

POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI



PRACA MAGISTERSKA

System bezpieczeństwa i zarządzania domem w IoT

Autor:

inż.

Mateusz Szymkowiak

Promotor:

dr inż.

Michał Melosik

9 września 2018

Streszczenie

W pracy zaprezentowano system bezpieczeństwa i zarządzania domem w Internet of Things. Głównymi obiektami badań, na którym skupiono się w tej pracy są systemy detekcji oraz rozpoznawania twarzy. Na potrzeby tej pracy utworzono aplikację internetową umożliwiającą sprawne testowanie wybranych algorytmów, sieci neuronowych oraz usług pozwalających na analizę obrazu. Z powodu niewystarczającej mocy obliczeniowej platformy Raspberry Pi, przetestowano środowiska Azure oraz AWS pozwalające na uruchomienie aplikacji w chmurze.

Abstract

english

Spis treści

1	Wstęp	6
2	Cel i zakres prac	7
3	Wstęp teoretyczny	8
3.1	Technologie	8
3.2	Raspberry Pi	10
3.3	Przetwarzanie obrazu	11
3.4	Azure	15
3.5	AWS	16
4	Przegląd dostępnych metod rozpoznawania twarzy	17
4.1	Open Cv	18
4.2	Azure Cognitive Services	21
4.3	AWS Rekognition	21
5	System zarządzania metodami rozpoznawania twarzy	22
5.1	Budowa aplikacji	22
5.2	Konfiguracje uruchomieniowe	23
6	Aplikacja internetowa	24
6.1	Technologie	25
6.2	Detekcja twarzy	26
6.3	Profile	28
6.4	Sieci neuronowe	29
6.5	Rozpoznawanie tożsamości	31
6.6	Odczyty sensorów	32
6.7	Powiadomienia	33
7	Aplikacja konsolowa	34
7.1	Technologie	34
7.2	Proces wykrywania twarzy	35

7.3	Proces trenowania sieci neuronowych cos o tej sieci napisac lub omowic jej zrodla	37
7.4	Proces rozpoznawania twarzy	40
8	Aplikacja konsolowa do zarządzania domem	44
8.1	Odczyt wartości sensorów	44
8.2	Proces detekcji ruchu	45
9	Porównanie wykorzystanych technologii TODO	48
9.1	Detekcja	48
9.2	Dane treningowe	48
9.3	Trenowanie sieci neuronowych	48
9.4	Rozpoznawanie twarzy	49
9.5	Ocena przydatności wybranych usług IoT	49
10	Podsumowanie	50
	Bibliografia	50
	Dodatki	52
	Spis rysunków	53
	Spis tablic	55

Rozdział 1

Wstęp

Internet rzeczy (Internet of Things -IoT) to koncepcja przedstawiająca sieć urządzeń fizycznych połączonych inteligentną siecią KNX lub internetową, komunikujących się między sobą i wymieniających dane. Podstawowym celem IoT jest stworzenie inteligentnych przestrzeni- miast, budynków lub systemów związanych z życiem codziennym. Jednym z zastosowań takich systemów są inteligentne domy (smart homes), wykorzystujące czujniki oraz akulatory do zarządzania domem.

Na potrzeby tej pracy opracowano system bezpieczeństwa oparty o system detekcji i rozpoznawania twarzy na obrazie.

W rozdziale „Cel i zakres pracy” przedstawiono główne założenia projektu oraz zakres prac autora.

W rozdziale „Wstęp teoretyczny” omówiono podstawowe zagadnienia związane z treścią tej pracy magisterskiej.

W rozdziale „Przegląd dostępnych metod rozpoznawania twarzy” porównano możliwości kilku z dostępnych usług lub oprogramowania pozwalającego na rozpoznawanie twarzy.

W rozdziale „System zarządzania metodami rozpoznawania twarzy” przedstawiono strukturę stworzonej aplikacji, jej podział na moduły oraz wybrane dla każdego z nich środowisko uruchomieniowe.

W rozdziale „Aplikacja internetowa” zaprezentowano wszystkie strony utworzone na potrzeby projektu.

W rozdziale „Aplikacja konsolowa” omówiono sposób wykorzystania wcześniej opisanych usług oraz algorytmy odpowiedzialne za działanie systemu do testowania różnych rozwiązań dla problemu identyfikacji twarzy.

W rozdziale „Aplikacja konsolowa do zarządzania domem” opisano sposób wykorzystania czujników oraz kamery podłączonej bezpośrednio do platformy Raspberry Pi.

W rozdziale „Porównanie wykorzystanych technologii” zbadano oraz porównano skuteczność działania technologii wybranych na potrzeby tej pracy.

W rozdziale „Podsumowanie” zawarto podsumowanie zgromadzonych informacji oraz przedstawiono wnioski wynikające z przeprowadzonych badań.

Rozdział 2

Cel i zakres prac

Celem pracy było stworzenie systemu bezpieczeństwa i zarządzania domem w IoT. Głównymi założeniami było opracowanie systemu pozwalającego na porównanie wybranych metod i usług pozwalających na analizę obrazu oraz kontrolę stanu czujników podłączonych do systemu. IoT jest bardzo prężnie rozwijającą się ideą, dlatego postanowiono porównać wybrane technologie i usługi ułatwiające zarządzanie oraz integrację z takimi aplikacjami.

Jako platformę sprzętową wybrano bardzo popularne w zastosowaniach IoT urządzenie Raspberry Pi 3. Z powodu jego ograniczonych zasobów obliczeniowych zintegrowano chmurowe usługi Azure oraz AWS. Zakres pracy obejmuje następujące zagadnienia:

- dobór odpowiedniej platformy sprzętowej oraz oprogramowania,
- porównanie działania wybranych algorytmów detekcji oraz rozpoznawania twarzy,
- wybór i porównanie wybranych technologii umożliwiających integrację z systemami IoT,
- wykrywanie zdarzeń rejestrowanych przez wybrane czujniki,
- opracowanie systemu przesyłania powiadomień,
- ocena przydatności wybranych usług w zastosowaniu dla systemu bezpieczeństwa domu w IoT.

Rozdział 3

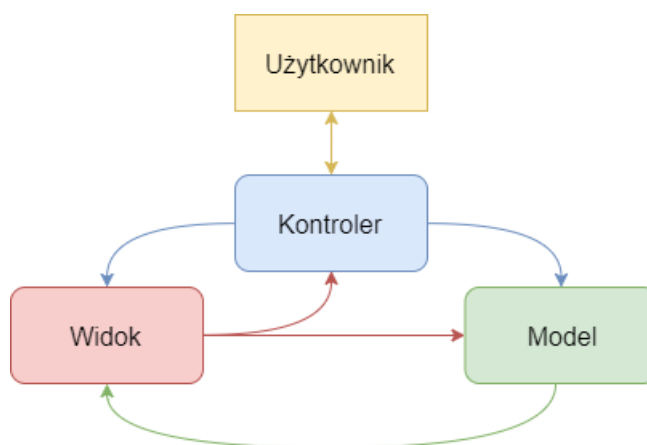
Wstęp teoretyczny

3.1 Technologie

3.1.1 Wzorzec projektowy MVC (Model-View-Controller)

Strona internetowa powstała na bazie bardzo popularnego wśród programistów wzorca projektowego MVC. Założenia wzorca Model-Widok-Kontroler przedstawionego na rysunku 3.1 są bardzo proste, ich składowymi są:

- Model- reprezentuje logikę biznesową. Tutaj znajdują się wszelkie obiekty, które służą do wykonywania zaimplementowanej funkcjonalności danej aplikacji,
- Widok- jest warstwą prezentacji. Odpowiada za prezentację logiki biznesowej (Modelu) użytkownikowi w przystępny sposób,
- Kontroler- obsługuje żądania użytkownika. Odebrane zadania oddelegowuje do odpowiednich modeli.



Rysunek 3.1: Schemat klasycznego wzorca MVC

3.1.2 Single Page Application

Single Page Application (SPA) to aplikacja lub strona internetowa, która w całości wczytuje się za jednym razem. Cały potrzebny do działania strony kod (HTML, CSS, JavaScript) przesyłany jest na początku lub dodawany dynamicznie w kawałkach, zwykle w odpowiedzi na interakcje generowane przez użytkownika. Sposób działania takiej aplikacji jest zbliżony do odczuć towarzyszących korzystaniu z aplikacji desktopowej lub mobilnej.

3.1.3 Chmura obliczeniowa

Chmura obliczeniowa jest określeniem oznaczającym model przetwarzania danych oparty na użytkowaniu usług dostarczonych przez usługodawcę. Reprezentację budowy takiego systemu przedstawiono na rysunku 3.2. Użytkownik nie musi opłacać licencji, a płaci jedynie za użytkowanie danej usługi. W przypadku takich usług najpopularniejszym sposobem naliczania kosztów jest opłata od czasu użytkowania usługi. Do najpopularniejszych udostępnianych usług należą:

- bazy danych,
- maszyny wirtualne,
- serwery,
- hosting dla aplikacji webowych.

Przedstawicielami usługodawców udostępniających szeroki zakres usług, które zostały wykorzystane podczas tworzenia tej pracy magisterskiej jest Azure od Microsoft oraz AWS stworzony przez Amazon. Usługi obu podmiotów, które mogą być przydatne podczas tworzenia rozwiązania zostały szerzej opisane w podpunkcie 3.4 i 3.5.



Rysunek 3.2: Obliczenia chmurowe

Źródło: <http://www.justscience.in>

3.1.4 1-Wire

One Wire to systemowa magistrala komunikacji elektronicznej pomiędzy urządzeniami, zapewniająca przesyłanie danych oraz zasilanie urządzenia przez pojedynczy kabel. Proces ten jest możliwy dzięki stopniowemu ładowaniu kondensatora znajdującego się w odbiorniku, a następnie wykorzystanie zgromadzonej energii do zasilenia urządzenia. Do magistrali może zostać podłączonych wiele urządzeń. Każdemu z nich przydzielany jest indywidualny adres 64-bitowy. Komunikację z urządzeniami inicjuje master, w tym przypadku Raspberry Pi. Przedstawiony protokół jest bardzo podobny do I2C, ale ze względu na wykorzystanie jedynie jednej linii danych, charakteryzuje się niższą prędkością przesyłania. Układ zazwyczaj zasilany jest napięciem o wartości 5V i służy do komunikacji pomiędzy niewielkimi urządzeniami, takimi jak np. termometr cyfrowy i mikrokontroler.

3.2 Raspberry Pi

Raspberry Pi jest platformą komputerową stworzoną przez Raspberry Pi Foundation, na którą składa się pojedynczy obwód drukowany widoczny na zdjęciu 3.3. Pierwsza wersja tego urządzenia została zaprezentowana w 2012 roku. Na potrzeby tej pracy wykorzystano nowszą wersję urządzenia w wersji 3 B, którą została wyposażona w 4 rdzeniowy procesor i 1 GB pamięci RAM. Raspberry pozwala na podłączenie wielu urządzeń peryferyjnych za pomocą 4 portów USB lub 40 pinów GPIO. Dodatkowe złącze pozwala na podłączenie dedykowanej kamery. Dzięki znacznej ilości portów GPIO istnieje możliwość komunikacji cyfrowej z sensorami. Budowa urządzenia nie pozwala na podłączenie czujników analogowych.



Rysunek 3.3: Raspberry Pi 3 B

Źródło: <https://www.amazon.ca>

3.2.1 Czujnik DHT 11



Rysunek 3.4: Czujnik DHT11

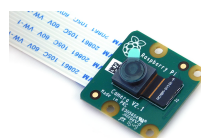
Czujnik temperatury i wilgotności powietrza DHT11 jest bardzo popularnym czujnikiem z interfejsem cyfrowym. Do komunikacji wykorzystuje on protokół 1-wire opisany w rozdziale 3.1.4.

Tablica 3.1: Parametry czujnika

Parametr	Wartość
Napięcie zasilania	3,3V do 5,5V
Średni pobór prądu	0,2 mA
Zakres pomiaru temperatury	od - 20°C do +50°C
Zakres pomiaru wilgotności	od 5% do 95% wilgotności względnej

3.2.2 Raspberry Pi Camera HD

Raspberry Pi Camera HD jest dedykowaną kamerą przeznaczoną tylko i wyłącznie do urządzenia Raspberry Pi w wersji 3, 2 oraz B+. Szybszy transfer danych niż przez port USB zapewnia dedykowane złącze minikomputera (w formie taśmy na zdjęciu 3.5). Kamera posiada matrycę o rozdzielczości 8 Mpx, wspiera tryb HD 1090p, 720p oraz 640 x 480. Moduł umożliwia wykonywanie zdjęć w rozdzielczości nawet 3280 x 2464 px. Wymagane sterowniki są preinstalowane na kompatybilnych urządzeniach.



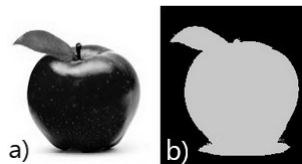
Rysunek 3.5: Raspberry Pi Camera Hd

3.3 Przetwarzanie obrazu

3.3.1 Thresholding **TODO**

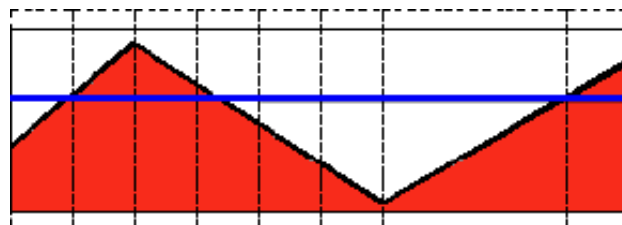
Thresholding jest operacją polegającą na segmentacji obrazu. Segmentacja to proces podziału obrazu na części określane jako obszary, które są jednorodne pod względem pewnych wybranych

własności. Podczas thresholdingu dochodzi do porównania wartości pixeli, tak by oddzielić ważne piksele od tych mniej interesujących. Przed operacją ustalana jest granica, która będzie stanowić o tym jak rozdzielone zostaną wartości.

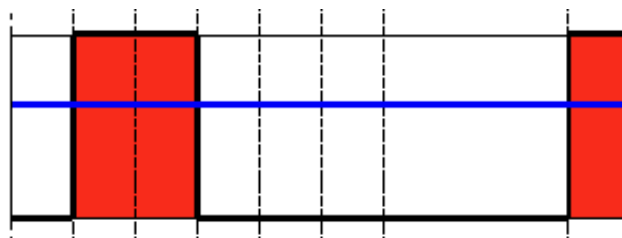


Rysunek 3.6: Efekt działania thresholdingu a) przed b) po

Zakładając, że obraz wejściowy to src , wartość danego piksela oznaczono jako $src(x,y)$, a przyjętą granicę jako $thresh$. Na rysunku 3.7 niebieską linią oznaczono $thresh$, a czerwonym kolorem obszar oznaczający wartość piksela.



Rysunek 3.7: Wykres wartości obrazu src przed thresholdingiem



Rysunek 3.8: Wykres wartości pikseli obrazu dst

W tej pracy zastosowano thresholding binarny konwertujący wartość każdego piksela do minimalnej lub maksymalnej. Działanie funkcji przedstawiono za pomocą poniższego wzoru.

