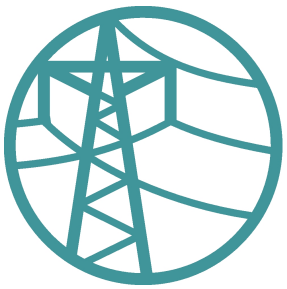


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Katedra Informatyki i Automatyki

Bartosz Gąsior

Zastosowanie metod uczenia maszynowego do rozpoznawania pozy postaci ludzkiej

Praca dyplomowa inżynierska

Opiekun pracy:
Dr inż. Tomasz Krzeszowski

Rzeszów, 2017

Spis treści

1. WSTĘP

Wśród wielu obszarów, którymi zajmuje się współczesna informatyka, rozpoznawanie obiektów na obrazach jest sektorem stale i intensywnie rozwijającym się oraz zyskującym coraz większą popularność. Wykorzystując komputerowe przetwarzanie obrazów możliwe jest pozyskanie informacji, których człowiek sam może nie wychwycić, dlatego coraz więcej dziedzin znajduje zastosowanie dla tego typu operacji. W medycynie rozpoznawanie obrazów służy głównie do rozpoznawania zdjęć medycznych. Dzięki temu możliwe jest wykrycie różnego rodzaju zmian w strukturze czy budowie badanych narządów i późniejsza szybka diagnoza. Wizje komputerową można również wykorzystywać do przewidywania wydarzeń, które mają wpływ na bezpieczeństwo. W miejscach publicznych zainstalowana jest niezliczona ilość kamer, a obraz możliwy jest do uzyskania z praktycznie każdego miejsca na ziemi. Jeszcze innym zastosowaniem, całkowicie różniącym się od wspomnianych wcześniej jest rozrywka, w której rozpoznawanie ruchów i zachowań przez konsole komputerowe lub automaty do gier umożliwia jeszcze większe uczucie realizmu w trakcie rozgrywki.

Rozpoznawanie obiektów i zachowania postaci jest jednak bardzo skomplikowanym i podatnym na błędy zadaniem. Możliwe jest, że jeden niewielki szczegół jak np. inny sposób ułożenia ciała w trakcie wykonywania czynności całkowicie zmieni jej odbiór. Kluczowa też może być jakość zdjęcia czy filmu wideo lub nawet oświetlenie otoczenia. Jak widać znalezienie odpowiedniej metody, która umożliwiała by klasyfikacje obrazów ze stuprocentową dokładnością jest niemal niemożliwe biorąc pod uwagę jak wiele czynników wpływa na to jak obraz jest traktowany przez komputery. Możliwe jest jednak wykonanie tego zadania z pewnym błędem. Tworząc system rozpoznawania należy zadbać o to, żeby błąd ten był jak najmniejszy dzięki czemu jest on bardziej wydajny i skuteczny.

Celem pracy jest stworzenie aplikacji służącej do identyfikacji pozy postaci ludzkiej, z zastosowaniem wybranych klasyfikatorów i metod opisu obiektów na obrazie oraz umożliwiającej późniejsze testy skuteczności przeprowadzonych operacji. W pracy do rozpoznawania pozy postaci zostały wykorzystane trzy klasyfikatory takie jak maszyna wektorów nośnych, k-Najbliższych Sąsiadów oraz perceptron wielowarstwowy, które zostały zaimplementowane oraz udostępnione przez bibliotekę OpenCV. Aby możliwe było wykorzystanie wspomnianych klasyfikatorów obiekty znajdujące się na

obrazach należy przedstawić w postaci liczbowej. Algorytmy służące do uzyskiwania tych wartości można podzielić na te, które charakteryzują się długim czasem obliczeń, ale wysoką dokładnością oraz takie, dla których czas działania będzie krótszy kosztem dokładności opisu obiektu na obrazie. Jako deskryptory obiektów zastosowane zostały współczynniki kształtu, shape context oraz histogram zorientowanych gradientów.

Praca złożona jest z sześciu rozdziałów, które opisują wykorzystane metody, algorytmy i fragmenty kodu. Rozdział pierwszy oraz drugi przedstawiają istotę pracy oraz pozwalają na zapoznanie się z teorią wykorzystaną w kolejnych rozdziałach. W rozdziale trzecim zawarty jest opis wykorzystanych metod służących do klasyfikacji oraz opisu obiektów znalezionych na obrazie. Rozdział czwarty prezentuje funkcje udostępniane przez bibliotekę `OpenCV`, które zostały wykorzystane w trakcie tworzenia pracy. W rozdziale piątym przedstawione zostały funkcje zaimplementowane na potrzeby zwiększenia skuteczności i szybkości działania programu. W rozdziale tym opisano także sposób implementacji deskryptorów obiektów obrazów oraz zaprezentowano działanie przygotowanego programu. Rozdział szósty opisuje wykorzystane bazy danych oraz przedstawia uzyskane wyniki klasyfikacji.

2. CYFROWE PRZETWARZANIE OBRAZÓW

Wszystkie obrazy można przedstawić w postaci matematycznej, jako dwuwymiarową funkcję $f(x,y)$, gdzie x oraz y są współrzędnymi wybranego punktu. Wartość funkcji dla każdej pary współrzędnych (x,y) nazywana jest stopniem natężenia lub jasnością. W przypadku, gdy wartości oraz argumenty funkcji przyjmują wartości dyskretne taki obraz można zakwalifikować jako obraz cyfrowy. Każdy z obrazów cyfrowych zbudowany jest ze skończonej ilości elementów, z których każdy znajduje się na określonej pozycji, taki element nazywany jest pikselem. Piksel jest najmniejszą, jednokolorową niepodzielną częścią obrazu [1].

W pracy wykorzystane zostały trzy formaty obrazów cyfrowych:

- **kolorowe** (zob. rys. 2.1a) - każdy z pikseli przechowuje trzy wartości odpowiadające odpowiednio czerwonemu, zielonemu oraz niebieskiemu natężeniu koloru punktu obrazu. Zakładając, że każdy z trzech kolorów może reprezentować 256 rodzajów natężenia, jeden piksel przechowywany jest w trzech bajtach pamięci, dzięki czemu możliwych do utworzenia jest około 16.7 mln kolorów.
- **w odcieniach szarości (grayscale)** (zob. rys. 2.1b) - piksele przechowują wartość poziomu szarości w danym punkcie. Standardowo przyjmuje się 256 odcieni szarości, dzięki czemu każdy z pikseli może być przechowywany jako pojedynczy bajt pamięci.
- **binarne** (zob. rys. 2.1c) - do reprezentacji pikseli wykorzystywany jest tylko jeden bit, który może przyjmować dwie wartości 1 lub 0. Przez co obraz jest czarno-biały.

Rys. 2.1. (a) Obraz kolorowy, (b) w odcieniach szarości, (c) binarny

Jedną z najważniejszych cech obrazu jest jego rozdzielczość. Rozdzielczość określa liczbę pikseli na obrazie, im jest ona wyższa tym obraz zawiera więcej informacji. W przypadku, gdy zmniejszona zostanie rozdzielczość obrazu przy jednoczesnym zachowaniu jego rozmiaru staje się on mniej ostry. Dla obrazów o dużej rozdzielczości możliwe jest stworzenie obrazu o większym rozmiarze z taką samą ilością detali.

