POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Technologie informacyjne w systemach automatyki (ART)

PRACA DYPLOMOWA INŻYNIERSKA

Biometryczny system kontroli dostępu - przetwarzanie obrazów i przygotowanie danych dla sieci neuronowych

Biometric access control system - image and data preprocessing for neural nets

AUTOR:
Jędrzej Kozal

PROWADZĄCY PRACĘ:
dr inż. Piotr Ciskowski

OCENA PRACY:

WROCŁAW 2017

Rozdział 1

Wstęp

Celem pracy jest porównanie metod i algorytmów umożliwiających wykrycie twarzy na zdjęciu oraz wyznaczenie wektora uczącego dla sieci neuronowej na podstawie wykrytego obszaru zdjęcia zawierajacego twarz. Projekt ten jest częścią systemu biometrycznej kontroli dostępu. Za implementację sieci neuronowych oraz analizę algorytmów związanych z sieciami odpowiadał Filip Guzy. Za architekturę systemu, oraz komunikację pomiędzy komponentami odpowiadał Michał Leś.

Celem całego projektu było skonstruowanie prostego systemu kontroli dostępu opartego na rozpoznawaniu twarzy. System ma określać czy na wykonanym zdjęciu występuje twarz. Po wykryciu twarzy następuje ekstrakcja cech - wyznaczenie wektora cech i przygotowanie wektora uczącego dla sieci. Zadaniem sieci neuronowej jest klasyfikacja użytkownika i podjęcie decyzji czy danemu użytkownikowi powinien zostać przyznany dostęp, czy też nie.

System oryginalnie został zaprojektowany na mikrokomputer Raspberry Pi. W projekcie wykorzystano Raspberry Pi 3. Przy projektowaniu systemu fakt ten miał niebagatelne znaczenie ze względu na ograniczone pokłady mocy obliczeniowej. Architekutra systemu została zaprojektowana w taki sposób, aby system można było z łatwością przenosić na inne platformy.

1.1 Omówienie zagadnienia

Biometria to nauka zajmująca się mierzeniem cech osobniczych. Jest wykorzystywana w systemach kontroli dostępu w celu zwiększenia komfortu. Użytkownik systemu korzystającego z takiego systemu nie musi pamiętać haseł, nosić przy sobie kart identyfikacyjnych, czy kluczy. Cechy osobnicze dzielmy na fizyczne i behawioralne. Najczęściej przedstawianymi cechami są:

1. Cechy fizyczne:
    - Siatkówka
    - Tęczówka
    - Odcisk palca
    - Twarz

- Geometria dłoni

2. Cechy behawioralne:
   - Głos
   - Podpis
   - Sposób chodzenia

Cechy najczęściej są unikatowe dla poszczególnych osobników, co zapewnia ich wiarygodność. Biometryka cieszy się rosnącą popularnością, ze względu na wiele możliwych zastosowań. Dodatkowo wzrost możliwości sprzętowych oraz rozwój technik Machine Learning i Pattern Recognition przyczyniają się do znaczących postępów w omawianej dziedzinie. Metody statystyczne umozliwiają ilościową analizę pomiarów wybranych cech osobniczych. Zastosowanie tanich czujników w urządzeniach produkcji masowej powoduje przenikanie tej dziedziny do życia codziennego. Niektóre smartfony zawierają biometryczne systemy kontroli dostępu.

Przetworzenie zebranych pomiarów i ich ilościowa analiza, umożliwiająca identyfikację osobnika jest trudnym zadaniem. Wymaga przeniesienia fizycznych pomiarów na bardziej abstrakcyjne poziomy i znalezienie ukrytych zależności. Często bardzo pomocna jest inna reprezentacja danych.

W realizowanym systemie zdecydowano się na analizowanie twarzy ze względu na łatwą akwizycję cech osobniczych. Przyjęto następujące kroki przetwarzania danych:

1. Akwizycja obrazu

   Jako czujnik wykorzystano kamerę mikrokomputera Raspberry Pi. Zawiera ona matrycę OV5647 i umożliwia wykonywanie zdjęć w rozdzielczości 5 MPx. Komunikacja z mikrokomputerem odbywa się przez port CSI. Jest to wygodne rozwiązanie i daje wiele możliwości rozwoju projektu.

2. Przetwarzanie obrazu

   Przed operacją wykrywania twarzy z wykorzystaniem kaskad Haara obraz jest konwertowany do skali szarości. W przypadku ekstrakcji głębi na podstawie obrazów z dwóch kamer, z obrazu są usuwane zakłócenia.

3. Wykrycie twarzy

   Następnym krokiem jest określenie, czy przetworzone zdjęcie zawiera twarz i oszacowanie jej współrzędnych. Jeśli zdjęcie nie zawiera twarzy system informuje o tym i proces przyznawania dostępu kończy się negatywnie.

4. Ekstrakcja cech

   Na podstawie obszaru zawierającego twarz przeprowadzana jest ekstrakcja cech. Cechy te stanowią wektor podawany na wejście sieci neuronowej.

5. Rozpoznawanie osobnika

   Wektor cech jest wykorzystywany przez sieć neuronową do klasyfikacji i podjęcia decyzji czy dostęp został przyznany czy nie.

W zakresie tej pracy szczegółowo zostaną omówione rozwiązane przyjęte w zakresie realizacji punktu 3 i 4.

1.2 Omawiany komponent jako część większego systemu

Jak wspomniano we wstępie, komponent opisany w niniejszej pracy jest częścią większego systemu. W tej sekcji pracy przedstawiono założenia projektowe, zaprezentowano szersze spojrzenie na cały system. Omówiono komponent z programistycznego punktu widzenia: omówiono wzorce projektowe, praktyki jakich przesztrzegano projektując komponent oraz uzasadniono wybór wykorzystanych narzędzi.

1.2.1 Wykorzystane wzorce projektowe

W celu ułatwienia pracy współpracownikom w projekcie został utworzony wydzielony łatwo wymienialny komponent. W celu ułatwienia pracy badawczej komponent został podzielony na mniejsze elementy z wykorzystaniem obiektowych wzorców projektowych. Do reprezentacji całego komponentu wybrano wzorzec "Fabryka". Zapewniono w ten sposób elastyczność użytkownikom klasy, ponieważ mogą oni zdecydować w jaki sposób współrzędne będące wynikiem działania algorytmów mają być przechowywane. Zapewniono klasę bazową do przechowywania wektora uczącego dla sieci, po której dziedziczą poszczególne reprezentacje, takie jak bezpośrednie przechowywanie danych w pamięci programu, czy zapis w pamięci nieulotnej w postaci pliku z rozszerzeniem .npy z biblioteki numpy. W celu zwiększenia elastyczności w obrębie komponentu do wyboru algorytmów dwukrotnie wykorzystano wzorzec "Strategia". Stworzono klasy bazowe do reprezentacji podstawowych własności algorytmu, które następnie implementowano w klasach pochodnych. Modyfikacja ta ułatwiła pracę badawczą i usystematyzowała strukturę projektu.

1.2.2 Wykorzystane zasady i dobre praktyki programowania

W trakcie realizacji projektu oprócz wykorzystania wzorców projektowych posługiwano się także dobrymi zasadami SOLID oraz Clean Code. Pozwalają one na zachowanie większego porządku oraz czytelności kodu. SOLID opisuje dobre praktyki programowania obiektowego i składa się z pięciu zasad:

1. Pojedyncza odpowiedzialność (Single resposibility)

   Zasada ta zawiera się w sformuowaniu: powinien istnieć jeden i tylko jeden powód do modyfikacji klasy. Każda z klas powinna mieć tylko jedną odpowiedzialność. Jednoznaczne zdefiniowanie odpowiedzialności klas umożliwia łatwiejszą modyfikację kodu i definiowanie nowych zachowań. Zasada ta prowadzi do powstania wielu klas w projekcie, jednakże rozsądne zaprojektowanie struktury klas pozwala na uniknięcie nieporządku w projekcie.

2. Otwarte/Zamknięte (Open/Closed)

   Zasada ta oznacza że należy projektować i tworzyć klasy w taki sposób, aby ich rozszerzanie było łatwe, jednocześnie nie wymagało modyfikacji już istniejącego kodu. W zachowaniu tej zasady mogą pomóc takie narzędzia jak polimorfizm, czy wzorzec projektowy constructor chaining.

3. Zasada podstawienia Liskov (Liskov substitution)

   "Jeżeli q(x) jest udowadnialną własnością obiektu x typu T, to q(y) powinno być prawdziwe dla obiektu y typu S, gdzie S jest podtypem T." Zasada ta mówi że w dowolnym momencie klasa bazowa może się zamienić z klasą pochodną. Przestrzeganie tej zasady umożliwia stosowanie wskaźników na klasę bazową wskazujące na klasy pochodne bez jakichkolwiek obaw. Klasa pochodna zachowa się w podobny sposób do klasy bazowej, bez niepożądanych bądź ukrytych efektów ubocznych wykonania funkcji. Najczęściej podawanym przykładem nieprzestrzegania tej zasady jest klasa Kwadrat dziedzicząca po klasie Prostokąt. Zmiana wysokości lub długości podstawy w klasie bazowej odpowiada jednej operacji zmiany długości boku w klasie pochodnej, przez co jedna z operacji z klasy bazowej traci sens, a użycie klasy pochodnej w miejsce bazowej może mieć niepożądane skutki.

4. Zasada niezależności interfejsów (Interface segregation principle)

   Segregacja interfejsów polega na ograniczeniu rozmiarów interfejsów. Klasy powinny implementować tylko te metody z interfejsów które są im potrzebne. Jeśli tak nie jest, należy podzielić interfejsy na mniejsze i dać użytkownikowi interfejsu opcję dziedziczenia tylko potrzebnych interfejsów. Przestrzeganie tej zasady pozwala na tworzenie minimalistycznego kodu, który ma wykonywać to co zakładano, a dodatkowo jest łatwy w refaktoryzacji.

5. Odwrócenie zależności (Dependency inversion)

   "Wszystkie zależności powinny zależeć od abstrakcji, a nie od konkretnego typu." Wysokopoziomowe moduły nie mogą posiadać zależności do niskopoziomowych modułów. Zamiast tego oba powinny polegać na abstrakcjach. Zasada ta jest mocno powiązana z pierwszą zasadą. Jeśli każda klasa posiada tylko jedną odpowiedzialność zdefiniowanie jasnych interfejsów między nimi jest łatwiejsze. Definiowanie abstrakcyjnych interfesjsów pozwala na jasne oddzielenie od siebie różnych komponentów i delegowanie zadań. W przestrzeganiu tej zasady pomaga taka technika jak wstrzykiwanie zależności.

Clean Code to zbiór zasad dotyczących tworzenia nazewnictwa i pisania oprogramowania w taki sposób, aby był on jak najbardziej czytelny dla innych programistów. Według założeń Clean Code wszystkie elementy oprogramowania takie jak klasy, struktury, funkcje, metody, obiekty miały nazwę adekwatną do realizowanej czynności lub przeznaczenia. Dodatkowo nazwy obiektów o lokalnym zakresie i krótkim czasie życia powinny być jak najkrótsze. Nazwy obiektów o dużym zasięgu powinny być jak najdłuższe. W przypadku funkcji zasda ta jest odwrócona: projektując interfejs klasy powinniśmy udostępniać użytkownikowi jak najkrótsze nazwy. Nazwy metod prywatnych powinny być jak najdłuższe i dokładnie opisywać za co odpowiada dana metoda. Metody powinny być krótkie. Robert C. Martin zaleca aby metody miały maksymalnie 6 linijek kodu. Jedną z podstawowych zasad tej ideologi jest unikanie komentowania kodu. W założeniach Clean Code, kod powinien być napisany w taki sposób, aby używanie komentarzy było zbędne. Mówi się wręcz, że "Dobrze napisany kod czyta się jak prozę". Jednym z podstawowych założeń omawianej metodologii jest duże pokrycie kodu testami jednostkowymi. Nie zdecydowano się na wykorzystanie testów jednostkowych przy pracy nad omawianym komponentem ze względu na dynamiczny charakter projektu. Definiowanie zachowań kodu w testach jednostkowych jest czasochłonne, a częste zmiany w systemie powodują, że należałby usuwać

część testów. Ideologia Clean Code zawiera także wiele innych zasad, których stosowanie prowadzi do zwiększenia porządku i czytelności kodu. Metodyka ta cieszy się dużą popularnością wśród wielu programistów, dodatkowo niektóre firmy wymagają od swoich programistów przestrzegania zasad tej metodyki.

Dobra organizacja kodu oraz porządek ułatwiają rozwijanie projektu oraz systematyzują pracę.

1.2.3 Wykorzystane biblioteki, narzędzia i zasoby

Projekt systemu biometrycznej kontroli dostępu został zrealizowany całkowicie w języku programowania Python. Decyzja ta była podyktowana głównie znaczącą liczbą gotowych bibliotek, które bardzo ułatwiają pracę. Wykorzystana została wersja języka 2.7.9, ponieważ biblioteki języka Python często wymagają wersji 2.7 lub wyższej. W omawianym komponencie wykorzystano biblioteki NumPy, SciPy, scikit-learn, openCV oraz Dlib.

