Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej
i Systemów Informacyjno-Pomiarowych
Zakład Elektrotechniki Teoretycznej
i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria oprogramowania

Rozpoznawanie gestów na przykładzie gry papier, kamień, nożyce

Adrian Szewczyk
nr albumu 279074

promotor
mgr inż. Marek Wdowiak

WARSZAWA 2018

Rozpoznawanie gestów na przykładzie gry papier, kamień, nożyce

Streszczenie

W pracy inżynierskiej zostało przedstawione podejście do problemu rozpoznawania gestów przez wszystkie jego etapy. Na początku zostało pokazane zabranie i przetwarzanie danych w celu uzyskania jak najlepszego zbioru uczącego jak i testowego. Potem zostały opisane etapy filtracji, segmentacji dłoni. Po uzyskaniu zbioru danych zostały stworzone klasyfikatory. Pierwszym podejściem było stworzenie klasyfikatora KNN, a następnie stworzenie klasyfikatora bazującego na sieciach neuronowych. Końcowym etapy pracy to porównanie wyników uzyskanych przez wymienione klasyfikatory na przykładzie gry papier, kamień, nożyce.

Słowa kluczowe: rozpoznawanie gestów, klasyfikacja, KNN, OpenCV, TensorFlow, sieci neuronowe

ENG Rozpoznawanie gestów na przykładzie gry papier, kamień, nożyce

Abstract

Keywords: ENG rozpoznawanie gestów, klasyfikacja, python, sieci neuronowe

WARSZAWA, 1 lutego 2018

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Rozpoznawanie gestów na przykładzie gry papier, kamień, nożyce:

- została napisana przeze mnie samodzielnie,
- nie narusza niczyich praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Adrian Szewczyk.....

Spis treści

Rozdział 1

Wstęp

1.1 Wprowadzenie

Istotną rolę w życiu społecznym pełnią gesty. Dodatkowa mowa niewerbowana może przekazać zdecydowanie więcej informacji niż tylko słowa wypowiedziane przez nadawcę. Gesty zazwyczaj służą do uzupełnienia mowy, do wspomożenia lub wzmocnienia przekazywanej wiadomości[1]. W języku migowym, który pozwala na wzajemną komunikację ludzi głuchych, gesty zastępują nawet słyszalne słowa. Oprócz samej mowy gesty mają też inne zastosowania. Przykładem takim może być gra papier, kamień, nożyce. W której każdy z graczy pokazuje jeden z trzech dostępnym gestów. W zależności od kombinacji gestów pokazanych przez graczy określany jest zwycięzca.

Interakcje człowiek-komputer są przedmiotem zainteresowania nauk na styku informatyki, nauk społecznych oraz projektowania technicznego i artystycznego. Interakcje pomiędzy użytkownikiem a komputer mają miejsce za pomocą interfejsów użytkownika, do których możemy zaliczyć zarówno sprzęt jak i oprogramowanie[2]. Szczególnym przypadkiem są interfejsy kinetyczne, których celem jest wykorzystywanie gestów ludzkich do interakcji człowiek-komputer.

Klasyfikacja gestów jest powszechnym problem przetwarzania obrazów. Jest to temat obszerny i coraz lepiej zbadany. Klasyfikacja gestów ma szerokie zastosowanie. Jedną z zalet rozpoznawania gestów jest możliwość sterowania naszym smartfonem za pomocą ruchu rąk. Po wykonaniu gestu, a następnie po rozpoznaniu go przez oprogramowanie telefonu wykonywana jest określona akcja. Innym wykorzystaniem klasyfikacji gestów jest sterowanie gier, gdzie jako przykład można podać urządzenie Kinect używane jako sensor dla konsoli Xbox 360[3].

1.2 Cel pracy

W swojej pracy dyplomowej będę chciał przedstawić sposób podejścia do problemu klasyfikacji gestów, a następnie zaimplementować działające rozwiązanie i wykorzystać go w praktyce na przykładzie gry papier, kamień, nożyce. Ogólnym celem będzie stworzenie klasyfikatora, który pozwoli na automatyczne rozpoznawanie gestów, które są wykonywane podczas gry.

1.3 Gra papier, kamień, nożyce

Jest to klasyczna gra towarzyska, w tradycyjnym wariancie przeznaczona dla dwóch graczy. W każdej rundzie gracze pokazują gesty przez siebie wybrane z trzech dostępnych. Wszystkie rundy odbywają się na ustalony wcześniej sygnał, a pokazywane gesty powinny być jak najbardziej zsynchronizowane. Gra kończy się wtedy, jeżeli jeden z graczy osiągnie określoną wcześniej ilość zwycięstw [4].

Trzy gesty dozwolone w grze:

- Papier
- Kamień
- Nożyce

(a) Gest papieru (b) Gest kamienia (c) Gest nożyc

Rysunek 1.1: Gesty papieru, kamienia, nożyce

Zwycięzca w danej rundzie określany jest w zależności od kombinacji gestów pokazanych przez graczy. Jeżeli oba gracze pokazali ten sam gest, wówczas następuje remis.

W przypadku kombinacji:

- papier-kamień - zwycięzcą zostaje gracz, który pokazał gest papieru
- papier-nożyce - zwycięzcą zostaje gracz, który pokazał gest nożyc
- kamień-nożyce - zwycięzcą zostaje gracz, który pokazał gest kamienia

Zachowując takie zasady, każdy z gestów ma takie samo prawdopodobieństwo zwycięstwa. W grze liczy się w dużej mierze szczęście, ale także umiejętność przewidywania gestów przeciwnika.

Rozdział 2

Wprowadzenie teoretyczne

2.1 Obrazy cyfrowe

Obrazem cyfrowym f nazywamy odwzorowanie dyskretne:

$$\mathbf{f}\colon D \to W$$

przy czym D jest dziedziną obrazu - skończonym zbiorem współrzędnych, a W jest jego przeciwdziedziną - zbiorem wartości [5]. W związku z dyskretną reprezentacją obrazu *funkcję obrazu* można określać pojęciem *macierzy obrazu*.

Dziedzina D jest iloczynem kartezjańskim skończonych zbiorów pojedynczych współrzędnych:

$$D = D_1 \times D_2 \times \ldots \times D_n$$

gdzie n jest rozmiarem przestrzeni obrazu, a D_i jest zbiorem wartości i-tej współrzędnej punktu obrazu.

Przeciwdziedzina W jest również iloczynem kartezjańskim:

$$W = W_1 \times W_2 \times \ldots \times W_n$$

Każdy ze zbiorów W_i w zależności od rodzaju obrazu może być podzbiorem zbioru liczb naturalnych, całkowitych lub liczb rzeczywistych.

Obrazy dwuwymiarowe są to obrazy dla których n przyjmuje wartość równą 2. Funkcja obrazu wówczas jest funkcją dwóch zmiennych **f**(x,y), w której odwzorowanie ma zazwyczaj postać:

$$\mathbf{f}\colon D = \{0, \ldots, x_{max} - 1\} \times \{0, \ldots, y_{max} - 1\} \to W$$

Wartość x_{max} oznacza rozmiar wzdłuż osi x, zaś wartość y_{max} rozmiar wzdłuż osi y. Punkty obrazów dwuwymiarowych nazywane są pikselami.

Obrazy które mogą przyjmować tylko jedną z dwóch wartości, wynoszących zazwyczaj 0 i 1 nazywamy **obrazami binarnymi**. Wówczas:

$$W = \{0, 1\}$$

Obrazy takie możemy traktować jako strukturę składającą się z dwóch zbiorów punktów obrazu. Pierwszy z nich A jest zbiorem punktów „włączonych", które możemy interpretować jako te znajdujące się na pierwszym planie - punkty pierwszoplanowe:

$$A = \{p\colon \mathbf{f}(p) = 1\}$$

gdzie f jest funkcją obrazu. Drugi ze zbiór B składa się z pozostałych punktów tzw. punktów „wyłączonych", które są również nazywane punktami tła:

$$B = \{p\colon \mathbf{f}(p) = 0\}$$

Rysunek 2.1: **Przykładowy obraz binarny dwuwymiarowy**

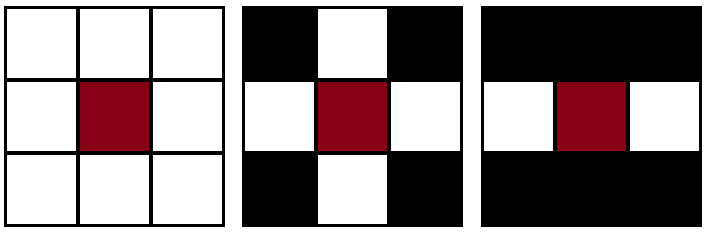
2.2 Operacje morfologiczne

Operacje morfologiczne pozwalają na złożone czynności związane z analizą kształtu poszczególnych elementów obrazu oraz położeniem ich względem siebie. W wyniku operacji struktura obiektu na obrazie zostaje zmieniona, w celu osiągnięcia określonych rezultatów. Operacje morfologiczne można zastosować dla obrazów binarnych oraz dla obrazów o wielu odcieniach szarości[6]. Dzięki nim jesteśmy wstanie wyszczególnić interesujące nas części czy też przefiltrować nasz obraz.

Podstawowymi operacjami morfologicznymi są: erozja, dylatacja, otwarcie oraz zamknięcie. Wymienione operacje można wykonywać sekwencyjnie, tworząc zaawansowane systemy analizy.

Operacje morfologiczne modyfikują wartości pikseli biorąc pod uwagę jego otoczenie. Liczbę punktów otoczenia określa tzw. element strukturalny, który definiuje wartości i ich rozmieszenie w otoczeniu. Posiada on jeden wyróżniony punkt, nazywany punktem centralnym, który nie musi być pikselem „włączonym".

Najczęściej stosowanym elementem strukturalnym jest element strukturalny elementarny. Obejmuje on najbliższe sąsiedztwo piksela. Elementarny element strukturalny o rozmiarze n jest elementem zawierający sąsiedztwo danego piksela o promieniu n[5].