Zakładając, że obraz cyfrowy przedstawiamy w postaci funkcji matematycznej możliwe jest wykonywanie na nim przekształceń. Operacje te pozwalają uzyskać z obrazu pożądany element bądź przekształcić go w wybrany sposób. Do najistotniejszych operacji zaliczane są:

- **binaryzacja** - proces konwertujący obraz kolorowy lub w odcieniach szarości do obrazu binarnego. Najczęściej stosowaną metodą binaryzacji jest progowanie polegające na wybraniu wartości, do której porównywane są piksele. Jeśli wartość danego piksela nie przekracza wybranej wartości progowej klasyfikowany jest on jako piksel obiektu, w przeciwnym przypadku klasyfikowany jest jako piksel tła. W wyniku wykonanej binaryzacji ilość zawartych w obrazie informacji diametralnie się zmniejsza.
- **segmentacja** - segmentacja dzieli obraz na obszary, które odpowiadają poszczególnym obiektom lub ich częściom. Każdy z pikseli obrazu jest przypisywany do jednego z takich obszarów. Obraz postrzegany jest jako zbiór obiektów [2].
- **erozja** - jedna z podstawowych operacji morfologii matematycznej, polegająca na sprawdzaniu otoczenia każdego z punktów obrazu. Dla obrazów kolorowych lub w odcieniach szarości sprawdzany punkt przyjmuje minimalną wartość otaczających go pikseli. W przypadku obrazu binarnego, jeśli jeden z pikseli sąsiedztwa przyjmuje wartość równą *0*, sprawdzany punkt również przyjmuje taką wartość. W innym przypadku wartość wybranego piksela pozostaje bez zmian.
- **dylatacja** - jest operacją przeciwną do erozji, powoduje zwiększenie powierzchni obiektów. Wybrany punkt przyjmuje maksymalną wartość z otaczających go pikseli. W obrazach binarnych w przypadku, gdy choć jeden piksel z obszaru sąsiedztwa przyjmuje wartość *1* to wybrany punkt również przyjmuje taką wartość [2].
- **opening** - polega na wykonaniu na obrazie operacji erozji, a następnie dylatacji. W wyniku tego przekształcenia z obrazu usuwane są niewielkie obiekty, a krawędzie zostają wygładzone. W wyniku openingu rozmiar dużych obiektów pozostaje bez zmian [19].

- **closing** - różni się od openingu kolejnością wykonania operacji erozji i dylatacji. W tym przypadku na obrazie w pierwszej kolejności wykonywana jest dylatacja, a później erozja. Metoda ta ma na celu wypełnienie niewielkich obszarów wewnątrz obiektów [20].

3. METODY ANALIZY OBRAZÓW

W celu dokonania analizy obrazu niezbędne jest przeprowadzenie na nim operacji mających za zadanie wydobycie z niego pożądanych informacji. W przeciwieństwie do przetwarzania obrazów wynikiem analizy są dane w postaci liczbowej, a nie przekształcony obraz. Dla zdjęć o dużej rozdzielczości lub dużym rozmiarze proces ten może być bardzo złożony i długotrwały. Po uzyskaniu danych liczbowych opisujących wszystkie obiekty należące do klas uczących możliwa jest późniejsza klasyfikacja.

W celu klasyfikacji obrazu konieczna jest jego wcześniejsza obróbka. Najczęściej proces ten przebiega w następującej kolejności [3]:

- wstępne przetwarzanie;
- wyodrębnienie cech;
- klasyfikacja.

Logiczne wydaje się więc, że obrazy należące do tej samej klasy powinny charakteryzować się podobnymi zestawami cech, natomiast wyniki analizy obrazów różnych klas powinny być możliwe do odróżnienia.

3.1. Metody klasyfikacji

Klasyfikacja jest to proces polegający na identyfikacji obiektów na podstawie zestawu jego cech. Aby klasyfikacja była możliwa konieczne jest podzielenie zbioru danych na podzbiory, które charakteryzują się wspólnymi cechami, tzw. klasy danych treningowych. Wykorzystując dane treningowe budowany jest klasyfikator wykorzystujący jeden z wybranych algorytmów, który służy do ustalenia przynależności badanego obiektu do jednej z klas [5]. Klasyfikacja składa się więc z dwóch etapów:

- **trening** - zbudowanie klasyfikatora w oparciu o zbiór danych uczących;
- **testowanie** - wykorzystanie zbudowanego wcześniej klasyfikatora do przyporządkowania sprawdzanego obiektu do jednej z istniejących klas.

Klasyfikator na podstawie zbioru obiektów oraz odpowiadającym im kategorii tworzy zestaw zasad wykorzystywanych później do przewidzenia klas testowanych próbek. Problem klasyfikacji sprowadza się więc do uzyskania wektora cech x z danego obiektu i przydzielenia go do dokładnie jednej z klas [4].

3.1.1. Maszyna wektorów nośnych

Maszyna wektorów nośnych (*ang. support vector machine, SVM*) [13] jest jednym z binarnych modeli wykorzystujących uczenie nadzorowane. Zakłada podział przestrzeni dwóch klas (-1;+1) na osobne, liniowo separowalne obszary przy użyciu hiperpłaszczyzny (zob. rys. 3.1) $g(x)$ określonej wzorem:

$$g(x) = w^T x + w_0 = 0, \tag{3.1}$$

gdzie wektor w jest wektorem normalnym do hiperpłaszczyzny.

$$d = \frac{|w_0|}{||w||} \tag{3.2}$$

Wzór 3.2 przedstawia odległość hiperpłaszczyzny od początku układu współrzędnych [5].

Odległość między hiperpłaszczyzną, a najbliższym punktem nazywana jest marginesem p. Algorytm znajduje najbliższe punkty z każdej klasy i wyznacza hiperpłaszczyznę maksymalną, dla której marginesy są jak największe. Marginesy można uzyskać poprzez tworzenie prostych równoległych do hiperpłaszczyzny optymalnej.

W wyniku ustalenia równania hiperpłaszczyzny możemy stwierdzić, do której z klas należy badany obiekt. Obiekt należy do klasy -1, w przypadku gdy:

$$g(x_i) < 0, \tag{3.3}$$

natomiast do klasy 1, gdy:

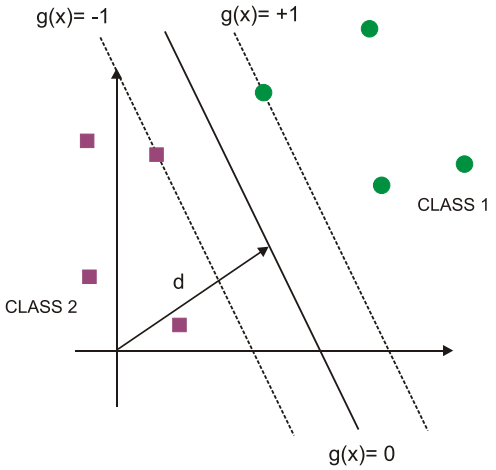
$$g(x_i) > 0. \tag{3.4}$$

Klasyfikator SVM można również wykorzystać do klasyfikacji wieloklasowej. Do osiągnięcia tego celu najczęściej wykorzystywana jest jedna z dwóch metod:

- **jeden przeciw wszystkim** (ang. *one-against-all);*
- **jeden przeciw jednemu** (ang. *one-against-one).*

W pierwszym przypadku tworzonych jest k modeli SVM, gdzie k odpowiada liczbie klas. Do elementów i-tej klasy przypisywane są etykiety z wartością 1 (próbki pozytywne), a do pozostałych obiektów bazy danych etykiety -1 (próbki negatywne). Po nauczeniu wszystkich klasyfikatorów elementy budujące zbiór danych testowych są klasyfikowane. W idealnym przypadku każda z próbek zostanie przydzielona do właściwej klasy przez dokładnie jeden klasyfikator.

