wynosi ok. 1,3.

Trzecim testem było sprawdzenie skuteczności rozpoznawania twarzy dla wszystkich trzech metod z uwzględnieniem rozdzielczości obrazów w bazie uczącej. Największym współczynnikiem poprawności cechowała się metoda FisherFaceRecognizer, gdyż dla obrazów uczących o rozdzielczości 92x112 pikseli rozpoznała poprawnie 95% (suma liczba na diagonali podzielona na maksymalną możliwą sumę na diagonali) próbek, zaś dla mniejszej rozdzielczości (46x56 pikseli) 92,5% próbek. Skuteczność metody EigenFaceRecognizer wynosiła 79% dla rozdzielczości większej i 80% dla mniejszej. Na wyróżnienie zasługuje metoda LBPH, gdyż jej skuteczność dla obu przypadków rozdzielczości wynosi 86,25%, a czas jej uczenia nie przekracza półtorej sekundy.

Czwartym testem było określenie miar metody pomiaru – czułości, precyzji oraz miary F1.

Czułość metod LBPHFaceRecognizer i FisherFaceRecognizer dla obydwu rozdzielcość utrzymywała się na poziomie 100 %, co świadczy o prawidłowym przypisaniu próbki do danej etykiety, jeżeli próbka faktycznie należy do danej etykiety. Metoda EigenFaceRecognizer cechowała się czułością na poziomie 92 % dla większej rozdzielczości oraz 92,19 % dla mniejszej rozdzielczości, co również jest wynikiem zadowalającym, jednak nie pozwalającym stwierdzić zależności pomiędzy rozdzielczością obrazu, a czułością metody.

Największą precyzję cechowała się metoda FisherFaceRecognizer dla obydwu rozdzielcości. Kolejno dla większej (92x112px) była równa 94,29 %, a dla mniejszej (46x56px) – 91,43 %. Duża wartość precyzji świadczy o powtarzalności wykonywanych pomiarów. Dla pozostałych przypadków precyzja utrzymywała się na poziomie ok. 84 %.

Kolejną obliczoną przez nas miarą była miara F1. Najlepszy wynik w tym teście uzyskała metoda FisherFaceRecognizer – 97 % dla większej rozdzielczości oraz 95,52 % dla mniejszej. Dla metody LBPHFaceRecognizer wynosiła ona 91,48 % dla obydwu rozdzielcości, a dla metody EigenFaceRecognizer – 86 % dla większej oraz 88 % dla mniejszej rozdzielczości próbek.

## 6 Możliwości rozbudowy

Program mógłby zostać wyposażony w detekcję nastroju osoby stojącej w kadrze kamery i jej płci. Można by również dodać interfejs graficzny do programu, przy pomocy biblioteki HighGUI.

Aby program rozpoznawał twarze w sposób poprawny z jak najmniejszymi możliwymi pomyłkami w przypisywaniu etykiety do odpowiedniej twarzy, należałoby dobrąć odpowiedni próg (threshold) oraz rozbudować bazę o większą ilość zdjęć danej osoby. Wtedy program wykrywając nieznaną twarz, nie będzie przypisywał jej etykiety z bazy, ale zostanie przypisana wartość -1.

## Bibliografia

- Źródła drukowane

[1] <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

[2] Adrian Kaehler, Gary Bradski,  
*OpenCV 3. Komputerowe rozpoznawanie obrazu w C++ przy użyciu biblioteki OpenCV.*

[3] Sebastian Wojas,  
*Metody przetwarzania obrazów z wykorzystaniem biblioteki OpenCV.*

[4] <http://www.michalbereta.pl/dydaktyka/KPO/Rozpoznawanie%20twarzy.pdf>

- Materiały dostępne w sieci

[5] <https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

[6] <http://vis-www.cs.umass.edu/lfw/#deepfunnel-anchor>

[7] <http://www.learnopencv.org>, ostatni dostęp: 20.06.2019

- <https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/>, ostatni dostęp: 24.06.2019

[8] <https://forbot.pl/blog/opencv-3-detekcja-i-rozpoznawanie-twarz-id5664>,  
ostatni dostęp: 25.06.2019