(a) (b) (c)

Rysunek 2.2: Dwuwymiarowy element strukturalny: (a) ośmiospójny, (b) czterospójny, (c) liniowy. Punkt w kolorze białym oznacza piksel „włączony", w kolorze czarnym piksel „wyłączony", a w kolorze czerwonym oznacza piksel „włączony", który jest traktowany jako punkt centralny

Etapy przekształceń morfologicznych dla obrazów binarnych:

1. Szablon strukturalny jest przesuwany po całym obiekcie, tak aby punkt centralny szablonu był analizowanym pikselem
2. Następuje porównanie otoczenia analizowanego piksela z elementem strukturalnym
3. W zależności od stosowanej operacji morfologicznej wartość analizowanego piksela zmienia się lub też pozostaje bez zmian

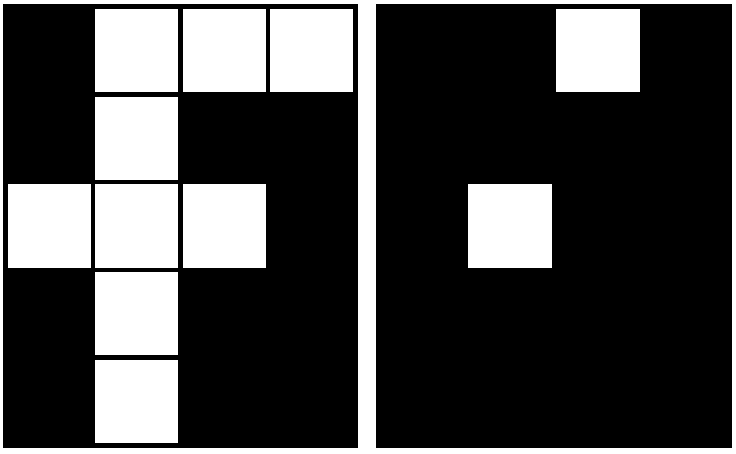
Erozja dla obrazów binarnych:

Erozja dla obrazów binarnych jest operacją zwężania i zmniejszania poprzez usunięcie pikseli granicznych. Usuwa wszystkie mniejsze obiekty, które możemy zinterpretować jako szumy. Definiujemy ją następująco:

$$A \ominus B = \{a\colon a + b \in A \text{ dla każdego } b \in B\}$$

Etapy operacji erozji:

1. Element strukturalny przesuwany jest iteracyjnie po całym obiekcie, tak aby punkt centralny szablonu był analizowanym pikselem
2. Porównuje się otoczenie analizowanego piksela z elementem strukturalnym
3. Jeżeli przynajmniej jeden piksel z otoczenia objętego przez szablon strukturalny ma wartość równą 0, to punkt centralny przyjmuje wartość 0. Jeżeli zaś taki przypadek nie wystąpił piksel centralny zachowuje swoją poprzednią wartość



(a) Obraz początkowy (b) Obraz wyjściowy

Rysunek 2.3: Operacja erozji na obrazie binarnym z elementem strukturalnym z rysunku 2.2c

Dylatacja dla obrazów binarnych:

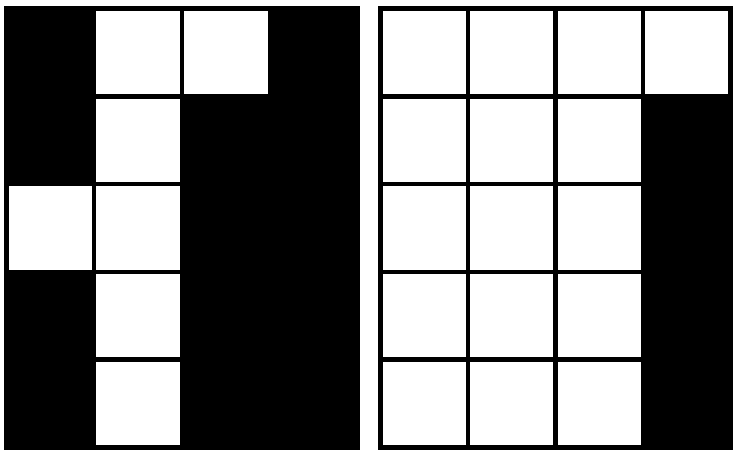
Dylatacja dla obrazów binarnych jest operacją rozszerzania i zwiększania. Pozwala na wypełnienie dziur binarnych w obiektach. Jeżeli dwa obiekty są położone blisko siebie, może nastąpić złączenie w jeden obiekt. Definiujemy ją następująco:

$$A \oplus B = \{p\colon p = a + b, a \in A, b \in B\}$$

Etapy operacji dylatacji:

1. Element strukturalny przesuwany jest iteracyjnie po całym obiekcie, tak aby punkt centralny szablonu był analizowanym pikselem

2. Następuje porównanie otoczenia analizowanego piksela z elementem strukturalnym
3. Jeżeli przynajmniej jeden piksel z otoczenia objętego przez szablon strukturalny ma wartość równą 1, to punkt centralny przyjmuje również wartość 1. Jeżeli zaś wszystkie piksele z otoczenia piksela centralnego określonego przez element strukturalny mają wartość 0 to wówczas piksel centralny przyjmuje wartość 0.



(a) Obraz początkowy (b) Obraz wyjściowy

Rysunek 2.4: Operacja dylatacji na obrazie binarnym z elementem strukturalnym z rysunku 2.2c

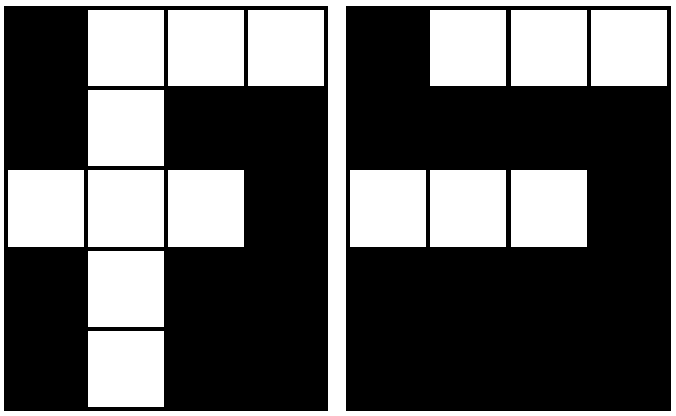
Otwarcie dla obrazów binarnych:

Operacja otwarcia dla obrazów binarnych jest sekwencyjnym wykonaniem metod erozji oraz dylatacji. Metoda ta wygładza obiekt, usuwa pomniejsze obiekty, a w dodatku nie modyfikuje znacząco wielkości obiektów tak jak to jest w przypadku zastosowania operacji erozji czy dylatacji. Definiujemy ją następująco:

$$A \circ B = (A \ominus B) \oplus B$$

Etapy operacji otwarcia:

1. Wykonywana jest operacja erozji na zadanym obrazie
2. Wykonywana jest operacja dylatacji na obraz, który jest wynikiem erozji z punktu 1.



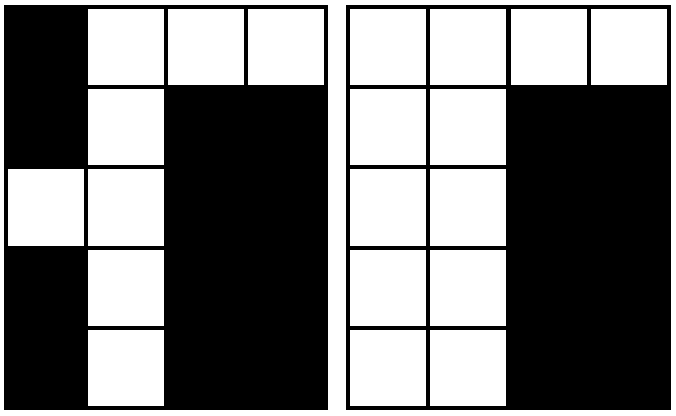
(a) Obraz początkowy (b) Obraz wyjściowy

Rysunek 2.5: Operacja otwarcia na obrazie binarnym z elementem strukturalnym z rysunku 2.2c

Zamknięcie dla obrazów binarnych:

Operacja zamknięcia dla obrazów binarnych jest sekwencyjnym wykonaniem metod dylatacji oraz erozji. W rezultacie uzyskujemy połączenie obiektów o zbliżonych odległościach, wypełnienie dziur w obiektach, jednakże końcowy kształt obiektu zostaje w dużym stopniu zmieniony w stosunku do kształtu przed zastosowaniem operacji zamknięcia. Definiujemy ją następująco:

$$A \bullet B = (A \oplus B) \ominus B$$



(a) Obraz początkowy (b) Obraz wyjściowy

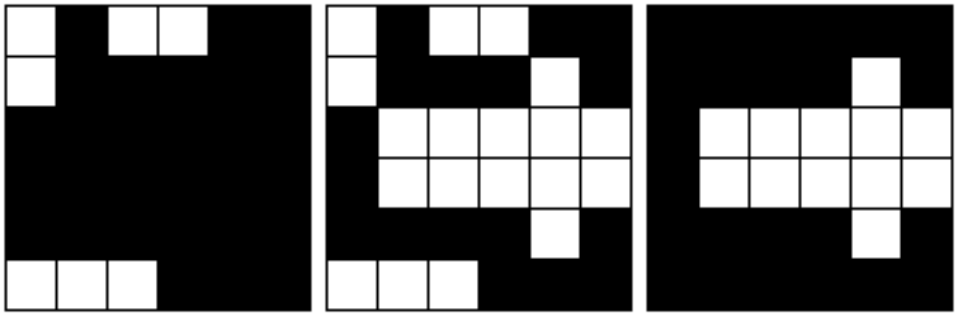
Rysunek 2.6: Operacja zamknięcia na obrazie binarnym z elementem strukturalnym z rysunku 2.2c

Etapy operacji zamknięcia:

1. Wykonywana jest operacja dylatacji na zadanym obrazie
2. Wykonywana jest operacja erozji na obraz, który jest wynikiem dylatacji z punktu 1.

2.3 Filtry ruchu

Filtry ruchu są wykorzystywane do generowania maski binarnej zawierającej poruszające się obiekty. Odejmowanie dwóch klatek z plików pozwala nam prześledzić, które z części obrazów zmieniły się.



(a) Tło (b) Obraz binarny (c) Różnica

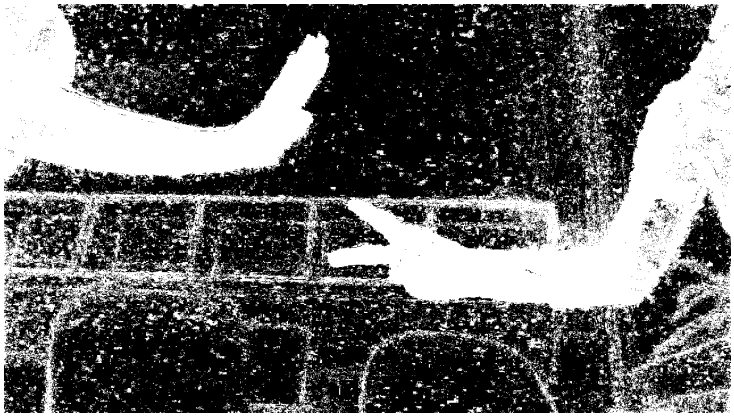
Rysunek 2.7: Różnica obrazu binarnego oraz jego tła

Biblioteka OpenCV posiada wiele gotowych rozwiązać do wykrywania obiektów w ruchu[7]. Jako przykładowe metody możemy wyróżnić następujące:

- BackgroundSubtractorMOG
- BackgroundSubtractorMOG2
- BackgroundSubtractorGMG
- BackgroundSubtractorKNN
- BackgroundSubtractorGSOC
- BackgroundSubtractorLSBP

BackgroundSubtractorKNN

BackgroundSubtractorKNN jest metodą segmentacji tła oraz pierwszego planu bazująca na K-najbliższych sąsiadach. Algorytm stosowany w metodzie jest bardzo wydajny, jeżeli liczba pikseli na pierwszym planie jest niska[8].



Rysunek 2.8: **Przykładowy obraz po zastosowaniu metody BackgroundSubtractorKNN z biblioteki OpenCV**

Segmentacja oczywiście nie jest całkowicie dokładna, dlatego w celu polepszenia efektów, warto pozbyć się niedokładności oraz możliwych szumów, które mogą wystąpić podczas analizy. Są one spowodowane różnymi odbiciami, cieniami czy innymi niezauważalnymi ruchami. Wówczas należy wykorzystać przedstawione wcześniej operacje morfologiczne, które poprawią nam wykonaną detekcję.

2.4 Detekcja kolorów

Celem detekcji kolorów jest wygenerowanie obrazu binarnego, w którym piksele „włączone" oznaczają te, które na zadanym obrazie są w określonym przedziale kolorów. Jeżeli dany piksel nie należy do określonego przedziału, wówczas staje się „wyłączony". Detekcja taka pozwala na wyłuskanie tylko obiektów o określonej barwie.