Druga metoda polega na zbudowaniu $k(k-1)/2$ klasyfikatorów, z których każdy tworzony jest przez unikalną kombinację elementów dwóch różnych klas. Do sklasyfikowania próbki najczęściej stosowana jest metoda przez głosowanie. W wyniku przydzielenia próbki do jednej z dwóch klas, do otrzymanej klasy dodawany jest "głos". Ostatecznie obiekt jest przypisywany do tej klasy, która uzyskała największą liczbę głosów [24].



Rys. 3.1. Maszyna wektorów nośnych [18]

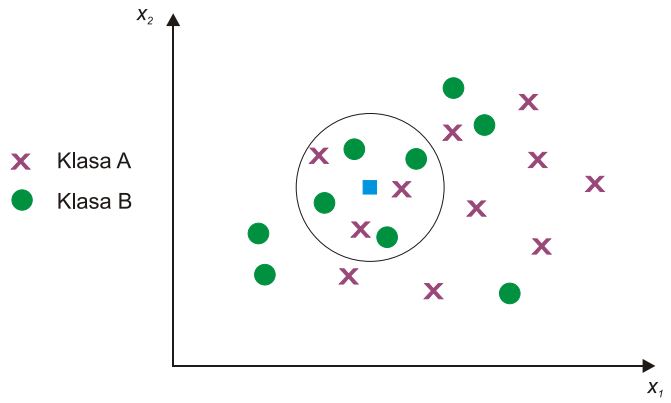
3.1.2. K-Najbliższych Sąsiadów

Klasyfikator k-Najbliższych Sąsiadów (ang. *k-Nearest Neigbours, k-NN*) [14] jest jednym z tych algorytmów, które są łatwe do zrozumienia, a do tego są bardzo skuteczne. Zaliczany jest do algorytmów leniwych (ang. *lazy algorithms*), ponieważ nie wykonuje operacji na danych wejściowych. Metoda opiera się na rozpoznaniu klasy obiektu na podstawie obserwacji jego najbliższego otoczenia [6]. Najważniejszym parametrem k-NN jest liczba sąsiadów brana pod uwagę w czasie klasyfikacji. Następnie w sąsiedztwie zliczane są wystąpienia obiektów z bazy danych treningowych. Obiekt jest przyporządkowywany do tej klasy, której liczba wystąpień była największa.

Na rysunku 3.2 zaprezentowano przykładowe działanie klasyfikatora. Dla sprawdzanego obiektu oznaczonego jako niebieski kwadrat, znajdującego się w centrum okręgu, znaleziono 7 sąsiadów. W sąsiedztwie badanej próbki znajdują się 3 obiekty klasy A oraz 4 obiekty klasy B, w wyniku czego obiekt zostanie rozpoznany jako obiekt klasy B.

Specjalnym przypadkiem klasyfikatora k-NN jest klasyfikator najbliższego są-

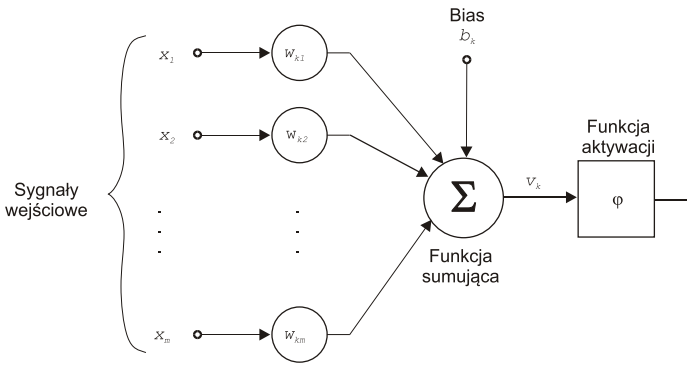
siada (*ang. nearest neighbour, NN*), w którym brana pod uwagę jest tylko klasa najbliższego sąsiada badanej próbki (*k=1*).



Rys. 3.2. K-Najbliższych Sąsiadów

3.1.3. Perceptron wielowarstwowy

Perceptron wielowarstwowy (*ang. multi-layer perceptron, MLP)* [12] jest jednym z najpopularniejszych przykładów sieci neuronowych *(ang. neural network)*, jest to kolejny przykład klasyfikatora wykorzystującego uczenie nadzorowane. Zasada działania sieci neuronowej wzorowana jest na pracy ludzkiego systemu nerwowego. Elementem odpowiedzialnym za przetwarzanie pobranych informacji jest neuron (zob. rys. 3.3).



Rys. 3.3. Model neuronu [7]

Każdy z neuronów zbudowany jest z [8]:

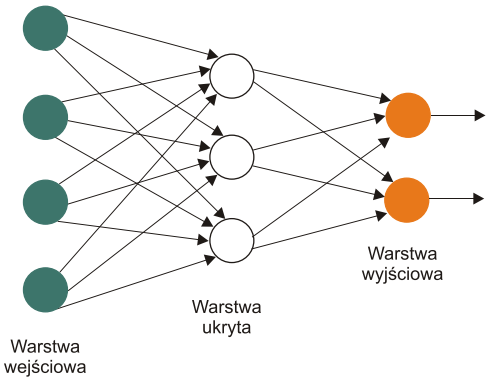
- **sygnałów wejściowych** x_j;

- zbioru **synaps**, czyli połączeń neuronów, z których każda ma przypisaną wagę w_{kj}. Sygnał wejściowy x_j na wejściu synapsy j połączony z neuronem k jest mnożony z wagą synapsy;
- **sumatora** zliczającego sygnały wejściowe oraz wagi odpowiadających synaps: $u_k = \sum_{j=1}^{m} w_{kj} x_j$;
- **funkcji aktywacji** (φ), odpowiedzialnej za ograniczanie wartości wyjścia neuronu. Najczęściej wartość wyjścia y_k neuronu ogranicza się do przedziału [0,1];
- opcjonalnie **biasu** b_k, który jest zewnętrznym sygnałem zwiększającym lub obniżającym wejście funkcji aktywacyjnej.

Ostatecznie wartość na wyjściu neuronu można opisać wzorem:

$$y_k = \varphi(u_k + b_k) \tag{3.5}$$

Rysunek 3.4 przedstawia schemat budowy perceptronu wielowarstwowego zawierającego jedną warstwę ukrytą, warstwę wejściową oraz wyjściową. Każdy neuron w dowolnej warstwie jest połączony do wszystkich neuronów lub węzłów w warstwie poprzedniej, a sygnał przechodzi przez sieć od strony lewej do prawej warstwa po warstwie [7]. Wagi neuronów w czasie tworzenia sieci ustalane są losowo, następnie dobierane są tak aby błąd neuronów warstwy wyjściowej był jak najmniejszy. Ilość neuronów w warstwie wyjściowej najczęściej odpowiada liczbie klas danych wejściowych.