1. NumPy

   Jest to biblioteka zawierająca wiele algorytmów do realizacji obliczeń numerycznych oraz implementację wielu matematycznych narzędzi związanych z algebrą liniową. Implementacja algorytmów macierzowych z NumPy cieszy się dużą popularnością, dlatego warto poświęcić czas na poznanie tej biblioteki, ponieważ stanowi ona fundament wielu innych projektów, a operacje macierzowe z jej wykorzystaniem są bardzo dobrze zoptymalizowane.

2. SciPy

   SciPy jest biblioteką wykorzystywaną do obliczeń naukowych i inżynierskich. Zawiera algorytmy umożliwiające przeprowadzanie obliczeń w wielu dziedzinach, dlatego również jest wykorzystywana jako baza innych projektów. Instalacja SciPy jest prerekwizytem do instlacji scikit-learn.

3. Scikit-learn

   Jest to biblioteka zawierająca wiele algorytmów z dziedziny machine learning oraz pattern recognition. Jest to kluczowa biblioteka w omawianym projekcie, ponieważ dostarcza najistotniejszych narzędzi. Jej zastosowanie znacznie ułatwiło pracę.

4. OpenCv

   Biblioteka ta stanowi potężny zbiór narzędzi do analizy oraz przetwarzania obrazów. Twórcy biblioteki skupiają się na działaniu w czasie rzeczywistym i zapewniają narzędzia do zrównoleglania obliczeń. Jest to znaczące ułatwienie, biorąc pod uwagę podstawową platformę sprzętową, na której był realizowany projekt.

5. Dlib

   Dlib jest biblioteką przewidzianą głównie dla języka C++, jednakże możliwe jest także wykorzystanie jej w Pythonie. Zawiera głównie algorytmy z dziedziny Machine Learning (ML). W omawianym projekcie została wykorzystana do zbadania możliwości alternatywnych metod detekcji twarzy.

Wszystkie wyżej wymienione biblioteki są rozpowszechniane w ramach licencji new-BSD lub 3-clause BSD, co oznacza, że wolno je wykorzystywać w celach akademickich

lub komercyjnych. Zastosowanie bibliotek dostarczających zoptymalizowanych algorytmów znacznie ułatwiło i usystematyzowało pracę nad projektem.

Na początku pracy nad projektem podjęto próbę implementacji własnej wersji algorytmu PCA, aby lepiej zrozumieć i dokładnie prześledzić jego działanie. Algorytm ten działał poprawnie, ale ze względu na dużą kosztowność obliczeniową zdecydowano się na korzystanie z wersji udostępnionej w bibliotece scikit-learn. W trakcie późniejszych prac nad systemem znaleziono informację na temat tego, co powodowało tak wysoką złożoność obliczeniową oraz sposób na jej uniknięcie. Zagadnienie to zostało opisane w Rozdziale 3.1.2 "Realizacja algorytmu".

Diagramy i rysunki wykonano z wykorzystaniem strony draw.io. Do wszystkich zapożyczonych obrazów zostały podane źródła.

Do przeprowadzenia badań jako bazy twarzy wykorzystano zbiór zdjęć dostępny w ramach biblioteki scikit-learn The Olivetti faces dataset. Posłużył on jako baza do wyznaczenia wartości własnych przestrzeni zdjęć twarzy. Informacje na temat bazy zdjęć, oraz link do strony bazy zostały podane w [17].

Rozdział 2

Implementacja algorytmów – detekcja twarzy

Pierwszym wymaganiem, po zdolności do akwizycji obrazów jest określenie czy uzyskane zdjęcie przestawia twarz. W niniejszym rozdziale omówiono możliwe metody detekcji twarzy na dwuwymiarowym obrazie. Szczegółowo omówiono przyjęte rozwiązanie oraz podano uzasadnienie dlaczego zostało ono wybrane.

2.1 Kaskady Haara

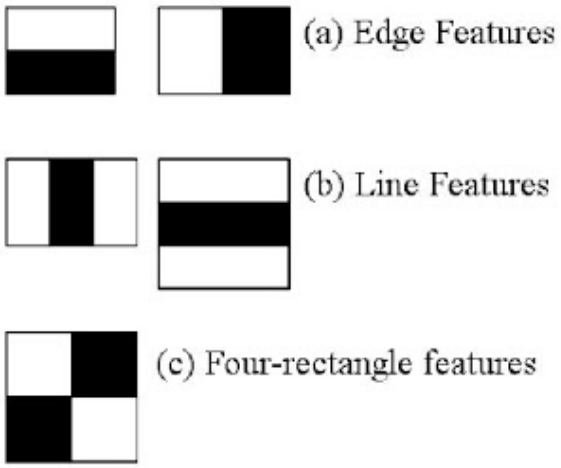
Algorytm Kaskad Haara (detektor Viola-Jones) wykorzystuje serię cech do określenia czy na zdjęciu w określonym miejscu znajduje się szukany obiekt. Cecha jest tutaj rozumiana jako prostokąt podzielony na części. Pod każdą z części dodawane są wartości pikseli, następnie wartości z obu obszarów są od siebie odejmowane. Koncepcja ta przypomina falkę Haara, stąd druga część nazwy algorytmu. Tak uzyskana wartość musi przekroczyć odpowiedni próg, który decyduje o tym czy w danym miejscu rozpoznano twarz. Jak podano w [7], aby móc szybciej obliczać wartości pikseli pod odpowiednim obszarem, wykorzystuje się obraz kumulacyjny (ang. integral image). Jest on zdefiniowany jako:

$$ii(x, y) = \sum_{x' \leqslant x, y' \leqslant y} i(x', y') \tag{2.1}$$

Funkcja ta przypomina dyskretną dystrybuantę dwuwymiarowej zmiennej losowej. Jej zastosowanie pozwala na obliczenie sumy jasności pod dowolnie wielkim obszarem w dowolnym miejscu obrazu w stałym czasie.

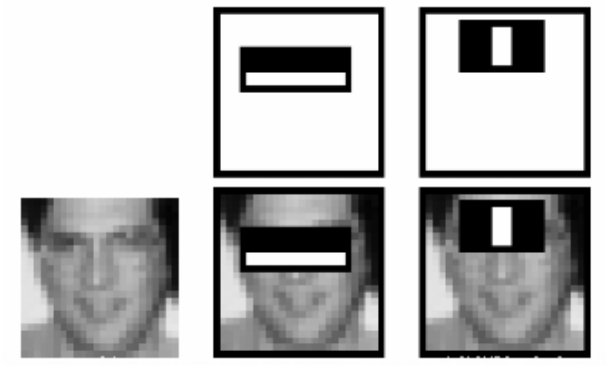
Nawet dla najmiejsznych obrazów można rozważać wiele cech. W celu ograniczenia ilości analizowanych cech wykorzystano alogrytm AdaBoost [6]. Algorytm ten posiadając zbiór uczącym oraz zbiór słabych hipotez, przyporządkowywuje wagi hipotezom, oceniając ich skuteczność. Dodatkowo wagi otrzymują także przykłady ze zbioru uczącego. Każdorazowo kiedy dany przykład zostanie źle zaklasyfikowany, jego waga jest zwiększana.

Wykorzystanie AdaBoost pozwoliło na wybranie cech generujących najmniejszy błąd. Ilość użytych cech wpływa na jakość algorytmu, jednocześnie zwiększając czas potrzebny na obliczenia. Jak można zauważyć na rys. 2.2 najlepsze cechy mają dość intuicyjną inter-



Rysunek 2.1: Różne rodzaje cech. Od sumy wartości pikseli pod czarnym obszarem, jest odejmowana wartość sumy pikseli pod białym polem.

Źródło: https://docs.opencv.org3.3.0d7d8btutorial_py_face_detection.html



Rysunek 2.2: Dwie cechy generujące najmniejszy błąd z wykorzystaniem AdaBoost.

Źródło: https://docs.opencv.org3.3.0d7d8btutorial_py_face_detection.html

Rysunek 2.3: Wykrywanie punktów charakterystycznych twarzy.
W lewej części - oryginalny obraz z kamertki, w prawej - części, obraz z zaznaczonym rezultatami detekcji. Wykrozystano 68 cech.

pretację: spodziewamy się, że obraz w okolicach oczu będzie ciemniejszy niż w okolicach policzków oraz, że środek nosa będzie jaśniejszy niż obszary po bokach.

Dysponując zbiorem cech można przystąpić do klasyfikacji obrazu. Odbywa się to kaskadowo. W pierwszym etapie wykorzystuje się klasyfikator złożony najczęściej z jednej lub dwóch cech. Klasyfikacja jest przeprowadzana na całym obrazie, aby wykryć gdzie może znajdować się twarz. W kolejnych etapach ilość cech jest zwiększana (najczęściej wykładniczo). Jeśli w danym obszarze nie została wykryta twarz, zostaje on pominięty w dalszych etapach. Pozwala to znacząco ograniczyć ilość wykonywanych operacji dla kolejnych klasyfikatorów. Zaleca się aby w pierwszym etapie wykorzystywać cechę z wysokim współczynnikiem fałszywych negatywnych rezultatów (ang. false negative) – w tym przypadku obszarów zawierających twarz, błędnie zaklasyfikowanych jako nie będących twarzą.

Kaskady Haara są szybkie, przez co chętnie wykorzystuje się je w projektach wymagających przetwarzania obrazów w czasie rzeczywistym. Było to także motywacją przy wyborze tej metody w omawianym systemie. Rozwiązanie daje zadowalające wyniki przy jednoczesnym szybkim działaniu. W rozważanym systemie złożoność obliczeniowa musi być rozpatrywana, biorąc pod uwagę, że oryginalna platforma sprzętowa na którą został przewidziany system to Rasbery Pi.

2.2 Inne sposoby rozwiązania problemu

Do rozwiązania problemu można zastosować inne podejście, polegające na znajdowaniu charakterystycznych punktów twarzy. W pracy [8] wykorzystano boosting drzewa gradientu do stworzenia kaskady regresorów. Dysponując zbiorem uczącym zawierającym zdjęcia twarzy oraz współrzędne punktów charakterystycznych można wyznaczyć kaskadę regresorów liniowych. Zakładamy, że estymowany kształt punktów charakterystycznych dowolnej twarzy może być wyrażony jako kombinacja liniowa kształtu punktów charakterystycznych z odpowiednio dużego zbioru uczącego. Każdy regresor to drzewo decyzyjne, które w liściach przechowuje wektor, który powinien zostać dodany do aktualnej estymacji punktów charakterystycznych (zbiór współrzędnych x i y, które powinny być kolejno dodane do każdej współrzędnej). W pierwszym kroku jako estymacja jest wykorzystywany średni kształt z bazy zdjęć uczących. Następnie każdy regresor delikatnie poprawia

estymację, przesuwając punkty w odpowiednią stronę. Na rys. 2.3 przedstawiono efekty detekcji twarzy z wykorzystaniem implementacji z biblioteki Dlib.

Omawiane rozwiązanie umożliwia całkowicie inne podejście do problemu ekstrakcji cech. Mając estymatę, charakterystyczne punkty twarzy oraz mierząc odległości między poszczególnymi punktami można przygotować wektor uczący, mający bardzo fizyczną interpretację. Dodając informację na temat koloru oczu czy skóry, można skonstruować zupełnie inne wektory cech niż proponowany w tej pracy.

Rozdział 3

Implementacja algorytmów – ekstrakcja cech

Niniejszy rozdział traktuje o algorytmach, umożliwiających ekstrakcję cech twarzy ze zdjęcia. Dysponując zdjęciem twarzy, należy zaproponować sposób utworzenia wektora, będącego reprezentatywnym odzwierciedleniem ilościowych cech ze zdjęcia i jednocześnie umożliwiającym sieci neuronowej dokonanie klasyfikacji. Zadanie to jest bardzo abstrakcyjne, zależy od wielu czynników i trudno jest określić rezultaty.

W rozdziale tym skupiono się głównie nad podejściem polegającym na zastosowaniu algorytmów redukcji wymiarowości, takich jak PCA czy SVD. Jest to dość popularne rozwiązanie i w przypadku PCA można uzyskać wyniki mające wizualną interpretację, co jest bardzo pomocne przy pracy nad tego typu problemem.

Obraz po wykryciu twarzy jest obcinany do kwadratu, na którym wykryto twarz następnie obraz jest obcinany do rozmiaru 64×64 piksele, w celu ujednolicenia rozmiarów przetwarzanych obrazów oraz zwiększenia szybkości przetwarzania.

3.1 Principle Components Analysis

Principle Components Analysis (PCA) jest algorytmem redukcji wymiarowości danych. Mając określony zbiór obserwacji, będący reprezentatywną reprezentacją próbą analizowanej przestrzeni należy wyznaczyć zbiór wektorów bazowych, tak aby wariancja danych ze zbioru obserwacji, względem nowo wyznaczonych wektorów bazowych, była jak największa.

3.1.1 Wstęp matematyczny

Przedstawione tu wyprowadzenie jest podawane za Pattern recognition and machine learning [1].