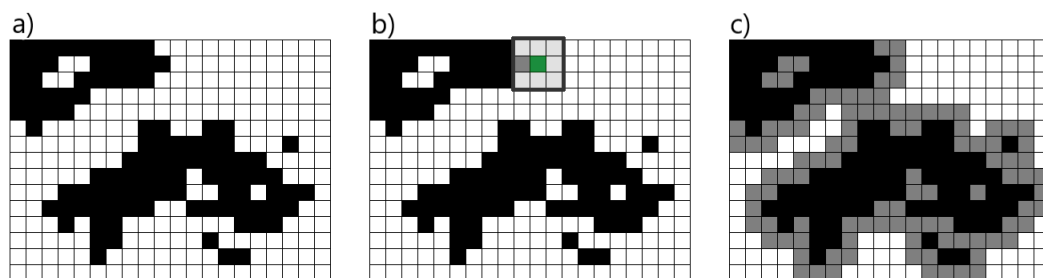
$$dst(x,y) = \begin{cases} maxValue & \text{gdy } src(x,y) > thresh \\ 0 & \text{pozostałe} \end{cases} \quad (3.1)$$

POPRAWIĆ WZÓR I OPISgdzie $maxValue$ - wartość maksymalna piksela
 dst - plik wyjściowy

Porównując rysunki 3.7 oraz 3.8 można zaobserwować że zgodnie ze wzorem 3.1 każdy piksel obrazu wejściowego poniżej wartości $thresh$ (niebieska linia), na obrazie wyjściowym ma wartość 0, a powyżej linii ma wartość maksymalną.

3.3.2 Dylatacja

Dylatacja jest operacją wykorzystywaną do przetwarzania obrazów binarnych, określaną rozszerzaniem. Proces dylatacji polega na przyłożeniu do każdego pixela na obrazie struktury o wybranym rozmiarze. Jeżeli choć jeden pixel sąsiadujący z punktem centralnym wybranej struktury ma wartość jeden to punkt centralny również otrzymuje taką samą wartość. W przeciwnym wypadku punktowi przypisywana jest wartość zero. Najkorzystniejsze efekty przynosi dylatacja strukturą przypominającą kształt koła, np 3x3 px. W efekcie dylatacji dochodzi do zwiększenia się obiektu, zniknięcia detali oraz wypełnienia pustek w niespójnym obszarze. Na rysunku 3.9 b) do obrazu przyłożono strukturę 3x3 px. W jej obszarze znaleziono jeden pixel o wartości maksymalnej (oznaczony szarym kolorem), dlatego pikselowi oznaczonemu kolorem zielonym zostaje przydzielona ta sama wartość. Przeprowadzenie operacji na całym obrazie, powoduje uzyskanie rysunku 3.9 c).



Rysunek 3.9: Przebieg przykładowej dylatacji a) przed b) w trakcie c) po
Źródło: https://pl.wikipedia.org/wiki/Cyfrowe_przetwarzanie_obraz%C3%B3w_binarnych

3.3.3 Rozmycie gaussowskie **DODAC wzór i więcej opisu**

Rozmycie gaussowskie inaczej nazywane wygładzaniem gaussowskim to operacja przetwarzania obrazu polegająca na modyfikacji go z użyciem filtru Gaussa. Rozmycie wykorzystywane jest w celu zmniejszenia szumów oraz zakłóceń w obrazie oraz w celu zamazania detali.



Rysunek 3.10: Przykład zastosowania rozmycia gaussowskiego na zdjęciu

3.3.4 Klasyfikator kaskad Haar’a

Detekcja obiektów klasyfikatorem kaskad Haar’a została oparta o metodę Paul’a Viola i Michael’a Jones’a – w skrócie Viola-Jones.

Algorytm Viola-Jones jest jednym z pierwszych algorytmów pozwalających na osiągnięcie zadowalających wyników w detekcji obiektów na obrazie. Został on zaproponowany w 2001 roku i został przemyślany z głównym przeznaczeniem dla detekcji twarzy. Na kluczowe koncepcje tej metody składają się:

- wyszukiwanie cech Haar’a,
- integralność obrazu,
- metoda uczenia AdaBoost (podstawowy algorytm do boostingu, metoda dzięki której z dużej liczby słabych klasyfikatorów można otrzymać jeden lepszy)

Cechy wykorzystywane w metodzie Viola-Jones opierają się na falkach Haara. Falki Haara są to sekwencje przeskalowanych kwadrato-podobnych funkcji, które razem tworzą falę (falo-podobną oscylację), podstawę z której można zbudować kwadrat. W dwóch wymiarach, fala kwadratu jest parą przylegających do siebie prostokątów, gdzie jeden jest jaśniejszy, a drugi ciemniejszy. Podstawowe szablony przedstawiono na rysunku 3.11.

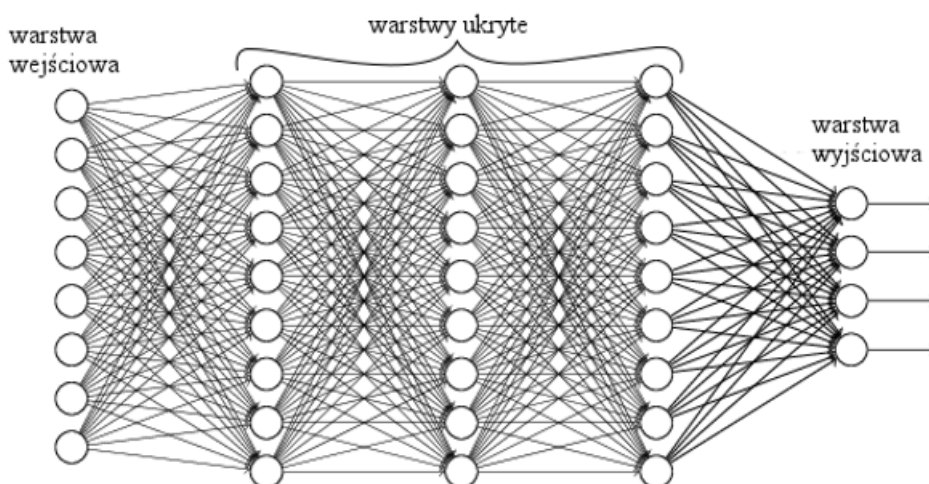


Rysunek 3.11: Szablony cech Haar’a a) dwuprostokątny b) trójpłaski c) czteropłaski d) centralny

Wartość każdej cechy jest obliczana jako różnica sumy poziomów szarości pikseli pokrywanych przez biały oraz czarny prostokąt oraz sumy poziomów szarości pikseli pokrywanych przez czarny prostokąt. Składniki tej różnicy posiadają wagi odwrotnie proporcjonalne do rozmiarów, dzięki czemu różnice wielkości dwóch obszarów są kompensowane. **MOŻE JESZCZE WZÓR?**

3.3.5 Głęboka sieć neuronowa

Pojęciem uczenia maszynowego określa się dziedzinę nauk związanych ze sztuczną inteligencją, zajmującą się badaniem algorytmów i systemów, które usprawniają swoje działanie wraz ze zdobywaniem nowej wiedzy lub też doświadczeniem. Wiedzą określamy dane uczące, które zostały wykorzystane do nauki. System może zostać usprawniony poprzez zwiększenie wiedzy systemu, które powinno pozwolić na usprawnienie procesu rozwiązywania podstawowych problemów.



Rysunek 3.12: Budowa głębokiej sieci neuronowej

Algorytmy głębokiego uczenia maszynowego są jednymi z najbardziej zaawansowanych. Zbudowane są one w sposób przypominający biologiczne sieci neuronowe. Głęboka sieć neuronowa składa się z neuronów rozmieszczonych na warstwach. Wiele warstw jest powodem dla którego taki rodzaj algorytmu określa się mianem głębokiego. Przykładowa budowa głębokiej sieci neuronowej została przedstawiona na rysunku 3.12.

Każdy z takich “sztucznych neuronów” jest tak naprawdę informacją, która jest przesyłana dalej do następnych neuronów i ich warstw. Poszczególne warstwy “uczą” się przetwarzać kolejne cechy obiektów/obrazów/dźwięków itp., dzięki czemu są w stanie odtwarzać całe obiekty w bardzo rzeczywisty sposób. W tej pracy wykorzystano głęboką sieć neuronową do zadania wykrywania twarzy na obrazie.

3.4 Azure

Azure jest platformą chmurową firmy Microsoft stworzoną w modelu Paas (Platform as a Service). Platforma zbudowana jest z grupy trzech technologii zapewniających specjalizowany zestaw możliwości dla programistów, które mogą być wykorzystywane zarówno przez aplikacje uruchamiane lokalnie na komputerach użytkowników oraz aplikacje uruchamiane w chmurze. W tej pracy wykorzystano:

- App Services,
- Azure Cognitive Services,
- bazy danych SQL oraz NoSQL (MSSQL, MySql, CosmicDB)
- maszyny wirtualne oparte o system Windows oraz kilka popularnych dystrybucji Linuxa.

3.4.1 Web App Services

Web App Services jest usługą pozwalającą na hostowanie aplikacji internetowych napisanych w jednym z wybranych języków (między innymi .Net, Node.js, Python) bez zarządzania infrastrukturą. Do największych zalet należy automatyczna skalowalność serwisu oraz wysoka dostępność. Podczas tworzenia środowiska istnieje możliwość wyboru preferowanego systemu operacyjnego. Dużą zaletą jest możliwość wdrożenia aplikacji za pomocą jednego kliknięcia bezpośrednio z Visual Studio. Sprawne korzystanie z usługi jest możliwe dzięki szerokiemu zakresowi dokumentacji i tutoriali udostępnionych przez producenta.

3.5 AWS

Amazon Web Services jest platformą chmurową firmy Amazon stworzoną w modelu Paas (Platform as a Service). Jest to bardzo podobne rozwiązanie do bliźniaczego Azure. Platforma udostępnia obliczenia chmurowe na żądanie. Szeroki zakres usług dostępny jest w darmowej wersji demo, jedynym wymaganiem jest założenie konta w serwisie. Po wykorzystaniu darmowego okresu, opłaty naliczane są według zużycia danej usługi. Do podstawowych usług, które mogłyby zostać wykorzystane dla celów tej pracy należą:

- Elastic Beanstalk,
- Rekognition,
- bazy danych (RDS),
- maszyny wirtualne (EC2).

3.5.1 Elastic Beanstalk

Elastic Beanstalk jest odpowiednikiem Azure Web App Services. Zakres dostępnych języków jest równie szeroki jak w przypadku Azure'a, ale w odróżnieniu od Microsoftu nie zawsze wspiera on najnowszych wersji produktów wymienionej firmy, a zakres możliwości konfiguracyjnych z poziomu strony internetowej jest uboższy. Podobnie jak w przypadku Azure istnieje możliwość wdrożenia aplikacji jednym kliknięciem po doinstalowaniu dedykowanego pluginu do Visual Studio.

Rozdział 4

Przegląd dostępnych metod rozpoznawania twarzy

Przez ostatnie lata dziedzina cyfrowego przetwarzania obrazu bardzo prężnie rozwijała się. W wyniku tego aktualnie dostępnych jest wiele usług pozwalających na uzyskiwanie danych z obrazu. W przypadku tej pracy najbardziej użytecznymi informacjami jest lokalizacja oraz identyfikacja twarzy. Na rysunku 4 przedstawiono cechy kilku wybranych systemów, stworzonych przez jedne z największych firm zajmujących się technologiami informatycznymi na świecie.

Tablica 4.1: Dostępne systemy przetwarzania obrazu

	OpenCv	Azure CS	AWS Rekognition	Google	face recognition	open face
Detekcja twarzy	tak	tak	tak	tak	tak	tak
Identyfikacja twarzy	tak	tak	tak	nie	tak	tak
Rozpoznawanie emocji	nie	tak	nie	nie	nie	nie
SDK	tak	tak	tak	tak	Python	Python
API	nie	tak	tak	tak	nie	nie
licencja	open source	płatna/demo	płatna/demo	płatna/demo	open source	open source

Zgodnie ze wzrostem popularności usług chmurowych, aktualnie dostępnych jest wiele usług rozpoznawania twarzy, które początkowo były dostępne za pomocą API(Application Programming Interface). Z czasem większość z nich udostępniła również SDK(Software Development Kit) dla najbardziej popularnych języków (między innymi C#, Java, Python). OpenCv jest podstawową open sourceową bibliotekę pozwalającą na identyfikację twarzy. Większość darmowych rozwiązań dostępnych jest na platformie GitHub:

- face_recognition (źródło: https://github.com/ageitgey/face_recognition)
- open face (źródło: https://github.com/ageitgey/face_recognition)

, ale niestety żadna z nich nie mogła równać się dojrzałością podobną do OpenCv, a na dodatek większość z nich można wykorzystać jedynie tworząc oprogramowanie w języku Python. dostępna jest jedynie dla Pythona (face recognition oraz open face).

Poza usługami przedstawionymi w powyższej tabeli, równie ciekawym rozwiązaniem, aczkolwiek znacznie bardziej rozbudowanym i czasochłonnym jest wykorzystanie Tensorflow (framework do machine learningu) w celu zbudowanie własnej implementacji sieci neuronowej rozpoznającej twarze. Podstawowym frameworkiem, o którego użyciu zdecydowano zostało OpenCv. Przemawiała za tym open sourcowa licencja, dojrzałość frameworku oraz liczne źródła wiedzy o implementacji. Podstawowe dostępne funkcje i zastosowania opisano w rozdziale 4.1.

W celu możliwości porównania rozwiązań lokalnych i chmurowych za drugi obiekt badań obrano usługę Azure Cognitive Services, którą szerzej opisano w rozdziale 4.2. O wyborze tego rozwiązania zdecydowała rozbudowana dokumentacja usługi oraz licencja studencka umożliwiająca darmowe wykorzystanie usługi przy zwiększonych limitach zapytań.

4.1 Open Cv

OpenCv (Open Source Computer Vision Library) jest open sourcowa biblioteką napisaną w języku C. Udostępniono liczne interfejsy biblioteki pozwalające na pracę z nią między innymi w języku C++ i Python. Biblioteka wspiera systemy operacyjne Linux oraz Windows. Biblioteka została ukierunkowana na przetwarzanie obrazu w czasie rzeczywistym. W licznie udostępnionych funkcjach można znaleźć moduły pozwalające na detekcję i rozpoznawanie twarzy na obrazie, które zostały szerzej opisane w kolejnym punkcie.

4.1.1 Rozpoznawanie twarzy **ZRODŁA WZORY I CYTOWANIA**

Przed zidentyfikowaniem tożsamości, należy najpierw rozwiązać problem detekcji twarzy. Podstawowym sposobem detekcji twarzy wykorzystywanym przez OpenCv jest kaskadowy klasyfikator Haar’a oparty o cechy Haar’a opisane w rozdziale 3.3.4. Do innych dostępnych rozwiązań wykorzystanych do stworzenia aplikacji należy wytrenowany model głębokiej sieci neuronowej przygotowany przez autorów OpenCv.

Eigenfaces

Eigenfaces, inaczej nazywany algorytmem twarzy własnych opiera się na metodzie analizy głównych składowych, która rozpoczyna się wyznaczeniem średniej wartości dla obrazów z tą samą etykietą identyfikującą daną osobę. Weźmy n wektorów x_i powstałych jako reprezentacje obrazów przedstawiających daną twarz. Dla tych wektorów wyznaczona zostaje wartość średnia μ .