Rys. 3.4. Model sieci neuronowej [8]

Podstawowym algorytmem służącym do uczenia jednokierunkowych sieci neuronowych jest algorytm **wstecznej propagacji błędu** (*ang. backpropagation*). Opiera się on na zmianie wartości wag synaps w celu minimalizacji funkcji błędu uczenia.

3.1.4. Walidacja krzyżowa

Walidacja krzyżowa (sprawdzian krzyżowy, *ang. Cross-Validation*) [9] jest sposobem selekcji danych. Zakładając, że baza danych zawiera n próbek dzielona ona jest na dwie grupy. Pierwsza z nich zawiera n_c danych używanych do tworzenia modelu (klasyfikatora) zaś druga n_v ($n_v = n - n_c$) danych służących do sprawdzania jego wiarygodności. Bazę danych można podzielić na $\binom{n}{n_c}$ sposobów. Dane wykorzystywane do tworzenia modelu nazywa się **zbiorem treningowym** (**uczącym**) natomiast dane sprawdzające to **zbiór testowy**. Złożoność i czas obliczeń metody zwiększa się wraz ze wzrostem n_v, z tego powodu najmniej złożonym obliczeniowo rodzajem sprawdzianu krzyżowego jest walidacja, gdy $n_v = 1$.

W przypadku korzystania z baz danych składających się z dużej ilości elementów pożądana jest możliwie najbliższa prawdzie wiarygodność klasyfikatorów, dlatego aby wartość ta nie była przypadkowa, przeprowadza się **walidację k-krotną**. W tym celu dane dzieli na k podzbiorów, gdzie jeden z podzbiorów traktowany jest jako zbiór testowy, a pozostałe wspólnie tworzą zbiór uczący. Następnie schemat ten powtarza się dla każdego z podzbiorów. Po wykonaniu wszystkich sprawdzianów krzyżowych otrzymane wyniki są uśredniane.

3.2. Metody opisu obiektów

Aby możliwe było nauczenie klasyfikatora oraz późniejsze przeprowadzenie testów sprawdzających, należy z obrazów wchodzących w skład bazy danych pozyskać informacje w postaci liczbowej. Do uzyskana tych wartości potrzebne jest przeprowadzenie różnego rodzaju operacji. Algorytmy służące do opisu kształtu obiektów można podzielić na te, w których istotna jest szybkość działania oraz te, dla których najważniejsza jest dokładność. Główne deskryptory opisujące obiekty to:

- współczynniki kształtu;
- histogram zorientowanych gradientów;
- shape context.

Kolejne podrozdziały opisują ww. deskryptory.

3.2.1. Współczynniki kształtu

Współczynniki kształtu (*ang. shape coefficients)* są najczęściej wykorzystywaną oraz najbardziej znaną metodą opisu obiektów. Można podzielić je na dwie części.

Współczynniki, które wykorzystywane, są w przypadkach gdy zależy nam na szybkości obliczeń oraz te, które powinny charakteryzować się dużo większą dokładnością opisu obiektów [2].

Do pierwszej grupy należy zaliczyć:

- współczynnik bezwymiarowy

$$R_S = \frac{L^2}{4\pi * S}, \tag{3.6}$$

gdzie:

L - obwód obiektu;

S - pole powierzchni obiektu.

- współczynnik Fereta

$$R_F = \frac{L_h}{L_v}, \tag{3.7}$$

gdzie:

L_h - maksymalna średnica obiektu w poziomie;

L_v - maksymalna średnica obiektu w pionie.

- współczynnik cyrkularności **W1** oraz **W2**

$$W1 = R_{C1} = 2 * \sqrt{\frac{S}{\pi}}, \tag{3.8}$$

$$W2 = R_{C2} = \frac{L}{\pi}. \tag{3.9}$$

- współczynnik Malinowskiej **W3**

$$W3 = R_M = \frac{L}{2 * \sqrt{\pi * S}} - 1. \tag{3.10}$$

W drugiej grupie znaleźć można:

- współczynnik Blaira-Blissa **W4**

$$W4 = R_B = \frac{S}{\sqrt{2\pi * \sum_i r_i^2}}, \tag{3.11}$$

gdzie:

r_i - odległość piksela obiektu od jego środka ciężkości;

i - numer piksela obiektu.

- współczynnik Danielssona **W5**

$$W5 = R_D = \frac{S^3}{(\sum_i l_i)^2}, \tag{3.12}$$

gdzie:

l_i - minimalna odległość piksela obiektu od konturu obiektu.

- współczynnik Harlicka **W6**

$$W6 = R_H = \sqrt{\frac{(\sum_i d_i)^2}{n * \sum_i d_i^2 - 1}}, \tag{3.13}$$

gdzie:

d_i - odległość pikseli konturu obiektu od jego środka ciężkości;

n - liczba pikseli konturu.

- współczynnik Lp1 **W7**

$$W7 = \frac{r_{min}}{R_{max}}, \tag{3.14}$$

gdzie:

r_{min} - minimalna odległość konturu od jego środka ciężkości;

R_{max} - maksymalna odległość konturu od jego środka ciężkości.

- współczynnik Lp2 **W8**

$$W8 = \frac{L_{max}}{L}, \tag{3.15}$$

gdzie:

L_{max} - maksymalny gabaryt obiektu.

- współczynnik Mz **W9**

$$W9 = \frac{2\sqrt{\pi S}}{L}. \tag{3.16}$$

Charakterystyka współczynników kształtu została opracowana na podstawie pozycji [2].

3.2.2. Histogram zorientowanych gradientów

Histogram zorientowanych gradientów (*ang. Histogram of oriented gradients, HOG*) [15] zakłada, że obiekt jest możliwy do opisania za pomocą rozkładu krawędzi pod określonym kątem. Metoda ta opiera się na podzieleniu obrazu na mniejsze, równe części (komórki) i wyszukiwaniu oraz zliczaniu lokalnych, znormalizowanych gradientów w wybranej części obrazu. Dla każdej z komórek obrazu obliczany jest jednowymiarowy histogram występowania orientacji krawędzi lub kierunku gradientów

Rys. 3.5. (a) Obraz testowy, (b) obliczony deskryptor [24]

dla wszystkich pikseli wewnątrz komórki. W celu normalizacji, obraz dzielony jest na bloki, tj. grupy sąsiadujących ze sobą komórek. Zakładając, że komórka ma rozmiar 8x8 pikseli otrzymujemy 64 wektory gradientów. Następnie dla każdej z komórek budowany jest histogram, który jest normalizowany. Połączone histogramy tworzą deskryptor HOG [10].

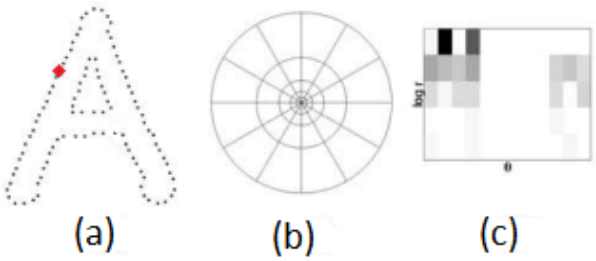
HOG w przeciwieństwie do innych podobnych algorytmów używa jednego ogólnego zestawu cech, a nie zestawu cech lokalnych. Najogólniej można stwierdzić, że jeden obiekt jest reprezentowany przez jeden wektor wartości.