Zakładamy, że dany jest zbiór D-wymiarowych wektorów obserwacji: $\mathbf{x}_n, n \in 1, ..., N$. Celem jest znalezienie nowej M-wymiarowej bazy ($M < D$) takiej, że wariancja zbioru wektorów $\mathbf{x}_n$ po zrzutowaniu na nową bazę jest największa. W celu pokazania procesu i rozumowania przyjmijmy, że szukamy jednego wektora $\mathbf{u}_1$. Wektor ten, ponieważ ma

stanowić bazę, musi być D-wymiarowy. Dodatkowo przyjmiemy, że wektor $\mathbf{u}_1$ jest jednostkowy, co zapiszemy w postaci warunku:

$$\mathbf{u}_1^T\mathbf{u}_1 = 1 \tag{3.1}$$

Rzut wektora $\mathbf{x}$ na kierunek wyznaczany przez wektor $\mathbf{u}$ jest dany przez: $u^T x$. Fakt ten ma proste uzasadnienie geometryczne:

$$\begin{aligned}\mathbf{u}^T\mathbf{x} &= \mathbf{u} \circ \mathbf{x} \\ &= ||\mathbf{u}||||\mathbf{x}||cos\angle(\mathbf{u},\mathbf{x}) \\ &= ||\mathbf{x}||cos\angle(\mathbf{u},\mathbf{x})\end{aligned} \tag{3.1}$$

Przyjmimy średnią jako estymator wartości oczekiwanej:

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{3.2}$$

Wariancja zmiennej losowej X jest dana przez:

$$\begin{aligned}var[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2] \\ &= \mathbb{E}[X^2] - \mathbb{E}[X]^2\end{aligned}$$

Wariancja jest miarą rozrzutu wartości zmiennej losowej. Im większe różnice między wartościami przyjmowanymi przez zmienną losową, tym większa jest wariancja.

Estymator wariancji dla próby losowej x_n:

$$s^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \overline{x})$$

Kowariancja między dwoma zmiennymi losowymi jest dana wzorem:

$$\begin{aligned}cov[X,Y] &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]\end{aligned} \tag{3.3}$$

Wzór ten można uogólnić dla wektorów losowych:

$$\begin{aligned}cov[\mathbf{X},\mathbf{Y}] &= \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y}^T - \mathbb{E}[\mathbf{Y}^T])] \\ &= \mathbb{E}[\mathbf{X}\mathbf{Y}^T] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{Y}^T]\end{aligned}$$

Definiujemy macierz kowariancji S dla zbioru obserwacji $\mathbf{x}_n$:

$$S = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T \tag{3.4}$$

Jak łatwo zauważyć, w powyższym wzorze wartości oczekiwane zostały zastąpione średnimi, ponieważ nie dysponujemy rozkładami zmiennnych losowych, tylko zbiorem obserwacji.

Definiujemy wariancję zbioru obserwacji względem wektora u_1:

$$s^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}})^2 = \mathbf{u}_1^T S\mathbf{u}_1 \tag{3.5}$$

Chcąc maksymalizować wariancję s^2 musimy pamiętać o warunku 3.1. Otrzymujemy więc problem optymalizacji polegający na znalezieniu maksimum warunkowego funkcji $f(\mathbf{u}_1)$ z ograniczeniem $g(\mathbf{u}_1)$:

$$\begin{aligned} f(\mathbf{u}_1) &= \mathbf{u}_1^T S \mathbf{u}_1 \\ g(\mathbf{u}_1) &= 1 - \mathbf{u}_1^T \mathbf{u}_1 = 0 \end{aligned}$$

Jest to problem, który można łatwo rozwiązać z wykorzystaniem mnożników Lagrange'a. W analizowanym przypadku Lagrangian przyjmuje postać:

$$\begin{aligned} \mathcal{L}(\mathbf{u}_1, \lambda) &= f(\mathbf{u}_1) - \lambda g(\mathbf{u}_1) \\ &= \mathbf{u}_1^T S \mathbf{u}_1 - \lambda(1 - \mathbf{u}_1^T \mathbf{u}_1) \end{aligned}$$

Różniczkując względem $\mathbf{u}_1$ oraz λ oraz przyrównując do zera uzyskujemy układ równań:

$$\begin{cases} \nabla_{\mathbf{u}_1} \mathcal{L}(\mathbf{u}_1, \lambda) & = \nabla_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 - \nabla_{\mathbf{u}_1} \lambda \mathbf{u}_1^T \mathbf{u}_1 = 0 \\ \frac{\partial}{\partial \lambda} \mathcal{L}(\mathbf{u}_1, \lambda) & = 1 - \mathbf{u}_1^T \mathbf{u}_1 = 0 \end{cases} \tag{3.6}$$

Zajmijmy się pierwszym równaniem z układu 3.6. Odjemnik można bardzo łatwo uprościć:

$$\nabla_{\mathbf{u}_1} \lambda \mathbf{u}_1^T \mathbf{u}_1 = \lambda \nabla_{\mathbf{u}_1} (u_{1_1}^2 + u_{1_2}^2 + ... + u_{1_D}^2) = 2\lambda \mathbf{u}_1 \tag{3.7}$$

Odjemna wymaga wiecej przekształceń:

$$\nabla_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 = \nabla_{\mathbf{u}_1} \begin{pmatrix} u_{1_1} & u_{1_2} & ... & u_{1_D} \end{pmatrix} \begin{pmatrix} s_{1_1} u_{1_1} + s_{1_2} u_{1_2} + ... + s_{1_D} u_{1_D} \\ s_{2_1} u_{1_1} + s_{2_2} u_{1_2} + ... + s_{2_D} u_{1_D} \\ \vdots \\ s_{D_1} u_{1_1} + s_{D_2} u_{1_2} + ... + s_{D_D} u_{1_D} \end{pmatrix} \tag{3.8}$$

$$\begin{aligned} &= \nabla_{\mathbf{u}_1} [u_{1_1}(s_{1_1} u_{1_1} + s_{1_2} u_{1_2} + ... + s_{1_D} u_{1_D}) \qquad (3.9) \\ &+ u_{1_2}(s_{2_1} u_{1_1} + s_{2_2} u_{1_2} + ... + s_{2_D} u_{1_D}) \\ &+ ... + u_{1_D}(s_{D_1} u_{1_1} + s_{D_2} u_{1_2} + ... + s_{D_D} u_{1_D})] \\ &= \begin{pmatrix} 2s_{1_1} u_{1_1} + 2s_{1_2} u_{1_2} + ... + 2s_{1_D} u_{1_D} \\ 2s_{2_1} u_{1_1} + 2s_{2_2} u_{1_2} + ... + 2s_{2_D} u_{1_D} \\ \vdots \\ 2s_{D_1} u_{1_1} + 2s_{D_2} u_{1_2} + ... + 2s_{D_D} u_{1_D} \end{pmatrix} \qquad (3.10) \\ &= 2S\mathbf{u}_1 \qquad (3.11) \end{aligned}$$

W przejściu między (9) a (10) korzystamy z faktu, że macierz S (macierz kowariancji) jest symetryczna.

Wracając do (3.6):

$$\begin{aligned} 2S\mathbf{u}_1 - 2\lambda \mathbf{u}_1 &= 0 \\ S\mathbf{u}_1 &= \lambda \mathbf{u}_1 \end{aligned} \tag{3.12}$$

Jest równaniem na wartości własne macierzy S, więc u_1 musi być wektorem własnym macierzy S, λ wartością własną macierzy S.

Mnożąc (11) lewostronnie przez u_1^T otrzymujemy:

$$\mathbf{u}_1^T S \mathbf{u}_1 = \lambda \tag{3.13}$$

Lewa część (13) to wariancja, która ma być maksymalizowana. Aby uzyskać jak największą wariancję zbioru obserwacji, po zrzutowaniu na kierunek wyznaczany przez $\mathbf{u}_1$, należy wybrać największą wartość własną macierzy S. Odpowiadający jej wektor własny jest szukanym wektorem $\mathbf{u}_1$. Dla kolejnych wektorów $\mathbf{u}$ należy znaleźć kolejne największe wartości własne i wektory własne.

3.1.2 Realizacja algorytmu

Korzystając z PCA można zmniejszyć wymiarowość danych w taki sposób aby zachować najwięcej informacji. Działanie algorytmu można zasadniczo podzielić na dwa etapy. Pierwszy to uzyskanie bazy wektorów, która zostanie wykorzystana w drugim etapie do uzyskania nowej reprezentacji wektorów z oryginalnej przestrzeni.

W pierwszym etapie należy obliczyć wartości własne macierzy kowariancji, jak to zostało pokazane w poprzednim dziale. Przyjmijmy, że dysponujemy macierzą pomiarów:

$$X_{D\times N} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} x_{1_1} & x_{1_2} & \dots & x_{1_N} \\ x_{2_1} & x_{2_2} & \dots & x_{2_N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{D_1} & x_{D_2} & \dots & x_{D_N} \end{pmatrix} \tag{3.14}$$

Zdefiniujmy macierz A, powstałą w wyniku odjęcia od każdego wiersza średniej:

$$A_{D\times N} = \begin{pmatrix} \mathbf{x}_1 - \overline{\mathbf{x}} & \mathbf{x}_2 - \overline{\mathbf{x}} & \dots & \mathbf{x}_N - \overline{\mathbf{x}} \end{pmatrix} \tag{3.15}$$

Dysponując macierzą A, można zapisać S w postaci:

$$S_{D\times D} = \frac{1}{N} A A^T \tag{3.16}$$

Co jest całkowicie zgodnie z definicją 3.3. Kolejnym krokiem będzie wyznaczenie wektorów i wartości własnych, posortowanie wektorów własnych według nierosnącej wartości własnych oraz wybranie M wektorów własnych.

Należy zwrócić uwagę na wymiar macierzy S. Przypomnijmy, że D to wymiar wektora pomiarowego, co w przypadku zdjęć oznacza wartość każdego piksela przeniesioną kolejno do elementów wektora. Zdjęcia w tej pracy, przed podaniem na wejście algorytmu ekstrakcji cech były konwertowane do skali szarości i obcinane do rozmiaru 64 × 64 pikseli, co oznacza, że wektor pomiarowy zawierał 4096 współrzędnych. Biorąc pod uwagę złożoność obliczeniową algorytmów wyznaczania wektorów własnych może to powodować problemy z wydajnością. Można temu zapobiec poprzez zamianę miejscami macierzy A i A^T w iloczynie. Obliczając wartości własne macierzy $A^T A$ musimy wrócić do D-wymiarowości przestrzeni, co jest możliwe z wykorzystaniem następującej własności:

$$u_i = A v_i \tag{3.17}$$

gdzie v_i oznacza wartość własną macierzy $A^T A$. Poprzez zamianę miejscami macierzy A i A^T zyskujemy inny wymiar macierzy. Iloczyn ma teraz rozmiar NxN. Przypomnijmy, że N to liczność zbioru obserwacji. Zazwyczaj N jest znacznie niższe od D, co może przyśpieszyć obliczenia. Korzystając z tego uproszczenia należy pamiętać, że zostaje nałożone ograniczenie co do wielkości M. Jako, że zostają obliczone wektory własne macierzy o rozmiarze NxN, może ich być maksymalnie N. Oznacza to, że ilość wymiarów danych po

redukcji nie może przekraczać ilości danych ze zbioru obserwacji ($M < N$). Ograniczenie to nie jest zbyt dotkliwe, gdy dysponujemy odpowiednio wielkim zbiorem obserwacji. W przypadku tej pracy, rozmiar bazy zdjęć nie był problemem.

Po konwersji do oryginalnej wymiarowości dysponujemy bazą M wektorów D-wymiarowych. Zbiór M wektorów jest nazywany wartościami własnymi i zostanie wykorzystany w drugim etapie. Dla uproszczenia wektory własne zostaną włączone do jednej macierzy:

$$U = \begin{pmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_M^T \end{pmatrix} \tag{3.18}$$

W pewnym sensie macierz U wyznacza reguły, przy pomocy których konwertujemy wektory D-wymiarowe na M-wymiarowe.

Drugim etapem działania algorytmu jest redukcja wymiarowości wektora $\mathbf{w}$ z poza zbioru obserwacji. Obrazowo można powiedzieć, że na podstawie nowego zdjęcia, niewykorzystanego w poprzednim etapie chcemy uzyskać reprezentujący je wektor liczb w przestrzeni o niższym wymiarze. Odbywa się to zasadniczo w dwóch bardzo prostych krokach: Pierwszym jest odjęcie od nowego wektora średniej. Drugim jest rzutowanie różnicy $\mathbf{w} - \overline{\mathbf{x}}$ na wcześniej uzyskane wartości własne. Uzyskanie współrzędnych w M-wymiarowej przestrzeni wektora w można więc zapisać jako:

$$\mathbf{v} = U(\mathbf{w} - \overline{\mathbf{x}}) \tag{3.19}$$

Jak łatwo można zauważyć $\overline{\mathbf{x}}$ oraz U zależą bezpośrednio od zbioru obserwacji, jakimi dysponujemy na początku. Dobranie zbioru reprezentatywnego dla danej przestrzeni jest kluczowe. Pierwszy etap działania algorytmu, polegający na generacji wektorów własnych, definiuje przekształcenie. Drugi etap działania pozwala na redukcję wymiarowości dowolnego D-wymiarowego wektora zgodnie z regułami ustalonymi w pierwszym kroku.