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Następnie dla każdego wektora x_i wyznaczana jest różnica od wartości średniej μ .

$$\psi = x_i - \mu$$

Na podstawie wartości ψ_i wyznacza się macierz kowariancji C .

$$C = \frac{1}{n} \sum_{i=1}^n \psi_i \psi_i^T$$

Kolejnym krokiem jest wyznaczenie wartości λ_i oraz v_i tak by spełniony był warunek:

$$C v_i = \lambda_i v_i \text{ dla } i = 1, 2, \dots, n$$

Wartość λ_i nazywana wartością własną macierzy C , a odpowiadający jej wektor v_i wektorem własnym C . W przypadku algorytmu Eigenfaces branych jest pod uwagę k głównych składowych λ_i , wyznaczonych na podstawie odrzucenia $n - k$ wartości najmniejszych. Dla każdego wektora wyznaczany jest wektor następujący:

$$y_i = W^T (x_i - \mu) \text{ gdzie } W = (v_1, v_2, \dots, v_k)$$

Identyfikacja twarzy algorytmem Eigenfaces polega na wykonaniu kroków:

1. Wartości głównych składowych zostają wyznaczone dla wszystkich obrazów uczących.
2. Wartość głównych składowych zostaje wyznaczona dla badanego zdjęcia
3. Dla obrazu wejściowego zostaje wyznaczony wektor najbliższy dla wartości obliczonych dla obrazów z bazy danych.

Fisherfaces

Algorytm Fisherfaces 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. Wyznaczanie poszukiwanego wektora rozpoczyna się od wyznaczenia wartości średniej μ wszystkich obiektów znajdujących się w rozpatrywanym zbiorze, oraz wartości średniej μ_i wewnątrz poszczególnych klas.

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

Gdzie N oznacza liczebność rozpatrywanego zbioru x_i

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j$$

Gdzie X_i jest klasą obiektów o indeksie i , dla $i = 1, 2, \dots, n$ dla n oznaczającego liczbę klas. Następnie wyznaczana jest macierz rozproszenia wewnątrzklasowego S_B oraz macierz średniego rozproszenia wewnątrzklasowego S_W .

$$S_B = \sum_{i=1}^n N_i (\mu - \mu_i) (\mu - \mu_i)^T$$

$$S_W = \sum_{i=1}^n \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^T$$

Rozwiązanie problemu sprowadza się do znalezienia danych, które pozwolą na otrzymanie największej wartości określającej stosunek rozproszenia wewnątrzklasowego do średniego wewnętrznego rozproszenia klas. Stąd algorytm poszukuje wektora d , dla którego poniższa funkcja osiąga maksimum.

$$f(d) = \frac{d^T S_B d}{d^T S_W d}$$

W celu zmniejszenia złożoności problemu, stosuje się modyfikację powyższego kryterium, wykorzystując macierz W złożoną z k głównych wektorów własnych wyznaczonych dla macierzy kowariancji C wszystkich elementów zbioru.

$$f(d) = \frac{d^T W^T S_B d}{d^T S_W d}$$

Znalezienie wektora d pozwala na wyznaczenie optymalnego kierunku rozdzielającego dwie klasy obiektów.

Local Binary Patterns Histograms

W odróżnieniu od holistycznego podejścia w dwóch poprzednich metodach, algorytm histogramów lokalnych binarnych wzorców 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. W ogólności wartość funkcji LBP dla piksela o współrzędnych (x_c, y_c) wyznacza się następująco:

$$LBP(x_c, y_c) = \sum_{p=0}^{p-1} 2^p S(i_p - i_c)$$

Gdzie p jest rozmiarem rozpatrywanego sąsiedztwa o środku w punkcie (x_c, y_c) i jasności o wartości i_c , a i_n jest wartością jasności dla n -tego punktu sąsiedztwa. $S(x)$ jest funkcją znaku definowaną następująco:

$$S(x) = \begin{cases} 1 & \text{dla } x > 0 \\ 0 & \text{w p.p.} \end{cases}$$

Działanie algorytmu polega na podzieleniu obrazu wejściowego na m różnych części i wyznaczenia dla niego wartości LBP jasności pikseli. Następnie dla każdego regionu wyznaczany jest histogram wyliczonych wartości. Tak wyznaczone wartości są konkatelowane do postaci wektora. Dla próbki wejściowej zostaje wyznaczony wektor histogramów, który następnie jest porównywany z wektorami obrazów użytych do uczenia sieci. Przewidywana etykieta jest wyznaczana na podstawie etykiety wektora najbliższego sąsiada.

4.2 Azure Cognitive Services

Cognitive Services jest częścią platformy Azure stworzonej przez Microsoft. Usługa jest standardowo płatna, ale w przypadku użytku na potrzeby studenckie przyznawany jest darmowy dostęp na ograniczony czas. Cognitive Services zajmuje się rozwiązywaniem problemów biznesowych dzięki sztucznej inteligencji. Do dostępnych modułów między innymi należą:

- obraz- algorytmy przetwarzania obrazów umożliwiające inteligentne identyfikowanie, podpisywanie i moderowanie grafik,
- mowa- konwertowanie wypowiedzi audio na tekst, weryfikacja głosowa,
- język- przetwarzanie języka naturalnego.

Na cele tego projektu wykorzystano moduł dotyczący przetwarzania obrazu, a dokładniej wykrywanie i rozpoznawanie twarzy w nim dostępne. Producent zadbał o prostą możliwość integracji z większością popularnych języków programistycznych poprzez udostępnienie paczek deweloperskich. W przypadku braku SDK dla wybranego języka istnieje możliwość skorzystania z udostępnionego REST Api. Na stronie producenta można znaleźć obszerną dokumentację oraz tutoriale.

4.2.1 Detekcja twarzy

Detekcja twarzy została opiera się o pojedyncze zapytanie do Api Azure’a. Dobierając odpowiednie parametry wejściowe klient może uzyskać znaczną ilość dodatkowych parametrów i cech związanych z przesłanym zdjęciem. W podstawowym przypadku odpowiedź serwisu ogranicza się do przydzielenia znalezionej twarzy identyfikatora oraz zwrócenia informacji o obszarze zawierającym twarz w postaci JSON’a.

4.2.2 Rozpoznawanie twarzy

Na potrzeby rozpoznawania twarzy została stworzona funkcja LargeGroup, która zgodnie ze swoją nazwą jest grupą, do której użytkownik może dodawać dowolną ilość osób oraz przypisywać każdej z nich zdjęcia. Na podstawie tak utworzonej grupy zostaje stworzony model umożliwiający rozpoznawanie tożsamości osób w niej zawartej. Sposób wykorzystania tej usługi został znacznie szerzej opisany w rozdziale 7.3.2.

4.3 AWS Rekognition

Usługa Rekognition jest odpowiednikiem Azure Cognitive Services ograniczonym do rozwiązań związanych z przetwarzaniem obrazu i video. Do wielu dostępnych funkcji należy analiza obrazu, detekcja twarzy i tekstu, porównywanie oraz rozpoznawanie twarzy. Dla wybranych języków programistycznych udostępniono SDK oraz obszerną dokumentację z licznymi przykładami kodu.

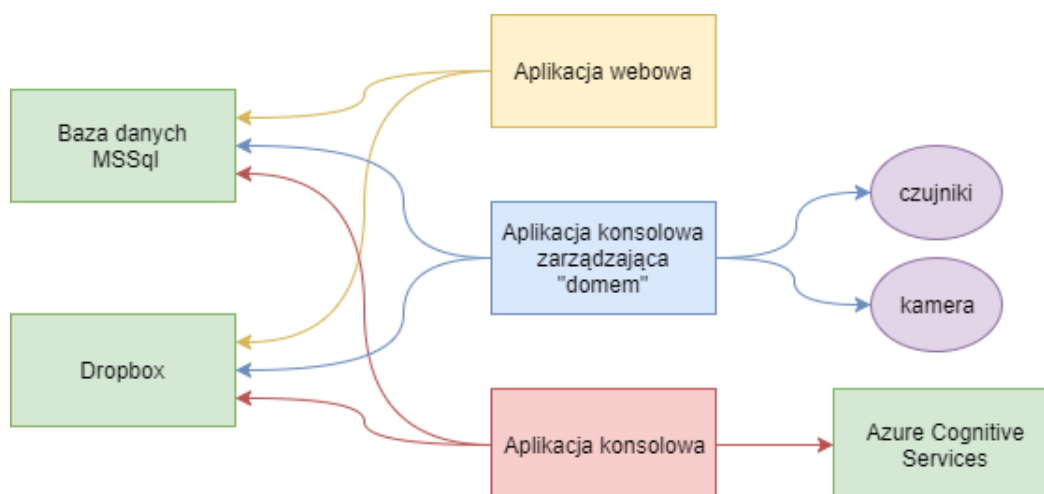
Rozdział 5

System zarządzania metodami rozpoznawania twarzy

5.1 Budowa aplikacji

Aplikację powstałą na potrzeby tej pracy można podzielić na 3 moduły, które zostały przedstawione na rysunku 5.1, a składają się na nie :

- Aplikacja webowa- interfejs pozwalający na zlecanie nowych zadań aplikacji konsolowej, odczyt wyników przesłanych przez obie aplikacje konsolowe,
- Aplikacja konsolowa- przetwarza zadania detekcji oraz rozpoznawania twarzy zlecone za pomocą aplikacji webowej,
- Aplikacja konsolowa zarządzająca domem- przekazuje cyklicznie odczytywane dane z czujników



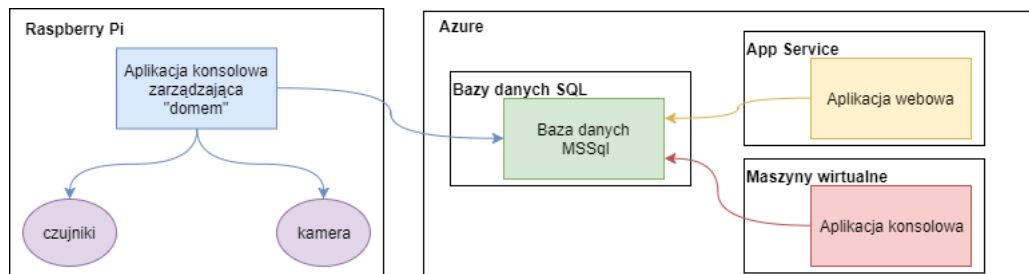
Rysunek 5.1: Budowa systemu zarządzania domem

We wczesnej fazie projektu istniała jedna aplikacja konsolowa, ale ze względu na złożoność programu przetwarzającego zadania, została ona rozdzielona na aplikację, która musi być uruchamiana na Raspberry Pi (program zarządzający domem) oraz drugą, którą można uruchomić w dowolnym innym środowisku. Na usługi pomocnicze wykorzystane w projekcie składają się

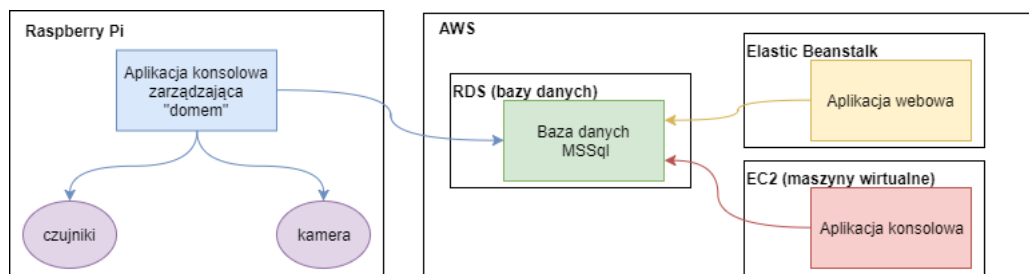
- baza danych MSSql- przechowywanie danych o dodanych zadaniach, wynikach, nauczonych sieciach neuronowych oraz osobach,
- Dropbox- przechowywanie większych plików- obrazów oraz nauczonych modeli sieci,
- Azure Cognitive Services- Web Api usługi przetwarzającej obrazy opisanej w rozdziale 4.2.

5.2 Konfiguracje uruchomieniowe

W celu zmaksymalizowania wydajności modułów, podjęto decyzję o uruchomieniu każdego z nich w odrębnym środowisku. System został uruchomiony w dwóch konfiguracjach. Niezależnie dla konfiguracji niezmienny pozostał moduł aplikacji konsolowej zarządzającej "domem", który ze względu na wymaganie fizycznego dostępu do urządzeń podłączonych do Raspberry Pi zawsze był uruchamiany w tym środowisku. Pierwsza z konfiguracji została oparta o środowisku chmurowe Azure, a druga AWS. Jakie usługi wybrano dla poszczególnych modułów przedstawiono na rysunku 5.2 i 5.3.



Rysunek 5.2: Podział systemu na usługi Azure

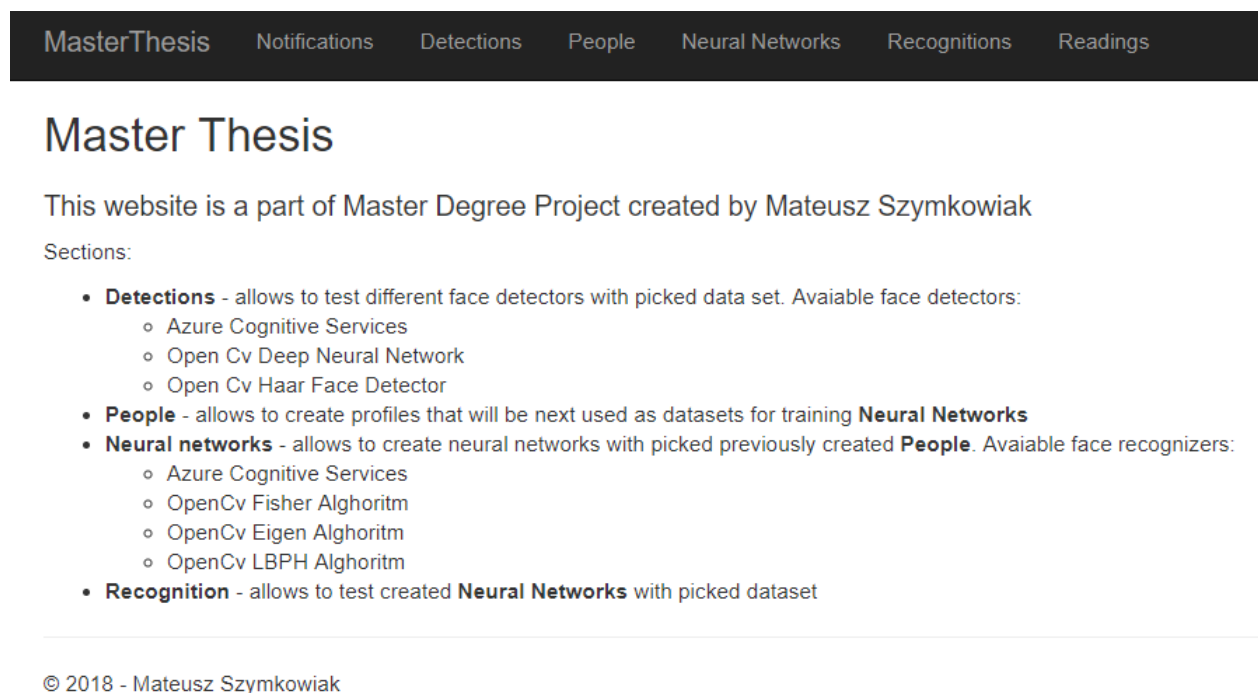


Rysunek 5.3: Podział systemu na usługi AWS

Rozdział 6

Aplikacja internetowa

Aplikacja webowa jest jedyną częścią systemu, do której użytkownik może mieć bezpośredni dostęp. Strona powstała w celu maksymalnego uproszczenia procesu badania kolejnych algorytmów i usług, które różniły się sposobem podawania danych wejściowych, sposobem uczenia oraz formatem zwracanych odpowiedzi. Do pozostałych zalet takiego rozwiązania należy ułatwienie przechowywania danych, poprzez umieszczenie ich we wspólnym miejscu co pomaga, w późniejszej interpretacji wyników.