3.2.3. Shape context

Shape context [16] jest deskryptorem wyznaczającym związek między punktami konturu obiektu. Względem każdego z punktów obliczane jest rozmieszczenie pozostałych. Charakterystyki shape context wszystkich punktów dwóch podobnych do siebie obiektów będą do siebie bardzo zbliżone.

Aby otrzymać deskryptor shape context, dla każdego z punktów obiektu należy wyznaczyć zbiór wektorów mających swój początek w wybranym wcześniej punkcie i zakończonych w pozostałych. Wektory te opisują kompozycję całego kształtu względem wybranego punktu. Oczywiście im wielkość, rozdzielczość obrazu oraz rozmiar badanego obiektu będą większe tym złożoność algorytmu będzie rosła, dlatego dla skrócenia działania tworzone są diagramy log-polar[1] (zob. rys. 3.6b). Do utworzenia,

[1] diagram log-polar - diagram przedstawiający położenie punktów względem środka, podzielony na biny odległości oraz kąta.



Rys. 3.6. (a) Wybrane punkty konturu obiektu, (b) diagram log-polar, (c) histogram shape context dla zaznaczonego na czerwono punktu [11]

których potrzebne jest wyznaczenie ilości binów odległości log_r oraz kąta ϕ względem punktu i następne zliczenie punktów leżących w utworzonych obszarach. Na podstawie diagramów log-polar tworzone są histogramy (zob. rys. 3.6c) przedstawiające rozmieszczenie punktów w poszczególnych binach. Dzięki zastosowaniu diagramów log-polar dokładność deskryptora zwiększa się dla punktów leżących w pobliżu badanej próbki [11].

4. OPENCV

OpenCV (*ang. Open Source Computer Vision Library)* jest otwartą, stale rozwijaną biblioteką, posiadającą setki zaimplementowanych algorytmów wizji komputerowej. Do jej głównych cech należy wieloplatformowość (Mac OS X, Windows, Linux, Android) oraz możliwość programowania w językach takich jak C++, Java oraz Python [20]. W celu skutecznego działania programu wykorzystanych zostało kilka wbudowanych funkcji oraz typów, które zostaną opisane w dalszej części rozdziału.

- **Mat** - typ reprezentujący n-wymiarową tablicę. W pracy użyto macierzy dwuwymiarowych, które najczęściej wykorzystywane są do przechowywania obrazów;
- **Point** - typ przechowujący współrzędne punktu;
- **FileStorage** - klasa służąca do zapisywania oraz odczytywania plików w formacie XML/YAML.

Pierwszym krokiem w pracy z deskryptorami obiektów jest wczytanie zdjęcia. Odpowiada za to metoda `imread` zwracająca obiekt typu `Mat`. Wczytanie obrazu z bazy danych testowych przedstawiono na listingu 4.1.

Listing 4.1. Funkcja wczytania obrazu

```
Mat img = imread(daneTestowe[k][i]);
```

Dla zwiększenia szybkości działania programu rozmiar wszystkich obrazów został zmieniony do wartości 128x96. Funkcję `resize` odpowiedzialną za tą operację przedstawia listing 4.2. Jej parametry to odpowiednio: obraz, którego rozmiar będzie modyfikowany, zmienna typu `Mat`, do której przypisany będzie wynik funkcji oraz przyjęty rozmiar wynikowego zdjęcia.

Listing 4.2. Funkcja służąca do zmiany rozmiaru obrazu

```
resize(imgTmp, img, Size(128,96));
```

Kolejna funkcja udostępniona przez OpenCV służy do znalezienia wszystkich konturów na badanym obrazie (zob. listing 4.3). Funkcja pobiera wczytany obraz, a następnie, pozyskuje punkty konturów i zapisuje je do dwuwymiarowego wektora punktów. Wywołując funkcję należy również zdefiniować sposób aproksymacji zarysu

obiektu. Listing 4.3 przedstawia użycie algorytmu `CHAIN_APPROX_NONE`, który zapisuje wszystkie wykryte punkty.

Listing 4.3. Znajdowanie konturów na obrazie binarnym

```
vector<vector<Point>> contours;
findContours(img.clone(), contours, CV_RETR_CCOMP,
   CV_CHAIN_APPROX_NONE);
```

Istnieje możliwość, że zdjęcia budujące bazę danych zawierają niewielkie zakłócenia, które sprawiają, że obiekt jest nieczytelny. W celu usunięcia wspomnianych zakłóceń, przed znalezieniem konturów, należy na obrazach przeprowadzić operacje openingu i closingu. Odpowiedzialna za to jest funkcja `morphologyEx` (zob. listing 4.4). Przy jej wywołaniu należy podać zdjęcie źródłowe, macierz do której zapisany będzie wynik operacji, typ operacji morfologicznej, element strukturalny oraz należący do niego punkt, dla którego przeprowadzana będzie operacja (przy podaniu wartości `Point(-1,-1)` punkt znajdować się będzie w środku elementu strukturalnego).

Funkcja `getSructuringElement` odpowiedzialna jest za stworzenie elementu strukturalnego o określonym rozmiarze oraz kształcie.

Listing 4.4. Funkcje openingu oraz closingu

```
/*element strukturalny*/
Mat element = getStructuringElement(MORPH_RECT, Size(3,3),
   Point(-1,-1));
/* closing */
morphologyEx(img, dst, MORPH_CLOSE, element, Point(-1, -1));
/* opening */
morphologyEx(img, dst, MORPH_OPEN, element, Point(-1, -1));
```

Obliczenie wartości współczynników kształtu wymaga takich wartości jak obwód obiektu, jego pole czy środek ciężkości. Do uzyskania wartości obwodu oraz pola konieczna jest tablica z punktami wcześniej obliczonego konturu (zob. listing 4.5). Funkcja `arclength` (obliczająca obwód) wymaga również stwierdzenia czy krzywa opisująca obiekt jest zamknięta czy otwarta.

W celu wyznaczenia współrzędnych środka ciężkości konieczne jest obliczenie momentów geometrycznych obiektu. Służy do tego funkcja `moments`. Użycie tej funkcji oraz wyznaczenie środka ciężkości przedstawia listing 4.6.

Listing 4.5. Obliczenie obwodu oraz pola

```
vector<Point> contourTmp;
vector<vector<Point>> contours;
//funkcja obliczająca obwód
L = arcLength(contours[0], true);
//funkcja obliczająca pole
S = abs(contourArea(contours[0], true));
```

Listing 4.6. Wyznaczenie momentów geometrycznych oraz obliczenie środka ciężkości

```
vector<Moments> mu(contours.size());
for (int i = 0; i < contours.size(); i++) {
   //momenty geometryczne
   mu[i] = moments(contours[i], false);
}

vector<Point2f> mc(contours.size());
for (int i = 0; i < contours.size(); i++) {
   //środek ciężkości obiektu
   mc[i] = Point2f(mu[i].m10 / mu[i].m00,
      mu[i].m01 / mu[i].m00);
}
```

Do obliczenia histogramu zorientowanych gradientów również wykorzystano funkcje zaimplementowane w bibliotece OpenCV. Przed obliczeniem HOG należało wykonać kilka kroków:

- wybrać z obrazu największy obiekt;
- wyznaczyć dla niego współrzędne punktów konturu;
- wybrany obiekt wypełnić kolorem.