[9] [https://docs.opencv.org/4.0.1/dd/d65/classcv\\_1\\_1face\\_1\\_1FaceRecognizer.html](https://docs.opencv.org/4.0.1/dd/d65/classcv_1_1face_1_1FaceRecognizer.html),  
ostatni dostęp: 26.06.201



## Dodatek

### 6.2 *pch.cpp*

```
// pch.cpp: plik źródłowy odpowiadający wstępnie skompilowanemu nagłówkowi, niezbędny do powodzenia komplikacji
```

```
#include "pch.h"
```

### 6.3 *projekt.cpp*

```
#include "pch.h"
```

### 6.4 *źródło.cpp*

```
#include "pch.h"
```

```
#include "opencv2/core/core.hpp"
```

```
#include "opencv2/highgui/highgui.hpp"
```

```
#include "opencv2/imgproc/imgproc.hpp"
```

```
#include "opencv2/objdetect/objdetect.hpp"
```

```
#include <string>
```

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <opencv2/face.hpp>
```

```
#include <numeric>
```

```
#include "opencv2\opencv.hpp"
```

```
#include <chrono>
```

```
using namespace cv;
```

```
using namespace std;
```

```
using namespace face;
```

```
using namespace dnn;
```

```
using namespace std::chrono;
```

```
string face_cascade_name = "haarcascade_frontalface_alt.xml";
```

```
CascadeClassifier face_cascade;
```

```
void read_csv(const string & filename, vector<Mat> & images, vector<int> & labels)
```

```
{
```

```
    ifstream file(filename.c_str(), ifstream::in);
```

```

if (!file)
{
    cout << "Otwarcie pliku " << filename << " nie powiodlo sie.";
    return;
}

string line, path, classlabel;
while (getline(file, line))
{
    stringstream liness(line);
    getline(liness, path, ';');
    getline(liness, classlabel);
    //classlabel.erase(0, 1);
    //path.erase(path.length() - 1, path.length());
    if (!path.empty() && !classlabel.empty())
    {
        images.push_back(imread(path, 0));
        labels.push_back(atoi(classlabel.c_str()));
        cout << path << "=" << atoi(classlabel.c_str()) << endl;
    }
}
}

int rozpoznawanie_twarzy_kamera() {

vector<Mat> images;
vector<int> labels;

string filename = "twarze_92_112.csv";

try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku '" << filename << "'. Powod: " << e.what() << endl;
    exit(1);
}
}

```

```

//Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
//Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

double current_threshold = model->getThreshold();
cout << "obecny prog" << current_threshold << endl;

model->setThreshold(150);
double current_threshold2 = model->getThreshold();

cout << "prog po zmianie:" << current_threshold2 << endl;

//model->train(images, labels);

high_resolution_clock::time_point t3 = high_resolution_clock::now();

model->train(images, labels);
cout << "model wytrenowany" << endl;

high_resolution_clock::time_point t4 = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(t4 - t3).count();

cout << "czas trwania uczenia modelu: " << duration << "microseconds" << endl;

model->save("model_92_112_LBPHFaceRecognizer.yaml");
/*
high_resolution_clock::time_point t1 = high_resolution_clock::now();
model->read("model.yaml");
high_resolution_clock::time_point t2 = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(t2 - t1).count();

cout << "czas trwania czytania modelu: " << duration << endl;
*/
//VideoCapture kamera("nadia.mp4");
VideoCapture kamera(0);
//VideoCapture kamera("magda_720_1280_bezokularow_wyciety.mp4");

```

```

if (!kamera.isOpened())
{
    printf("error to initialize camera");
    return 1;
}

kamera.set(CAP_PROP_FRAME_WIDTH, 1280);
kamera.set(CAP_PROP_FRAME_HEIGHT, 720);

Mat obraz_z_kamery;

// stworzenie i załadowanie parametrów detektora twarzy
Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
"opencv_face_detector.pbtxt");

Mat testSample = imread("Luiz_Inacio_Lula_da_Silva_twarz_0046.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

int nr_fotki = 0;

while (waitKey(1) != 27)
{
    kamera >> obraz_z_kamery;