Rysunek 6.1: Wygląd strony głównej aplikacji internetowej (rozwiązanie 1)

Zgodnie z interfejsem przedstawionym na rysunku 6.1, strona została podzielona na 5 głównych sekcji:

- Notifications- powiadomienia,
- Detections- detekcje,
- People- profile,
- Neural Networks- sieci neuronowe,
- Recognitions- identyfikacja tożsamości,
- Sensor Readings- odczyty sensorów.

6.1 Technologie

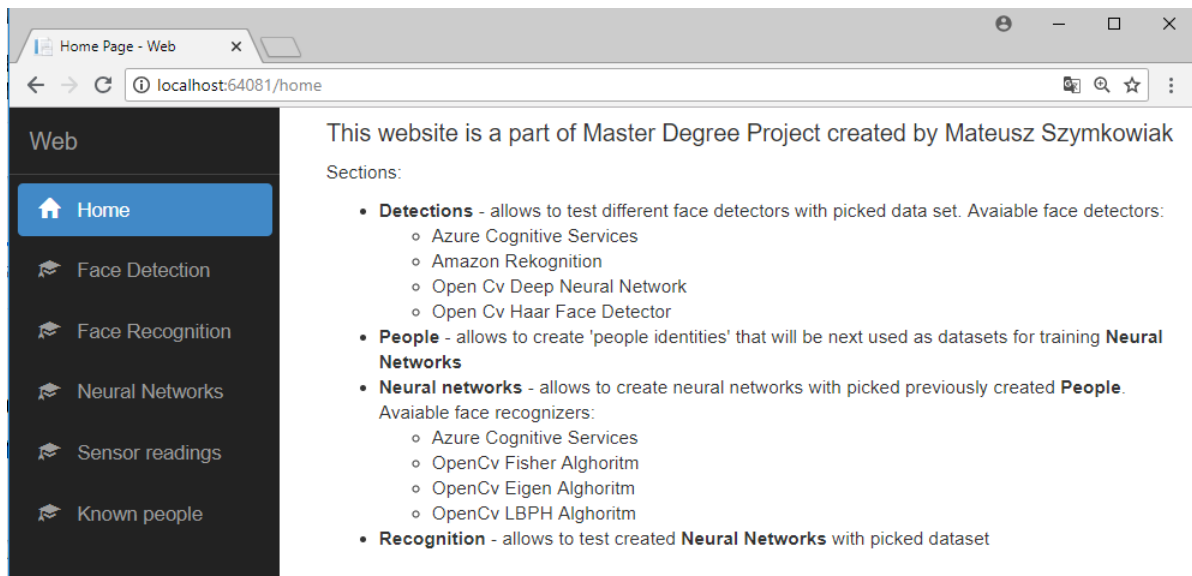
Aplikacja webowa powstała w najnowszej kompilacji .NET Core 2, będącej międzyplatformową strukturą open source o wysokiej wydajności służącą do tworzenia nowoczesnych aplikacji internetowych opartych na usługach chmurowych. Logika biznesowa aplikacji została zaprogramowana w języku C#. Warstwa widoku powstała w dwóch dostępnych rozwiązaniach, nieznacznie różniących się wyglądem, ale znacznie odbiegających od siebie sposobem działania. Przed omówieniem poszczególnych rozwiązań przedstawiono, krótkie definicje wykorzystanego wzorca projektowego MVC oraz technologii SPA.

6.1.1 Rozwiązanie 1- Razor Pages

Pierwsza wersja została oparta o strony tworzone w technologii Razor Pages opartej o składnię Razor oraz podstawowe technologie webowe: HTML i CSS. Taki sposób tworzenia warstwy prezentacji jest zalecany dla aplikacji .NET Core, ponieważ pozwala zminimalizować ilość pracy wymaganej na jej utworzenie oraz zapewnia bardzo prosty proces wdrożenia. Aplikacja utworzona z pomocą Razor’a została zaprezentowana na rysunku 6.1.

6.1.2 Rozwiązanie 2- Angular 4

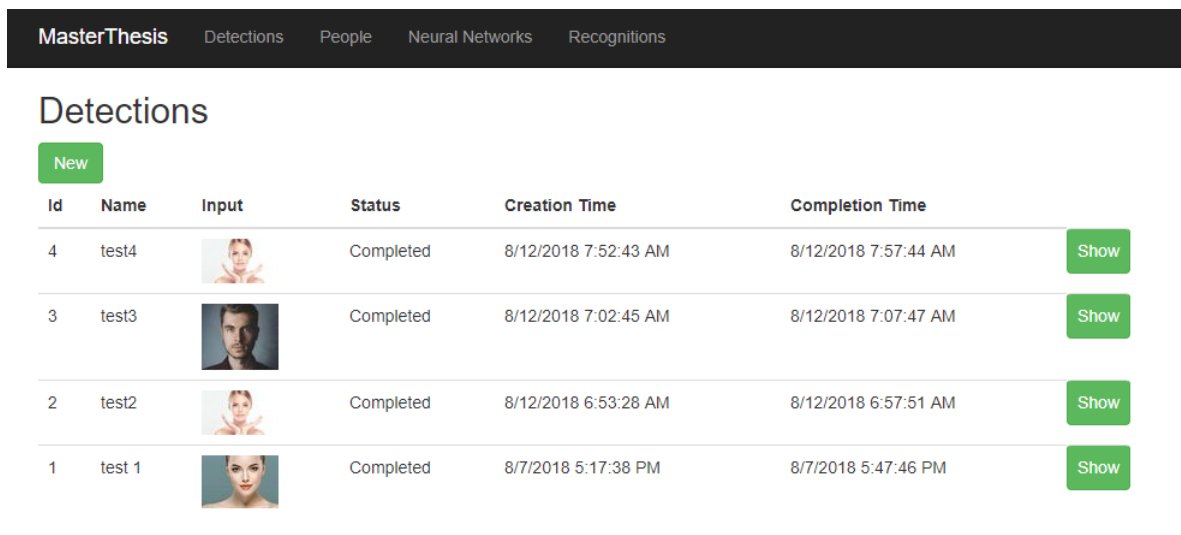
Druga wersja widoku aplikacji oferuje dostęp do tych samych możliwości co pierwsze rozwiązanie, ale powstała przy pomocy frameworka webowego- Angular 4. Strona główna widoczna jest na rysunku 6.2. Angular jest open sourcowym frameworkiem używanym do tworzenia aplikacji SPA (Single Page Application), napisany w języku TypeScript i wspierany oraz rozwijany przez Google.



Rysunek 6.2: Wygląd strony głównej aplikacji internetowej (rozwiązanie 2)

6.2 Detekcja twarzy

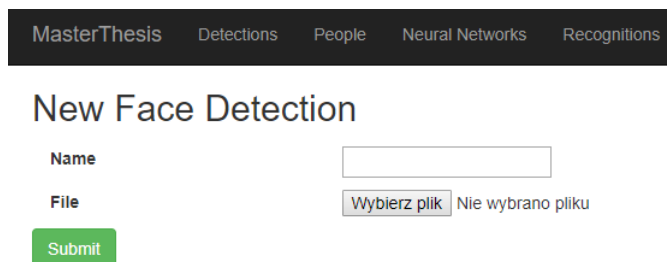
Detections jest stroną odpowiedzialną za wykrywanie twarzy na obrazach przesłanych do systemu. Na głównej stronie możemy zobaczyć wszystkie zlecone detekcje, zarówno nowe jak i już zakończone.



Rysunek 6.3: Widok detekcji twarzy

Przycisk 'New' widoczny na 6.3 pozwala na stworzenie nowego zadania wykrycia twarzy na obrazie, które zostanie przetworzone przez moduł aplikacji konsolowej. Na formularzu z rysunku

6.4 należy podać nazwę zadania oraz za pomocą przycisku 'Wybierz plik' wybrać obraz w formacie png,jpg lub jpeg znajdujący się na dysku użytkownika.



Rysunek 6.4: Tworzenie nowej detekcji



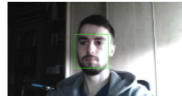
Użycie przycisku 'Show' widocznego przy każdym zadaniu detekcji na rysunku 6.3, pozwoli na wyświetlenie szczegółów związanych z requestem, w tym wyników jeśli zadanie zostało zakończone.

Detection

Id 2
Name mateusz
Creation time 9/9/2018 5:40:25 AM
Completion time 9/9/2018 7:40:54 AM
Status Completed
Input image



Results

Detection type	Faces found	(seconds) Processing time	startX	endX	startY	endY	Area	Image
haar	1	0.026928186416625977	493	702	242	451	43681	
dnn	1	0.044879913330078125	491	740	149	502	87897	
azure	1	0.7141578197479248	480	724	232	476	59536	

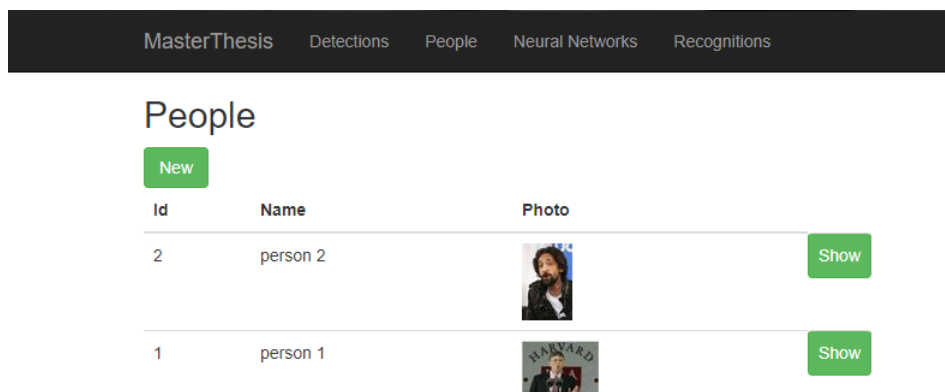

Rysunek 6.5: Widok zakończonej detekcji

Strona ze zdjęcia 6.5 pozwala sprawdzić datę utworzenia oraz zakończenia zadania, obraz wejściowy oraz tabelę z informacjami o wynikach detekcji, na które składają się:

- detection type- typ detekcji (haar, głęboka sieć neuronowa, azure),
- faces found- ilość wykrytych twarzy,
- processing time- czas potrzebny na wykrycie twarzy daną metodą,
- startX, endX, startY, endY- współrzędne lewego dolnego oraz prawego górnego wierzchołka prostokąta obejmującego wykrytą twarz,
- Area- rozmiar obszaru obejmującego twarz w pikselach.

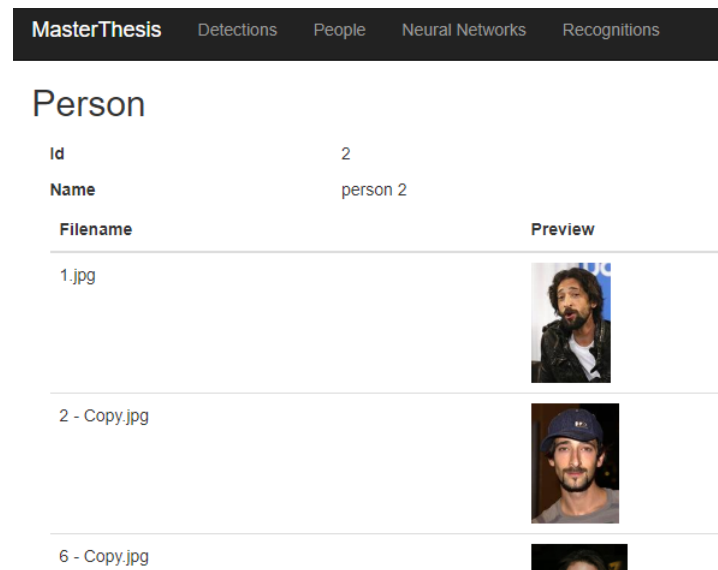
6.3 Profile

Strona 'People' służy do tworzenia nowych profili, które później mogą zostać wykorzystane jako dane uczące podczas trenowania sieci neuronowych. Do każdego profilu osoby musi zostać przypisany zasób minimum 2 zdjęć. W przypadku braku możliwości wykrycia twarzy na zdjęciu, zostanie ono zignorowane podczas procesu nauczania.

MasterThesis Detections People Neural Networks Recognitions				
People				
New				
Id Name Photo Show				
2	person 2		Show	
1	person 1		Show	

Rysunek 6.6: Lista utworzonych profili

Nowa osoba może zostać utworzona w podobny sposób jak request detekcji twarzy. Jediną różnicą jest wymóg wyboru kilku obrazów.



Rysunek 6.7: Widok profilu

Utworzona osoba nie może być modyfikowana. Pierwsze załadowanie widoku osoby może trwać wydłużony czas z powodu procesu generowania linków do plików magazynowanych w usłudze Dropbox.

6.4 Sieci neuronowe

W zakładce 'Neural Networks' użytkownik może utworzyć zadanie trenowania sieci neuronowych (OpenCv: Fisherfaces, Eigenfaces, LBPH, Azure: Cognitive Services) wybranymi profilami utworzonymi w zakładce 'People'. Użytkownik nie ma możliwości wyboru typu sieci, która zostanie nauczona, domyślnie wszystkie dostępne rozwiązania (wymienione powyżej) zostaną wykorzystane.

New							
Id	Name	Trained people count	Number of files	Status	Creation Time	Completion Time	
22	all people 15 photos	100	4	Completed	08.09.2018 17:13:08	08.09.2018 19:49:08	Show
21	all people 10 photos	100	4	Completed	08.09.2018 17:11:46	08.09.2018 19:31:26	Show

Rysunek 6.8: Strona przedstawiająca istniejące grupy usług rozpoznawania tożsamości

Podczas tworzenia nowej sieci każdy istniejący profil zostanie wyświetlony jako checkbox, który należy zaznaczyć jeśli dany profil ma zostać wykorzystany w procesie nauczania. W wyniku szybkiego rozrostu bazy profili dodano checkbox 'Check all' pozwalający na wybranie wszystkich profili poprzez jedno kliknięcie. Kolejnym parametrem, który musi zostać wypełniony jest 'Photos

per person used’ określający maksymalną ilość zdjęć przypisanych do danego profilu, które mogą zostać wykorzystane do nauki sieci. Podana wartość powinna znajdować się w przedziale od 2 do 20. W przypadku gdy niedostępna będzie określana ilość zdjęć w danym profilu, to wykorzystane zostanie tyle zdjęć ile jest dostępnych.