Wykorzystano do tego funkcję `drawContours`. Aby uzyskać zdjęcie z pożądanym obiektem, krzywą wyznaczoną przez współrzędne punktów konturu należy wypełnić

kolorem. W tym celu jako jeden z argumentów funkcji należy podać wartość `CV_FILLED` (zob. listing 4.7).

Listing 4.7. Uzyskanie wymaganego obiektu obrazu

```
vector<vector<Point>> contours;

//wypełnienie macierzy o rozmiarze badanego zdjęcia zerami
Mat mask = Mat::zeros(img.size(), CV_8UC1);
drawContours(mask, contours, 0, Scalar(255), CV_FILLED);
```

Mając obraz z wymaganym obiektem można przejść do obliczania histogramu zorientowanych gradientów (zob. listing 4.8). Pierwszym krokiem jest stworzenie deskryptora, w tym celu należy ustalić:

- rozmiar okna, na którym szukany jest obiekt;
- rozmiar bloku;
- wartość o jaką przesuwany jest blok;
- rozmiar komórki (zob. podrozdział 3.2.2);
- liczbę klastrów tworzonych histogramów.

Listing 4.8. Parametry HOG

```
HOGDescriptor d(imgTmp.size(), Size(32, 32), Size(16, 16),
   Size(16, 16), 9);
```

Mając stworzony deskryptor oraz znając jego rozmiar możliwe jest obliczenie samych współczynników. Odpowiedzialna jest za to funkcja `compute` (zob. listing 4.9). Do jej wywołania konieczny jest wektor typu `float`, w którym przechowywane są obliczone wartości.

Listing 4.9. Obliczenie wartości współczynników HOG

```
vector< float> descriptorsValues;
vector< Point> locations;
d.compute(imgTmp, descriptorsValues, Size(8, 8),
   Size(0, 0), locations);
```

W bibliotece OpenCV zaimplementowano również funkcje służące do klasyfikacji. Użyte w pracy klasyfikatory udostępniane przez OpenCV to: SVM, k-NN oraz perceptron wielowarstwowy (MLP).

Maszyna wektorów nośnych

W celu utworzenia klasyfikatora SVM należało zdefiniować:

- typ klasyfikatora, `C_SVC` odpowiada za klasyfikację n-klasową (n≥2);
- typ jądra;
- liczbę iteracji algorytmu oraz błąd pomiarów.

Listing 4.10. Ustalenie parametrów SVM

```
Ptr<ml::SVM> svm = ml::SVM::create();
svm->setType(ml::SVM::C_SVC);
svm->setKernel(ml::SVM::LINEAR);
svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER,
   100, 1e-6));
```

W celu nauczenia klasyfikatora wywołano funkcję `train` (zob. listing 4.11), która za argumenty przyjmuje odpowiednio tablicę obliczonych współczynników dla każdego obrazu, sposób odczytu danych z tablicy (`ROW_Sample`, definiujący każdy wiersz jako osobną próbkę) oraz tablicę, w której przechowywane są odpowiadające obrazom klasy. Dodatkowo wykorzystano funkcję `save`, która zapisuje stworzony klasyfikator w wybranym miejscu.

Listing 4.11. Nauczenie SVM

```
/*wektor macierzy, w którym przechowywane są wartości
obliczonych współczynników oraz ich klasy*/
vector<Mat> dataTmp;
svm->train(dataTmp[0], ROW_SAMPLE, dataTmp[1]);
path = path + string("ShapeContextTrainedSVM.xml");
svm->save(path);
```

Wykonanie predykcji klasy testowanego obrazu sprowadza się do wywołania funkcji `predict`, której jedynym argumentem jest wektor deskryptorów próbki (zob. listing 4.12).

Listing 4.12. Klasyfikacja obrazu

```
float result = svm->predict(shape);
```

Perceptron wielowarstwowy

Przed utworzeniem klasyfikatora MLP należy ustalić z ilu warstw ukrytych zbudowany będzie model oraz z ilu neuronów każda z nich będzie się składać. W pracy wykorzystano jedną warstwę, która zawiera 7 neuronów (zob. listing 4.13). Aby ustalić liczbę neuronów przeprowadzono szereg testów (zob. roz. 6). Wielkość warstwy wejściowej zależy od ilości danych treningowych, sam klasyfikator natomiast ma tyle wyjść ile jest określonych klas.

Listing 4.13. Warstwy MLP

```
/*Przykładowe wielkości warstw perceptronu dla 5-klasowej
bazy danych*/
Mat layers = cv::Mat(5, 1, CV_32F);
layers.row(0) = Scalar(dataTmp[0].cols); //warstwa wejściowa
layers.row(1) = Scalar(7); //warstwa ukryta
layers.row(2) = Scalar(5); //warstwa wyjściowa
```

W pracy z perceptronem wielowarstwowym istotne jest aby macierz przechowująca klasy obiektów zapisana była w postaci binarnego wektora o rozmiarze równym ilości klas. Wartość wektora na pozycji odpowiadającej klasy jest równa 1, pozostałe elementy zaś zostają wypełnione zerami (zob. listing 4.15). Następnie należy zdefiniować pozostałe parametry. Funkcja aktywacji, która została użyta to funkcja sigmoid, przyjmująca na wyjściu wartości z zakresu $[0, 1]$, jest to jedyna funkcja aktywacji w pełni wspierana przez OpenCV w wersji 3.1. Jako algorytm uczenia zastosowano algorytm wstecznej propagacji błędu (zob. podrozdział 3.1.3), a w celu zwiększenia skuteczności algorytmu wartości `MomentumScale` oraz `WeightScale` ustalono odpowiednio na 0.0001 oraz 0.03.

Listing 4.14. Parametry MLP

```
vector<Mat> dataTmp;
Ptr< ANN_MLP >  ann = ANN_MLP::create();
ann->setLayerSizes(layers);
```

```
ann->setActivationFunction(cv::ml::ANN_MLP::SIGMOID_SYM,
   1.0, 1.0);
ann->setTrainMethod(cv::ml::ANN_MLP::BACKPROP);
ann->setBackpropMomentumScale(0.0001);
ann->setBackpropWeightScale(0.03);
ann->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER,
   (int)3, 0.00001));
ann->train(dataTmp[0], ROW_SAMPLE, dataTmp[2]);
ann->save(path);
```

Listing 4.15. Obiekt sklasyfikowany do drugiej klasy bazy danych zbudowanej z pieciu klas

```
[0 1 0 0 0]
```

W przypadku perceptronu wielowarstwowego funkcja klasyfikująca dane testowe nie zwraca przewidywanej klasy obiektu lecz wektor zawierający prawdopodobieństwo przynależności obiektu do danej klasy. Aby wybrać pożądaną wartość wystarczy znaleźć indeks elementu wektora o największej wartości (zob. listing 4.16).