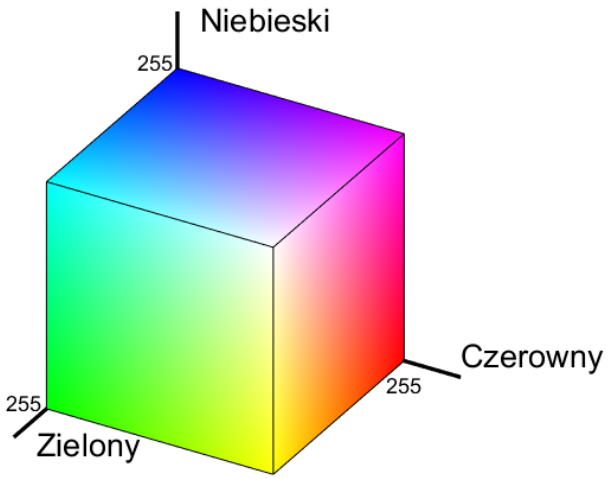
Detekcja jest bardzo wrażliwa na różnego rodzaju szumy. Częstymi powodem występowanie ich jest zmienne oświetlenie. Obiekt przy różnym świetle możemy odbierać kolorystycznie całkowicie inaczej. W przypadku, kiedy analizujemy dynamiczny ruch, wówczas dynamika również może wpływać na kolorystykę danego obiektu.

Aby polepszyć efekt końcowy detekcji obiektów o określonym kolorze, należy analogicznie jak w przypadku detekcji ruchu, wykorzystać odpowiednie operacje morfologiczne.

Model RGB

Model RGB składa się z trzech części: R (*ang. Red*) określa wartość barwy

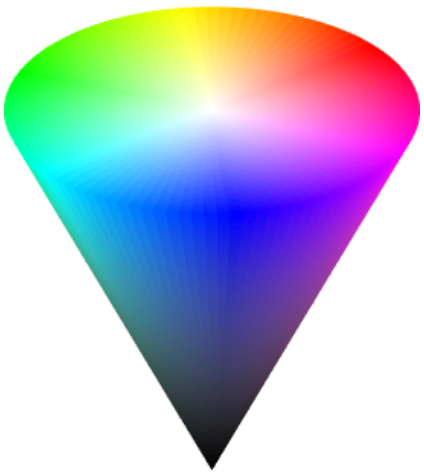
czerwonej, G (*ang. Green*) określa wartość barwy zielonej, B (*ang. Blue*) określa wartość barwy niebieskiej. Wartości najczęściej są podawane w przedziale liczb całkowitych 0-255 lub w przedziale liczb rzeczywistych 0-1. Model RGB jest szeroko stosowany w urządzeniach wyświetlających obraz oraz w urządzeniach analizujących obraz.



Rysunek 2.9: **Model RGB**

Model HSV

Przy detekcji kolorów tradycyjny model RGB najczęściej nie jest wystarczająco dobrym modelem. W zadaniach takich zdecydowanie lepiej sprawuje się model HSV.



Rysunek 2.10: **Model HSV**

HSV (*ang. Hue Saturation Value*) jest to model opisu przestrzeni barw, wprowadzony przez Alveya Raya Smitha w 1978 roku.

Model składa się z następujących części: barwa, nasycenie, wartość. Geometrycznie możemy go przedstawić jako stożek. Wszystkie barwy wywodzą się ze światła białego, czyli ze środka stożka. Składowa H, oznacza barwę w postaci kąta od 0 do 360 stopni. Składowa S czyli nasycenie, geometrycznie to odległość od środka na promieniu podstawy. Decyduje o bliskości naszego koloru do koloru białego. Ostatnia składowa V, to wartość oznaczająca wysokość na stożku, którą możemy traktować jako jasność naszego koloru. Czym mniejsza jej wartość tym kolor ciemniejszy[9].

Konwersja z modelu RGB na model HSV: [10]

Jeżeli wartość barw jest w przedziale liczb całkowitych 0-255, wówczas należy zmienić przedział na liczby rzeczywiste w zakresie 0-1:

$$
\begin{aligned}
R' &= R/255 \\
G' &= G/255 \\
B' &= B/255 \\
C_{max} &= max(R', G', B') \\
C_{min} &= min(R', G', B') \\
\Delta &= C_{max} - C_{min}
\end{aligned}
$$

$$
H = \begin{cases}
0^\circ & \text{,gdy } \Delta = 0 \\
60^\circ * (\frac{G'-B'}{\Delta} mod 6) & \text{,gdy } C_{max} = R' \\
60^\circ * (\frac{B'-R'}{\Delta} + 2) & \text{,gdy } C_{max} = G' \\
60^\circ * (\frac{R'-G'}{\Delta} + 4) & \text{,gdy } C_{max} = B'
\end{cases}
$$

$$
S = \begin{cases}
0 & \text{,gdy } C_{max} = 0 \\
\frac{\Delta}{C_{max}} & \text{,gdy } C_{max} \neq 0
\end{cases}
$$

$$
V = C_{max}
$$

2.5 Klasyfikatory

Klasyfikacja to wyznaczanie klasy decyzyjnej do której należy nowy, nieznany dotąd obiekt. Metody klasyfikacji możemy podzielić na dwie kategorie. Pierwsza z nich jest klasyfikacją nadzorowaną, druga zaś to klasyfikacja nienadzorowana.

Klasyfikacja nadzorowana w momencie uczenia klasyfikatora tzn. podawania zbioru danych treningowych, musi w sobie zawierać etykietę(atrybut

decyzyjny), która oznacza do jakiej klasy należy ten konkretny przypadek. Zaś w przypadku ucznia nienadzorowanego atrybut decyzyjny nie istnieje.

Elementami wejścia dla stworzenia klasyfikatora jest zbiór uczący, wyjściem jest atrybut decyzyjny, oznaczający przynależność do danej klasy [11].

2.5.1 Klasyfikator KNN

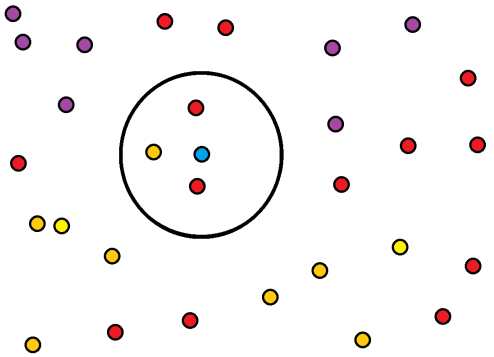
Klasyfikator KNN jest przykładem klasyfikacji nadzorowanej. Należy on do grupy algorytmów leniwych tzn. tych które nie tworzą wewnętrznej reprezentacji danych uczących, lecz określają rozwiązanie dopiero przy podaniu do klasyfikatora danych [12].

Po utworzeniu klasyfikatora bazującego na zbiorze uczącym, szacowanie atrybutu decyzyjnego odbywa się w następujących krokach:

1. Ustalamy wartość k
2. Znajdujemy k obiektów treningowych, najbliższych naszemu obiektowi
3. Analizowany obiekt należy do klasy najliczniejszej w znalezionym zbiorze z podpunktu 2

Ocenianie odległości pomiędzy dwoma obiektami polega na umieszczaniu obiektów w przestrzeni d-wymiarowej, gdzie d opisuje ilość atrybutów dla obiektów. Metryka miary może odbywać się w różny sposób. Najczęściej jest to miara euklidesowa, jednakże możemy również wykorzystać inne miary tj. Manhattana czy Minkowskiego.

Po znalezieniu k-najbliższych sąsiadów, atrybut decyzyjny przyjmuje wartość od najbardziej licznej klasy w wyznaczonym zbiorze.

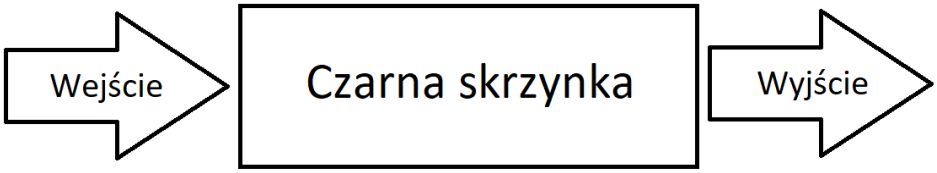


Rysunek 2.11: Przykład klasyfikatora KNN dla trzech sąsiadów

Na zaprezentowanym przykładzie możemy przeanalizować działanie klasyfikatora KNN dla trzech sąsiadów. Dla wyuczonego klasyfikatora KNN próbkami, które na grafice są zaprezentowane jako kółeczka, zaś ich kolor pokazuje do jakiej klasy obiektu należą. Naszym zadaniem jest określenie atrybuty decyzyjnego dla próbki, która została zaznaczona kolorem niebieskim. W pierwszym kroku zaznaczamy najbliższe sąsiedztwo, na ilustracji zatoczono go dużym okręgiem. Można zaobserwować, że w sąsiedztwie mamy dwa obiekty klasy czerwonej oraz jeden klasy żółtej. Wybór klasy naszego obiektu jest większościowy, co oznacza, że nasza nowa, nieznana dotąd próbka danych zostanie zakwalifikowana do klasy czerwonej.

2.6 Sieci neuronowe

Sieci neuronowe możemy traktować jako model czarnej skrzynki. Dla określonych danych wejściowych uzyskujemy wartości wyjściowe.

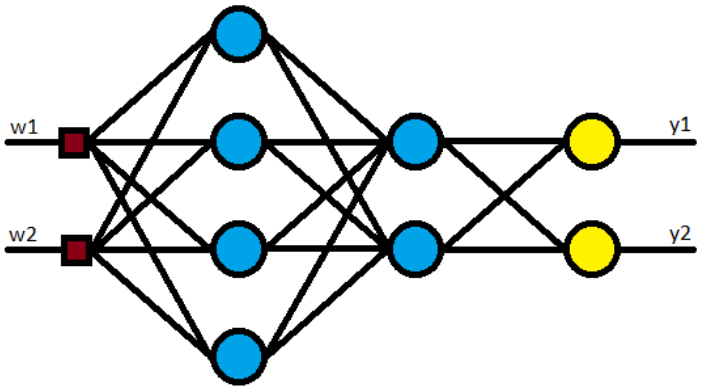


Rysunek 2.12: **Sieć neuronowa jako model czarnej skrzynki**

Sztuczne sieci neuronowe są to systemy komputerowe zainspirowane biologiczną siecią neuronowa ludzkiego mózgu. Sieci neuronowe nie są algorytmami, jednakże są ogólnym modelem wykorzystywanym przy uczeniu maszynowym i przy przetwarzaniu danych wejściowych.

Sieci neuronowe są to zbiory połączonych ze sobą sztucznych neuronów [13]. W biologicznym układzie mózgu, połączenia pomiędzy dwoma neuronami nazywamy synapsami. W sztucznych sieciach neuronowych połączenie to często nazywamy krawędzią.

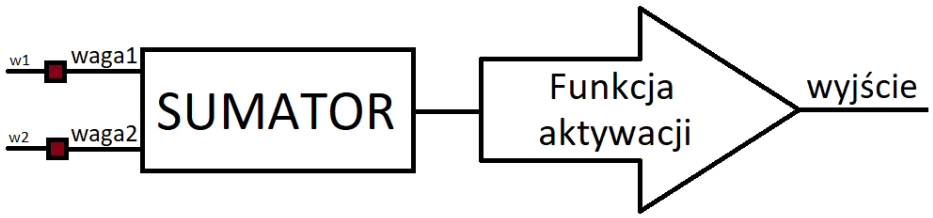
Czarna skrzynka to tzw. warstwa ukryta. Znajdują się w niej neurony ściśle ze sobą powiązane. Każdy z neuronów ma w sobie pewną wartość, powszechnie oznaczaną przez wartość liczby rzeczywistej. Każda krawędź w sieciach neuronowych posiada pewną określoną wagę. Zwiększanie lub zmniejszanie wag zmienia siłę sygnału w danym połączeniu pomiędzy neuronami.