Name

Submit

Photos per person used

0

☐ Check all

People included in neural network training

Include?	Id	Name
<input type="checkbox"/>	2	001
<input type="checkbox"/>	3	002
<input type="checkbox"/>	4	003

Rysunek 6.9: Tworzenie nowej grupy sieci neuronowych

Wykorzystując przycisk 'Show' z widoku na rysunku 6.8 można wyświetlić szczegółowe informacje o wybranej grupie sieci neuronowych. Dostępny widok ukazano na rysunku 6.10. Widoczne są na nim wszystkie parametry grupy, na które składają się: ilość wykorzystanych zdjęć i profili oraz tabela z informacjami o przygotowanych sieciach. Informacje dostępne w tabeli to nazwa i typ sieci, rozmiar utworzonego modelu, czas potrzebny na wytrenowanie oraz pełny czas(czas trenowania + przygotowanie danych wejściowych) potrzebny na stworzenie sieci.

Neural Network

Id 22
Name all people 15 photos
Photos per person 15
Completion time 08.09.2018 19:49:08

Trained files

Id	Name	Type	File size	(seconds) Processing time	(seconds) Training time
74	22_Eigen.xml	Eigen	2.10 GB	293.4762122631073	272.4587895870209
75	22_Fisher.xml	Fisher	196.25 MB	161.35560297966003	140.3381803035736
76	22_LBPH.xml	LBPH	121.06 MB	29.96478033065796	8.947357654571533
77	22	AzureLargeGroup	0	590.5403499603271	1>

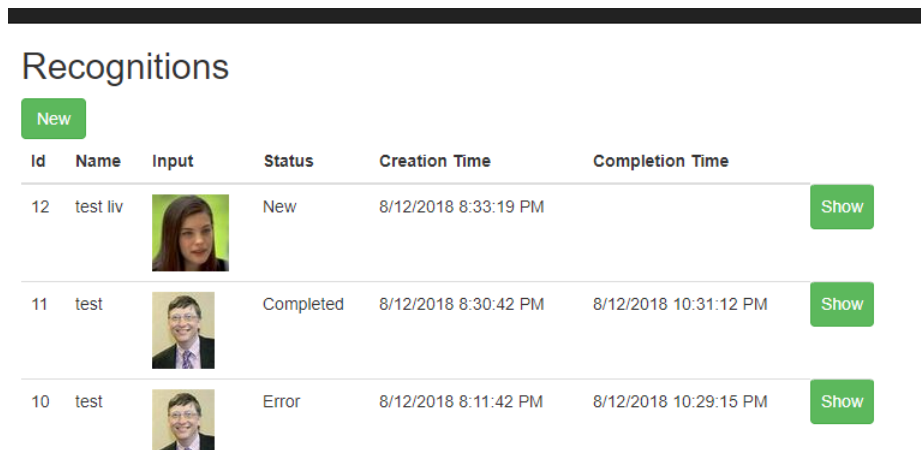
People data used for training




Id	Name	
2	001	Show
3	002	Show

Rysunek 6.10: Szczegółowe informacje o grupie sieci neuronowych

6.5 Rozpoznawanie tożsamości

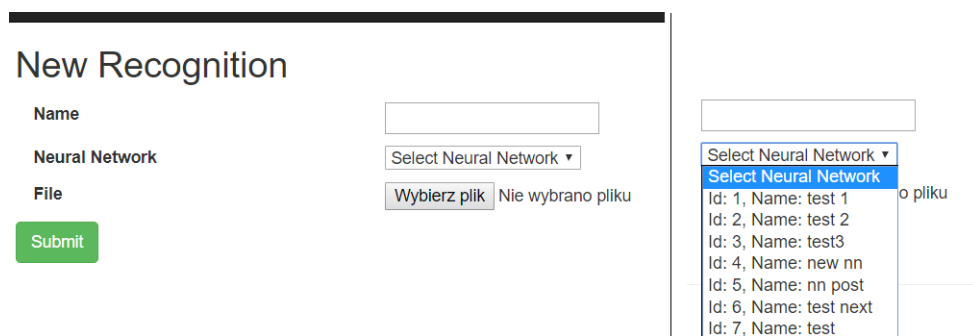
W sekcji 'Recognitions' użytkownik ma możliwość wykorzystać wcześniej utworzone zbiory sieci neuronowych w celu identyfikacji tożsamości na zdjęciu przedstawiającym pojedynczą osobę.



Recognitions					
New					
Id	Name	Input	Status	Creation Time	Completion Time
12	test liv		New	8/12/2018 8:33:19 PM	Show
11	test		Completed	8/12/2018 8:30:42 PM	8/12/2018 10:31:12 PM Show
10	test		Error	8/12/2018 8:11:42 PM	8/12/2018 10:29:15 PM Show

Rysunek 6.11: Lista zadań identyfikacji osoby

Podobnie jak na pozostałych stronach, podczas tworzenia nowego zadania użytkownik będzie musiał uzupełnić prosty formularz. W formularzu przedstawionym na rysunku 6.12 należy załączyć jedno zdjęcie oraz wybrać grupę sieci neuronowych, która ma zostać wykorzystana do identyfikacji tożsamości osoby.



New Recognition

Name

Neural Network

Select Neural Network ▾

File

Wybierz plik

Nie wybrano pliku

Submit

Select Neural Network ▾

Select Neural Network

Id: 1, Name: test 1

Id: 2, Name: test 2

Id: 3, Name: test3

Id: 4, Name: new nn

Id: 5, Name: nn post

Id: 6, Name: test next

Id: 7, Name: test

Rysunek 6.12: Formularz tworzenia zadania identyfikacji

Po zakończonym procesie identyfikacji opisanym w kolejnym podrozdziale, użytkownik może wyświetlić wynik uzyskany przez każdą sieć dostępną w grupie. Przykładowy rezultat widoczny jest na zdjęciu 6.13.

Recognition

Id 13
Name test
Creation time 8/12/2018 8:48:14 PM
Completion time 8/12/2018 10:48:48 PM
Status Completed
Input image



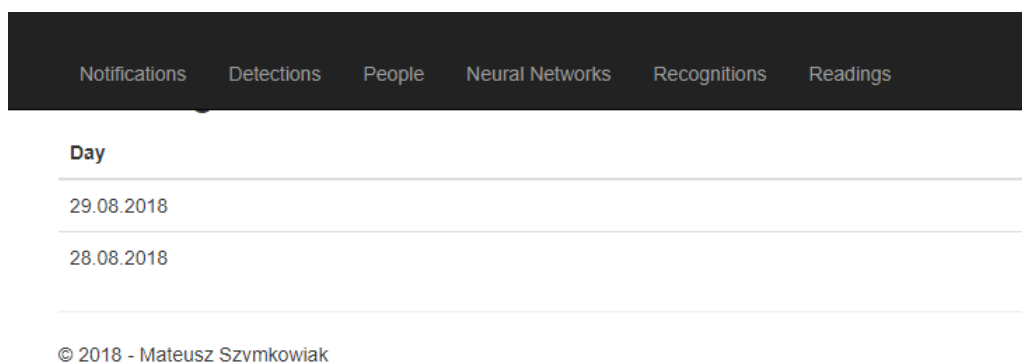
Results

Recognised Identity	Confidence	Neural Network Name	Neural Network Type	Comments
3	0	10_Eigen.xml	Eigen	
3	0	10_Fisher.xml	Fisher	
3	0	10_LBPH.xml	LBPH	
3	1	10	AzureLargeGroup	

Rysunek 6.13: Zakończony request identyfikacji

6.6 Odczyty sensorów

Zakładka Readings powstała w celu czytelnej prezentacji odczytów zebranych z sensorów w dniach funkcjonowania systemu. Odczyty zbierane są co minutę co generuje znaczną ilość wpisów do wyświetlenia. Z tego powodu po wejściu na stronę należy wybrać dzień, którym zainteresowany jest użytkownik. Przykładową listę zaprezentowano na rysunku 6.14. Po wybraniu dnia ukazuje się widok, na którym przedstawiono wszystkie odczyty zebrane w wybranym okresie czasu. Do dostępnych informacji należą temperatura, wilgotność powietrza oraz czas wykonania pomiaru.





Rysunek 6.14: Widok pozwalający na wybór dnia

Notifications Detections People Neural Networks Recognitions Readings			
Id	Humidity	Temperature	Time
1013	20	24	16:52:38.0166667
1012	62	8	16:52:33.5366667
1011	57	2	16:41:18.3633333
1010	72	27	15:29:33.6800000
1009	44	24	15:29:32.1700000
1008	42	30	15:29:22.7533333
1007	61	40	15:29:21.2666667
8	25	33	14:26:12.8000000
7	75	39	14:26:11.2933333

Rysunek 6.15: Widok wyświetlający wszystkie odczyty z wybranego dnia

6.7 Powiadomienia

Ostatnią ze stworzonych zakładek są Powiadomienia. Widok służy do wyświetlania wszystkich powiadomień utworzonych przez system, na które składają się powiadomienia sensorów oraz wykrycia ruchu.

Notifications Detections People Neural Networks Recognitions Readings			
29.08.2018 16:52:33	SensorReading	Humidity value is to high. Required value should fit between 30 and 60	
29.08.2018 16:52:33	SensorReading	Temperature value is to low. Required value should fit between 15 and 30	
29.08.2018 16:41:18	SensorReading	Temperature value is to low. Required value should fit between 15 and 30	
29.08.2018 16:40:25	Movement	Movement detected	
29.08.2018 16:40:21	Movement	Movement detected	

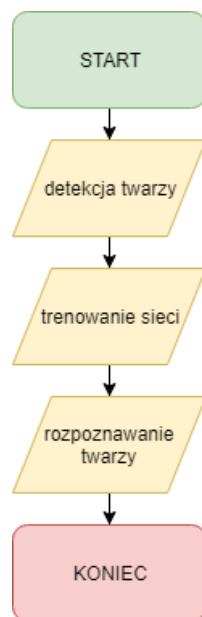
Rysunek 6.16: Widok wyświetlający wszystkie odczyty z wybranego dnia

Typy oraz sposób ich generowania zostały szerzej opisany w rozdziale 8. Każde wyświetlane powiadomienie posiada wiadomość, typ oraz czas utworzenia. W przypadku wykrycia ruchu dodatkowo wyświetlone zostaje zdjęcie uchwyconego wydarzenia. Kilka przykładowych powiadomień można zaobserwować na rysunku 6.16.

Rozdział 7

Aplikacja konsolowa

Aplikacja konsolowa powstała w celu przetwarzania czasochłonnych zadań w tle, tak by użytkownik aplikacji webowej nie doświadczał długich czasów ładowania oraz ewentualnych błędów podczas przerwania sesji lub połączenia internetowego ze stroną. Aplikacja konsolowa uruchamiana jest co 5 minut i przetwarza zadania trzech typów w kolejności widocznej na rysunku 7.1. Poszczególne procesy zostały szerzej opisane w kolejnych podrozdziałach.



Rysunek 7.1: Proces działania aplikacji konsolowej

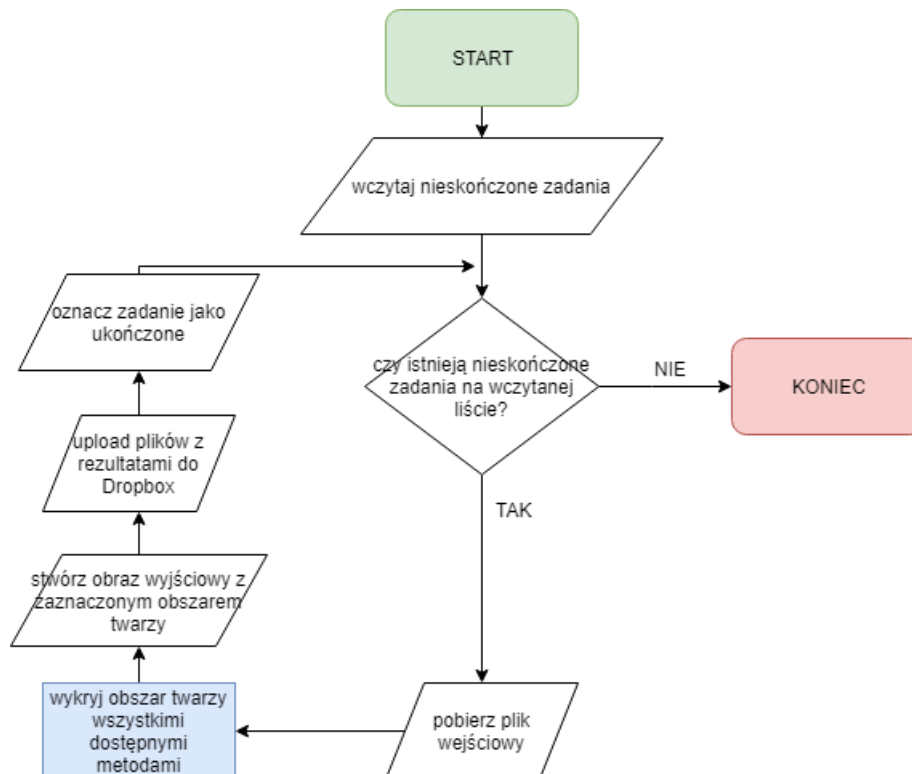
7.1 Technologie

Aplikacja konsolowa powstała w języku programowania Python w wersji trzeciej. W początkowej fazie projektu za wyborem tego języka przemawiała wieloplatformowość, możliwości wprowadzania szybkich zmian w kodzie i brak potrzeby go kompilowania. C#, który został wybrany do stworzenia aplikacji webowej okazał się nie przystosowany do uruchomienia na plat-

formie Raspberry Pi z powodu braku dostępnego .NET Core SDK na procesory ARM oraz wymaganiu znacznie większej ilości zasobów niż Python. Ostatecznie aplikacja konsolowa została przeniesiona na zewnętrzny serwer, ale popularność Pythona pozwoliła na szybką integrację z usługami firm trzecich, dzięki przygotowanym przez firmy paczki deweloperskie.

7.2 Proces wykrywania twarzy

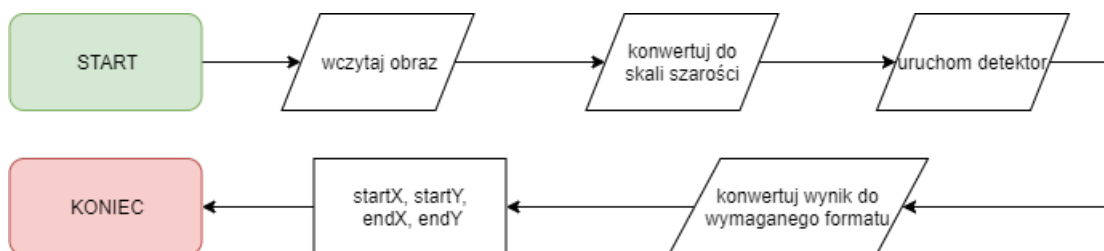
Uogólniony proces detekcji twarzy na obrazie został przedstawiony na grafie 7.2. Proces przetwarzania zadań detekcji rozpoczyna się od pobrania z bazy wszystkich żądań o statusie 'New'. Następnie każde zadanie przetwarzane jest osobno. Dla aktualnie procesowanego zadania pobierany jest obraz wejściowy z usługi Dropbox i zapisywany w lokalnym folderze. Następnym krokiem jest wywołanie procesu odpowiedzialnego za odpowiednie przygotowanie zdjęcia, a następnie pozyskanie oczekiwanych wyników, w sposób odpowiedni dla danego sposobu. Krok ten został szerzej opisany w podrozdziałach 7.2.1, 7.2.2 i 7.2.3. po uzyskaniu wyników każdą dostępną w programie metodą, zostaje utworzony obraz wyjściowy dla każdej metody (haar, azure, ..), które zostaje wgrany do odpowiedniego folderu w usłudze Dropbox. Po poprawnym wgraniu plików wynikowych informacja o rezultatach zostaje dodana do bazy danych. Po wykonaniu zadania bez żadnych błędów, request zostaje oznaczony jako zakończony, w innym przypadku zostaje nadany mu status 'Error'. Proces powtarzany jest dla każdego wpisu pozyskanego z bazy.