Listing 4.16. MLP, funkcja sprawdzająca klasę, do której z największym prawdopodobieństwem należy badany obiekt

```
/* zmienna przechowująca indeks klasy z największym
prawdopodobieństwem */
int cls = -1;
//wektor dla 5-klasowej bazy danych
Mat response = Mat::zeros(1, 5, CV_32F);
annTmp->predict(shape, response);
float max = -FLT_MAX;
   for (int i = 0; i < 5; i++) {
      float value(response.at<float>(0, i));
      if (value > max) {
         max = value;
         cls = i + 1;}}}
```

K-Najbliższych Sąsiadów

Najważniejszym parametrem metody k-NN jest to, ilu sąsiadów badanego obiektu jest branych pod uwagę przy klasyfikacji. Należy tą wartość ustalić przed rozpoczęciem pracy algorytmu. Dla wszystkich wykonanych testów $k = 7$. Typ algorytmu `BRUTE_FORCE` nie wykonuje żadnych przekształceń na zestawie danych treningowych.

Listing 4.17. Parametry k-NN

```
vector<Mat> dataTmp;
cv::Ptr<cv::ml::KNearest> knn(cv::ml::KNearest::create());
Ptr<TrainData> trainingData;
trainingData = TrainData::create(dataTmp[0],
   ml::ROW_SAMPLE, dataTmp[1]);
knn->setAlgorithmType(ml::KNearest::BRUTE_FORCE);
knn->setIsClassifier(1);
knn->setDefaultK(k);
knn->train(trainingData);
```

W celu wykonania klasyfikacji wykorzystana została funkcja `findNearest`, która za argumenty przyjmuje: obliczony wybrany deskryptor badanego obiektu, wartość k, oraz zmienną, do której zapisany zostaje wynik.

Listing 4.18. klasyfikacja k-NN dla deskryptora shape context

```
knn->findNearest(shapeContext(img, angleBins, radiusBins),
   knn->getDefaultK(), result);
```

5. PROJEKT PROGRAMU

W tym rozdziale opisano fragmenty kodu, które stworzono na potrzeby skutecznego i szybkiego działania programu. Część metod korzysta z danych wygenerowanych przy użyciu funkcji opisanych w rozdziale 4.

Listing 4.3 przedstawia sposób wykrywania wszystkich konturów na obrazie. Do działania programu niezbędny jest tylko zarys pozy postaci. Dzięki temu, że zdjęcia posiadają niewielkie zakłócenia przyjąć można, że sylwetka postaci jest największym obiektem na obrazie. Znalezienie pozy postaci sprowadza się więc do znalezienia największego czworokąta, opisanego na obiekcie o największym polu (zob. listing 5.1).

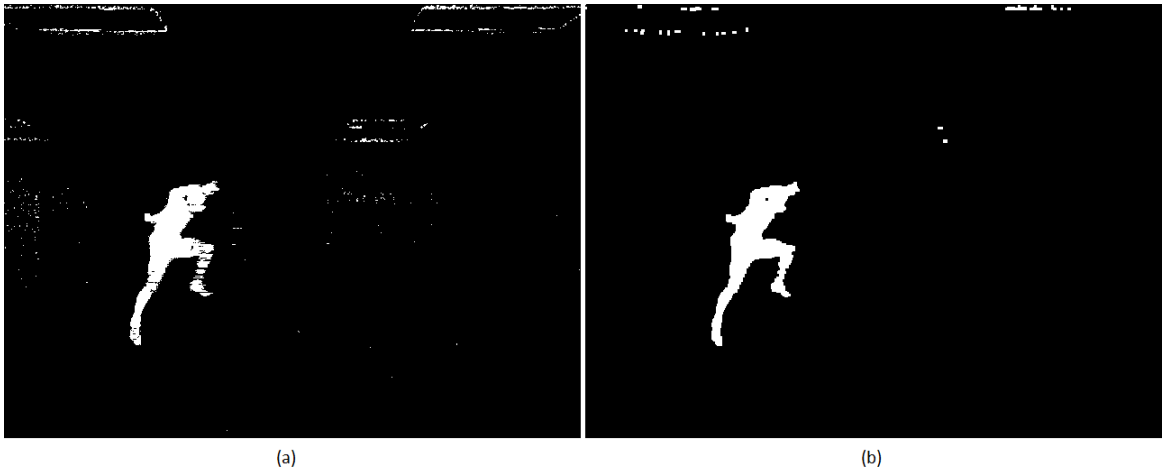
Listing 5.1. Funkcja `detectObject` znajdująca największy obiekt na ekranie

```
/*zmienna przechowująca pole największego obiektu*/
int largest_area = 0;
/*zmienna przechowująca indeks największego obiektu*/
int largest_contour_index = 0;
Rect bounding_rect;
/*contours -  punkty konturów wszystkich obiektów*/
for (int i = 0; i < contours.size(); i++) {
   double a = contourArea(contours[i], false);
   if (a > largest_area) {
      largest_area = a;
      largest_contour_index = i;
      bounding_rect = boundingRect(contours[i]);
      }
   }
```

Efekt działania funkcji przedstawionej na listingu 5.1 pokazuje rysunek 5.1.

Rys. 5.1. (a) Zdjęcie oryginalne wraz z zaznaczonym znalezionym obiektem, (b) wyznaczony kontur obiektu

Przed znalezieniem sylwetki postaci konieczne jednak jest przeprowadzenie na części obrazów operacji morfologicznych takich jak closing oraz opening (zob. listing 4.4). Co ma na celu stworzenie sylwetek, które będą składały się z jednego obiektu. Efekt wykonania wspomnianych operacji przedstawia rysunek 5.2.



Rys. 5.2. (a) Zdjęcie oryginalne, (b) zdjęcie po przeprowadzeniu operacji closingu i openingu

5.1. Implementacja deskryptorów obrazów

Współczynniki kształtu

W celu obliczenia wartości współczynników kształtu, najczęściej wykorzystywane są funkcje do obliczania pola, obwodu oraz momentów geometrycznych, które zostały

opisane w rozdziale 4. Wykorzystanie wymienionych funkcji zostanie zaprezentowane na przykładzie wyznaczenia wartości współczynnika Blaira-Blissa (zob. równanie 3.11), gdzie wymagane jest pole obiektu (zob. listing 4.5) oraz odległość pikseli obiektu od jego środka ciężkości (zob. listing 4.6). Pomocna będzie tu funkcja `pointPolygonTest`, która została zaimplementowana w OpenCV. Sprawdza ona czy wybrany punkt obrazu znajduje się wewnątrz konturu obiektu (zob. listing 5.2).

Listing 5.2. Fragment kodu funkcji obliczającej wartość współczynnika W4

```
/*sprawdzany jest każdy punkt należący do obrazu
zapisywany jest on w zmiennej pomocniczej p*/
Point p;
for (int y = 0; y < img.rows; y++) {
for (int x = 0; x < img.cols; x++) {
   p.x = x;
   p.y = y;
/*warunek sprawdzający czy punkt leży wewnątrz konturu*/
   if (pointPolygonTest(contours[0], p, false) > 0){
      tmpx = (p.x - mc[0].x);
      tmpy = (p.y - mc[0].y);
      tmp = sqrt((tmpx*tmpx) + (tmpy*tmpy));
      suma = suma + (tmp*tmp);}}}
return wsp=(S/(sqrt(2*M_PI*suma)));
```