    //Mat obraz_z_kamery_szarosciowy;
    //cvtColor(obraz_z_kamery, obraz_z_kamery_szarosciowy, COLOR_BGR2GRAY);

    //przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej
    twarze

    Mat blob = cv::dnn::blobFromImage(obraz_z_kamery, 1.0, Size(img_width, img_height),
Scalar(104.0, 177.0, 123.0), true, false);

    // uruchomienie głębszej sieci neuronowej wykrywającej twarze
    net.setInput(blob, "data");
    Mat detekcje = net.forward("detection_out");

```

```

// przeformatowanie wyniku: tyle wierszy ile twarzy,
// w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt - twarz,
// więc tam zawsze jest 1),
// pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków
// prostokąta zawierającego obiekt (x1, y1, x2, y2)
Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F, detekcje.ptr<float>());

vector<Rect> twarze;
for (int i = 0; i < macierz_detekcji.rows; i++)
{
    if (macierz_detekcji.at<float>(i, 2) > 0.5) //pewność wykrycia (w skali 0-1)
    {
        // odczyt punktów tworzących prostokąt wokół twarzy
        int x1 = (macierz_detekcji.at<float>(i, 3) * obraz_z_kamery.cols);
        int y1 = (macierz_detekcji.at<float>(i, 4) * obraz_z_kamery.rows);
        int x2 = (macierz_detekcji.at<float>(i, 5) * obraz_z_kamery.cols);
        int y2 = (macierz_detekcji.at<float>(i, 6) * obraz_z_kamery.rows);

        if (x1 < 0 || y1 < 0 || x2 > 1280 || y2 > 720)
            break;
        //zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze
        //czyli właśnie prostokąty otaczające twarze)
        twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));
        //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni
        element wektora twarze: twarze.back()

        Mat przeskalowana_twarz;
        Mat przeskalowana_twarz_szara;

        //funkcja rysująca prostokąt na ekranie
        rectangle(obraz_z_kamery, twarze.back(), CV_RGB(0, 255, 100), 1,
        LINE_8, 0);
        //resize(twarze, przeskalowana_twarz, Size(img_width, img_height), 1.0,
        1.0, 1);

        resize(obraz_z_kamery(twarze.back()), przeskalowana_twarz,
        Size(img_width, img_height), 1.0, 1.0, 1);
    }
}

```

```

imshow("przeskalowana twarz", przeskalowana_twarz);
cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara,
COLOR_BGR2GRAY);

//imwrite(format("Magdusia_%03d.jpg", nr_fotki++), przeskalowana_twarz);
//imwrite(format("Nadia_%03d.jpg", nr_fotki++), przeskalowana_twarz);

//int plabel = model->predict(przeskalowana_twarz);
//int plabel = model->predict(przeskalowana_twarz_szara);
int plabel;
double confidence;

model->predict(przeskalowana_twarz_szara, plabel, confidence);

cout << "etykieta " << plabel << endl << "odleglosc " << confidence << endl;

string box_text = format("Predykcja = %d", plabel);

int pos_x = max(twarze[i].x - 10, 0);
int pos_y = max(twarze[i].y - 10, 0);

putText(obraz_z_kamery, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

}

}

imshow("test", obraz_z_kamery);

}

}

int rozpoznawanie_twarzy_kamera_ladowanie() {

vector<Mat> images;
vector<int> labels;

string filename = "face_recognition_video.csv";

```

```
try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku '" << filename << "'. Powod: " << e.what() << endl;
    exit(1);
}
```

```
Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
//Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
//Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

double current_threshold = model->getThreshold();
cout << "obecny prog" << current_threshold << endl;

model->setThreshold((current_threshold / 100));
double current_threshold2 = model->getThreshold();

cout << "prog po zmianie:" << current_threshold2 << endl;

model->train(images, labels);

//VideoCapture kamera("nadia.mp4");
VideoCapture kamera(0);

if (!kamera.isOpened())
{
    printf("error to initialize camera");
    return 1;
}

kamera.set(CAP_PROP_FRAME_WIDTH, 1280);
kamera.set(CAP_PROP_FRAME_HEIGHT, 720);