Rysunek 7.2: Proces wykrywania twarzy zaimplementowany w aplikacji konsolowej

7.2.1 OpenCv Haar

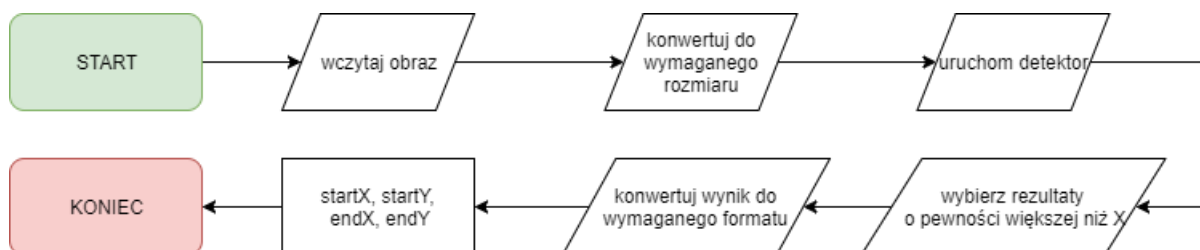
Pierwszą z metod detekcji twarzy, która została zintegrowana z programem jest detekcja metodą Haar’a. Algorytm został opisany w rozdziale 3.3.4. Przed uruchomieniem detekcji twarzy, obraz wejściowy należy odpowiednio przygotować. W tym celu wcześniej pobrany obraz zostaje wczytany do programu i przekonwertowany do odcieni szarości. Tak przygotowany obraz można poddać detekcji. Detektor zwraca współrzędne obszarów zawierających w sobie twarze. Tak uzyskane dane należy przekonwertować do formatu, który został przyjęty jako wspólny dla wszystkich metod.



Rysunek 7.3: Wykrywanie twarzy metodą Haar zaimplementowane w aplikacji konsolowej

7.2.2 OpenCv DNN (Deep Neural Network)

Proces detekcji z wykorzystaniem głęboko uczonej sieci neuronowej nie wymaga formatowania obrazu do skali szarości, ale za to obraz musi zostać przeskalowany do odpowiedniego, wcześniej przyjętego rozmiaru. Dodatkową zaletą tej metody jest format odpowiedzi detektora, który oprócz obszaru zawierającego twarz zwraca również 'pewność' z jaką twarz została wykryta, co pozwala na odfiltrowanie niezadowolających użytkownika wyników.

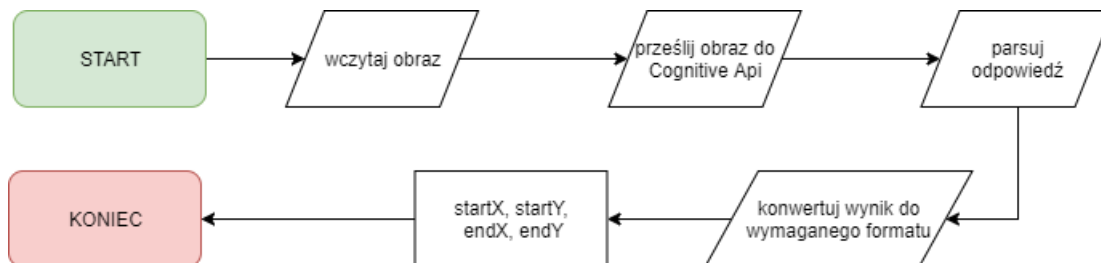


Rysunek 7.4: Wykrywanie twarzy metodą DNN (Deep Neural Network) zaimplementowane w aplikacji konsolowej

7.2.3 Azure Cognitive Services

Kolejna wykorzystana metoda detekcji twarzy, korzysta z usługi dostępnej na platformie Azure, dzięki czemu środowisko rozruchowe zostaje odciążone z obliczeń związanych z przetwarzaniem obrazu. Proces wykrycia twarzy ogranicza się do przesłania wybranego obrazu do Cognitive Se-

rvices Api. W odpowiedzi klient uzyskuje JSON zawierający parametry, które podano jako wymagane podczas wykonywania zapytania. JSON zostaje zparsowany na obiekt, którego wartości zostają przekonwertowane do wymaganego formatu [startX, startY, endX, endY].



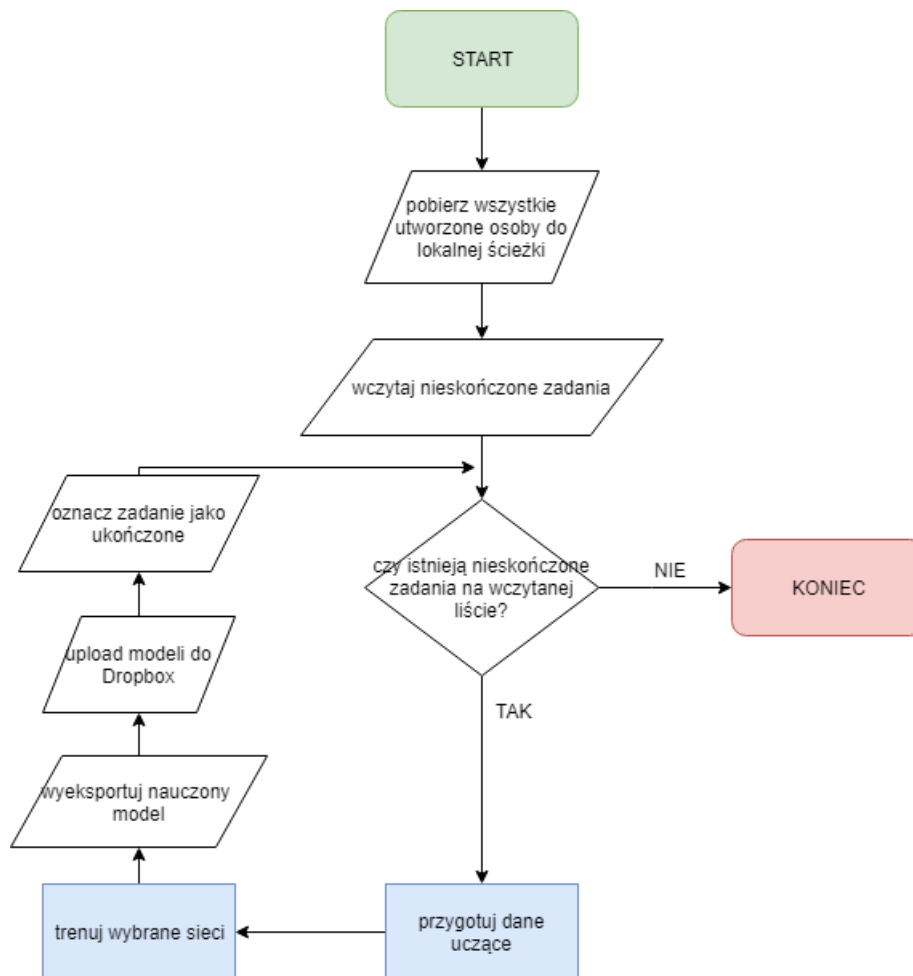
Rysunek 7.5: Wykrywanie twarzy używając Azure Cognitive Services zaimplementowane w aplikacji konsolowej

7.3 Proces trenowania sieci neuronowych **cos o tej sieci napisac lub omowic jej zrodla**

Proces trenowania sieci został przedstawiono na grafie 7.6. Po jego zakończeniu użytkownik otrzymuje grupę sieci neuronowych zawierającą gotowe do użycia metody rozpoznawania twarzy przedstawione w rozdziale 4.1 oraz 4.2. Proces zaczyna się od sprawdzenia jakie dane uczące (profile utworzone w zakładce 'People' aplikacji webowej) są dostępne lokalnie. Repozytorium zostaje porównane z danymi dostępnymi w bazie, a następnie zostają pobrane zdjęcia wszystkich brakujących osób.

Kolejnym krokiem jest pobrania z bazy wszystkich nowych grup sieci neuronowych, które nie zostały jeszcze zakończone. Następnie dla każdego zgłoszenia z listy wykonywany jest proces przygotowania danych uczących na podstawie zdjęć osób, które wybrano podczas tworzenia sieci neuronowej w aplikacji webowej. Po odpowiednim przygotowaniu danych grupa sieci zostaje nauczona i dla metod tego wymagających zostaje utworzony plik w formacie xml przechowujący wytrenowany model. Taki plik zostaje zapisany w bazie danych oraz wysłany do odpowiedniego folderu w Dropboxie.

W przypadku braku komplikacji, request zostaje oznaczony jako zakończony. Proces zostaje powtórzony dla każdego zadania wczytanego do listy na początku grafu 7.6. Sposób przygotowania danych oraz nauczania dla poszczególnych metod został opisany w kolejnych podrozdziałach.



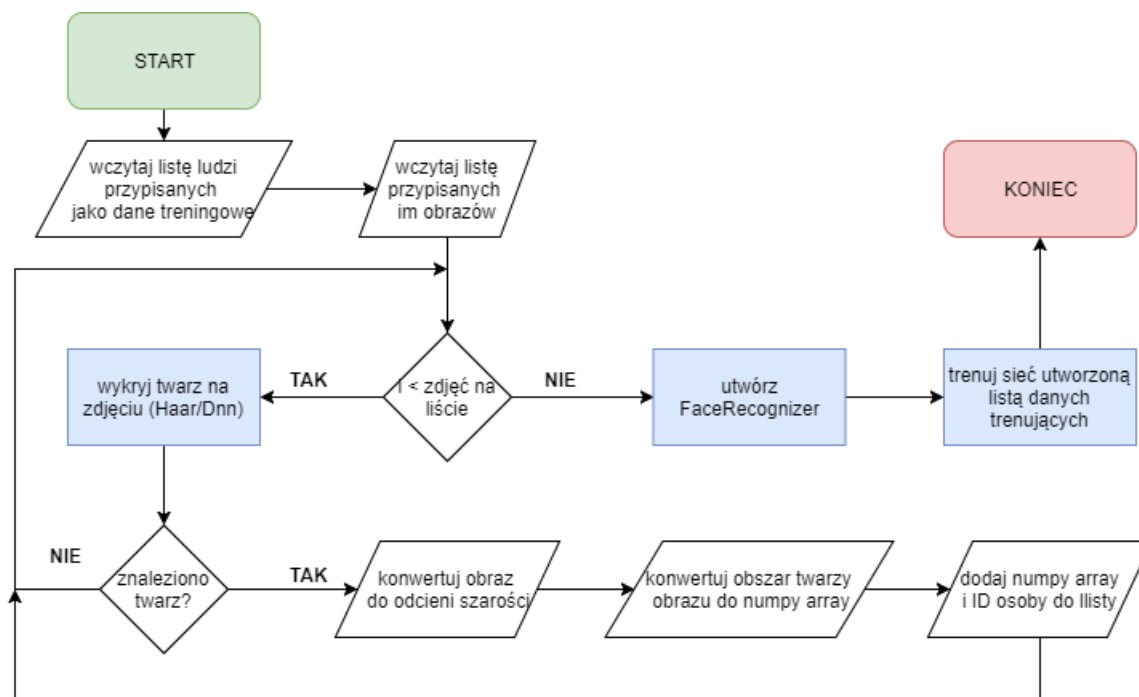
Rysunek 7.6: Proces trenowania sieci zaimplementowany w aplikacji konsolowej

7.3.1 Trenowanie identyfikatorów OpenCv

Ogólny schemat procesu trenowania identyfikatorów twarzy OpenCv (Eigenfaces, Fisherfaces, LHGP) został przedstawiony na rysunku 7.7. Zalecany proces trenowania udostępniony w dokumentacji¹ został poddany zmianą wymagany przez aplikację konsolową i zaimplementowany.

Proces rozpoczyna się od stworzenia listy zawierającej wszystkie przypisane do sieci profile oraz załączone do nich obrazy. Następnie dla każdego zdjęcia z listy zostaje wykonany pre-processing. Pierwszym krokiem jest próba wykrycia twarzy na obrazie. W przypadku nie znalezienia twarzy, obraz zostaje zignorowany i program przechodzi do przetwarzania kolejnego obrazu. Jeśli na zdjęciu zostanie zlokalizowana twarz, to obszar ją zawierający zostaje przeskalowany do odcieni szarości, a następnie przekonwertowany do postaci numpy array. Wektor zawierający informacje o twarzy zostaje dodany do listy uczącej wraz z odpowiadającym mu ID profilu, do którego należy zdjęcie. Po przetworzeniu wszystkich obrazów na liście, zostaje utworzone i nauczone wszystkie wcześniej wymienione identyfikatory dostępne w OpenCv.

¹https://docs.opencv.org/3.4.3/da/d60/tutorial_face_main.html#tutorial_face_conclusion



Rysunek 7.7: Trenowanie sieci neuronowej OpenCv zaimplementowany w aplikacji konsolowej

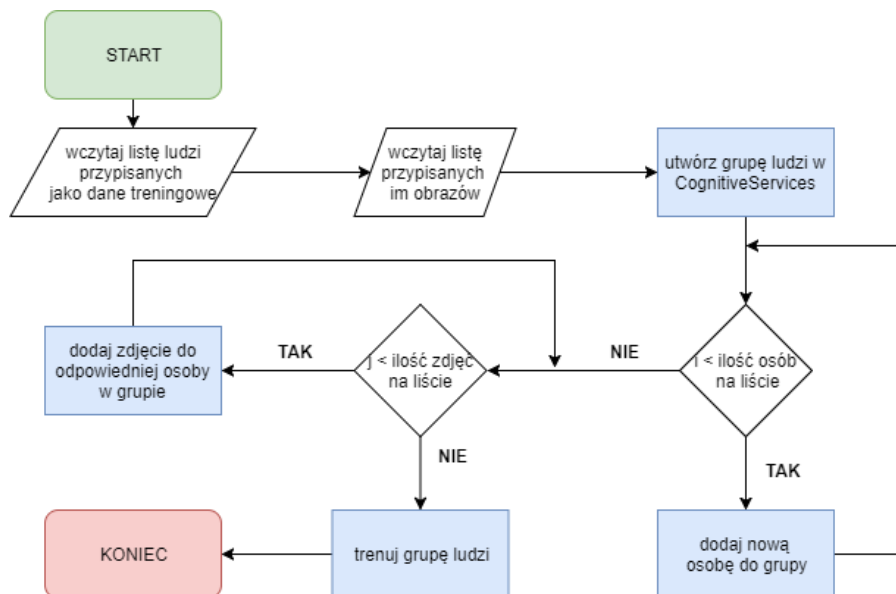
7.3.2 Trenowanie identyfikatora Azure

Proces rozpoczyna się od wczytania wszystkich osób oraz ich zdjęć, które zostały przypisane do zadania trenowania sieci. Na diagramie 7.8 niebieskim kolorem oznaczono zapytania do Azure Cognitive Services API(Application Programming Interface).