Rysunek 2.13: **Przykładowa sieć neuronowa**

Na rysunku 3.2 zaprezentowano przykładowy schemat sieci neuronowej. Sieć ta zawiera dwa elementy wejścia, które są przekazywane do każdego neuronu z pierwszej warstwy ukrytej. Model posiada dwie warstwy ukryte, które są oznaczone na modelu kolorem niebieskim. Pierwsza z nich zawiera cztery neurony, zaś druga warstwa posiada dwa neurony. Ostatnia warstwa modelu, czyli warstwa wyjściowa oznaczona kolorem żółtym posiada dwa neurony, których wyjście jest interpretowane jako rezultat działania całej sztucznej sieci neuronowej.

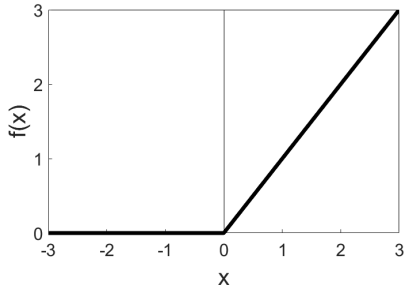
Sztuczne neurony na podstawie zsumowanych iloczynów wartości na wejściowych oraz wartości wag odpowiadających im krawędziom, wykonują pewną funkcję tak zwaną funkcję aktywacji, której rezultat jest traktowany jako wyjście dla pojedynczego neuronu.



Rysunek 2.14: **Przykład pojedynczego neuronu**

Funkcje aktywacji:

Wartość funkcji aktywacji dla danego neuronu, określa wartość wyjściową tego neuronu. Często używa się tej samej funkcji aktywacji dla tej samej warstwy, a niejednokrotnie wykorzystujemy tę samą funkcję aktywacji nawet do całej sieci neuronowej.

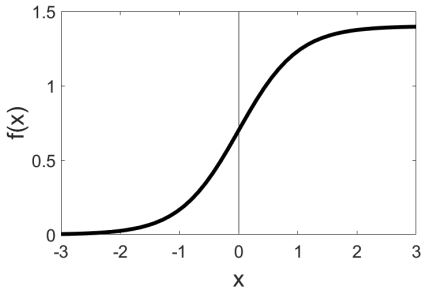


(a) Wykres

$$f(x) = max(0, ax)$$

(b) Wzór

Rysunek 2.15: **Funkcja ReLU**

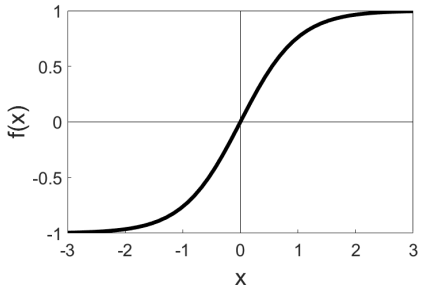


(a) Wykres

$$f(x) = \frac{1}{1+exp(-\beta x)}$$

(b) Wzór

Rysunek 2.16: **Funkcja Sigmoid**



(a) Wykres

$$f(x) = tanh(\beta x)$$

(b) Wzór

Rysunek 2.17: **Funkcja Tanh**

Softmax jest przykładem funkcji aktywacji, którą często wykorzystuje się jako funkcję aktywacji w warstwie wyjściowej. Suma wartości aktywacji dla całej warstwy jest równa 1. Wykorzystując tę normalizację, możemy wartości

wyjściowe neuronów zinterpretować jako prawdopodobieństwo przynależności do poszczególnych klas, które reprezentowane są przez neurony wyjściowe. Funkcję Softmax definiujemy następująco:

$$f(y_i) = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)}$$

Uczenie sieci neuronowych:

Uczenie sieci polega na dostrajaniu wartości wag poszczególnych neuronów. Przy uczeniu nadzorowanym sieci podstawowym algorytmem jest wsteczna propagacja błędu. Znając wartości wyjściowe dla danego zbioru danych wejściowych jesteśmy wstanie policzyć błąd działania sieci. Ten błąd jest przekazywany iteracyjnie do połączonych w poprzednich warstwach neuronów. Ważnym parametrem przy uczeniu sieci jest parametr szybkości uczeniu. Określa on szybkość uczenia się sieci neuronowych.

Wykorzystanie sieci neuronowych:

- Rozpoznawanie obrazów
- Sterowanie gestami
- Przewidywanie wartości akcji spółek giełdowych
- Generowanie plików wideo z obrazka
- Rozpoznawanie mowy
- Tworzenie opisu zdjęć
- Przewidywanie prognozy pogody
- Diagnostyka medyczna
- Ocena zdolności kredytowej
- Rekomendowanie reklam

2.6.1 Dobór i ocena cech

Podstawowym elementem tworzenia klasyfikatora jest dobór odpowiednich do niego cech na których będzie bazował. Źle dobrane elementy wejścia spowodują niedokładną klasyfikację. Dlatego tak bardzo ważne jest odpowiedni dobór cech opisujących.

Istotnym etapem dla każdej cechy była jej normalizacja danych, aby dana cecha nie powodowała większego wpływu od innych czy też wpływała nieznacząco w stosunku do innych. Przykładem normalizacji jest standaryzacja, którą możemy przedstawić zgodnie ze wzorem:

$$Z = \frac{x - \mu}{\sigma}$$

x - wartość przed normalizacją
μ - średnia
σ - odchylenie standardowe

Warunki dobrej cechy diagnostycznej:

- Powinny być wielkościami z podobnego zakresu
- Wartości danej cechy w obrębie określonej klasy powinny być do siebie możliwie jak najbardziej zbliżone
- Powinny być mało wrażliwe na szum
- Ta sama cecha powinna przyjmować różne wartości w przypadku różnych klas
- Ogólna liczba cech diagnostycznych nie może być zbyt duża [13]

Warunkiem dobrej cechy diagnostycznej, jest to, że ta sama cecha w obrębie różnych klas powinna przyjmować jak najbardziej różne wartości. Przy diagnostyce tego warunku można wykorzystać średnią arytmetyczną wartości danej cechy w poszczególnych klasach.

Kolejnym warunkiem na jaki warto zwrócić uwagę podczas oceny cech jest jej odchylenie standardowe w obrębie określonej klasy. Odchylenie standardowe można określić wzorem:

$$\sigma = \sqrt{\mathrm{E}((X - \mathrm{E}(X))^2)}$$

X - zmienna
E - wartość oczekiwana

Opisuje ono rozproszenie cechy w obrębie określonej klasy co implikuje, że dobrze dobrana cecha charakteryzuje się jak najmniejszym rozproszeniem.

Przykładowe cechy możliwe do wykorzystania do opisu masek binarnych:

- Stosunek szerokości do wysokości obiektu binarnego
- Stosunek powierzchni obiektu binarnego do powierzchni całego obrazu
- Stosunek powierzchni otoczki wypukłej na obiekcie binarnym do powierzchni całego obrazu
- Stosunek półosi małej do dużej elipsy opisanej na obiekcie binarnym
- Wartości punktów końcowych obiektu binarnego
- Orientacja obiektu binarnego

Przy wyborze odpowiednich cech ważnym elementem jest ich ocena, aby sprawdzić czy wybrana cecha będzie pozytywnie wpływała na klasyfikację.

Rozdział 3

Wykorzystane narzędzia

3.1 OpenCV

OpenCV (Open Source Computer Vision Library) jest to biblioteka wykorzystywana przy rozpoznawaniu obrazów oraz przy uczeniu maszynowym. Została napisana w języku C przez programistów z firmy Intel w 1999 roku.

W późniejszym czasie kolejne części biblioteki były także pisane w języku C++. Biblioteka OpenCV umożliwia pisanie kodów nie tylko w językach C i C++ ale również mamy możliwość wykorzystywania tej biblioteki w językach Python, Java czy też Matlab. W związku z popularnością biblioteki udostępniono nakładki do języków takich jak C#, Perl, Ruby czy Haskell aby móc również wykorzystywać zalety biblioteki.

Biblioteka OpenCV udostępnia zaawansowaną funkcjonalność w szeroko rozumianym pojęciu rozpoznawaniu obrazów:

- przetwarzania obrazów
- klasyfikowaniu wzorców
- dokładnych pomiarów obrazów

Jako główne zalety biblioteki OpenCV możemy wyróżnić, że jest ona darmowa oraz że jest o otwartych źródłach. Zawiera bogaty zakres funkcjonalności, a kod biblioteki został napisany w sposób zoptymalizowanym, tak aby operacje wymagające dużej mocy obliczeniowej czy działające w czasie rzeczywistym mogły wykonywać się możliwie jak najszybciej.

Biblioteka OpenCV od 20 lat znajduje zastosowanie w wielu dziedzinach naszego codziennego życia:

- medycyna
- robotyka
- samochody autonomiczne
- systemy antywłamaniowe

- systemy zabezpieczające
- rozpoznawanie gestów
- segmentacja obiektów
- wykrywanie ruchu
- rozszerzona rzeczywistość
- rozpoznawanie obiektów

3.2 Python

Jest to interpretowany, obiektowy język wysokiego poziomu. Używany jest w szerokiej gamie aplikacjach. Jest językiem ogólnego przeznaczenia, co oznacza, że może zostać wykorzystany praktycznie do wszystkiego. Jest rozwijany jako projekt otwartoźródłowy. Pojawił się w roku 1991, zaprojektowany przez holenderskiego programistę Guido van Rossum.

Zalety używania Pythona:

- prosta, czytelna, klarowna składnia, zmniejszająca ilość potrzebnych linijek kodu w programach
- dynamicznie zarządza pamięcią oraz typy danych
- nie wymusza stylu programowania
- jest wieloplatformowy
- posiada bogaty zbiór różnego rodzaju bibliotek
- łatwy do nauczenia, nawet dla osób zaczynających programować
- szybkość działania w stosunku do innych języków skryptowych

Jeśli mielibyśmy dyskutować o wadach językach Python to ciężko jednoznacznie wskazać takie. Na pewno część z programistów może uznać zalety dynamicznego zarządzania typami jako wadę, ponieważ w niektórych przypadkach może spowodować błędy trudniejsze do znalezienia. Działania Pythona jest również wolniejsze w stosunku do języków takich jak C czy C++. Za jedną z wad na pewno można uznać sposób programowania obiektowego jakie odbywa się w Pythonie. W szczególności nie ma enkapsulacji, istnieją metody które symulują takie działa, jednakże w stosunku do innych języków obiektowych czytelność kodu zdecydowanie jest zmniejszona.

Język mimo swoich lat ciągle zyskuje na popularności. Ostatnio wszedł do pierwszej trójki najbardziej popularnych języków programowania według TIOBE[1], wyprzedzając między innymi język C++, a ulegając jedynie językowi Java oraz C. Wzrost języka Python na pewno można łączyć ze wzrostem

[1] stan na grudzień 2018

popularności uczenia maszynowego i głębokich sieci, gdzie język Python jest jednym z najlepszych, jak nie najlepszym językiem w tych dziedzinach. Bardzo bogata biblioteka oraz przyjemna składnia sprawia, że język Python jest częściej wykorzystywany.

Jako ciekawostkę można powiedzieć, że nazwa języka Python wzięła się nie od zwierzęcia, lecz od słynnego Brytyjskiego serialu "Monty Python's Flying Circus".

3.3 TensorFlow

Jest to biblioteka programistyczna wykorzystywana w uczeniu maszynowym i głębokich sieciach neuronowych. Została wydana jako otwarte oprogramowanie przez Google Brain Team w dniu 9 listopada 2015.