Mat obraz_z_kamery;
```

```

// stworzenie i załadowanie parametrów detektora twarzy
Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
"opencv_face_detector.pbtxt");

Mat testSample = imread("magda_720_1280_bezokularow_wyciety 010.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

int nr_fotki = 0;

while (waitKey(1) != 27)
{
    kamera >> obraz_z_kamery;

    //Mat obraz_z_kamery_szarościowy;
    //cvtColor(obraz_z_kamery, obraz_z_kamery_szarościowy, COLOR_BGR2GRAY);

    //przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej
    twarze

    Mat blob = cv::dnn::blobFromImage(obraz_z_kamery, 1.0, Size(img_width, img_height),
Scalar(104.0, 177.0, 123.0), true, false);

    // uruchomienie głębszej sieci neuronowej wykrywającej twarze
    net.setInput(blob, "data");
    Mat detekcje = net.forward("detection_out");
    // przeformatowanie wyniku: tyle wierszy ile twarzy,
    // w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt - twarz,
    więc tam zawsze jest 1),
    // pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków
    prostokąta zawierającego obiekt (x1, y1, x2, y2)
    Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F, detekcje.ptr<float>());

    vector<Rect> twarze;
    for (int i = 0; i < macierz_detekcji.rows; i++)
    {
        if (macierz_detekcji.at<float>(i, 2) > 0.4) //pewność wykrycia (w skali 0-1)

```

```

{

    // odczyt punktów tworzących prostokąt wokół twarzy
    int x1 = (macierz_detekcji.at<float>(i, 3) * obraz_z_kamery.cols);
    int y1 = (macierz_detekcji.at<float>(i, 4) * obraz_z_kamery.rows);
    int x2 = (macierz_detekcji.at<float>(i, 5) * obraz_z_kamery.cols);
    int y2 = (macierz_detekcji.at<float>(i, 6) * obraz_z_kamery.rows);

    // zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze
    // (czyli właśnie prostokąty otaczające twarze)
    twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));
    //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni
    element wektora twarze: twarze.back()

    Mat przeskalowana_twarz;
    Mat przeskalowana_twarz_szara;

    //funkcja rysująca prostokąt na ekranie
    rectangle(obraz_z_kamery, twarze.back(), CV_RGB(0, 255, 100), 1,
    LINE_8, 0);
    //resize(twarze, przeskalowana_twarz, Size(img_width, img_height), 1.0,
    1.0,1);

    resize(obraz_z_kamery(twarze.back()), przeskalowana_twarz,
    Size(img_width, img_height), 1.0, 1.0, 1);
    imwrite(format("Magda_twarz_%03d.jpg", nr_fotki++),
    przeskalowana_twarz);
    cout << "zdjęcie: " << format("Magda_%03d.jpg", nr_fotki) << endl;

    imshow("przeskalowana twarz", przeskalowana_twarz);
    cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara,
    COLOR_BGR2GRAY);

    //imwrite(format("Magdusia_%03d.jpg", nr_fotki++), przeskalowana_twarz);
    //imwrite(format("Nadia_%03d.jpg", nr_fotki++), przeskalowana_twarz);

    //int plabel = model->predict(przeskalowana_twarz);
    int plabel = model->predict(przeskalowana_twarz_szara);
}

```

```

cout << plabel << endl;

string box_text = format("Predykcja = %d", plabel);

int pos_x = max(twarze[i].x - 10, 0);
int pos_y = max(twarze[i].y - 10, 0);

putText(obraz_z_kamery, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

}

imshow("test", obraz_z_kamery);
}

}

```

```

int rozpoznawanie_twarzy_video_tensorflow() {

vector<Mat> images;
vector<int> labels;

Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
//Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
//Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

string filename = "face_recognition_video.csv";

try {
    read_csv(filename, images, labels);
}

catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku '" << filename << "'. Powod: " << e.msg << endl;
    exit(1);
}

```

```

model->train(images, labels);

//VideoCapture kamera("head-pose-face-detection-male.mp4");
//VideoCapture kamera("magda_720_1280_okulary.mp4");
VideoCapture kamera("magda_720_1280_bezokularow.mp4");

if (!kamera.isOpened())
{
    printf("error to initialize camera");
    return 1;
}

kamera.set(CAP_PROP_FRAME_WIDTH, 1280);
kamera.set(CAP_PROP_FRAME_HEIGHT, 720);