Zgodnie z dokumentacją² pierwszym i najważniejszym krokiem jest utworzenie nowej AzureLargeGroup, do której w kolejnym kroku zostaną dodane id/nazwy wszystkich profili znajdujących się na wczytanej liście. Następnie każdy obraz z listy zostaje przypisany odpowiedniej osobie w AzureLargeGroup. W tym rozwiązaniu klient nie musi martwić się detekcją twarzy na obrazie, bo jest ona wykonywana przez usługę Azure po przesłaniu obrazu. W przypadku problemów z wykryciem twarzy na obrazie, zostanie zwrócona informacja mówiąca o tym że plik zostanie zignorowany podczas procesu nauczania. Po dodaniu wszystkich zdjęć, program wywołuje funkcję trenowania nowej sieci na podstawie danych dołączonych do AzureLargeGroup. Już po około sekundzie, identyfikator tożsamości udostępniony przez usługę Azure jest gotowy do działania.

Ze względu na setki zapytań, które muszą zostać wykonane do zewnętrznego serwisu Azure, proces przygotowania danych przed trenowaniem może wydłużyć się nawet do kilku minut w przypadku wybrania dużej ilości profili i zdjęć.

²<https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/Quickstarts/python-analyze>



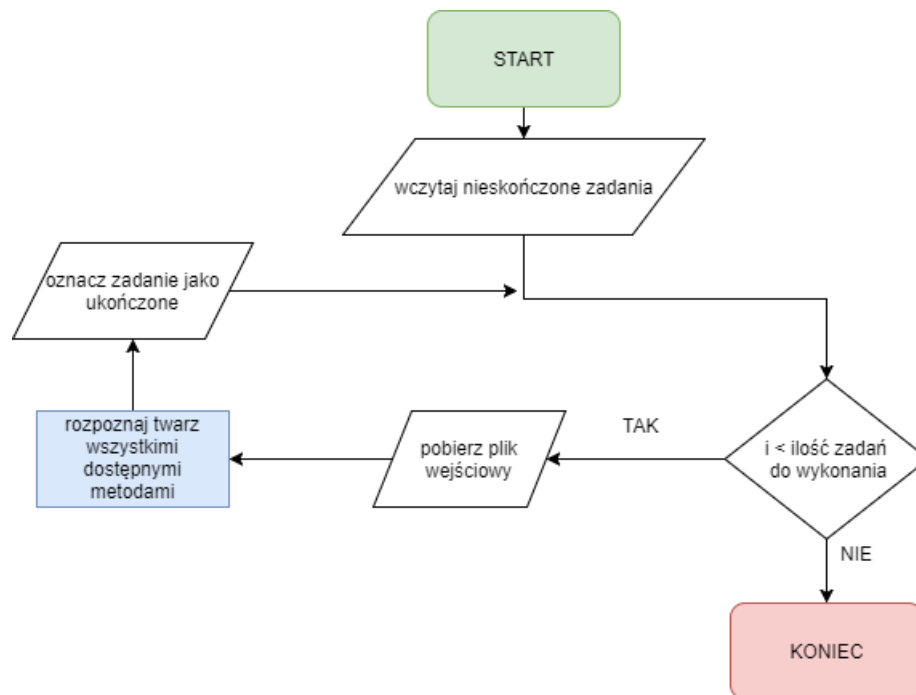
Rysunek 7.8: Trenowanie usługi Azure Cognitive Services zaimplementowane w aplikacji konsolowej

7.4 Proces rozpoznawania twarzy

Ogólny proces rozpoznawania twarzy przedstawiony na rysunku 7.9 jest zbliżony do wcześniej opisywanych procesów detekcji i trenowania.

Proces rozpoczyna się od pobrania wszystkich modeli sieci neuronowej, których aktualnie nie ma w lokalnym katalogu. W kolejnym etapie zostają wczytane wszystkie nowe zadania rozpoznania twarzy. Dla każdego z requestów na liście wykonywane są te same czynności, a pierwszą z nich jest pobranie pliku wejściowego z usługi Dropbox. Następnie twarz zostaje rozpoznana za pomocą każdego dostępnego algorytmu identyfikacji.

Poszczególne metody zostały szerzej opisane w kolejnych podrozdziałach. Po ukończeniu identyfikacji tożsamości, wszystkie uzyskane wyniki zostają wprowadzone do bazy danych, a zadanie oznaczone jako ukończone. Gdy zadania na liście skończą się to dochodzi do zakończenia procesu.



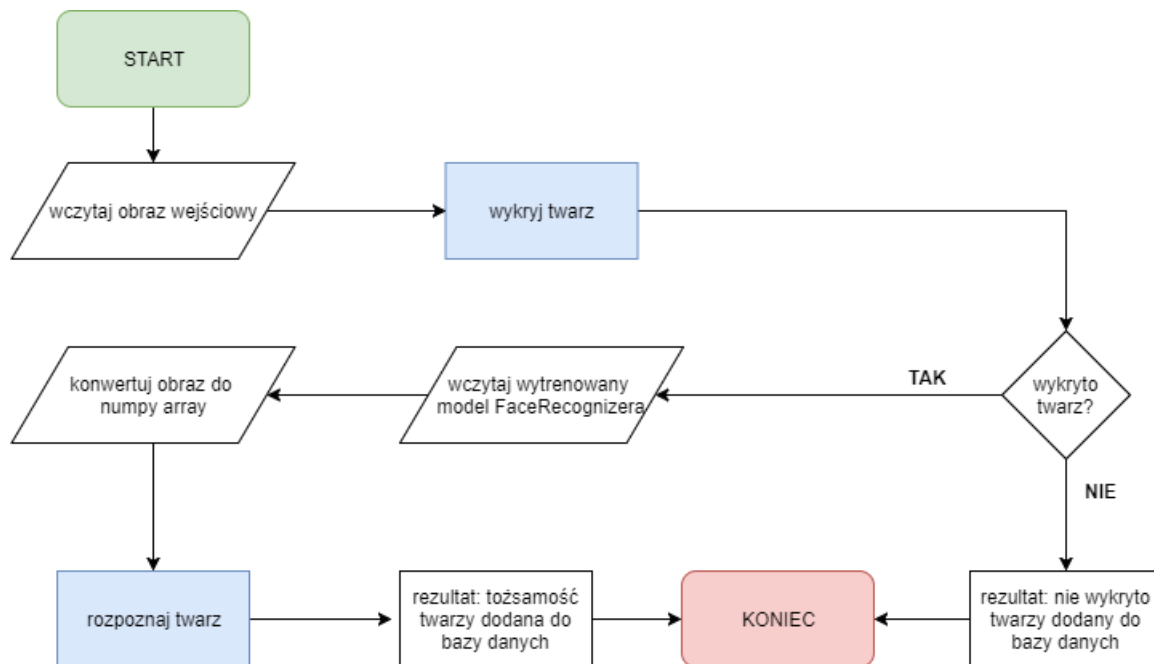
Rysunek 7.9: Proces identyfikacji tożsamości zaimplementowany w aplikacji konsolowej

7.4.1 Identyfikacja tożsamości metodami dostępnymi w OpenCv

Niezależnie od typu FaceRecognizera, proces rozpoznawania twarzy z użyciem algorytmów OpenCv wygląda identycznie. Przykładowe zastosowanie zostało opisane w dokumentacji³.

Obraz zostaje wczytany do programu, a następnie zostaje podjęta próba detekcji twarzy metodą Haara. W przypadku nie wykrycia żadnej twarzy do zadania zostaje dodany pusty rezultat z komentarzem informującym o braku twarzy na przekazanym obrazie. Jeśli na zdjęciu zostanie wykryta twarz to proces przechodzi do kolejnego kroku, którym jest wczytanie modelu odpowiadającego typowi identyfikatora i numerowi wybranej grupy sieci. Przed identyfikacją wycinek obrazu zawierający twarz zostaje przekonwertowany do numpy array, który jest formatem wymaganym przez FaceRecognizer. W rezultacie uzyskana zostaje tożsamość osoby ze zdjęcia prezentowana w postaci numeru Id z bazy oraz wartość wektora oddalenia od najbliższego obiektu w modelu (0.0 oznacza idealne dopasowanie, im większa wartość tym mniejsza pewność).

³https://docs.opencv.org/3.4.3/da/d60/tutorial_face_main.html#tutorial_face_conclusion



Rysunek 7.10: Proces identyfikacji osoby wykorzystując identyfikatory OpenCv zaimplementowany w aplikacji konsolowej

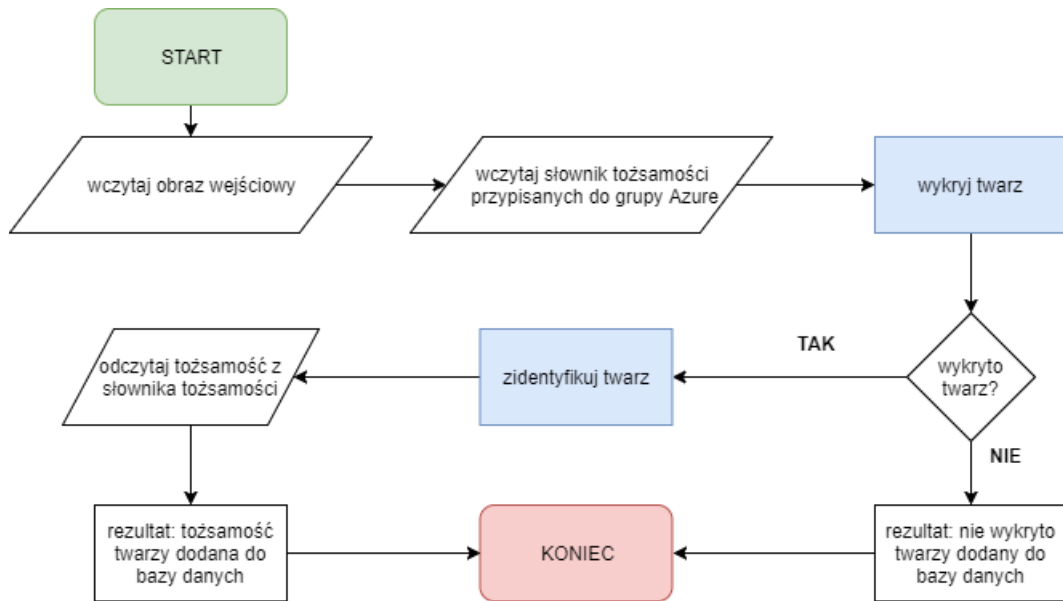
7.4.2 Identyfikacja tożsamości przy użyciu Azure Cognitive Services

Proces rozpoznawania twarzy przy pomocy Azure Cognitive Services jest mniej obciążający dla platformy, na której uruchomiono program, ale za to może być wolniejszy od pozostałych rozwiązań ze względu na ograniczenia jakości połączenia.

Kroki procesu identyfikacji zaimplementowano zgodnie z dokumentacją⁴, a pierwszym z nich jest wczytanie słownika tożsamości przypisanych do AzureLargeGroup. W kolejnym etapie zdjęcie wejściowe zostaje wysłane do usługi detekcji opisanej w 7.2.3 ale ze zmodyfikowanymi parametrami. Każdej wykrytej twarzy zostaje nadany numer identyfikacyjny Azure.

Jeśli jakaś twarz została znaleziona to kolejnym etapem jest wysłanie żądania identyfikacji uzyskanej twarzy przy pomocy AzureLargeGroup, która została przypisana do tego requesta. Tożsamość rozpoznana przez CS zostaje zwrócona w postaci numeru identyfikacyjnego usługi, dlatego musi ona zostać porównana z wcześniej wczytanym słownikiem, który pozwala na odczytanie Id osoby istniejącej w bazie danych programu.

⁴<https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/Quickstarts/python-analyze>



Rysunek 7.11: Proces identyfikacji osoby wykorzystując Azure Cognitive Services zaimplementowany w aplikacji konsolowej

Rozdział 8

Aplikacja konsolowa do zarządzania domem

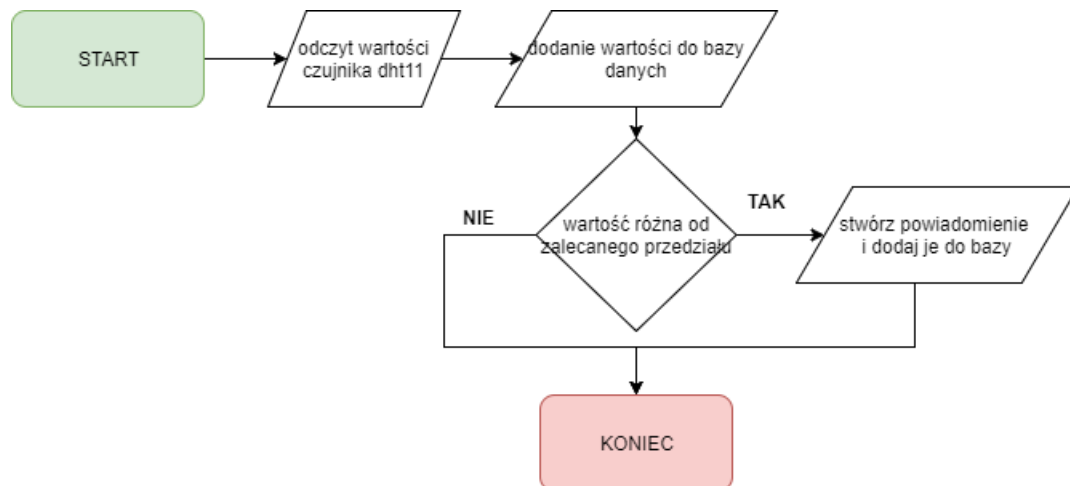
Aplikacja konsolowa do zarządzania domem służy jako aplikacja odpowiedzialna za wszystkie działania związane z Raspberry Pi oraz jego peryferiami. Na te zadania składają się 2 czynności:

- odczyt wartości sensorów,
- detekcja ruchu.

Obie czynności wykonywane są niezależnie od siebie. Funkcja wykrywania ruchu działa bez przerwy, w przypadku problemu zostanie ponownie uruchomiona po około minucie za pomocą zadania wpisanego do serwisu Crontab. Odczyt parametrów wskazywanych przez czujniki odbywa się cyklicznie co 1 minutę.