3.1.3 Interpretacja

Warto poświęcić uwagę na to, co tak na prawdę zyskujemy w wyniku działania algorytmu PCA. Przez odjęcie średniej przesuwamy środek nowo powstałego układu współrzędnych do punktu gdzie znajdowała się średnia zbioru obserwacji. Dodatkowo, po odjęciu średniej, wektor reprezentuje jedynie różnicę miedzy średnią (środkiem nowego układu współrzędnych), a punktem końcowym. Jest więc reprezentacją unikatowych cech w zbiorze obserwacji. Wybranie wektorów własnych, będących osiami - kierunkami z największą wariancją powoduje, że odrzucając część bazy wektorów tracimy najmniej zróżnicowane dane. W celach badawczych można wrócić do oryginalnej przestrzeni, aby porównać błąd przybliżenia oraz przeanalizować jak bardzo reprezentacyjne są uzyskane wyniki. W przypadku zdjęć możliwe jest wręcz wizualne porównanie jak bardzo oryginalne zdjęcie i zdjęcie odtworzone po redukcji wymiarowości są podobne do siebie. Rekonstrukcja odbywa się w następujący sposób:

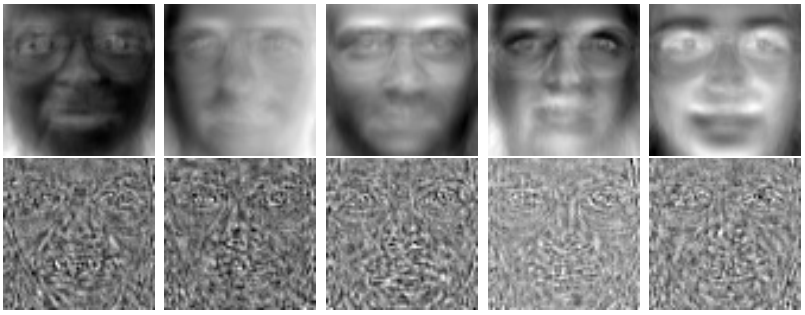
$$\mathbf{w} \approx \overline{\mathbf{x}} + \sum_{n=1}^{M} \mathbf{v}_n \mathbf{u}_n \tag{3.20}$$

Gdzie $\mathbf{v}_n$ oznacza n-ty element wektora v, a $\mathbf{u}_n$ oznacza n-ty wektor u.

Efekt rekonstrukcji przedstawiono na rys. 3.1. Dokonano redukcji wymiarowości zdjęcia twarzy autora systemu, oraz na podstawie uzyskanego wektora zrekonstruowano oryginalny obraz. Wraz ze zwiększeniem się ilości składowych jakość rekonstrukcji zdjęcia się

Rysunek 3.1: Porównanie rekonstrukcji zdjęcia twarzy autora pracy na postawie różnej liczby wykorzystanych składowych.

Z lewej strony oryginalne zdjęcie, następnie kolejno rekonstrukcja z wykorzystaniem 100, 200, 300 i 400 składowych.



Rysunek 3.2: Porównanie eigenfaces.

Górny wiersz - wektory 0-4, dolny wiersz - wektory 392-396.

polepsza. Użycie większej ilości składowych nie przynosi znaczącej poprawy jakości obrazu, co można zaobserwować dla rekonstrukcji z wykorzystaniem 300 i 400 składowych. PCA jest chętnie wykorzystywaną metodą, ponieważ jest bardzo intuicyjna oraz pozwala na wizualne sprawdzenie rezultatów.

W omawianym zagadnieniu wektory własne zyskały specjalną nazwę eigenfaces (twarze własne), można zauważyć jak wyglądają aplikując technikę odtwarzania zdjęcia na podstawie współrzędnych z przestrzeni o zredukowanej wymiarowości. Efekty zostały zaprezentowane na rys. 3.2.

Jak można łatwo zaobserwować, wektory 0-4 zawierają najbardziej istotne zmiany pomiędzy poszczególnymi obrazami ze zbioru obserwacji - odpowiadają one wartościom własnym o największej wariancji. Wektory 392-396 odpowiadają niższym wartościom własnym. Wektory te przypominają losowy szum i niosą informację o szczegółach twarzy. Można zauważyć, że pominięcie składowych z niższą wartością własną, powoduje mniejszą utratę informacji - dane w kierunkach wyznaczanych przez te wektory nie różnią się znacząco od siebie.

3.1.4 PCA jako samodzielny system rozpoznawania twarzy

Principle components analysis powoduje przeniesienie zdjęcia mającego wizualną reprezentację na abstrakcyjny wektor liczb. Umożliwia to skonstruowanie klasyfikatora z wykorzystaniem algorytmów umożliwiających klasyfikację. Jest to zagadnienie dobrze znane w dziedzinie Pattern Recognition (rozpoznawania wzorców) i opracowano wiele rozwiązań tego problemu. Zastosowanie algorytmów typu k-NN (k Nearest Neighbour - k najbliższych sąsiadów) lub NM (Nearest Mean - najbliższa średnia) umożliwia zaklasyfikowanie

danego algorytmu do jednej z grup. Jest to możliwie dzięki wykorzystaniu reprezentacji zdjęcia jako wektora liczb. Oba algorytmy dysponują zbiorem uczącym, który opisuje dane grupy, do których może zostać klasyfikowany dany wzorzec. W przypadku NM, dla każdej z grup, obliczana jest średnia wektorów, następnie obliczana jest odległość między wektorem podlegającym klasyfikacji, a średnimi zbiorów. Wektor zostaje zaklasyfikowany do zbioru, którego średnia jest w najmniejszej odległości od wektora. W przypadku algorytmu k-NN odległość jest obliczana pomiędzy wszystkimi wektorami ze zbioru uczącego. Następnie zostaje wybrane k najbliższych sąsiadów. Zbiór, którego najwięcej przedstawicieli zostało wybranych wygrywa. Dla przypadku dychotomii jeżeli k jest nieparzyste, to możliwa jest jednoznaczna klasyfikacja. Gdy ilość zbiorów, do których można zaklasyfikować obraz jest większa od dwóch, możliwe jest wystąpienie wypadku, w którym nie można jednoznacznie zaklasyfikować obiekt. W przypadku systemu kontroli dostępu może być wskazane wprowadzenie maksymalnej odległości między średnią lub najbliższym sąsiadem, a klasyfikowanym wektorem, przy której może nastąpić klasyfikacja. Powyżej tej odległości obraz nie powinien być zaklasyfikowany do którejkolwiek z grup. Dodatkowo warto zwrócić uwagę na brak kontroli PCA nad przetwarzanymi danymi. Zdjęcie przetwarzane przez metodę statystyczną może przedstawiać mysz lub samochód i zostanie podany wynik. Nie wiadomo jakie współrzędne można uzyskać, dzięki takiej operacji ani jak zostaną zinterpretowane wyniki przez system. Algorytm rozpoznawania twarzy powinien odrzucić takie zdjęcie na etapie wykrywania obecności twarzy.

3.2 Inne podejścia do problemu redukcji wymiarowości

Istnieje wiele algorytmów umożliwiających redukcję wymiarowości. Ze względu na rosnącą ilość danych do przetwarzania, algorytmy te zyskały na znaczeniu. W niniejszym fragmencie pracy zostaną krótko omówione zasady działania trzech algorytmów umożliwiających redukcję wymiarowości.

3.2.1 Nonnegative Matrix Factorization

Nonnegative Matrix Factorization (NMF) jest sposobem rozkładu macierzy na dwie mniejsze. Przyjmimy, że dana jest macierz $V_{m\times n}$, która ma zostać podana redukcji wymiarowości. Macierz ta zostaje wyrażona jako iloczyn dwóch mniejszych macierzy:

$$V \approx WH \tag{3.21}$$

Zakłada się rozmiary macierzy: $W_{m\times k}$ oraz $H_{k\times n}$, dla $k \leqslant min(m,n)$. Dodatkowo przyjmuje się że zarówno W jak i H są nieujemne. Łatwo można zauważyć że stosując taki zapis i przyjmując stosunkowo małą wartość k można zapisać V z wykorzystaniem mniejszej ilości zmiennych. Ponadto korzystając ze spostrzeżenia, że wektory kolumnowe V mogą zostać zapisane jako:

$$\mathbf{v}_i = W\mathbf{h}_i, \tag{3.22}$$

można przyjąć W, jako bazę przestrzeni definiowanej przez V.

Dokładny rozkład we wzorze 3.21 nie musi istnieć dla dowolnego k, dlatego nie może zostać umieszczony znak równości. Zwykle w trakcie szukania W i H przyjmuje się, że ich iloczyn ma się znajdować w odpowiednio niewielkiej odległości od V. Najczęściej rozważane wartości błędu to:

$$||A - B||^2 = \sum_{ij}(A_{ij} - B_{ij})^2$$

$$D(A||B) = \sum_{ij}(A_{ij} log \frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij})$$

Pierwsza z prezentowanych norm to znana norma L2, czyli odległość Euklidesowa. Drugie równanie to zmodyfikowana wersja dywergencji Kullbacka-Leiblera dla nieujemnych macierzy. Jest to wielkość informująca nas o tym jak wiele informacji tracimy korzystając z estymacji B rozkładu A. Wielkość ta nie jest normą, ponieważ nie jest symetryczna.

W pracy [9] zaproponowano iteracyjne metody znajdowania W i H dla obu miar błędu. Polegały one na stopniowym poprawianiu wartości obu macierzy. Pierwszy rodzaj polegał na przemnażaniu aktualnej wartości W i H przez odpowiednie współczynniki. Drugi rodzaj opierał się na dodawaniu odpowiedniej poprawki (rozwiązanie przypominające szukanie optimum funkcji z wykorzystaniem gradientu). W obu przypadkach początkowe wartości były losowe co powodowało, że rezultaty uzyskiwane przez algorytm mogły być różne dla tej samej macierzy V. Dodatkowym ryzykiem jest utknięcie w minimum lokalnym.

W pracy [10] zaproponowano metodę wyznaczania W i H, która nie inicjalizuje losowo wartości początkowych, tylko korzysta z rozkładu SVD (singular-value decomposition - rozkład według wartości osobliwych), który dla macierzy A o wartościach rzeczywistych zapisujemy jako:

$$A = U\Sigma V^T \tag{3.23}$$

U oraz V są ortogonalne, a Σ jest macierzą diagonalną, na przekątnej, której leżą tzw. wartości osobliwe. Wektory kolumnowe U są nazywane lewymi wektorami osobliwymi, wektory V są nazywane prawymi wektorami osobliwymi. Autorzy [10] wykorzystali lewe i prawe wektory osobliwe, odpowiadające największym wartościom osobliwym Σ do inicjalizacji W i H. Umożliwiło to poprawienie wyników oraz uzyskanie algorytmu dającego te same rezultaty dla takiego samego wejścia. Dodatkowo czas działania programu, korzystającego z tej metody stał się bardziej deterministyczny.

3.2.2 Independent Components Analysis

Rozważmy następujący problem: dana jest obserwacja x będąca liniową kombinacją wielu sygnałów s o niegaussowskim rozkładzie:

$$x_j = a_{j1}s_1 + a_{j2}s_2 + ... + a_{jn}s_m \tag{3.24}$$

Powższe równanie opisuje wartość j-tej obserwacji. Zarówno x i s są n-wymiarowymi wektorami. Zapis może zostać uogólniony przez zapis macierzowy:

$$\mathbf{x} = A\mathbf{s} \tag{3.25}$$

$A_{n\times n}$ jest nazywane macierzą mieszania (ang. mixing matrix), $\mathbf{s}$ jest nazywane nieznaleźną składową. Celem jest odzyskanie niezależnych składowych $\mathbf{s}$ na podstawie $\mathbf{x}$. Zauważmy, że: $\mathbf{s} = A^{-1}\mathbf{x} = W\mathbf{x}$. W jest nazywana macierzą rozdzielania (ang. unmixing matrix).

Można sobie wyobrazić, że problem ten występuje w kontekście analizy sygnału mowy: dysponujemy sygnałem, w którym wiele osób mówi jednocześnie, a chcemy uzyskać osobne wypowiedzi poszczególnych osób (problem cocktail party).