Umożliwia pisanie programów m.in. w językach takich jak Python czy C. Jest dostępny na 64-bitowych systemach operacyjnych: Windows, Linux, macOS oraz na platformach mobilnych: Android oraz iOS. Zaś w maju 2017 został wydany TensorFlow Lite jako dedykowane rozwiązanie specjalnie dla użytkowników Androida.

Olbrzymią zaletą TensorFlow jest to, że reprezentuje on paradygmat Dataflow, w którym program ma postać grafu skierowanego modelującego przepływ danych pomiędzy niezależnymi operacjami w węzłach. W pierwszym kroku należy zdefiniować model, a następnie stworzyć tzw. TensorFlow session, która pozwala na uruchomienie programu. Takie podejście do pisania programów posiada ogromną zaletą jaką jest możliwość efektywnego programowania równoległego oraz rozproszonego.

TensorFlow daje możliwość nie tylko wykorzystania mocy obliczeniowej procesora, ale równie dobrze korzystania z mocy kart graficznych. Wszystko to powoduje pełne wykorzystanie komputerów przy obliczeniach równoległych.

TensorFlow oprócz niskopoziomowych struktur posiada również moduły wyższego poziomu. Do modułów niskiego poziomu możemy odwoływać się poprzez warstwę API, która umożliwia łatwy do używania interfejs przeznaczony do modelów głębokiego uczenia. Zaś nad warstwą API znajduje się warstwa wysokiego poziomu, przykładem jej jest Keras.

3.4 Keras

Otwartoźródłowa biblioteka programistyczna napisana w języku Python wydana w dniu 27 marca 2015 napisana przez pierwotnego autora François

Chollet.

Jej głównym przeznaczeniem jest to, aby w jak najprostszy i jak najszybszy sposób umożliwić pracę w głębokich sieciach neuronowych. Co więcej, biblioteka została zaprojektowana w sposób przyjemny do użytkowania, skupiona na modularności oraz rozszerzalności. Zaś od 2017 roku TensorFlow wspiera w swojej bibliotece Kerasa. Dzięki czemu Keras jest wysokopoziomową warstwą tworzenia modelów sieci neuronowych i jej uczenia.

Rozdział 4

Rozpoznawanie gestów na przykładzie gry papier, kamień, nożyce

4.1 Aktywizacja danych

Pierwszym etapem mojej pracy była aktywizacja danych potrzebnych do uczenia maszynowego. Zbieranie danych polegało na nagrywaniu plików wideo, na których dana osoba wykonywała określony gest. W swojej pracy dyplomowej zakładałem, że każdy wykonywany gest, jest pokazywany w niebieskiej rękawiczce. To założenie wykorzystywałem w całej swojej pracy inżynierskiej.

Rysunek 4.1: Rękawiczki wykorzystywane w pracy inżynierskiej

Osoba podczas nagrywania wykonuje ciąg określonych gestów, w dowolnym odstępie czasie. Jest ona ustawiona prostopadle do kamery, co pozwala, aby gest odbywał się poprzez wyciągnięcie ręki oraz jego pokazanie. Na nagraniu zostaje zarejestrowany wykonywany gest, tak aby jedyną widoczną częścią ciała była prawa dłoń oraz ewentualnie cześć przedramienia.

Rysunek 4.2: Przykładowa klatka z nagrania

Do nagrywania plików wideo używałem dwóch kamer dedykowanych w laptopach:

- USB2.0 UVC HD Webcam
- HD Webcam

Obie kamery pozwoliły na zadowalające efekty zbierania danych. Jakość nagrań przy dostatecznie dobrym oświetleniu była bardzo zadowalająca, pozwalająca bez problem rozpoznawać wykonywany gest. Również liczba klatek na sekundę pozwoliła, aby dynamiczny ruch był płynny i łatwy do prawidłowego wysegmentowania.

W celu zebrania jak najszerszego zakresu danych, zbieranie gestów nie ograniczyło się do jednej osoby, jednakże do pięciu różnych osób:

- trzech kobiet
- dwóch mężczyzn

Zbieranie danych na różnych osobach pozwala na zmniejszenie wpływu charakterystycznych cech dla danej osoby:

- długości dłoni
- szerokości dłoni
- ułożenia palców
- nachylenia dłoni

Tak zebrane dane dają ogólny pogląd na różnorodność cech danego gestu. Sprawia, że uczenie maszynowe jest skuteczniejsze i daje oczekiwane rezultaty.

Po zgromadzeniu całego materiału wideo przeszedłem do etapu wyznaczania numerów klatek nagrań, na których był widoczny moment wykonanego gestu. Informacje o numerach klatek nagrania wykorzystywałem w późniejszym etapie, przy segmentacji dłoni z wykonanym gestem.

W mojej pracy inżynierskiej wykorzystywałem zbiór danych składający się z:

- 350 próbek gestu papieru
- 350 próbek gestu kamienia
- 350 próbek gestu nożyc

Aby zapewnić prawidłowe rozpoznawanie, należy zgromadzić jak największą liczbę próbek gestów, która będzie w późniejszych etapach wykorzystywana przy uczeniu oraz weryfikacji klasyfikatorów.

4.2 Detekcja ruchu

Po zebraniu danych zająłem się ich przetwarzaniem. Pierwszym krokiem było wysegmentowanie obiektów, które znajdowały się w ruchu. Aby to osiągnąć, należało wygenerować maskę binarną, która rozdziela elementy obrazu, która są w ruchu, od tych, które są elementami statycznymi. Do tego wykorzystałem tzw. background subtraction czyli tłumacząc z angielskiego „odejmowanie tła".

W swojej pracy inżynierskiej w analizie wziąłem pod uwagę następujące dostępne funkcje w bibliotece OpenCV:

- BackgroundSubtractorMOG
- BackgroundSubtractorMOG2
- BackgroundSubtractorGMG
- BackgroundSubtractorKNN
- BackgroundSubtractorGSOC
- BackgroundSubtractorLSBP

(a) BackgroundSubtractorKNN (b) BackgroundSubtractorGSOC (c) BackgroundSubtractorLSBP

Rysunek 4.3: Przykładowe obrazy biorące udział w wyborze odpowiedniej funkcji z biblioteki OpenCV usuwającej tło

Po przeprowadzeniu licznych testów wywnioskowałem, że najlepszą metodą w mojej pracy inżynierskiej będzie funkcja z biblioteki OpenCV BackgroundSubtractorKNN. Uważam, że na przetestowanych próbkach segmentacja ruchu metodą tą dawała najlepsze rezultaty w porównaniu z innymi przetestowanymi metodami. Parametry funkcji przyjęły następującą końcową wartość:

1. history: 200
2. dist2Threshold: 8
3. detectShadows: false

(a) Znak papieru (b) Gest kamienia (c) Gest nożyc

Rysunek 4.4: Uzyskane rezultaty metodą BackgroundSubtractorKNN z biblioteki OpenCV

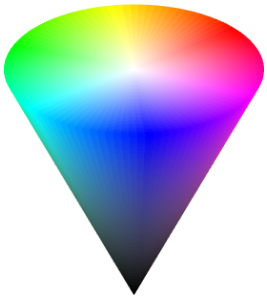
Można zauważyć, że wykonywany ruch dłonią jest bardzo widoczny, jednakże dodatkowo zawarta jest duża ilość szumu. Pozbycie się elementów dodatkowych, nieinteresujących nas części obiektów przedstawię w kolejnych podrozdziałach o detekcji kolorów oraz w podrozdziale o filtracji szumu.

4.3 Detekcja kolorów

Równolegle do segmentacji obiektów w ruchu, zająłem się detekcją obiektów o określonym kolorze. W swojej pracy inżynierskiej założyłem, że każdy

wykonywany gest jest pokazywany w niebieskiej rękawiczce, w celu łatwiejszej segmentacji dłoni. Nie użyłem tradycyjnego koloru skóry ze względu na fakt, że oprócz detekcji dłoni, spowodowałby także segmentację innych części ciała, takich jak twarz czy klatka piersiowa. Wszystko to mogłoby spowodować mniej precyzyjne rozpoznawanie gestu.

Wybrałem kolor niebieski ze względu na fakt, że jest to kolor dobrze rozróżniający się od innych, na który nie wpływa aż tak znacząco moc oświetlenia. Jako że każdy gest wykonywany jest w niebieskiej rękawiczce, mogłem wykorzystać ten fakt i przy pomocy modelu HSV określić zakres kątów barwy w tym modelu.



Rysunek 4.5: **Model HSV**

Na powyżej przedstawionym modelu HSV, można łatwo zaobserwować, że kolor niebieski znajduje się w pewnym przedziale składowej H modelu HSV. To pozwala na zdecydowanie łatwiejsze określenie zakresu niż w przypadku modelu RGB. Chciałem tak dobrać przedział kolorów w modelu HSV, aby był możliwie jak najszerszy w celu dobrej detekcji dłoni. Zakres ten zawierał dodatkowo kolory zbliżone do niebieskiego oraz większość z jego odcieni. To pozwoliło na zwiększoną liczbę wykrytych pikseli, a w tym lepiej wysegmentowaną dłoń. Jednakże wszystko to również niesie za sobą różnego rodzaju szumy, które nie są związane z obiektem dłoni.

(a) Znak papieru (b) Gest kamienia (c) Gest nożyc

Rysunek 4.6: Rezultaty detekcji dłoni w niebieskiej rękawiczce

Zakres modelu HSV jaki wykorzystałem w swojej pracy inżynierskiej jest następujący:

1. od HSV (98,50,20)
2. do HSV (130,255,255)

Segmentacje niosą ze są różnego rodzaju szum, jednakże jeżeli połączymy detekcję koloru niebieskiego z detekcją ruchu, która została opisana w poprzednim podrozdziale, może pozbyć się w większości tych niechcianych elementów.

(a) Znak papieru (b) Gest kamienia (c) Gest nożyc

Rysunek 4.7: Rezultaty detekcji dłoni w niebieskiej rękawiczce oraz wykrywania ruchu

Jak widzimy większość z niepożądanych szumów została usunięta za sprawą połączania dwóch masek binarnych. Możemy też zaobserwować, że nie wszystko zostało prawidłowo przefiltrowane, mogły też pozostać tzw. dziury binarne czyli części obiektów, które nie zostały zakwalifikowane jako części obiektu, a nimi faktyczne są.

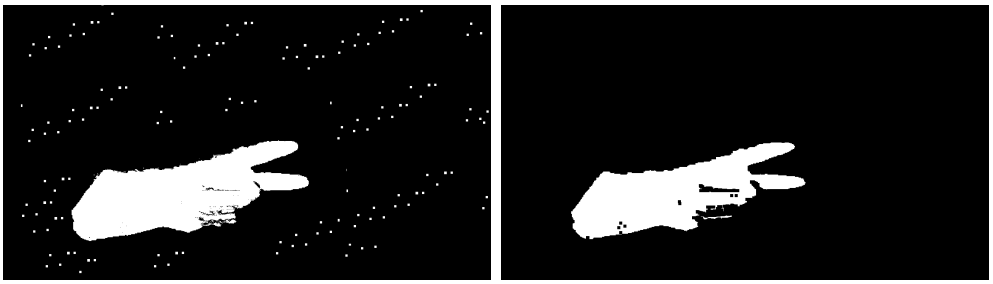
Wszystko to powoduje, że konieczne są kolejne filtracje szumów. W tym przypadku wykorzystałem operacje morfologiczne na obrazach binarnych, które opiszę w następnym podrozdziale.