Mat obraz_z_kamery, obraz_z_kamery_szary;

// stworzenie i załadowanie parametrów detektora twarzy
Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
"opencv_face_detector.pbtxt");

Mat testSample = imread("person4_1.jpg", 0);
while (waitKey(1) != 27)
{
    kamera >> obraz_z_kamery;

    //przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej
    twarze
    Mat blob = cv::dnn::blobFromImage(obraz_z_kamery, 1.0, Size(300, 300), Scalar(104.0,
    177.0, 123.0), true, false);

    // uruchomienie głębszej sieci neuronowej wykrywającej twarze
    net.setInput(blob, "data");
    Mat detekcje = net.forward("detection_out");
    // przeformatowanie wyniku: tyle wierszy ile twarzy,
    // w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt - twarz,
    więc tam zawsze jest 1),
}

```

```

    // pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków
    prostokąta zawierającego obiekt (x1, y1, x2, y2)

    Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F, detekcje.ptr<float>());

    vector<Rect> twarze;
    for (int i = 0; i < macierz_detekcji.rows; i++)
    {
        if (macierz_detekcji.at<float>(i, 2) > 0.6) //pewność wykrycia (w skali 0-1)
        {
            // odczyt punktów tworzących prostokąt wokół twarzy
            int x1 = (macierz_detekcji.at<float>(i, 3) * obraz_z_kamery.cols);
            int y1 = (macierz_detekcji.at<float>(i, 4) * obraz_z_kamery.rows);
            int x2 = (macierz_detekcji.at<float>(i, 5) * obraz_z_kamery.cols);
            int y2 = (macierz_detekcji.at<float>(i, 6) * obraz_z_kamery.rows);
            //zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze
            (czyli właśnie prostokąty otaczające twarze)
            twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));
            //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni
            element wektora twarze: twarze.back()

            Mat przeskalowana_twarz;
            int img_width = testSample.cols;
            int img_height = testSample.rows;
            //funkcja rysująca prostokąt na ekranie
            rectangle(obraz_z_kamery, twarze.back(), CV_RGB(0, 255, 100), 1,
LINE_8, 0);
            resize(macierz_detekcji, przeskalowana_twarz, Size(img_width, img_height),
1.0, 1.0, 1);

            int plabel = model->predict(przeskalowana_twarz);
            cout << plabel << endl;

            string box_text = format("Predykcja = %d", plabel);

            int pos_x = max(twarze[i].x - 10, 0);
            int pos_y = max(twarze[i].y - 10, 0);

            putText(obraz_z_kamery, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

        }
    }

```

```

        }

        imshow("test", obraz_z_kamery);

    }

}

int rozpoznawanie_twarzy_video_haar() {

    vector<Mat> images;
    vector<Mat> images_gray;
    vector<int> labels;

    string filename = "face_recognition_video.csv";

    try {
        read_csv(filename, images, labels);
    }

    catch (cv::Exception& e) {
        cerr << "Problem z otwarciem pliku '" << filename << "'. Powod: " << e.msg << endl;

        exit(1);
    }

    Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
    //Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
    //Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

    model->setThreshold(150);
    //model->train(images, labels);
    model->read("model.yaml");

    //VideoCapture kamera("head-pose-face-detection-male.mp4");
    //VideoCapture kamera("magda_720_1280_okulary.mp4");
    //VideoCapture kamera("magda_720_1280_bezokularow.mp4");

    VideoCapture kamera(1);

    if (!kamera.isOpened())

```

```

{
    printf("Problem z ladowaniem kamery.");
    return 1;
}

kamera.set(CAP_PROP_FRAME_WIDTH, 1280);
kamera.set(CAP_PROP_FRAME_HEIGHT, 720);

Mat obraz_z_kamery, obraz_z_kamery_szary;