W [11] podano rozwiązanie problemu opierające się na centralnym twierdzeniu granicznym:

Załóżmy że wartość oczekiwana $\mathbf{x}$ i $\mathbf{s}$ jest równa 0 oraz wariancja niezależnych składowych jest równa 1. Możemy to zrobić, ponieważ dla rozkładów o niezerowej średniej można odjąć średnią. Ograniczenie nałożone na wariację $\mathbf{s}$ jest spowodowane przez równanie 3.25, konkretniej, wartość $\mathbf{s}$ może zostać obliczona z dokładnością do stałej ($\mathbf{s}$ może zostać przemnożone przez dowolne p, jeśli A zostanie przemnożone przez $\frac{1}{p}$). Stosując to ograniczenie unikamy zbyt wielkich wartości niezależnych składowych Dodatkowo załóżmy, że wszystkie niezależne składowe $\mathbf{s}$ mają taki sam rozkład i są niezależnymi zmiennymi losowymi. Wprowadźmy $\mathbf{y}$ jako liniową kombinację $\mathbf{x}$:

$$\mathbf{y} = \mathbf{w}^T\mathbf{x} \tag{3.26}$$

Jeśli $\mathbf{w}$ jest fragmentem macierzy W, to $\mathbf{y}$ musi odpowiadać jednej z niezależnych składowych. Nie możemy bezpośrednio wyznaczyć $\mathbf{y}$, ponieważ nie znamy A i $\mathbf{s}$, możemy co najwyżej je estymować. Przyjmijmy:

$$\mathbf{z} = A^T\mathbf{w} \tag{3.27}$$

Wykorzystując nową zmienną możemy zapisać:

$$\mathbf{y} = \mathbf{w}^T\mathbf{x} = \mathbf{w}^T A\mathbf{s} = \mathbf{z}^T\mathbf{s} \tag{3.28}$$

Możemy zauważyć, że y jest kombinacją liniową $\mathbf{s}$. Zgodnie z centralnym twierdzeniem granicznym, suma dowolnych rozkładów prawdopodobieństwa dąży do rozkładu Gaussa, jeśli jest spełniony warunek Lindeberga, a w szczególności gdy $E[X] = 0$ i $\sum_{k=1}^{n} var[X] = 1$ to:

$$\sum_{k=1}^{n} X_k \to N(0,1) \tag{3.29}$$

Widzimy więc, że $\mathbf{y}$ będące kombinacją liniową wektorów $\mathbf{s}$, będzie bardziej "gaussowskie", niż dowolna z niezależnych składowych osobno. Założyliśmy, że s nie ma rozkładu Gaussa, więc $\mathbf{y}$ będzie równe $\mathbf{s}$, gdy będzie najmniej gaussowskie, co jest jednoznaczne z tylko jedną niezerową wartością w wektorze $\mathbf{z}$.

Autorzy [11] proponują miary niegaussowskości oparte na kurtozie, bądź ujemnej entropii. Negatywna entropia może być dość dobrym estymatorem stopnia w jakim rozkład nie jest gaussowski, jest nieujemna i równa zero tylko dla rozkładu Gaussa.

Oprócz zastosowania w analizie mowy, często podawaną aplikacją ICA jest ocenianie aktywności mózgu w poszczególnych obszarach na podstawie sygnałów z elektrod rozmieszczonych w różnych miejscach głowy. W kontekście tej pracy, znalezienie wektorów s, mogących definiować bazę nowej przestrzeni prowadzi do redukcji wymiarowowści danych.

Rozdział 4

Omówienie implementacji

W tym rozdziale przedstawiono szczegóły implementacji kodu realizującego omawiane funkcjonalności komponentu. Omówiono podstawowe koncepcje i założenia kryjące się za przyjętymi rozwiązaniami. Omówiono funkcjonalności oraz zakres odpowiedzialności poszczególnych klas. Warto nadmienić że wszystkie przedstawione tutaj rozwiązania dotyczą najnowszej wersji komponentu, która nie została jeszcze zintegrowana z resztą systemu.

4.1 Klasa zarządzająca komponentem

W najnowszej wersji projektu najważniejszą klasą w omawianym komponencie jest AbstractFactory. Wzorzec projektowy Fabryki Abstrakcyjnej dobrze oddaje działanie komponentu z punktu widzenia reszty systemu, choć biorąc pod uwagę zakres odpowiedzialności klasy odpowiedni jest także wzorzec Budowniczy. Jest to klasa zawierająca wszystkie niezbędne algorytmy, które są wykorzystywane przy procesach wykrywania twarzy i ekstrakcji cech. Zadaniem tej klasy jest przykrycie wszystkich zależności i szczegółów związanych z operacją przekształcania zdjęcia w wektor liczb. Efektem tej operacji jest zwrócenie obiektu Product, omówionego w późniejszej części tego rozdziału.

4.1.1 Interfejs do komunikacji z resztą systemu

W obiektowych językach programowania często można się spotkać z podziałem metod na publiczne i prywatne. Podział ten wymusza dobre praktyki na programistach, ale ma jeden bardzo praktyczny aspekt. Wszystkie metody publiczne klasy to interfejs jaki zostaje udostępniony użytkownikowi. Jest to dość naturalny podział - do interfejsu klasy zaliczają się wszystkie metody z jakich może skorzystać użytkownik. W języku Python nie przewidziano podziału na metody prywatne i publiczne. Wszystkie metody klasy są publiczne z założenia. Powoduje to brak jasno określonego interfejsu klasy. W najnowszej wersji komponentu klasa AbstractFactory jako interfejs udostępnia metodę:

```
def fabricate(self, img_filename):
    prod = self.prouctType(self.n_dim)
    prod.set(self.compute_coor(self.detect_face(img_filename)))
    return prod
```

Jest to metoda pozwalająca na przeprowadzanie całego procesu detekcji twarzy i ekstrakcji cech. Jako arguemnt przyjmuje nazwę pliku ze zdjęciem, a zwraca produkt.

4.1.2 Konstruktor

Dobór algorytmów wykorzystywanych w tej klasie, oraz typ zwracanego produktu jest wybierany przez użytkownika przez wstrzykiwanie zależności w argumentach konstruktora.

Konstruktor pełni ważną rolę w klasie, ponieważ wykorzystano w nim technikę wstrzykiwania zależności.

```
def __init__(self, FeatureExtractionStrategy, \
FaceDetectionStrategy, CoordinatesProduct):
    self.n_dim = 4096
    self.n_components = 400
    self.face_detection_strategy = FaceDetectionStrategy()
    self.extraction_strategy = FeatureExtractionStrategy(self.n_dim, \
    self.n_components)
    self.prouctType = CoordinatesProduct
```

Użytkownik klasy przy tworzeniu obiektu typu AbstractFactory podaje w argumentach konstruktora jakie warianty klas do detekcji twarzy, ekstrakcji cech oraz przechowywania informacji na temat produktu powinny zostać wykorzystywane. Dodatkowo w konstruktorze są inicjalizowane stałymi rozmiary wektora przed i po redukcją wymiarowości. Wartości te wynoszą odpowiedni 4096 i 400. Pierwsza z tych wartości odpowiada rozmiarowi obrazka 64×64 piksele i została przyjęta w celu dopasowania rozmiarów przetwarzany wektorów do tych ze zbioru Olivetti faces dataset. Druga wartość jest efektem eksperymentów z siecią neuronową – dla 400-elementowych wektorów uzyskiwano najlepsze rezultaty. Wartości te nie powinny być przyjęte jako stałe i użytkownik powinien mieć możliwość zmiany ich wartości, nie jest to jednakże rażące niedopatrzenie.

4.1.3 Pozostałe funkcje

```
def detect_face(self, img_filename):
    return self.face_detection_strategy.detect_face_proxy(img_filename)

def compute_coor(self, img):
    return self.extraction_strategy.extract_features(img)
```

Pozostałe dwie funkcje klasy służą do wywoływania operacji detekcji twarzy i ekstracji cech z odpowiednich strategii.

4.2 Klasa przechowująca wyniki działania Fabryki

Zadaniem klasy CoordinatesProduct jest nałożenie abstrakcji na sposób przechowania wektora liczb zwracanego przez AbstractFactory. Posiada ona dwie metody: getter i setter. Po klasie tej dziedziczą: CoordinatesProductOnDisc i CoordinatesProductInClass. Każda z nich przechowuje wektor liczb w inny sposób: pierwsza na dysku, druga bezpośrednio w

pamięci programu, jako wektor z biblioteki NumPy. Umożliwienie przechowywania wartości na dysku pozwala na łatwy sposób debugowania aplikacji i analizę uzyskanych danych. O wiele prostszym rozwiązaniem mogłoby być wykorzystanie struktury danych podawanej sieci neuronowej, ale jednym z celów takiego rozwiązania jest uniezależnienie wyników zwracanych przez AbstractFactory od frameworku używanego do uczenia sieci neurowej.

4.3 Strategia detekcji twarzy

FaceDetectionStrategy jest klasa abstrakcyjną po której mają dziedziczyć wszystkie implementacje algorytmów detekcji twarzy. Zawiera metody ogólnego przeznacznia, które mogą być wykorzystywane przez klasy potomne. Do metod tych zaliczają się:

```
read_image_and_convert_to_greyscale(self, filename),
crop_img(self, image, x, y, w, h),
debug_show(self, img)
```

Klasa ta posiada metodę łączącą wszystkie operacje wykonywane w ramach działania algorytmu wykrywania twarzy:

```
def detect_face_proxy(self, input_img_filename, \
output_img_filename=None):
    grayscale_image = \
    self.read_image_and_convert_to_greyscale(input_img_filename)
    (x, y, w, h) = self.detect_face(grayscale_image)
    resized_face = self.crop_img(grayscale_image, x, y, w, h)
    if output_img_filename != None:
            self.debug_show(resized_face)
        cv2.imwrite(output_img_filename, resized_face)
    return resized_face
```

Jak łatwo można zauważyć podanie do metody dodatkowego parametru w postaci ścieżki i nazywy pliku do zapisania, umożliwia łatwe debugowanie aplikacji. Metoda ta jest wywoływana w detect_face w AbstractFactory.

Klasa HaarCascadesStrategy dziedziczy po FaceDetectionStrategy i implementuje detekcje twarzy z wykorzystaniem implementacji z biblioteki OpenCv. Kiedy na zdjęciu nie zostanie znaleziona twarz zostanie zgłoszony wyjątek: Found 0 faces!

4.4 Strategia ekstrakcji cech

Zasada działania FeatureExtractionStrategy jest bardzo podobna do poprzedniej klasy. Stanowi ona klasę bezową dla wszystkich implementacji redukcji cech. W jej skład wchodzą metody pomocnicze, takie jak:

```
load_img(self, name)
convert_image_to_vector(self, img)
load_matrix(self, name)
```

Klasa ta udostępnia jako interfejs metodę:

```
extract_features(self, img)
```

Klasa PCA dziedziczy po FeatureExtractionStrategy. PCA przeprowadza redukcję wymiarowości z wykorzystaniem pliku eig.npy. Plik ten zawiera wszystkie wektory własne potrzebne do redukcji wymiarowości. Dodatkowo PCA wczytuje średni wektor ze zbioru i konwertuje go do wektora. Operacja ekstrakcji cech przebiega w następujący sposób:

```
def scale_vector(self, vec):
    scaleVec = ScaleVector()
    return scaleVec.scale_vector(vec)

def extract_features(self, img):
    vec = self.image_to_vector(img)
    scaled_vec = self.scale_vector(vec)
    mean_substracted_vec = scaled_vec - self.mean
    coordinates = np.zeros((self.n_components))
    for i in range(0,self.n_components):
        coordinates[i] = np.dot(self.components[i,:],\
        mean_substracted_vec)
    return coordinates
```

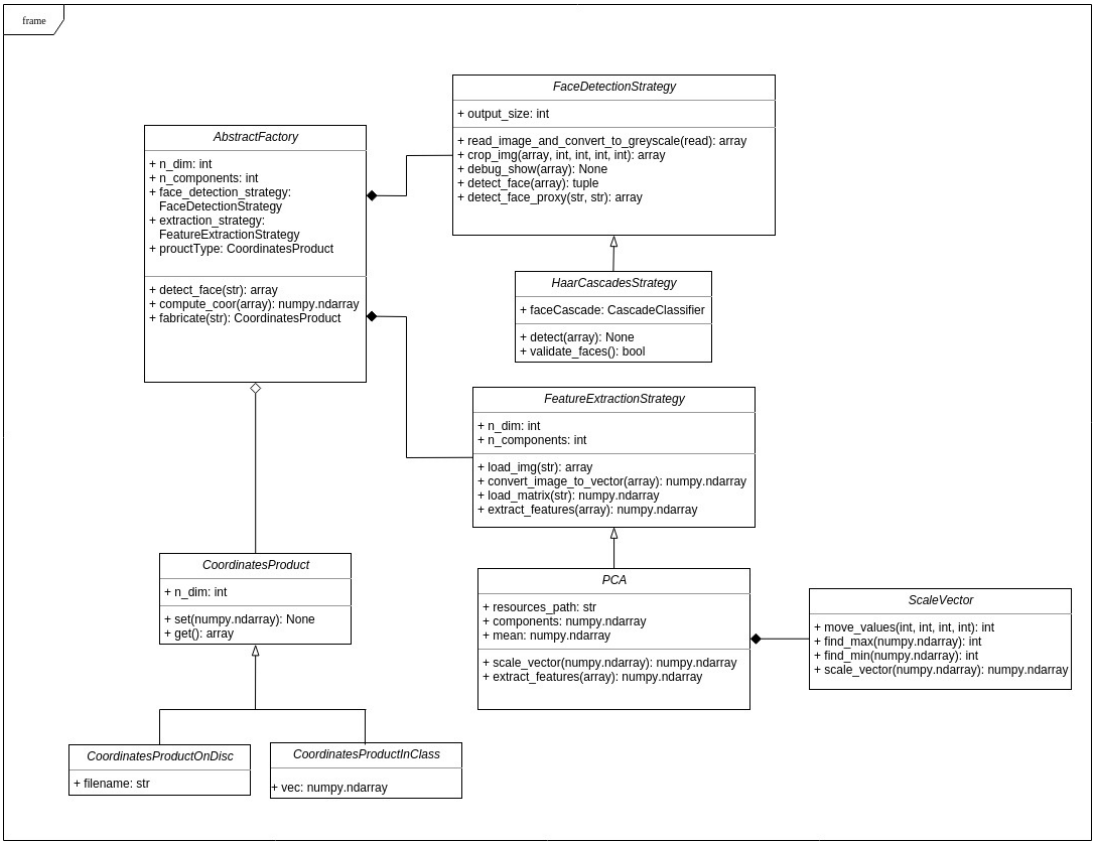
W klasie PCA wykorzystano klasę ScaleVector odpowiedzialną za skalowanie wektora, tak aby przyjmował wartości od zera do 255:

```
def move_values(self, x, xmin, xmax, max):
    return max*(x-xmin)/(xmax-xmin)

def scale_vector(self, vec):
    min_ = self.find_min(vec)
    max_ = self.find_max(vec)
    for i in range(len(vec)):
        vec[i] = self.move_values(vec[i], min_, max_,255.0)
    return vec
```

4.5 Pozostała część oprogramowania

Przedstawione wcześniej klasy stanowią całośc komponentu działającego w systemie. Nie jest to jednakże cały kod, jaki został sporządzony w trakcie realizacji projektu. Wyznacznie wektorów własnych, oraz skrypty umożliwiające rekonstrukcję zdjęć na podstawie uzyskanych współrzędnych odgrywały bardzo ważną rolę przy powstawaniu systemu. Nie zostały one włączone bezpośrednio do kodu komponentu, ponieważ są one tam zbędne. Po obliczeniu wektrów własnych dla PCA są one zapisywane w pliku z rozszerzeniem .npy, z którego następnie korzysta klasa PCA w trakcie działania systemu. Obecny wygląd komponentu jest efektem wielu prób i błędów, próbowania możliwości jakie oferują wykorzystane biblioteki oraz wieloma poprawkami na przestrzeni życia całego projektu.