4.4 Filtracja szumów

Połączenie iloczynowe maski binarnej uzyskanej z segmentacji obiektów w ruchu oraz maski binarnej z obiektami o określonym kolorze niesie za sobą możliwe niedokładności, które dla polepszenia efektowności rozpoznawania gestów należy zminimalizować. Odbywa się to poprzez zastosowanie operacji morfologicznych na obrazie binarnym.

Dodatkowe elementy:

Pierwszym problemem po wygenerowaniu maski binarnej można uznać wszystkie dodatkowe elementy, które nie powinny być zakwalifikowane jako część dłoni. Segmentacja elementów obrazu w ruchu może nieść ze sobą różnego rodzaju szum spowodowany tym, że inne obiekty, różne od naszej dłoni też mogą się poruszać, co w wyniku generacji maski binarnej zostaną też wzięte pod uwagę. Kolejnym problem są także wszystkie te piksele, które są koloru niebieskiego lub też są zbliżone do niego, a więc przy wyłuskiwaniu wszystkich pikseli o kolorze rękawiczki, czyli o kolorze niebieskim, zostaną również uwzględnione jako obiekt.



(a) Maska binarna przed operacją morfologiczną erozji

(b) Maska binarna po operacji morfologicznej erozji

Rysunek 4.8: **Zastosowanie operacji morfologicznej erozji z elementem strukturalnym kwadratu o boku 8 pikseli**

Zastosowanie operacji morfologicznej erozji na masce binarnej powoduje usunięcie elementów niechcianych, które są traktowane jako dodatkowy, niepotrzebny szum. W swojej pracy inżynierskiej wykorzystałem w pierwszym kroku filtracji masek binarnych operację morfologiczną erozji z małym elementem strukturalnym, z kwadratem o boku 4. Wykorzystanie takiej operacji w mojej ocenie dało zadowalające efekty, które pozwolą na dokładniejsze rozpoznawanie gestów.

Ubytku w dłoni:

Oprócz pozbycia się dodatkowych elementów na obrazie binarnym należy także, wypełnić luki, które mogły się pojawić podczas segmentacji obiektów w ruchu oraz przy uzyskiwaniu maski binarnej z obiektami o określonym kolorze. Są one spowodowane tym, że pewne części z obrazów podczas tych operacji nie zostały prawidło zakwalifikowane. Zbyt mocne lub też zbyt słabe światło czy też pojawiające się cienie mogły wpłynąć na kolor niebieskiej rękawiczki, a przez to spowodować niedokładne wykrycie. Również gest ręki

mógł być bardzo powolny, a przez co nie zostać prawidło uznanym przez operacje segmentacji ruchu. Jednakowo też dynamiczny ruch ręku w połączeniu z występujących w otoczeniu światłem wpływa na odbierany kolor rękawiczki. Należy też dodatkowo wziąć pod uwagę dziury w obrazie binarnym, który mogły wyniknąć z operacji morfologicznej erozji, która została zastosowana jako pierwsza.

(a) Maska binarna przed operacją morfologiczną dylatacji

(b) Maska binarna po operacji morfologicznej dylatacji

Rysunek 4.9: **Zastosowanie operacji morfologicznej dylatacji z elementem strukturalnym kwadratu o boku 11 pikseli**

Zastosowanie operacji morfologicznej dylatacji na masce binarnej powoduje wypełnienie dziur, które mogły wystąpić w masce binarnej. W swojej pracy inżynierskiej do pozbycia się różnego rodzaju dziur w maskach binarnych użyłem dwukrotnej operacji morfologicznej dylatacji z elementem strukturalnym kwadratu o boku 11.

(a) Gest kamienia przed filtracją

(b) Gest kamienia po filtracji

Rysunek 4.10: **Przykładowe maski binarne po zastosowaniu operacji morfologicznych erozji oraz dylatacji z elementami strukturalnymi kwadratu o boku 8 dla erozji oraz kwadratu o boku 11 dla operacji dylatacji**

W swojej pracy inżynierskiej zastosowałem następujące operacje do przefiltrowania masek binarnych:

1. Operację morfologiczną erozji na masce binarnej z elementem strukturalnym kwadratu o boku 4
2. Operację morfologiczną zamknięcia z elementem strukturalnym kwadratu o boku 8 na wyjściowej masce binarnej uzyskanej w podpunkcie nr 1
3. Operację morfologiczną dylatacji z elementem strukturalnym kwadratu o boku 11 na wyjściowej masce binarnej uzyskanej w podpunkcie nr 2
4. Operację morfologiczną dylatacji z elementem strukturalnym kwadratu o boku 11 na wyjściowej masce binarnej uzyskanej w podpunkcie nr 3

(a) Gest papieru (b) Gest kamienia (c) Gest nożyc

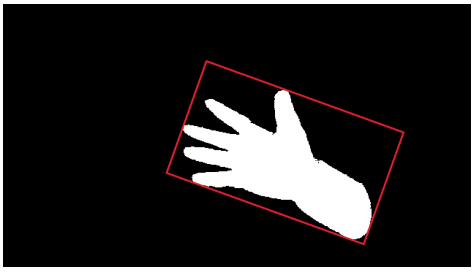
Rysunek 4.11: **Przykłady efektów wykonanej filtracji, która została powyżej opisana**

Cała filtracja w mojej ocenie dała bardzo dobre rezultaty, które znacząco polepszyły prawidłową segmentację dłoni, a przez to w późniejszych krokach dokładniejsze rozpoznawanie dłoni.

4.5 Znajdowanie obiektów

Ostatnim etapem przetwarzania obrazów w mojej pracy inżynierskiej było wydzielenie z maski binarnej prostokąta, zawierającego w sobie całą wysegmentową wcześniej dłoń, bez ogólnego tła. Dzięki czemu jesteśmy wstanie analizować jedynie tą część maski binarnej, która zawiera w sobie całą dłoń, bez zbędnych innych elementów.

Znajdowanie to polegało na wyszukaniu prostokąta na całej masce binarnej, która będzie miała największe pole ze wszystkich znalezionych konturów na obiektach z maski binarnej.



Rysunek 4.12: **Zaznaczenie na czerwono prostokąta o największym polu, który zawiera w sobie całościowo obiekt**

Po znalezieniu takiego prostokąta następowało przycięcie maski binarnej do odpowiednich rozmiarów zgodnie ze znalezionym prostokątem o największym polu. Prostokąt ten jest przyległy przez obiekt binarny z każdej swojej strony, co oznacza, że większe przycięcie nie jest możliwe, bez zmniejszania znalezionego obiektu.

Rysunek 4.13: **Obcięcie zgodnie ze znalezionym prostokątem**

Etapy przycięcia maski binarnej:

1. Przypisz do obecnie znalezionego największego pola prostokąta wartość 0
2. Znajdź wszystkie kontury obiektów na masce binarnej
3. Dla każdego konturu wykonaj:
   (a) Oblicz współrzędne lewego górnego oraz prawego dolnego wierzchołka prostokąta, który będzie otaczał analizowany kontur
   (b) Oblicz pole powierzchni tak utworzonego prostokąta
   (c) Jeżeli pole powierzchni utworzonego prostokąta jest większe niż pole powierzchni obecnie znalezionego największego prostokąta, to traktur tą powierzchnię jako największą i dodatkowo zapamiętaj współrzędne prostokąta

4. Przytnij maskę binarną o zapamiętanych wierzchołkach największego znalezionego prostokąta

Efekt końcowy wysegmentowania dłoni zawiera w sobie obiekt binarny reprezentujący dłoń, którą jest wykonywany gest. Zostanie on wykorzystany przez klasyfikatory opisane w następnych podrozdziałach.

(a) Gest papieru (b) Gest kamienia (c) Gest nożyc

Rysunek 4.14: **Końcowa segmentacja dłoni**

4.6 Dobór i ocena cech

Po wysegmentowaniu gestów dla uczenia maszynowego przeszedłem do etapu opisu odpowiednimi cechami zgromadzonego materiału. Skupiłem się, na właściwościach binarnych jakie one przedstawiają.

Zdecydowałem się, że gesty będę opisywał za pomocą następujących cech:

- Stosunek szerokości do długości rozmiarów maski binarnej
- Stosunek powierzchni znalezionego obiektu dłoni do powierzchni całego obrazu
- Stosunek powierzchni otoczki wypukłej utworzonej na znalezionej dłoni do powierzchni całego obrazu

Pierwszym krokiem przy analizowaniu cech opisujących dane było wyliczenie średnich w obrębie określonego gestu.

Nazwa cechy	**Papier**	**Kamień**	**Nożyce**
Stosunek szerokości do długości	0.43	0.59	0.78
Stosunek powierzchni obiektu dłoni do powierzchni całego obrazu	0.48	0.38	0.84
Stosunek powierzchni otoczki wypukłej utworzonej na znalezionej dłoni do powierzchni całego obraz	0.35	0.62	0.84

Tablica 4.1: Wartości średnie

Przy ocenie jakości dobranych cech wykorzystałem odchylenie standardowe. Dzięki niemu byłem wstanie porównać jak dana cecha jest porozrzucana w obrębie określonego gestu.

Nazwa cechy	**Papier**	**Kamień**	**Nożyce**
Stosunek szerokości do długości	0.10	0.18	0.05
Stosunek powierzchni obiektu dłoni do powierzchni całego obrazu	0.09	0.10	0.04
Stosunek powierzchni otoczki wypukłej utworzonej na znalezionej dłoni do powierzchni całego obraz	0.10	0.16	0.05

Tablica 4.2: Odchylenie standardowe

Odchylenie standardowe dla danej cechy, w obrębia określonego gestu nie były dużą wartością, co oznacza, że cecha ta nadaje się dobrze, aby ją wykorzystać przy rozpoznawaniu gestów.

Po przeanalizowaniu odchyleń standardowych doszedłem do wniosku, że wszystkie trzy, wcześniej wybrane cechy są odpowiednie, aby je wykorzystać do stworzenia klasyfikatora.

4.7 Rozpoznawanie gestów przy pomocy klasyfikatora KNN

Dla klasyfikatora KNN przygotowałem następujący zestaw cechy opisujących maskę binarną:

- Stosunek szerokości do długości obrazu
- Stosunek powierzchni znalezionego obiektu dłoni do powierzchni całego obrazu
- Stosunek powierzchni otoczki wypukłej utworzonej na znalezionej dłoni do powierzchni całego obrazu

Każda z cech została poddana etapowi normalizacji w postaci standaryzacji. Została ona przeprowadzona zgodnie ze wzorem: $Z = \frac{x - \mu}{\sigma}$, gdzie x - wartość przed normalizacją, μ - średnia, σ - odchylenie standardowe

Klasyfikator został oparty na klasycznej odległości euklidesowej. Zbiór danych, który został opisany w poprzednich podrozdziałach składał się z 1050 różnych próbek danych. Został on podzielony w stosunku 8:2, dzięki czemu zbiór uczący liczył 855 próbek, zaś zbiór weryfikacyjny zawierał 195 próbek.

W zależności od doboru parametry klasyfikator k, osiągnąłem następujące wyniki:

k = 3
Średnia dokładność klasyfikatora: **83.76%**

k = 5
Średnia dokładność klasyfikatora: **86.08%**

k = 7
Średnia dokładność klasyfikatora: **82.41%**

k = 9
Średnia dokładność klasyfikatora: **81.68%**

```
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='paper', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='paper', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
> Przewidywanie='scissors', Gest='scissors'
Dokładność: 86.31578947368422%
```

Rysunek 4.15: **Uruchomienie klasyfikatora KNN**

Z wykonanych analiz dokładności klasyfikatorów w zależności od parametru k wynika, że najlepsze efekty można było osiągnąć używając klasyfikatora KNN z parametrem k o wartości 5, który działał ze średnią dokładnością **86.08%**.