8.1 Odczyt wartości sensorów

Proces zbierania danych z sensorów rozpoczyna się od pobrania odczytów z czujnika DHT11. Raspberry komunikuje się z czujnikiem za pomocą protokołu 1-wire. Do pobrania wartości wykorzystano bibliotekę dedykowaną dla tego czujnika i języka Python. Po prawidłowym odczycie, każdy z parametrów (temperatura i wilgotność) zostaje zapisany w bazie danych z odpowiednim opisem. W kolejnym kroku odczytane wartości zostają porównane z wcześniej zdefiniowanym zakresem, który jest zapisany w bazie danych. W przypadku przekroczenia lub zbyt niskiej wartości system podejmie decyzję o utworzeniu nowego powiadomienia, które będzie widoczne na stronie opisanej w rozdziale 6.7



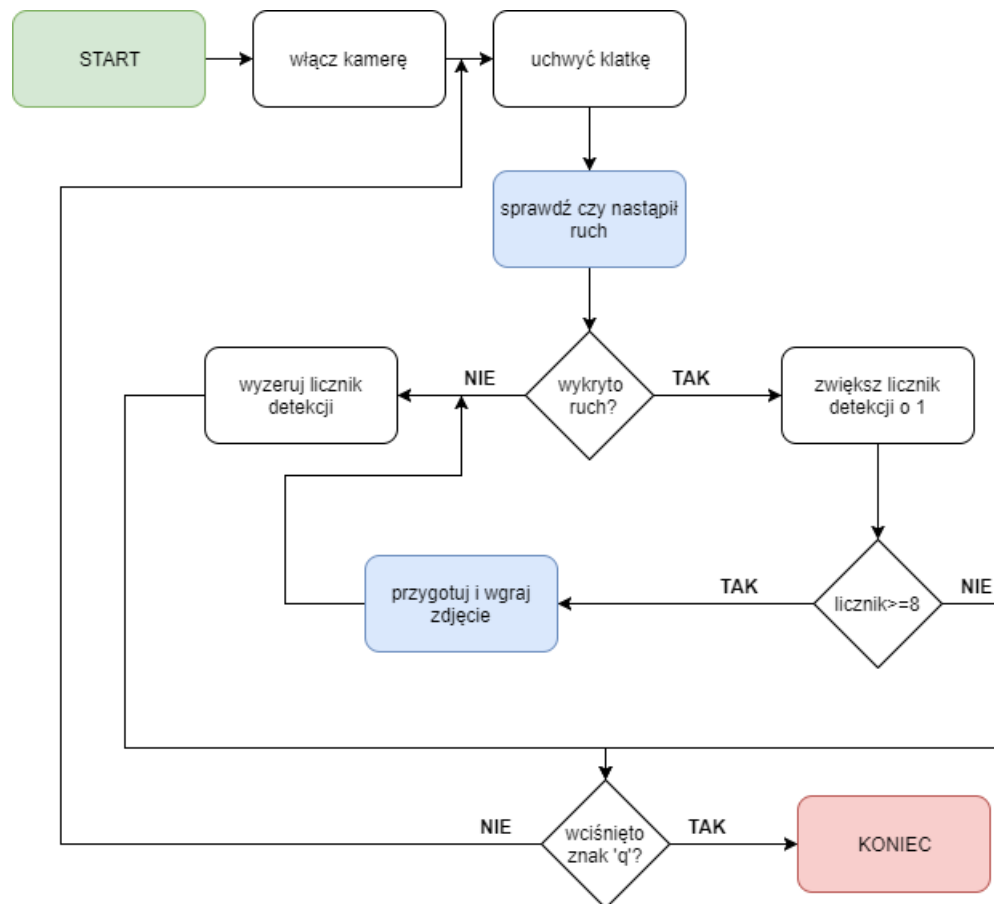
Rysunek 8.1: Proces pobierania danych z sensorów zaimplementowany w aplikacji konsolowej do zarządzania domem

8.2 Proces detekcji ruchu

Proces detekcji ruchu rozpoczyna się od podstawowej rzeczy, którą jest uruchomienie kamery. Jeśli kamera została uruchomiona i daje możliwość pobierania pojedynczych klatek to zostaje uruchomiona nieskończona pętla, której pierwszym krokiem jest uchwycenie najnowszej klatki obrazu. Następnie klatka zostaje przekazana specjalnej funkcji identyfikującej czy nastąpiły jakieś przemieszczenia względem poprzedniego obrazu. Proces ten został dokładnie opisany w rozdziale 8.2.1.

Jeśli moduł wykrył ruch to licznik detekcji zostaje zwiększony o 1, a w przeciwnym wypadku licznik zostaje wyzerowany. Żadne dodatkowe funkcje nie są wykonywane, aż do czasu osiągnięcia wartości równej 8 przez licznik. Liczba 8 została wyznaczona w celu ograniczenia ilości przechwytywanych ruchów do tych kilkakrotnie potwierdzonych.

Po osiągnięciu tej wartości program ostatecznie stwierdza, że nastąpił ruch i przekazuje uchwycony moment do funkcji zapisującej, której działanie opisano w kolejnej części tego rozdziału. Po udanym wykryciu licznik zostaje wyzerowany. Proces ten powtarza się tak długo, aż nie zostanie ręcznie zatrzymany poprzez wciśnięcie klawisza 'q'.



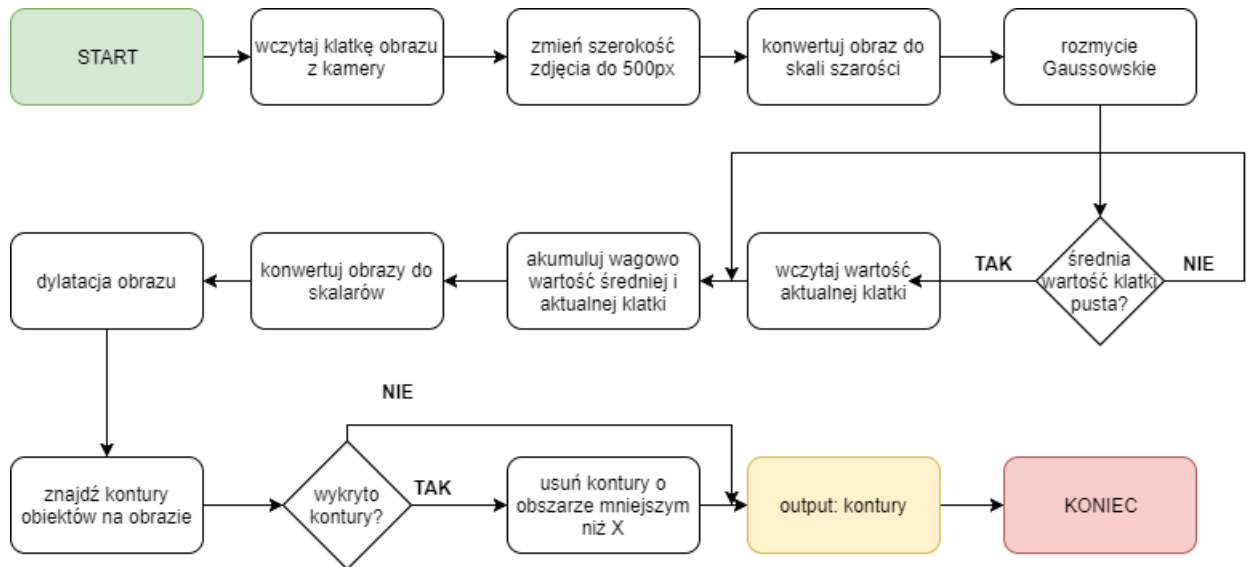
Rysunek 8.2: Proces wykrywania ruchu zaimplementowany w aplikacji konsolowej do zarządzania domem

8.2.1 Wykrywanie ruchu

Proces określający czy od poprzedniego wywołania funkcji nastąpił ruch zaprezentowano na rysunku 8.3/

Na początku wczytane zdjęcie zostaje przeskalowane do szerokości 500px z zachowaniem proporcji wymiarów. W kolejnym kroku obraz zostaje przekonwertowany do skali szarości. Następnie obraz zostaje wygładzony przy pomocy rozmycia gaussowskiego. Jeśli jest to pierwsze uruchomienie funkcji i zmienna przechowująca średnią wartość klatki jest pusta to zostanie do niej przypisany aktualny obraz. Przy pomocy wbudowanej funkcji OpenCv dochodzi do akumulacji w sposób wagowy wartości średniej klatki oraz aktualnej.

W wyniku tej operacji uzyskany zostaje obraz, na którym widoczne są jedynie obszary różniące się między aktualną klatką, a średnią. Następnie wszelkie braki/wypustki w obrazie zostają wypełnione dzięki dwukrotnie zastosowanej dylatacji strukturą o wymiarach 3x3 px. Ostatnim krokiem jest znalezienie konturów na przetworzonym obrazie, które są końcowym rezultatem działania funkcji.



Rysunek 8.3: Proces sprawdzenia czy wystąpił ruch zaimplementowany w aplikacji konsolowej do zarządzania domem

8.2.2 Powiadomienie o wykryciu ruchu

Zgodnie z wcześniej opisanym algorytmem, wykryty ruch zostaje przekazany do zapisania. W zależności od typu aktualnie włączonych opcji, zdjęcie może zostać zapisane w nienaruszonym stanie lub z dodatkowo oznaczoną strefą, w której nastąpił ruch. W kolejnym etapie zostaje utworzony nowy wpis w bazie danych przechowującej informacje o wykrytych ruchach, a zdjęcie zostaje wgrane do odpowiedniego folderu w serwisie Dropbox. Efekt utworzenia nowego wykrycia ruchu można zaobserwować na zdjęciu 6.16.

Rozdział 9

Porównanie wykorzystanych technologii **TODO**

9.1 Detekcja

9.1.1 Zdjęcie na wprost

9.1.2 Zdjęcie częściowo od boku

9.1.3 Przechylona głowa

9.1.4 Częściowo zakryta twarz

9.1.5 Wnioski

9.2 Dane treningowe

Wybierz jakas baze, delikatnie opisz i ile probek bierzesz per osoba teraz mam: thumbnails
features deduped sample

9.3 Trenowanie sieci neuronowych

dodać czas w pythonie

9.3.1 Wpływ ilości próbek na czas tworzenia modelu

kilka testów

9.3.2 Wpływ ilości próbek na rozmiar modelu

kilka testów

9.4 Rozpoznawanie twarzy

9.4.1 Wpływ ilości próbek na pewność rozpoznania

9.4.2 Porównanie wyników

9.4.3 Czas przetwarzania zapytania

9.5 Ocena przydatności wybranych usług IoT

Konfiguracja bazy danych oraz maszyny wirtualnej okazała się równie prosta w każdym ze środowisk. Proces konfiguracji odbywał się poprzez wypełnienie kilku formularzy niewymagających wprowadzania dużej ilości informacji (ze względu na ograniczenia studenckiej licencji). Na końcu procesu uzyskany zostaje connection string oraz konto za pomocą, którego można zalogować się na serwer.

Największa różnica między dwoma dostawcami wystąpiła w przypadku usług hostujących aplikację webową czyli Azure App Service oraz AWS Elastic Beanstalk. Podczas pierwszych testów konfiguracji aplikacja internetowa istniała jedynie w rozwiązaniu przygotowanym w języku Angular 4. Struktura aplikacji została przygotowana na podstawie wzoru przygotowanego przez Microsoft. Azure App Service bezproblemowo wspierał nawet najnowsze wersje rozwiązań przygotowanych dla .NET Core 2. Niestety AWS nie był przygotowany

Rozdział 10

Podsumowanie

Bibliografia

- [1] *angular-chart.js - beautiful, reactive, responsive charts for Angular.JS using Chart.js*. URL: <https://jtblin.github.io/angular-chart.js/> (term. wiz. 21.01.2017).
- [2] *LaTeX - A document preparation system*. URL: <https://www.latex-project.org/> (term. wiz. 21.01.2017).
- [3] *Zotero | Home*. URL: <https://www.zotero.org/> (term. wiz. 21.01.2017).

Dodatki

Dodatek do niniejszej pracy stanowi płyta CD zawierająca:

- pracę dyplomową w postaci źródłowej (LaTeX),
- pracę dyplomową w postaci pliku pdf.

Spis rysunków

3.1	Schemat klasycznego wzorca MVC	8
3.2	Obliczenia chmurowe	9
3.3	Raspberry Pi 3 B	10
3.4	Czujnik DHT11	11
3.5	Raspberry Pi Camera Hd	11
3.6	Efekt działania thresholdingu a) przed b) po	12
3.7	Wykres wartości obrazu <i>src</i> przed thresholdingiem	12
3.8	Wykres wartości pikseli obrazu <i>dst</i>	12
3.9	Przebieg przykładowej dylatacji a) przed b) w trakcie c) po	13
3.10	Przykład zastosowania rozmycia gaussowskiego na zdjęciu	13
3.11	Szablony cech Haar’a a) dwuprostokątny b) trójpłaskokątny c) czteropłaskokątny d) centralny	14
3.12	Budowa głębokiej sieci neuronowej	15
5.1	Budowa systemu zarządzania domem	22
5.2	Podział systemu na usługi Azure	23
5.3	Podział systemu na usługi AWS	23
6.1	Wygląd strony głównej aplikacji internetowej (rozwiązanie 1)	24
6.2	Wygląd strony głównej aplikacji internetowej (rozwiązanie 2)	26
6.3	Widok detekcji twarzy	26
6.4	Tworzenie nowej detekcji	27
6.5	Widok zakończonej detekcji	27
6.6	Lista utworzonych profili	28
6.7	Widok profilu	29
6.8	Strona przedstawiająca istniejące grupy usług rozpoznawania tożsamości	29
6.9	Tworzenie nowej grupy sieci neuronowych	30
6.10	Szczegółowe informacje o grupie sieci neuronowych	30
6.11	Lista zadań identyfikacji osoby	31
6.12	Formularz tworzenia zadania identyfikacji	31
6.13	Zakończony request identyfikacji	32

6.14	Widok pozwalający na wybór dnia	32
6.15	Widok wyświetlający wszystkie odczyty z wybranego dnia	33
6.16	Widok wyświetlający wszystkie odczyty z wybranego dnia	33
7.1	Proces działania aplikacji konsolowej	34
7.2	Proces wykrywania twarzy zaimplementowany w aplikacji konsolowej	35
7.3	Wykrywanie twarzy metodą Haar zaimplementowane w aplikacji konsolowej	36
7.4	Wykrywanie twarzy metodą DNN (Deep Neural Network) zaimplementowane w aplikacji konsolowej	36
7.5	Wykrywanie twarzy używając Azure Cognitive Services zaimplementowane w aplikacji konsolowej	37
7.6	Proces trenowania sieci zaimplementowany w aplikacji konsolowej	38
7.7	Trenowanie sieci neuronowej OpenCv zaimplementowany w aplikacji konsolowej	39
7.8	Trenowanie usługi Azure Cognitive Services zaimplementowane w aplikacji konsolowej	40
7.9	Proces identyfikacji tożsamości zaimplementowany w aplikacji konsolowej	41
7.10	Proces identyfikacji osoby wykorzystując identyfikatory OpenCv zaimplementowany w aplikacji konsolowej	42
7.11	Proces identyfikacji osoby wykorzystując Azure Cognitive Services zaimplementowany w aplikacji konsolowej	43
8.1	Proces pobierania danych z sensorów zaimplementowany w aplikacji konsolowej do zarządzania domem	45
8.2	Proces wykrywania ruchu zaimplementowany w aplikacji konsolowej do zarządzania domem	46
8.3	Proces sprawdzenia czy wystąpił ruch zaimplementowany w aplikacji konsolowej do zarządzania domem	47

Spis tablic

3.1	Parametry czujnika	11
4.1	Dostępne systemy przetwarzania obrazu	17