Rysunek 4.1: Diagram klas wkorzystywanych w komponencie.

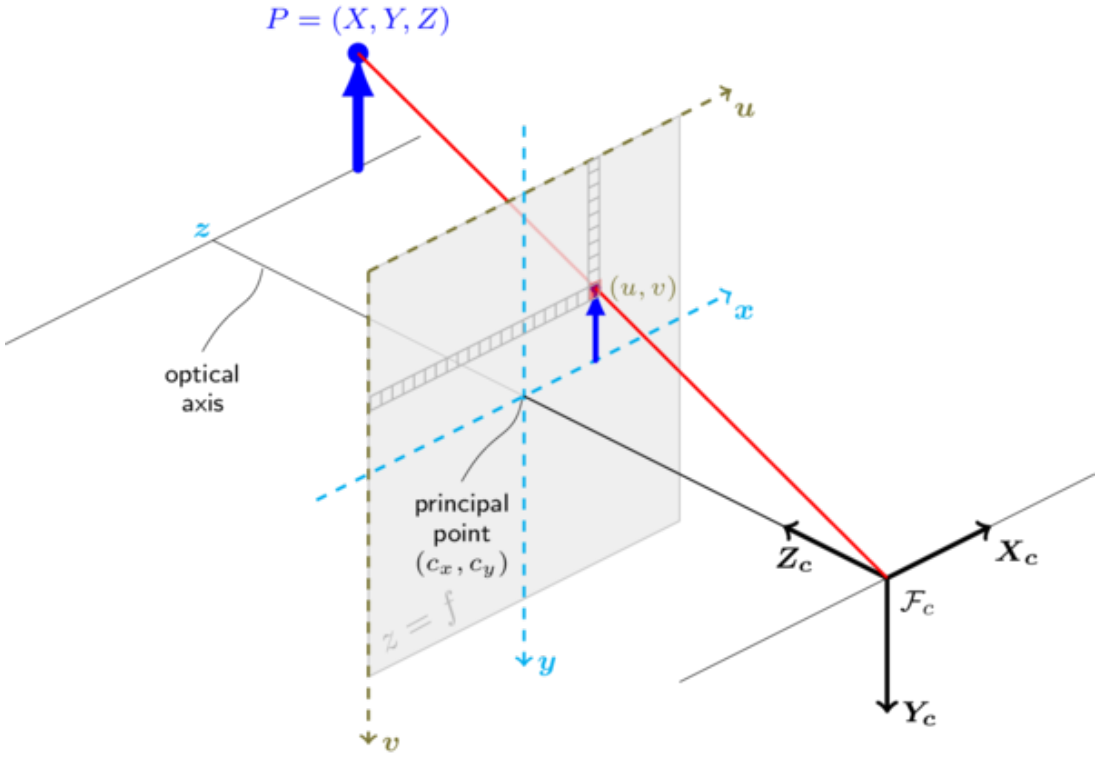
Rozdział 5

Zabezpieczenie przed możliwością oszukania systemu

W trakcie realizacji projektu specjalnościowego, jako zadanie dodatkowe zbadano jakie mozliwości może dać dołączenie do systemu ekstrakcji głębi. Był to dodatek mający wskazać możliwe strony rozwoju całego projektu przy jednoczesnej próbie zaadresowania kwestii bezpieczeństwa. W trakcie testów systemu odkryto, że istnieje możliwość oszukania całego systemu poprzez przedstawianie systemowi zdjęć ludzi, którym wcześniej przyznano prawo dostępu. Jest to realne zagrożenie, z którym twórcy systemów biometrycznych muszą się mierzyć. Dobrym przykładem może być tu system FaceID w IphoneX firmy Apple. Twórcy systemu zapewniali, że nie da się go oszukać poprzez wykonanie masek 3d, imitujących twarz człowieka. 9 listopada 2017 roku eksperci z firmy Bkav oświadczyli, że udało im się złamać zabezpieczenia poprzez wykonanie specjalnej maski, wykorzystując drukarkę 3D, silikonowe elementy, wydruki 2D oraz materiał imitujący skórę [18]. Dodatkowo system FaceID okazał się nie być odpornym na podobieństwo występujące między bliźniakami jednojajowymi. Bezpieczeństwo systemów biometrycznych jest więc bardzo ważną kwestią i podjęto próbę zbadania zagadnień z nim związanych.

5.1 Przyjęta metodyka

Zbadano możliwości jakie daje ekstrakcja głębi z wykorzystaniem dwóch kamer. W obecnym projekcie systemu zintegrowanie systemu ekstrakcji głębi byłoby trudne, ponieważ w projekcie wykorzystywana jest kamera Rasbery Pi, podłączona przez port CSI, a mikrokomputery Rasbery Pi posiadają tylko jeden port CSI. Do badań wykorzystano dwie kamerki internetowe Creative VF0770. Do ekstrakcji głębi wykorzystano funkcje biblioteki OpenCV. Celem było uzyskanie mapy punktów w 3D, które następnie miały być aproksymowane płaszczyzną. Następnie należało wyznaczyć błąd aproksymacji, oraz próg, od którego mapa punktów miała być uznawana za płaską.



Rysunek 5.1: Scena przy przyjętym modelu kamery.

Źródło:

https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

5.2 Podstawy teoretyczne

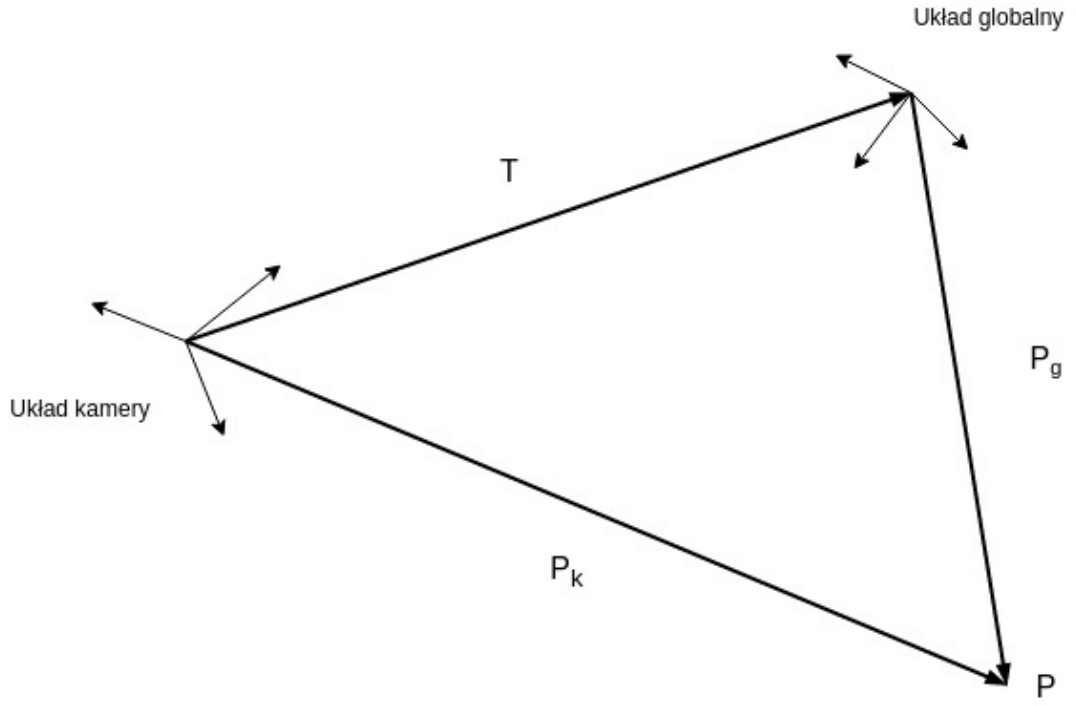
Przed przedstawieniem wyników omówiono przyjęte założenia teoretyczne. W nimniejszej sekcji zostały omówione takie koncepcje jak pinhole camera model, model zniekształceń w bibliotece OpenCv, podstawowe założenia geometrii epipolarnej oraz wyznaczanie mapy głębokości na podstawie dysparycji.

Model kamery

Przed przystąpieniem do ekstrakcji głębi należy ustalić model działania pojedynczej kamery oraz zrozumieć jakie zakłócenia obrazu występują.

Najbardziej rozpowszechnionym modelem jest tzw. pinhole camera model. W modelu tym kamera przekształca piksele z globalnego układu współrzędnych do układu współrzednych na płaszczyźnie, na którą są rzutowane punkty. Przekształcenie to można wyrazić wzorem:

$$Z \begin{pmatrix} u \\ v \end{pmatrix} = K(R \quad T) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{5.1}$$



Rysunek 5.2: Ilustracja zależności między układami współrzędnych a punktem.

W powyższym równaniu X, Y oraz Z, to współrzędne punktu w zewnętrznym układzie współrzędnych, u i v to współrzędne w układzie kamery. Macierz R i wektor T, to macierz rotacji i wektor translacji opisujące orientację i położenie kamery względem zewnętrznego układu współrzędnych. Zaliczają się one do parametrów zewnętrznych, ponieważ nie zależą od samej kamery, ale od jej położenia i orientacji. Parametry te opisują transformację punktu z globalnego układu współrzędnych do układu współrzędnych kamery.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = (R \quad T) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

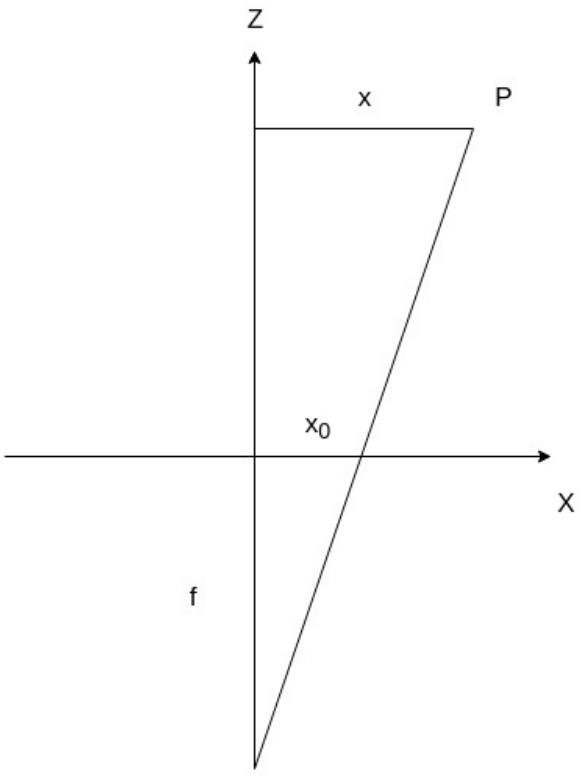
Gdzie x, y, z, to współrzędne punktu P w układzie kamery.

Warto zwrócić uwagę, że wektor T zawiera położenie globalnego układu współrzędnych względem układu współrzędnych kamery, a nie odwrotnie. Fakt ten może wydawać się nieco nieintuicyjny, ale można go łatwo zilustrować. Z 5.2 wynika jednoznacznie, że $P_k = T + P_g$. Oczywiście należy uwzględnić wzajmeną orientację układów, co zapewnia macierz R.