4.8 Rozpoznawanie gestów przy pomocy sieci neuronowej

4.8.1 Sieć neuronowa dla cech

Analogicznie jak w przypadku klasyfikatora KNN również i kolejny klasyfikator oparłem na wyznaczonych wcześniej cechach. Klasyfikator ten różnił się tym od poprzedniego, że bazował tym razem na sieciach neuronowych. A swoją skuteczność klasyfikacji zawdzięcza uczeniu maszynowemu.

Sieć została stworzona w języku Python, korzystając z biblioteki TensorFlow oraz jako wysokopoziomowe API sieci neuronowych wykorzystałem bibliotekę Keras.

Sieci neuronowe, które wykorzystywałem przy budowie klasyfikatora składały się z następujących części:

1. Warstwa wejściowa
2. Jedna warstwa ukryta
3. Warstwa wyjściowa

Do warstwy wejściowej były podawane wartości następujących cech:

- Stosunek szerokości do długości obrazu
- Stosunek powierzchni znalezionego obiektu dłoni do powierzchni całego obrazu
- Stosunek powierzchni otoczki wypukłej utworzonej na znalezionej dłoni do powierzchni całego obrazu

Każda z cech została poddana etapowi normalizacji w postaci standaryzacji, która zdecydowanie korzystnie wpływała na skuteczność klasyfikacji.

Zbiór uczący sieci neuronowe zawierał 855 próbek, zaś zbiór weryfikacyjny zawierał w sobie 195 próbek. Każdy z trzech gestów miał po 350 próbek, zaś przydział do odpowiedniego zbioru odbywał się w sposób losowy.

Warstwa wyjściowa każdej z przetestowanych sieci, składała się z funkcji aktywacji softmax, której wyjście traktowałem jako prawdopodobieństwo danego z trzech gestów. Każdy z trzech neuronów wyjściowych oznaczył właśnie to prawdopodobieństwo w następującej kolejności:

1. gest papieru
2. gest kamienia
3. gest nożyc

Pierwszym zbiorem sieci jaki przetestowałem były to sieci z jedną warstwą ukrytą. Do uczenia sieci wykorzystywałem metodę GradientDescentOptimizer z parametrem uczenia równym 0.01. Zaś sama sieć została uruchomiana dla 100 epok. Ilość neuronów w warstwie ukrytej oraz funkcje aktywacji dla tej warstwy i warstwy wejściowej były to parametry, które doświadczalnie zmieniałem.

Zanotowałem następujące dokładności rozpoznawania siecią przy wyżej wymienionych parametrach:

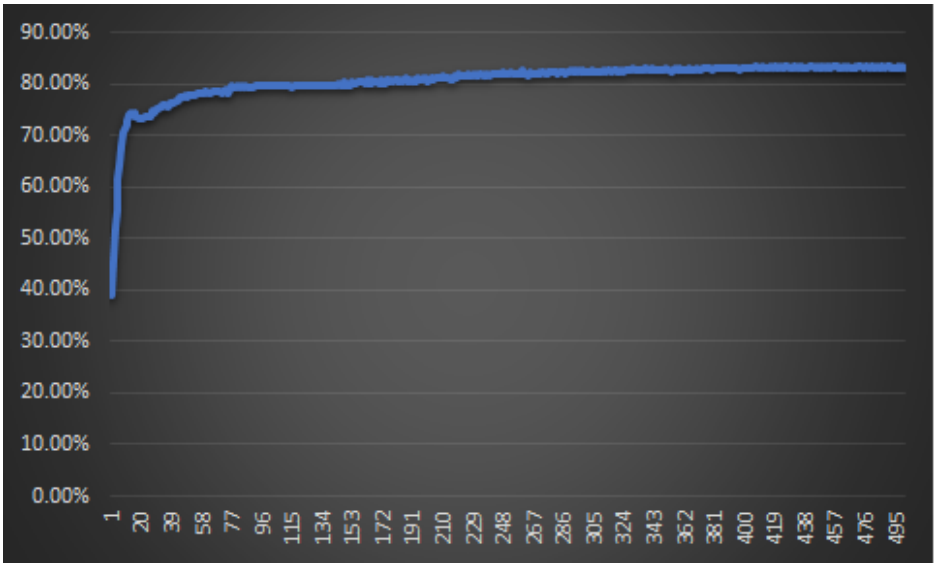
	ReLU funkcją aktywacji	**Sigmoid funkcją aktywacji**	**Tanh funkcją aktywacji**
2 neurony w warstwie ukrytej	57.54%	63.29%	74.28%
4 neurony w warstwie ukrytej	80.05%	64.52%	81.35%
6 neuronów w warstwie ukrytej	83.43%	69.01%	83.08%
8 neuronów w warstwie ukrytej	81.59%	67.54%	77.26%
10 neuronów w warstwie ukrytej	80.03%	64.21%	78.11%
12 neuronów w warstwie ukrytej	79.89%	57.63%	76.41%
15 neuronów w warstwie ukrytej	81.28%	65.59%	79.62%
20 neuronów w warstwie ukrytej	78.82%	61.77%	80.09%
25 neuronów w warstwie ukrytej	78.55%	54.23%	77.78%
50 neuronów w warstwie ukrytej	77.73%	60.19%	79.13%

Tablica 4.3: Skuteczność sieci neuronowej

Po przeprowadzonych doświadczeniach okazało się, że najlepsze wyniki można było uzyskać siecią neuronową składającą się z 6 neuronów w warstwie ukrytej, przy wykorzystywaniu funkcji aktywacji ReLU w warstwie wejściowej oraz w warstwie ukrytej. Wszystkie wartości zostały przedstawione dla 100 epok uczenia sieci.

W następnym etapie swojej pracy przeanalizowałem, wpływ ilości epok podczas uczenia sieci. Testowanie odbyło się na sieci, która dała najlepsze rezultaty w poprzednim doświadczeniu.

Analiza uczenia w poszczególnych epokach:



Rysunek 4.16: **Skuteczność uczenia sieci w zależności od ilości epok uczenia**

Na wykresie można zaobserwować bardzo dynamiczny wzrost skuteczności uczenia, dla pierwszych 50 epok. Kolejnym 50 epokach następuje również sporo wzrost skuteczności, jednakże już nie tak gwałtowny. Uczenie przy ponad 100 opokach daje to coraz mniejszy wzrost skuteczności uczenia, co można z łatwością zobaczyć na wykresie. Analiza ta spowodowała, że sieci neuronowe poddawałem uczeniu dla 100 epok.

Ostatnim parametrem, który przeanalizowałem był dobór parametru szybkości uczenia w metodzie GradientDescentOptimizer, wpływa on w jak szybki sposób zbliżamy się do optymalnego rozwiązania według wyliczonego gradientu.

Szybkość uczenia	Skuteczność sieci neuronowej
0.001	67.19%
0.010	83.43%
0.020	78.91%
0.050	80.37%
0.100	80.34%
0.150	79.88%
0.200	80.02%
0.250	81.11%
0.500	82.78%

Tablica 4.4: Skuteczność sieci neuronowej w zależności od współczynnika szybkości uczenia

Większość z prób była do siebie zbliżona skutecznością. W wykonanych doświadczeniach sieć dawała najlepsze rozwiązania dla parametru szybkości uczenia o wartości: 0.01.

```
855/855 [==============================] - 0s 48us/step - loss: 0.3762 - acc: 0.8632
Epoch 87/100
855/855 [==============================] - 0s 23us/step - loss: 0.3825 - acc: 0.8491
Epoch 88/100
855/855 [==============================] - 0s 28us/step - loss: 0.3757 - acc: 0.8526
Epoch 89/100
855/855 [==============================] - 0s 35us/step - loss: 0.3786 - acc: 0.8503
Epoch 90/100
855/855 [==============================] - 0s 35us/step - loss: 0.3771 - acc: 0.8561
Epoch 91/100
855/855 [==============================] - 0s 35us/step - loss: 0.3812 - acc: 0.8444
Epoch 92/100
855/855 [==============================] - 0s 35us/step - loss: 0.3801 - acc: 0.8480
Epoch 93/100
855/855 [==============================] - 0s 53us/step - loss: 0.3843 - acc: 0.8468
Epoch 94/100
855/855 [==============================] - 0s 39us/step - loss: 0.3816 - acc: 0.8526
Epoch 95/100
855/855 [==============================] - 0s 33us/step - loss: 0.3787 - acc: 0.8491
Epoch 96/100
855/855 [==============================] - 0s 34us/step - loss: 0.3786 - acc: 0.8561
Epoch 97/100
855/855 [==============================] - 0s 33us/step - loss: 0.3814 - acc: 0.8456
Epoch 98/100
855/855 [==============================] - 0s 32us/step - loss: 0.3763 - acc: 0.8515
Epoch 99/100
855/855 [==============================] - 0s 29us/step - loss: 0.3738 - acc: 0.8655
Epoch 100/100
855/855 [==============================] - 0s 55us/step - loss: 0.3752 - acc: 0.8585
195/195 [==============================] - 0s 277us/step
Dokładność 0.8102564105620751
```

Rysunek 4.17: **Uruchomienie sieci neuronowej**

Przeprowadzona analiza pozwoliła ustalić ostateczne parametry sieci:

- Sieć składa się z następujących warstw: warstwa wejściowa, jedna warstwa ukryta, warstwa wyjściowa
- Warstwa wejściowa składa się z trzech neuronów reprezentujące wartości cech dla danej próbki, zaś funkcją aktywacji neuronów tej warwie jest funkcja ReLU
- Warstwa ukryta składa się z 6 neuronów, a funkcja aktywacji każdego z nich to również ReLU
- Warstwa wyjściowa składa się z trzech neuronów, które reprezentują prawdopodobieństwo gestu w kolejności: papier, kamień, nożyce. Funkcja aktywacja dla każdego z neuronów to softmax.
- Sieć była uczona na 100 epokach
- Jako metodę uczenia uczenia wykorzystałem dostępny w bibliotece Keras GradientDescentOptimizer, ze współczynnikiem uczenia 0.01
- Sieć była uczona 855 próbkami zbioru uczącego
- Sieć była testowana 195 próbkami zbioru weryfikującego

Sieć z podanymi wyżej parametrami dała średnią skuteczność na poziomie **83.43%**. Jest to wartość zadowalająca, zbliżona do wartości jaka została uzyskana za pomocą klasyfikatora KNN.

4.8.2 Sieć neuronowa obrazów binarnych

Kolejnym podejściem do klasyfikacji była analiza wysegmentowanej dłoni gestu siecią neuronową tym razem bez opisywania gestu cechami, lecz podaniem do klasyfikatora całego otrzymanego wcześniej obrazu binarnego.

Po procesie segmentacji dłoni z gestem, opisanym w poprzednich podrozdziałach, należało ustandaryzować rozmiar obrazu binarnego, który został wykorzystany jako element wejścia dla sieci neuronowej. Ustaliłem, że rozmiar będzie wynosił 60 pikseli szerokości oraz 25 pikseli wysokości. Stosunek obu wartości chciałem tak dobrać, aby zachował jak najbardziej ogólną jego wartość dla wszystkich obrazów binarnych z wykorzystywanego zbioru danych. Zmniejszenie rozmiaru, ma również korzystny wpływ na szybkość działania sieci, jednakże w wyniku czego tracimy pewne dostępne szczegóły na obrazie.

Sieć została stworzona w języku Python, korzystając z biblioteki TensorFlow oraz jako wysokopoziomowe API sieci neuronowych wykorzystałem bibliotekę Keras.