Mat testSample = imread("magda_720_1280_bezokularow_wycięty 010.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

while (waitKey(1) != 27)
{
    kamera >> obraz_z_kamery;

    vector<Rect> faces;
    vector<Mat> images;

    if (!face_cascade.load(face_cascade_name))      //Ładowanie pliku ze sprawdzeniem
poprawnoci
    {
        cout << "Nie znaleziono pliku " << face_cascade_name << ".";
        return -2;
    }
    int i = 0;

    Mat img_gray;
    Mat img_cut;
    Mat img_resized;
    Mat resized_gray;

    vector<int> compression_params;

    compression_params.push_back(IMWRITE_JPEG_QUALITY);
    compression_params.push_back(100);
}

```

```

        cvtColor(obraz_z_kamery, img_gray, COLOR_BGR2GRAY); //Konwersja obrazu z
        kamery do odcieni szarosci

        face_cascade.detectMultiScale(img_gray, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE,
        Size(img_width, img_height)); //wykrycie twarzy o rozmiarach

        for (unsigned i = 0; i < faces.size(); i++)
        {
            resize(obraz_z_kamery(faces[i]), img_resized, Size(img_width, img_height), 1.0, 1.0,
1);

            cvtColor(img_resized, resized_gray, COLOR_BGR2GRAY); //przeskalowanie i
            przekształcenie na obrazy szarościowe

            imshow("przeskalowana twarz", img_resized);

            Point pt1(faces[i].x + faces[i].width, faces[i].y + faces[i].height); //kwadrat na
            twarzy

            Point pt2(faces[i].x, faces[i].y);
            rectangle(obraz_z_kamery, pt1, pt2, Scalar(0, 255, 0), 2, 8, 0);

            int plabel = model->predict(resized_gray);

            string box_text = format("Predykcja = %d", plabel);

            int pos_x = max(faces[i].x - 10, 0);
            int pos_y = max(faces[i].y - 10, 0);

            putText(obraz_z_kamery, box_text, Point(pos_x, pos_y), FONT_HERSHEY_PLAIN,
1.0, CV_RGB(0, 255, 0), 2.0);

            cout << plabel << endl;
        }

        imshow("test", obraz_z_kamery);
    }
}

int rozpoznawanie_twarzy_obrazy_test() {

```

```

Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
//Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();
//Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();

double current_threshold = model->getThreshold();
cout << "obecny prog" << current_threshold << endl;

model->setThreshold(150);
double current_threshold2 = model->getThreshold();

cout << "prog po zmianie:" << current_threshold2 << endl;

model->read("model.yaml");

string obraz_testowany_nazwa = "WIN_20190623_16_48_05_Pro 092_46_57.jpg";
Mat obraz_testowany = imread(obraz_testowany_nazwa, 1);

if (!obraz_testowany.data)
{
    cout << "Nie odnaleziono " << obraz_testowany_nazwa << ".";
    return -2;
}

// stworzenie i załadowanie parametrów detektora twarzy
Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
"opencv_face_detector.pbtxt");

Mat testSample = imread("David_Beckham_twarz_0011.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

int nr_fotki = 0;

while (waitKey(1) != 27)
{

```

```
//przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej twarze
```

```
Mat blob = cv::dnn::blobFromImage(obraz_testowany, 1.0, Size(img_width, img_height),  
Scalar(104.0, 177.0, 123.0), true, false);
```

```
// uruchomienie głębszej sieci neuronowej wykrywającej twarze  
net.setInput(blob, "data");  
Mat detekcje = net.forward("detection_out");  
// przeniesienie wyniku: tyle wierszy ile twarzy,  
// w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt - twarz,  
więc tam zawsze jest 1),
```

```
// pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków  
prostokąta zawierającego obiekt (x1, y1, x2, y2)
```

```
Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F, detekcje.ptr<float>());
```

```
vector<Rect> twarze;  
for (int i = 0; i < macierz_detekcji.rows; i++)  
{  
    if (macierz_detekcji.at<float>(i, 2) > 0.5) //pewność wykrycia (w skali 0-1)  
    {  
        // odczyt punktów tworzących prostokąt wokół twarzy  
        int x1 = (macierz_detekcji.at<float>(i, 3) * obraz_testowany.cols);  
        int y1 = (macierz_detekcji.at<float>(i, 4) * obraz_testowany.rows);  
        int x2 = (macierz_detekcji.at<float>(i, 5) * obraz_testowany.cols);  
        int y2 = (macierz_detekcji.at<float>(i, 6) * obraz_testowany.rows);  
  
        if (x1 < 0 || y1 < 0 || x2 > 1280 || y2 > 720)  
            break;  
        // zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze  
(czyli właśnie prostokąty otaczające twarze)  
        twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));  
        //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni  
element wektora twarze: twarze.back()
```