Macierz K zawiera parametry wewnętrzne kamery i jest definiowana jako:

$$K = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

gdzie f, to długość ogniskowej, c_x, c_y jest punktem, w którym oś optyczna przecina się z płaszczyzną obrazu (ang. principal point), zwykle znajdującym się w środku obrazu.



Rysunek 5.3: Rzut z góry na scenę.

Korzystając z wcześniejszych równań można zapisać:

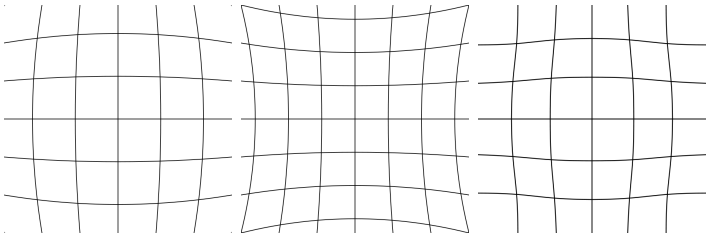
$$u = f\frac{x}{z} + c_x$$
$$v = f\frac{y}{z} + c_y$$

Na rys 5.3 można zauważyć, że równania te mają proste uzasadnienie geometryczne. Z twierdzenia Talesa otrzymujemy: $\frac{x_0}{f} = \frac{x}{z}$. Wartość u możemy uzyskać poprzez dodanie do x_0 z poprzedniego równania c_x. Dla y rozumowanie jest analogiczne.

Zakłócenia

Zaprezentowany wcześniej model zakłada brak zakłóceń przy procesie przetwarzania obrazu. W rzeczywistych kamerach występują różne rodzaje zniekształceń. W dystorsji linie proste z oryginalnej przestrzeni nie są proste na obrazie. Wynika to z fizycznej niedoskonałości układów optycznych. Im dalej od osi optycznej tym większe zniekształcenie.

W bibliotece OpenCV przewiduje się współczynniki radialne $k_1, ..., k_6$ oraz styczne p_1 i p_2: (podawane za dokumentacją OpenCV [14])



Rysunek 5.4: Różne rodzaje dystorcji (od lewej): beczkowa, poduszkowa oraz falista.

źródło: https://en.wikipedia.org/wiki/Distortion_%28optics%29

$$
\begin{aligned}
x' &= \frac{x}{z} \\
y' &= \frac{y}{z} \\
x'' &= x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\
y'' &= y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y'
\end{aligned}
$$

Współczynniki te następnie są wykorzystywane we wcześniejszym modelu w następujący sposób:

$$
\begin{aligned}
u &= f_x x'' + c_x \\
v &= f_y y'' + c_y
\end{aligned}
$$

Widać, że uzwględnienie współczynników dystorcji znacznie komplikuje przyjęty model, jednakże jest koniecznym krokiem przed przystąpieniem do dalszych operacji.

5.2.1 Kalibracja kamery

Kalibracja kamery odbywa się poprzez fotografowanie czarno-białej szachownicy pod różnymi kątami. Następnie zdjęcia są konwertowane do skali szarości i wykrywane są narożniki szachownicy. Pierwsze oszacowanie położenia narożników nie jest dokładne. Estymacja jest poprawiana z wykorzystaniem prostego faktu dotyczącego gradientu:

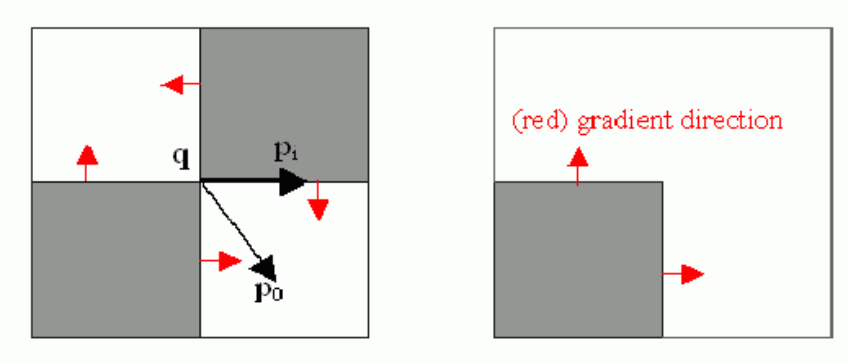
Załóżmy, że $\mathbf{q}$ jest narożnikiem szachownicy, a $\mathbf{p}_i$ jest punktem w bliskim sąsiedztwie. Przyjmijmy, że wartość błędu dana jest następującym równaniem:

$$E_i = \nabla f_{\mathbf{p}_i}^T (\mathbf{q} - \mathbf{p}_i) \tag{5.3}$$

gdzie $\nabla f_{\mathbf{p}_i}$ jest gradientem, w punkcie p_i. Jest to równanie na iloczyn skalarny i będzie ono równe zero, gdy $\mathbf{p}_i$ będzie prostopadłe do gradientu. Przyrównując powyższe równanie do zera oraz mnożąc obie strony przez $\nabla f_{\mathbf{p}_i}$ otrzymujemy:

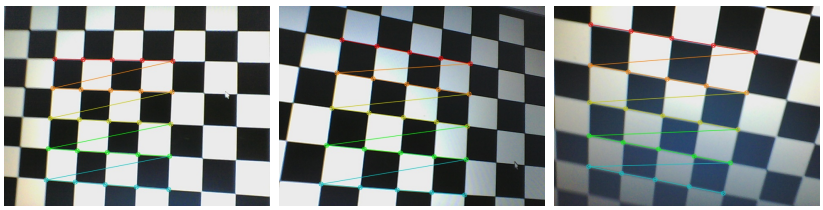
$$\nabla f_{\mathbf{p}_i} \nabla f_{\mathbf{p}_i}^T \mathbf{q} - \nabla f_{\mathbf{p}_i} \nabla f_{\mathbf{p}_i}^T \mathbf{p}_i = 0$$

W konsekwencji otrzymujemy:



Rysunek 5.5: Gradient szachownicy.

źródło: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html



Rysunek 5.6: Wykryte narożniki szachownicy.

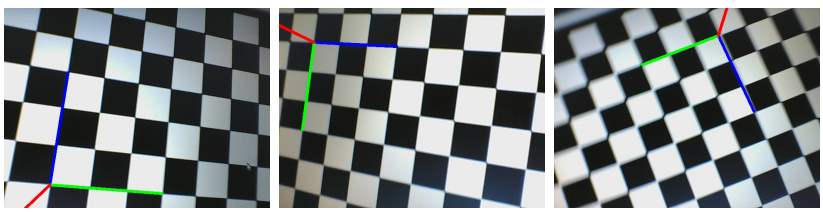
$$\mathbf{q} = (\nabla f_{\mathbf{p}_i} \nabla f_{\mathbf{p}_i}^T)^{-1} \nabla f_{\mathbf{p}_i} \nabla f_{\mathbf{p}_i}^T \mathbf{p}_i \tag{5.4}$$

Operacja ta jest powtarzana, aż punkt q uzyska stabilną wartość (jego współrzędne nie będą się zmieniać o większą wartość niż przyjęty próg).

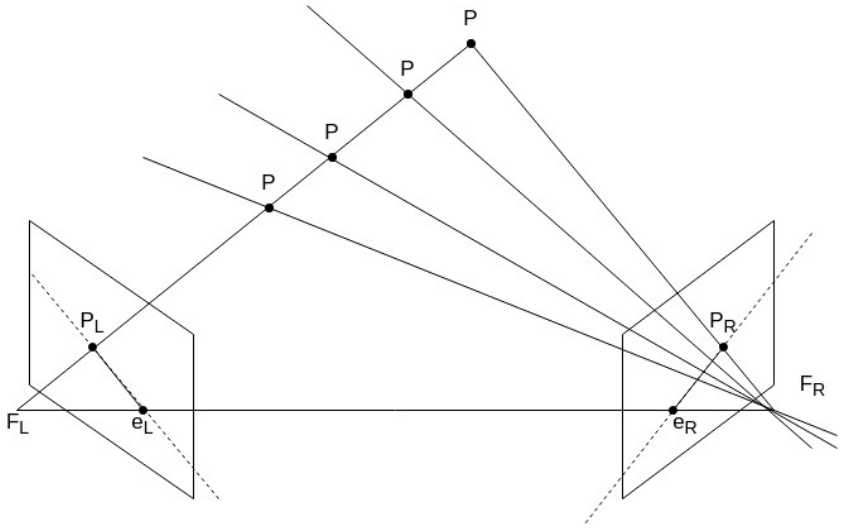
Dzięki wyznaczonym narożnikom można wyznaczyć wewnętrzne i zewnętrzne kamery. Wartości te są następnie wykorzystane do estymacji pozycji 5.7 oraz do usunięcia zakłóceń z obrazu.

5.2.2 Geometria epipolarna

Zakładając wcześniej przyjęty model rzeczywistej kamery oraz stosując uproszczenie polegające na przeniesieniu płaszczyzny obrazu przed ognisko (w rzeczywistych kamerach płaszczyzna ta znajduje się za ogniskiem) możemy zauważyć pewne zależności geometryczne dla punktu widzianego z dwóch kamer.



Rysunek 5.7: Estymacja orientacji szachownicy.



Rysunek 5.8: Podstawowa koncepcja geometrii epipolarnej.

Analizując scenę przedstawioną na rys. 5.8 można zauważyć, że ogniska obu kamer, punkt P oraz jego projekcje na płaszczyznę obrazu leżą na jednej płaszczyźnie. Linia poprowadzona pomiędzy ogniskami kamer jest nazywana linią główną (ang. baseline). Punkty przecięcia linii głównej z płaszczyzną główną (e) nazywane są epipolami (ang. epipoles). Jeśli obie kamery się widzą, to epipole są widoczne jako środek drugiej kamery na obrazie. Płaszczyzna przechodząca przez dowolny punkt P widziany w obu kamerach oraz przez linię główną jest nazywana płaszczyzną epipolarną. Linią epipolarną nazywamy linię, będącą na przecięciu płaszczyzny obrazu z płaszczyzną epipolarną. Wszystkie linie epipolarne przecinają się w epipolach.

Warto zwrócić uwagę na to, że linia FP jest widziana na drugim obrazie jako odpowiadająca linia epipolarna. Oznacza to więc, że szukając odpowiednika piksela z jednego obrazu na drugim, należy szukać wzdłuż epilinii, a nie na całym obrazie. Jest to znaczące ułatwienie, które będzie miało duże znaczenie przy obliczaniu dysparycji.

5.2.3 Szacowanie głębokości

Kolejnym krokiem jest obliczenie mapy dysparycji oraz na jej podstawie wyznaczenie głębokości.

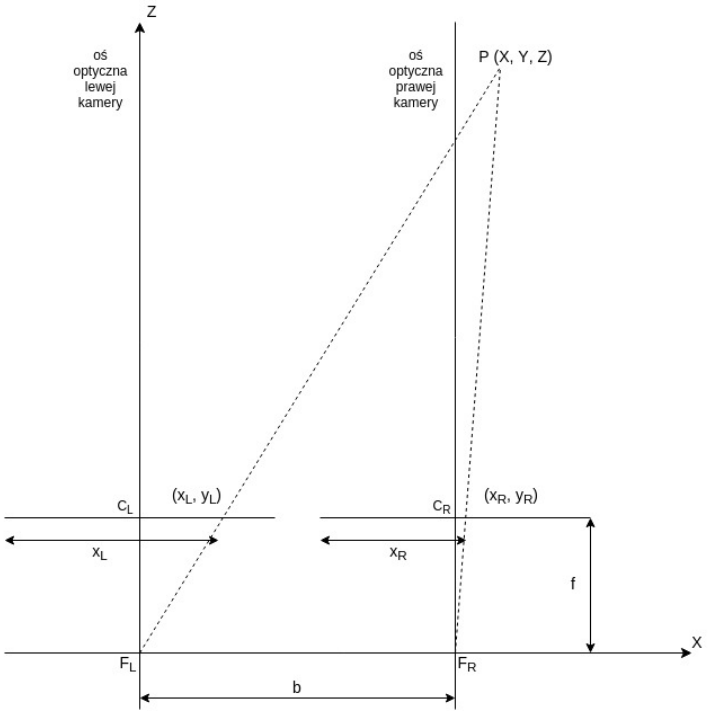
Przy ekstrakcji głęgokości jest wykorzystywane podobieństwo trójkątów PF_LF_R oraz PC_LC_R z rys. 5.9:

$$\frac{b}{Z} = \frac{b - x_L + x_R}{Z - f} \tag{5.5}$$

Po przekształceniach otrzymujemy:

$$Z = \frac{bf}{x_L - x_R} = \frac{bf}{d} \tag{5.6}$$

gdzie d jest dysparycją - różnicą wartości odpowiadających sobie pikseli na dwóch obrazach szukanych wzdłuż epilinii. Znając ogniskową, odległość między kamerami oraz



Rysunek 5.9: Ekstrakcja głębi na podstawie dysparycji obrazów z dwóch kamer.