Zbiór uczący sieci neuronowe zawierał 855 próbek, zaś zbiór weryfikacyjny zawierał w sobie 195 próbek. Każdy z trzech gestów miał po 350 próbek, zaś przydział do odpowiedniego zbioru odbywał się w sposób losowy.

Sieci neuronowe, które wykorzystywałem przy budowie klasyfikatora składały się z następujących części:

1. Warstwa wejściowa
2. Jedna warstwa ukryta
3. Warstwa wyjściowa

Jako elementy wejścia były wykorzystywane wysegmentowane wcześniej obrazy dłoni z gestem i skonwertowane do rozmiaru 60 pikseli szerokości i 25 pikseli wysokości. Wartości pikseli obrazu rgb zostały przeskalowane z przedziału 0-255 do wartości z przedziału 0-1. Do warstwy wejściowej sieci, podane zostały wartości tablicy o rozmiarze 25x60x3, która zawierała przeskalowane wartości rgb pikseli obrazu dłoni. Ilość neuronów w warstwie ukrytej był to parametr, który doświadczalnie zmieniałem. Funkcja aktywacji jaką wykorzystałem dla tej warstwy była to funkcja RuLU.

Warstwa wyjściowa każdej z przetestowanych sieci, składała się z funkcji aktywacji softmax, której wyjście traktowałem jako prawdopodobieństwo danego z trzech gestów. Każdy z trzech neuronów warstwy wyjścia oznaczał właśnie to prawdopodobieństwo w następującej kolejności:

1. gest papieru
2. gest kamienia
3. gest nożyc

Budowa sieci neuronowej korzystając z biblioteki Keras:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(25, 60, 3)),
    keras.layers.Dense(256, activation=tf.nn.relu),
    keras.layers.Dense(3, activation=tf.nn.softmax)
])
```

Do uczenia sieci neuronowych wykorzystywałem metodę dostępną w bibliotece Keras GradientDescentOptimizer z parametrem uczenia równym 0.01. Zaś sama sieć została uruchomiana dla 100 epok.

Zanotowałem następujące dokładności rozpoznawania siecią przy wyżej wymienionych parametrach:

Liczba neuronów w warstwie ukrytej sieci	**Dokładność klasyfikacji**
32	98.13%
64	98.46%
128	99.18%
256	99.58%
512	98.88%

Tablica 4.5: Skuteczność sieci neuronowej

Powyższe wartości dokładności rozpoznawania gestów przez sieć neuronową pokazują, że wyśmienicie sobie radzi z klasyfikacją gestów. Najlepsze rezultaty dawała sieć zbudowana z 256 neuronów w warstwie ukrytej. Średnia jej skuteczność to ponad 99% co sprawia, że rozpoznawanie gestów można uznać praktycznie za bezbłędne.

```
855/855 [==============================] - 0s 210us/step - loss: 0.0042 - acc: 1.0000
Epoch 87/100
855/855 [==============================] - 0s 194us/step - loss: 0.0041 - acc: 1.0000
Epoch 88/100
855/855 [==============================] - 0s 212us/step - loss: 0.0041 - acc: 1.0000
Epoch 89/100
855/855 [==============================] - 0s 210us/step - loss: 0.0040 - acc: 1.0000
Epoch 90/100
855/855 [==============================] - 0s 208us/step - loss: 0.0039 - acc: 1.0000
Epoch 91/100
855/855 [==============================] - 0s 202us/step - loss: 0.0038 - acc: 1.0000
Epoch 92/100
855/855 [==============================] - 0s 210us/step - loss: 0.0038 - acc: 1.0000
Epoch 93/100
855/855 [==============================] - 0s 185us/step - loss: 0.0038 - acc: 1.0000
Epoch 94/100
855/855 [==============================] - 0s 212us/step - loss: 0.0037 - acc: 1.0000
Epoch 95/100
855/855 [==============================] - 0s 201us/step - loss: 0.0036 - acc: 1.0000
Epoch 96/100
855/855 [==============================] - 0s 201us/step - loss: 0.0036 - acc: 1.0000
Epoch 97/100
855/855 [==============================] - 0s 220us/step - loss: 0.0035 - acc: 1.0000
Epoch 98/100
855/855 [==============================] - 0s 190us/step - loss: 0.0035 - acc: 1.0000
Epoch 99/100
855/855 [==============================] - 0s 208us/step - loss: 0.0034 - acc: 1.0000
Epoch 100/100
855/855 [==============================] - 0s 193us/step - loss: 0.0034 - acc: 1.0000
195/195 [==============================] - 0s 287us/step
Dokładność: 0.9846153846153847
```

Rysunek 4.18: **Uruchomienie sieci neuronowej**

Rozdział 5

Podsumowanie

W niniejszej pracy inżynierskiej opisano podejście do rozpoznawania gestów na przykładzie gry papier, kamień, nożyce. Wykorzystując bibliotekę OpenCV przedstawiono w jaki sposób można wysegmentować dłoń. Techniki opisane w pracy według mojej oceny dały bardzo dobre rezultaty. Segmentacja gestu pozwoliła na wyraźne wyłuskanie dłoni wykonującej gest.

Segmentacja była pierwszym etapem przy procesie rozpoznawania gestów. Po nim zaś nastąpił etap bezpośredniej klasyfikacji dłoni. W pierwszym sposobie podejścia do klasyfikowania był to klasyfikator KNN. Klasyfikator ten dla odpowiednich parametrów osiągnął średnią skuteczność 86.08%, zakładając użycie trzech cech opisujących wysegmentowaną dłoń.

Drugim zaś podejściem do klasyfikacji był klasyfikator oparty o sieci neuronowe, który również korzysta z wyekstrahowanych wcześniej cech. Przy jego wykorzystaniu skuteczność była bardzo zbliżona do klasyfikatora KNN.

Ostatnim wykorzystanym klasyfikatorem był klasyfikator również oparty o sieci neuronowe, jednakże tym razem jako element wejścia był podawany cały obraz binarny, bez jakikolwiek opisujących go cech. Tak stworzony klasyfikator zdecydowanie przewyższył klasyfikatory wykorzystujące wcześniej dobrane cechy. Rozpoznawanie takie było praktycznie bezbłędne na zgromadzonym zestawie danych, ponieważ klasyfikator przy wykorzystaniu uzyskanych najlepszych parametrów osiągał skuteczność powyżej 99%, co w moim przeświadczeniu jest wyśmienitym rezultatem, biorąc pod uwagę cały proces rozpoznawania gestów.

W mojej ocenie cel jaki na początku swojej pracy założyłem, czyli prawidłowe rozpoznawanie trzech gestów papieru, kamienia oraz nożyc, został osiągnięty z sukcesem. Klasyfikator oparty o sieci neuronowe pokazał jak wielkie możliwości daje wykorzystanie sztucznych sieci neuronowych, które coraz częściej są wykorzystywane w naszym codziennym życiu.

Jak można rozbudować tą pracę inżynierską:

1. Zgromadzenie większego zbioru danych
2. Wyekstrahowanie większej ilości cech dla klasyfikatora KNN
3. Porównanie klasyfikatora KNN z innymi klasyfikatorami opartymi o cechy opisujące gest
4. Wykorzystanie sieci CNN
5. Klasyfikacja dłoni bez użycia rękawiczki
6. Zaimplementowanie gry dla wielu graczy wykorzystując stworzone klasyfikatory
7. Dodanie do klasyfikacji kolejnych gestów

Bibliografia

[1] Andy Collins, „Mowa ciała", 2002

[2] Paweł Kostkiewicz, Michał Syfert, „Interakcja człowiek-komputer", 2010

[3] Kinect SDK – Wprowadzenie, https://msdn.microsoft.com/pl-pl/library/kinect-sdk–wprowadzenie.aspx, dostęp: styczeń 2019

[4] Len Fisher, „Rock, Paper, Scissors: Game Theory in Everyday Life", 2008

[5] Marcin Iwanowski, „Metody morfologiczne w przetwarzaniu obrazów cyfrowych", 2009

[6] Witold Malina, Maciej Smiatacz, „Cyfrowe przetwarzanie obrazów", 2008

[7] Background Subtraction, https://docs.opencv.org/3.4/db/d5c/tutorial_py_bg_subtraction.html, dostęp: styczeń 2019

[8] BackgroundSubtractorKNN, https://docs.opencv.org/3.4.1/db/d88/classcv_1_1BackgroundSubtractorKNN.html, dostęp: styczeń 2019

[9] Jankowski Michał, „Elementy Grafiki Komputerowej", 2006

[10] RGB to HSV color conversion, https://www.rapidtables.com/convert/color/rgb-to-hsv.html, dostęp: styczeń 2019

[11] Witold Malina, Maciej Smiatacz, „Rozpoznawanie obrazów", 2010

[12] Metoda K Najbliższych Sąsiadów, http://home.agh.edu.pl/ horzyk/lectures/miw/KNN.pdf, dostęp: styczeń 2019

[13] Osowski Stanisław, „Metody i narzędzia eksploracji danych", 2013

[14] Ewaryst Rafajłowicz, Wojciech Rafajłowicz, Andrzej Rusiecki, „Algorytmy przetwarzania obrazów i wstęp do pracy z biblioteką OpenCV", 2009

[15] Daniel Slater, Gianmario Spacagna, Peter Roelants, Valentino Zocca, „Deep Learning. Uczenie głębokie z językiem Python. Sztuczna inteligencja i sieci neuronowe", 2018

[16] Douglas McIlwraith, Haralambos Marmanis, Dmitry Babenko, „Algorithms of the Intelligent Web 2nd Edition", 2016

[17] François Chollet, „Deep Learning with Python", 2017

[18] Sebastian Raschka, „Python. Uczenie maszynowe", 2017

[19] Aurélien Géron, Krzysztof Sawka[tłum], „Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow", 2018

[20] Bharath Ramsundar, Reza Bosagh Zadeh, „TensorFlow for Deep Learning. From Linear Regression to Reinforcement Learning", 2018

[21] Jean Serra, „Image Analysis and Mathematical Morphology", 1983

[22] Wilhelm Burger, Mark J. Burge, „Principles of Digital Image Processing: Fundamental Techniques", 2009

[23] O'Reilly Media, „Learning Python 4th Edition", 2009

[24] Hetland Magnus Lie, „Beginning Python", 2016

[25] Tiobem, https://www.tiobe.com/tiobe-index/, dostęp: grudzień 2018

[26] Analizaobrazu, http://analizaobrazu.x25.pl/articles/19, dostęp: styczeń 2019

[27] OpenCV docs, https://docs.opencv.org/3.4/d1/d32/tutorial_py_contour_properties.html dostęp: styczeń 2019

[28] mimuw.edu.pl http://mst.mimuw.edu.pl/lecture.php?lecture=syd&part=Ch2, dostęp: styczeń 2019

[29] OpenCV docs, https://opencv.org/about.html, dostęp: styczeń 2019

[30] TensorFlow.org https://www.tensorflow.org/guide/graphs, dostęp: styczeń 2019

[31] Keras.io, https://keras.io/, dostęp: styczeń 2019

[32] Learningopencv, https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/, dostęp: styczeń 2019

[33] Pyimagesearch, https://www.pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/, dostęp: styczeń 2019

[34] OpenCV docs, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html, dostęp: styczeń 2019

[35] OpenCV docs, https://docs.opencv.org/3.4.2/df/d9d/tutorial_py_colorspaces.html, dostęp: styczeń 2019

[36] Mechanizm uwagi w sieciach neuronowych, http://moria.umcs.lublin.pl/sphinx/src/machine_learning/attention.html, dostęp: styczeń 2019