```
Mat przeskalowana_twarz;
```

```
Mat przeskalowana_twarz_szara;
```

```

        //funkcja rysujaca prostokat na ekranie
        rectangle(obraz_testowany, twarze.back(), CV_RGB(0, 255, 100), 1,
LINE_8, 0);
        //resize(twarze, przeskalowana_twarz, Size(img_width, img_height), 1.0,
1.0,1);

        resize(obraz_testowany(twarze.back()), przeskalowana_twarz,
Size(img_width, img_height), 1.0, 1.0, 1);

        imshow("przeskalowana twarz", przeskalowana_twarz);
        cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara,
COLOR_BGR2GRAY);

        int plabel;
        double confidence;

model->predict(przeskalowana_twarz_szara, plabel, confidence);

cout << "etykieta " << plabel << endl << "odleglosc " << confidence << endl;

string box_text = format("Predykcja = %d", plabel);

int pos_x = max(twarze[i].x - 10, 0);
int pos_y = max(twarze[i].y - 10, 0);

putText(obraz_testowany, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

}

}

imshow("test", obraz_testowany);

}

}

int rozpoznawanie_twarzy_obrazy() {

```

```

//tworzenie facerecognizer

Ptr<FaceRecognizer> model = FisherFaceRecognizer::create();
//Ptr<FaceRecognizer> model = EigenFaceRecognizer::create();
//Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();

//wersja 1: tworzenie modelu z pliku csv
vector<Mat> images;
vector<int> labels;

string filename = "face_recognition_video.csv";

try {
    read_csv(filename, images, labels);
}
catch (cv::Exception& e) {
    cerr << "Problem z otwarciem pliku '" << filename << "'. Powod: " << e.msg << endl;
    exit(1);
}

model->train(images, labels);

//zapisywanie modelu do pliku yaml
string save_filename = "model.yaml";
model->save(save_filename);

string obraz_testowy_nazwa = "angelina_jolie.jpg";

Mat testSample = imread("magda_720_1280_bezokularow_wyciety 010.jpg", 0);

int img_width = testSample.cols;
int img_height = testSample.rows;

Mat obraz_testowy = imread(obraz_testowy_nazwa, 0);
Mat obraz_testowy_przeskalowany;

```

```

resize(obraz_testowy, obraz_testowy_przeskalowany, Size(img_width, img_height), 1.0, 1.0, 1);

if (!obraz_testowy.data)
{
    cout << "Nie odnaleziono " << obraz_testowy_nazwa << ".";
    return -2;
}

//rozpoznawanie twarzy i wyrzucanie numeru
int plabel = model->predict(obraz_testowy_przeskalowany);
cout << plabel;

int i = 0;
imshow("okno testujące", obraz_testowy);

/* imshow("odpowiedz sieci", images[7 * plabel + (i % 7)]); //ta 7 to ile obrazow zaladowanych

while (waitKey(0) != 27)
{
    i++;
    imshow("odpowiedz sieci", images[7 * plabel + (i % 7)]);
}

*/
return 0;
}

int rozpoznawanie_twarzy_obrazy_ladowanie() {

    // stworzenie i załadowanie parametrów detektora twarzy
    Net net = cv::dnn::readNetFromTensorflow("opencv_face_detector_uint8.pb",
    "opencv_face_detector.pbtxt");

    Mat testSample = imread("Arnold_Schwarzenegger_0032_56_46.jpg", 0);

    int img_width = testSample.cols;
    int img_height = testSample.rows;

    int ilosc_zdjec = 48;
}

```

```

int nr_fotki = 24;

for(int i = 0; i < ilosc_zdjec; i++)
{
    ++nr_fotki;
    Mat obraz = imread(format("Arnold_Schwarzenegger_%04d.jpg", nr_fotki), 1);

    if (obraz.cols == 0) {
        cout << "Error reading file " << endl;
        return -1;
    }