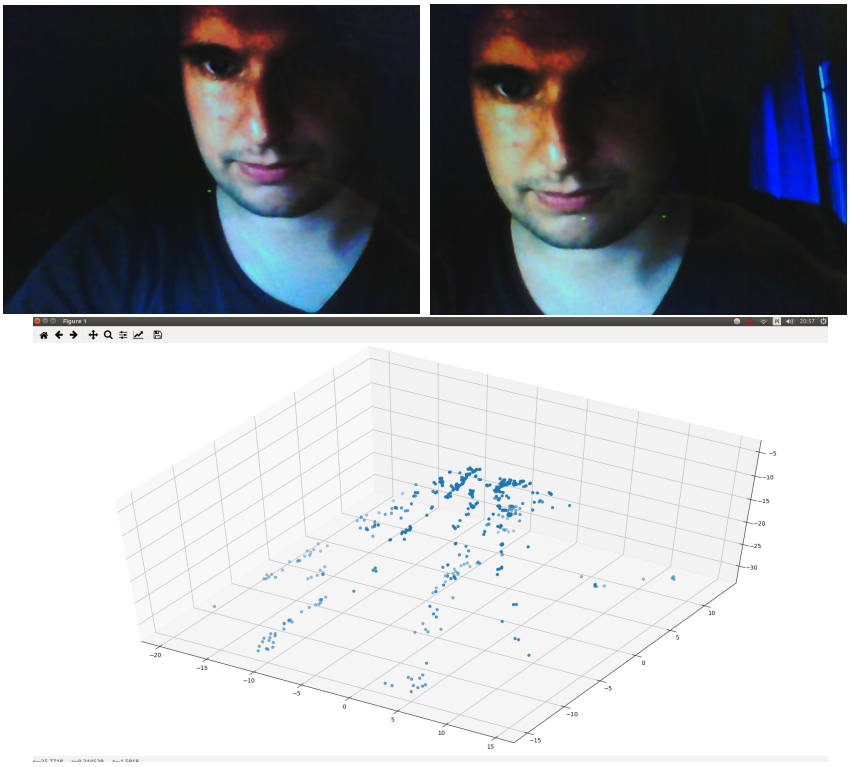
obliczając różnicę między analogicznymi pikselami z dwóch obrazów możemy szacować głębię.

Warto zwrócić uwagę, że wzór 5.5 nie traci na ogólności, gdy punkt P znajdzie się pomiędzy osiami optycznymi kamer, ponieważ do obliczenia podstawy trójkąta PC_LC_R zostanie wykorzystana odległość między punktem, w którym wiązka światła pada na prawą matrycę, a osią optyczną prawej kamery. Odległość ta może zostać w takim wypadku wyrażona jako $\frac{1}{2}l - x_R$, gdzie l to długość ekranu światłoczułego. Po podstawieniu podobnej wartości za odległość od osi optycznej lewej kamery do punktu przecięcia wiązki światła z lewą matrycą $\frac{1}{2}l$ uprości się i zostaniemy ze wzorem 5.5.

5.3 Wyniki

Dokonano kalibracji kamery, wyznaczono współczynniki wewnętrzne i zewnętrzne kamery. Jednakże nie udało się uzyskać zadowalających rezultatów. Po przedstawieniu systemowi zdjęcia osoby wyświetlonego na laptopie uzyskano mapę głębi przedstawioną na rys. 5.10. Jak można z łatwością zauważyć na mapie głębokości otrzymano punkty o różnej wartości współrzędnej z, czyli systemowi nie udało się poprawnie zidentyfikować płaskiej powierzchni. Gdyby uzyskana mapa głębi była z dopuszczalnym błędem aproskymowalna płaszczyzną, kolejnym krokiem byłoby zobserwowanie jak system zachowuje się dla normalnej twarzy. Na podstawie obserwacji oraz błędów aproksymacji płaszczyzną, planowano ustalić próg błędu aproksymacji. Po przekroczeniu takiego progu system mógłby uznać, że podejmowana jest próba oszustwa.

Trudno jest określić jedną przyczynę niepowodzenia. Na pewno ważną rolę odgrywa kalibracja kamer. Wielokrotnie podejmowano próbę poprawiania uzyskiwanych wyników poprzez ponowną kalibrację, ale nie wpłynęło to w znaczący sposób na rezultaty. W



Rysunek 5.10: Obraz płaskiego zdjęcia, sfotografowany lewą i prawą kamerą oraz wynik ekstrakcji głębokości.

związku z ograniczoną ilością czasu projekt nie mógł zostać ukończony, pomimo wielu godzin poświęconych na analizę problemu.

Rozdział 6

Zakończenie

6.1 Podsumowanie projektu

W trakcie pracy nad projektem udało się zrealizować następujące zadania:

- Ustalono protokoły komunikacji z innymi komponentami.
- Wykorzystano implementację alogrytmu kaskad Haara do wykrycia czy na zdjęciu jest twarz ludzka oraz do wyznaczenia współrzędnych twarzy.
- Przeanalizowano działanie algorytmu PCA, pozwalającego na redukcję wymiarowości i wykorzystano go do przygotowania wektora uczącego dla sieci neuronowej.
- Przetestowano poprawność działania całego systemu.
- Zaproponowano alternatywne metody rozwiązania postawionego problemu.
- Zbadano jakie możliwości w projekcie może oferować ekstrakcja głębi na podstawie obrazów z dwóch kamer.

Rozwiązanie zaproponowane w projekcie cieszy się dużą popularnością i daje wystarczająco dobre rezultaty. Praca z wykorzystaniem algorytmu PCA jest wygodna, ponieważ dane, na których operujemy mają geometryczną interpretację. Dodatkowym atutem w analizowanym zastosowaniu PCA jest możliwość wizualizacji rezultatów, co ułatwia weryfikację otrzymanych wyników. Ze względu na dużą dostępność bibliotek, w trakcie realizacji projektu przywiązano większą wagę do zrozumienia podstaw działania algorytmów. W pracy starano się unikać bezpośrednich odwołań do sposobu działania bibliotek, raczej skupiano się na fundamentach teoretycznych analizowanych algorytmów. Owocem takiego podejścia jest prostota zrealizowanego komponentu pod kątem programistycznym, przy jednoczesnym spełnieniu głównego celu postawionego przed rozpoczęciem realizacji projektu: zwiększenie wiedzy na temat stosowanych rozwiązań w dziedzinie Pattern Recognition.

W projekcie nie przeprowadzono testów modułowych komponentu, system był testowany jako całość. Powodem takiego stanu rzeczy jest trudność w testowaniu omawianego komponentu, ze względu na jego dość abstrakcyjny charakter, w porównanianu do reszty systemu. Testowanie rezultatów działania całego systemu jest czasochłonne. Dowolna

zmiana w systemie ekstrakcji cech powoduje konieczność rozpoczynania procesu uczenia sieci neuronowej od nowa. W dalszej części tego rozdziału przedstawiono pomysł na zaimplementowanie testów modułowych dla komponentu realizującego ekstrakcję cech.

Realizowany projekt umożliwił opracowanie i wykorzystanie wydajnych sposobów na pracę w grupie. Komponentowe podejście do systemu umożliwiło prowadzenie równoległych prac nad rozwojem całego systemu. Ustalenie abstrakcyjnych interfejsów między komponentami pozwoliłoby na przeprowadzanie daleko idących zmian w obrębie komponentu, bez konieczności wprowadzania zmian w innych częściach systemu.

6.2 Możliwe usprawnienia

System jako całość w obecnym stanie spełnia podstawowe założenia jakie zostały przed nim postawione, t.j. rozpoznawanie twarzy niewielkiej grupy osób oraz ich identyfikacja. Jednakże jest wiele aspektów, które można poprawić i rozwinąć. W niniejszej sekcji przeprowadzono dyskusję najważniejszych problemów, które należy zaadresować przy dalszej pracy nad projektem.

Testy modułowe

Jednym z pomysłów na rozwój systemu jest porównanie efektów działania algorytmów redukcji wymiarowości. Metodyka takich testów miałaby polegać na utworzeniu bazy zdjęć różnych osób i porównaniu odległości współrzędnych przyporządkowanych danym zdjęciom z bazy przez różne algorytmy. Możliwe do analizowania miary to: średnia odległość, największa i najmniejsza odległość, mediana oraz wariancja. Przy takim porównaniu należy oczywiście wybrać odpowiedni rodzaj normy. Takie podejście umożliwiłoby uniezależnienie wyników omawianego komponentu od sieci neuronowej, co znacznie ułatwiłoby badania właściwości proponowanych rozwiązań. Warto zwrócić uwagę, że takie rozwiązanie pozwala na testowanie dowolnego podejścia do problemu. Ustalenie wspólnego zbioru zdjęć dla porównywanych algorytmów, normy dla przestrzeni cech oraz miar i progów dla każdej z miar umożliwiłoby automatyzację testów.

Zbadanie wpływu czynników wizualnych na wyniki systemu

Mając dostęp do usystematyzowanych testów można dokładnie zbadać wyniki jakie osiąga system oraz jego odporność na czynniki wizualne. Do takich czynników zalicza się: różne ustawienie twarzy, oświetlenie kierunkowe twarzy, pory dnia, zarost czy okulary. Proponuje się przyjęcie zbioru zdjęć jednej osoby w różnych warunkach oświetleniowych i porównanie uzyskanych wektorów.

Geometryczne cechy

W rozdziale 2.2 przedstawiono koncepcję wykorzystania innego rodzaju cech. Polega to na przygotowaniu wektora cech z geometryczną interpretacją. Każdy element takiego wektora mógłby oznaczać odległość między charakterystycznymi punktami twarzy. Aby uniezależnić uzyskiwane wyniki od odległości, wszystkie odległości muszą być względne. W przypadku tego typu wektora należy przetestować, jak sieć neuronowa poradzi sobie z tego typu cechami oraz zbadać wpływ pozy twarzy na osiągane wyniki. Algorytmy estymujące

położenie punktów charakterystycznych z reguły potrafią sobie poradzić z oświetleniem, więc nie byłby to czynnik mający wielkie znaczenie w tym przypadku, można go jednakże dodać do testów dla podwójnego sprawdzenia.

Ekstrakcja głębi

Ekstrakcja głębi na podstawie obrazów z dwóch kamer w obecnym stanie projektu nie daje zadowalających rezultatów. Fakt ten prawdopodobnie jest spowodowany złą kalibracją kamery, niewłaściwym położeniem kamer bądź niewłaściwym określeniem położenia kamer. Jest to część projektu, która musi zostać dopracowana, następnie może zostać zintegrowana do systemu. Poza zwiększeniem bezpieczeństwa systemu można wykorzystać mapę głębi do ekstrakcji cech. Ponieważ taka mapa ma zbyt wiele punktów, konieczne byłoby wykorzystanie algorytmów redukcji wymiarowości. Bardzo ciekawe mogłoby być porównanie wyników uzyskiwanych przez system na podstawie obrazów 2D i 3D. Informacje uzyskane dzięki estymacji głębi otwierają bardzo wiele możliwości rozwoju systemu.

Bibliografia

[1] Christopher M. Bishop. *Pattern recognition and machine learning.* Springer Science + Business Media, New York 2009

[2] Krzysztof Ślot. *Rozpoznawanie biometryczne. Nowe metody ilościowej reprezentacji obiektów* Wydawnictwo Komunikacji i Łączności, Warszawa 2000

[3] Stanisław Osowski. *Sieci Neuronowe w ujęciu algorytmicznym* Wydawnictwo Naukowo-Techniczne, Warszawa 1996

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* Wydawnictwo Helion, Gliwice 2017

[5] Martin, Robert C. *Czysty Kod* Wydawnictwo Helion, Gliwice 2014

[6] Yoav Freund, Robert E. Schapire. *A Short Introduction to Boosting* Journal of Japanese Society for Artificial Intelligence, 1999

[7] Paul Viola, Michael Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features.* Cambridge 2001

[8] Vahid Kazemi, Josephine Sullivan. *One Millisecond Face Alignment with an Ensemble of Regression Trees* KTH, Royal Institute of Technology Computer Vision and Active Perception Lab, 2014

[9] Daniel D. Lee and Seung, H. Sebastian. *Advances in Neural Information Processing Systems 13: Algorithms for Non-negative Matrix Factorization* str. 556-562 MIT Press, 2001

[10] C. Boutsidis, E. Gallopoulos. *SVD based initialization: A head start for nonnegative matrix factorization* University of Patras 2007

[11] Aapo Hyvärinen, Erkki Oja *Independent Component Analysis: Algorithms and Applications* Helsinki University of Technology, 2000

[12] Hartley, R. I., Zisserman, A. *Multiple View Geometry in Computer Vision* Cambridge University Press, 2004

[13] Itseez. *Open Source Computer Vision Library*
`https://github.com/itseez/opencv`

[14] Dokumentacja biblioteki OpenCv
`https://docs.opencv.org/2.4/index.html`

[15] Zbiór tutoriali z OpenCV
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

[16] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. *Scikit-learn: Machine Learning in Python* Journal of Machine Learning Research, volume 12, strony 2825-2830, 2011

[17] Informacje na temat zbioru zdjęć twarzy zawartym w bibliotece scikit-learn.
http://scikit-learn.org/stable/datasets/olivetti_faces.html

[18] News o pokonaniu zabezpieczeń systemu FaceID
http://www.bkav.com/d/topnews//view_content/content/103968/faceid-beatenbymasknotaneffectivesecuritymeasure

Spis treści

Spis rysunków