    Mat obraz_przeskalowany;

    try {
        resize(obraz, obraz_przeskalowany, Size(img_width, img_height), 1.0, 1.0, 1);
    }
    catch (cv::Exception& e) {
        cerr << e.msg << endl;

        exit(1);
    }

    //przetworzenie obrazu na format dostosowany do głębszej sieci neuronowej wykrywającej
    twarze

    Mat blob = cv::dnn::blobFromImage(obraz_przeskalowany, 1.0, Size(img_width, img_height),
Scalar(104.0, 177.0, 123.0), true, false);

    // uruchomienie głębszej sieci neuronowej wykrywającej twarze
    net.setInput(blob, "data");
    Mat detekcje = net.forward("detection_out");
    // przeformatowanie wyniku: tyle wierszy ile twarzy,
    // w każdym wierszu: zero, nr klasy wykrytego obiektu (ta sieć wykrywa jeden obiekt - twarz,
    więc tam zawsze jest 1),
    // pewność wykrycia oraz znormalizowane do skali 0-1 współrzędne 2 wierzchołków
    prostokąta zawierającego obiekt (x1, y1, x2, y2)
    Mat macierz_detekcji(detekcje.size[2], detekcje.size[3], CV_32F, detekcje.ptr<float>());

```

```

vector<Rect> twarze;
for (int i = 0; i < macierz_detekcji.rows; i++)
{
    if (macierz_detekcji.at<float>(i, 2) > 0.6) //pewność wykrycia (w skali 0-1)
    {
        // odczyt punktów tworzących prostokąt wokół twarzy
        int x1 = (macierz_detekcji.at<float>(i, 3) * obraz.cols);
        int y1 = (macierz_detekcji.at<float>(i, 4) * obraz.rows);
        int x2 = (macierz_detekcji.at<float>(i, 5) * obraz.cols);
        int y2 = (macierz_detekcji.at<float>(i, 6) * obraz.rows);

        // zbudowanie prostokąta i wpisanie go do wektora zawierającego twarze
        // (czyli właśnie prostokąty otaczające twarze)
        twarze.push_back(Rect(Point(x1, y1), cv::Point(x2, y2)));
        //narysowanie ostatnio dodanego prostokąta do wektora twarzy - ostatni
        element wektora twarze: twarze.back()

        Mat przeskalowana_twarz;
        Mat przeskalowana_twarz_szara;

        //funkcja rysująca prostokąt na ekranie
        rectangle(obraz, twarze.back(), CV_RGB(0, 255, 100), 1, LINE_8, 0);
        //resize(twarze, przeskalowana_twarz, Size(img_width, img_height), 1.0,
        1.0,1);

        resize(obraz(twarze.back()), przeskalowana_twarz, Size(img_width,
        img_height), 1.0, 1.0, 1);

        imshow("przeskalowana twarz", przeskalowana_twarz);
        cvtColor(przeskalowana_twarz, przeskalowana_twarz_szara,
        COLOR_BGR2GRAY);

        //imwrite(format("Arnold_Schwarzenegger_twarz_%04d.jpg", nr_fotki++),
        przeskalowana_twarz);
        //imwrite(format("David_Beckham_twarz_%04d.jpg", nr_fotki++),
        przeskalowana_twarz);
        imwrite(format("Luiz_Inacio_Lula_da_Silva_twarz_%04d.jpg", nr_fotki),
        przeskalowana_twarz);
    }
}

```

```
        }
        imshow("test", obraz);
    }
    return 0;
}

int main()
{
    //rozpoznawanie_twarzy_kamera();
    //rozpoznawanie_twarzy_kamera_ladowanie();
    //rozpoznawanie_twarzy_obrazy();
    //rozpoznawanie_twarzy_video_tensorflow();
    //rozpoznawanie_twarzy_obrazy_ladowanie();
    //rozpoznawanie_twarzy_video_haar();
    rozpoznawanie_twarzy_obrazy_test();
}
```