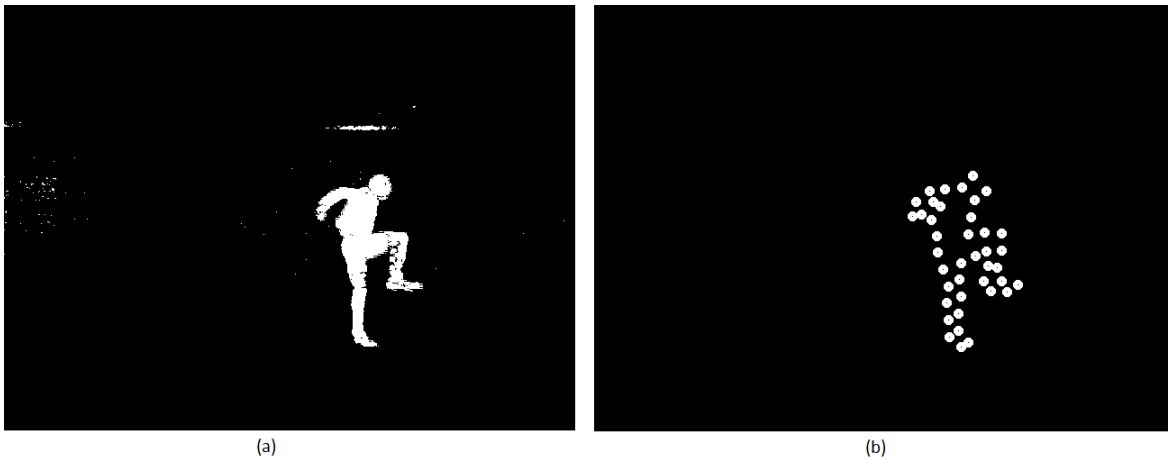
Shape context

Shape context, opisany w podrozdziale 3.2.3, wymaga ustalenia na ile binów podzielony będzie diagram log-polar. W pracy przyjęto, że ilość binów względem kąta wynosi 10, a względem odległości 5. Dodatkowo maksymalna odległość sprawdzanego punktu względem punktu, dla którego liczony jest shape context, wynosi 250, a minimalna odległość, czyli granica pierwszego binu to 10. Wyznaczenie granic binów pokazuje listing 5.3.

Listing 5.3. Obliczenie granic binów

```
double r_bins_edges[5];
double  r_inner = 10;
double  r_outer = 250;
double  nDist=(log10(r_outer)-log10(r_inner))/(rbins - 1);
for (int i = 0; i<rbins; i++)
   r_bins_edges[i] = pow(10, log10(r_inner) + nDist*i);
/* elementy tablicy r_bins_edges:
10, 22.3607, 50, 111.803, 250 */
```

Shape context nie wymaga aby badane były wszystkie punkty konturu dlatego dla każdego obiektu wyznaczono ich 40 (zob. rys. 5.3). Wartość ta została dobrana tak, aby zachować kształt obiektu, a złożoność obliczeniowa programu znacząco się nie zwiększyła. Do wybrania punktów posłużono się funkcją zaprezentowaną na listingu 5.4.



Rys. 5.3. (a) Zdjęcie oryginalne, (b) wyznaczone wybrane punkty

Listing 5.4. Funkcja odpowiedzialna za selekcje 40 punktów konturu

```
/*wektor zawierający wszystkie punkty konturu*/
vector<Point> contourQuery
/*wektor z 40 wybranymi punktami konturu*/
vector<Point> contour;
int n=40;
int tmp;
for (int add = 0; add < n; add++) {
```

```
    tmp = (contourQuery.size() / n) * (add);
    contour.push_back(contourQuery[tmp]);
}
return contour;
```

Kolejnym etapem jest stworzenie macierzy odległości, za co odpowiedzialna jest funkcja zaprezentowana na listingu 5.5. Każdy wiersz macierzy (wektor `tmpOdl`) zawiera odległość punktu (o indeksie równym numerowi wiersza) od pozostałych.

Listing 5.5. Odległości między punktami

```
vector<Point>contours;
vector<vector<double>> odl;
Point tmpPoint;
for (int i = 0; i < contours.size(); i++) {
    tmpPoint = contours[i];
    doubleodlEuklidesowa = 0;
    vector<double> tmpOdl;
    for (int ii = 0; ii < contours.size(); ii++) {
        tmpx = contours[ii].x - tmpPoint.x;
        tmpy = contours[ii].y - tmpPoint.y;
        odlEuklidesowa = sqrt((tmpx*tmpx) + (tmpy*tmpy));
        tmpOdl.push_back(odlEuklidesowa);
    }
    odl.push_back(tmpOdl);
}
```

Podobnie jak odległość między punktami obliczono kąt między nimi. W tym celu posłużono się funkcją `atan2`. Wynik tej funkcji zamieniany jest na stopnie i sprawdzany, w którym binie kąta leży testowany punkt.

Następnie dla elementów macierzy odległości należy sprawdzić, w którym binie odległości znajduje się badany punkt. Służy do tego krótka pętla:

Listing 5.6. Przynależność punktów do binów odległości

```
for (int k = 0; k < rbins; k++) {
    if (odl[i][j] <= r_bins_edges[k]) {
        odl[i][j] = k;
        break;
    }
}
```

Ostatnim krokiem jest zliczenie punktów w poszczególnych binach. Każdy wiersz macierzy `binPoints` zawiera deskryptor shape context danego punktu. Wiadomo, że wszystkich binów jest $5*10=50$, a każdy kontur zbudowany jest z 40 punktów, możliwe jest więc obliczenie rozmiaru wektora opisującego każdy z obrazów. W tym przypadku jest to $50*40=2000$.

Listing 5.7. Zliczanie punktów w binach

```
/*wektor binPoints wypełniony jest zerami*/
vector<vector<float>> binPoints;
for (int i = 0; i < odl.size(); i++) {
    for (int j = 0; j < odl[i].size(); j++) {
        if (i == j)
            continue;
        int index =  odl[i][j] * nbins + katy[i][j];
        binPoints[i].at(index)++;
    }
}
```

5.2. Instrukcja obsługi programu

Program wywoływany jest z linii poleceń. W celu uruchomienia działania programu należy wpisać komendę z poprawnymi argumentami (zob. rys. 5.4), możliwe także jest wywołane menu pomocy (zob. rys. 5.5). W celu uruchomienia programu należy wybrać:

1) Bazę danych
   - db1;

- db2.

2) Deskryptor obrazu
   - wsp - współczynniki kształtu;
   - hog - histogram zorientowanych gradientów;
   - shape context.
3) Ścieżkę zapisu pliku z wynikami.
4) Opcjonalnie, w przypadku wybrania współczynników kształtu, kombinacje współczynników
   - wsp_1 - W1, W3, W5, W6, W7;
   - wsp_2 - W3, W5, W7, W9;
   - wsp_3 - W5, W6.

```
shapeContext1 "db1" "wsp" "wsp_1Results.txt" "wsp_2"
```

Rys. 5.4. Przykładowe wywołanie programu z wybranymi parametrami

```
C:\Users\Bartek\Documents\Visual Studio 2015\Projects\shapeContext1\x64\Debug>shapeCont
ext1 "help"

***** VIDEOINPUT LIBRARY - 0.1995 - TFW07 *****

Do uruchomienia programu potrzebne jest polecenie w postaci "baza_danych" "deskryptor"
"sciezka pliku zapisu".
Przykladowy schemat : shapeContext1 "db1" "hog" "hog.txt".

W celu uruchomienia pomocy nalezy uzyc polecenia "help":
shapeContex1 "help"
```

Rys. 5.5. Wywołanie menu pomocy

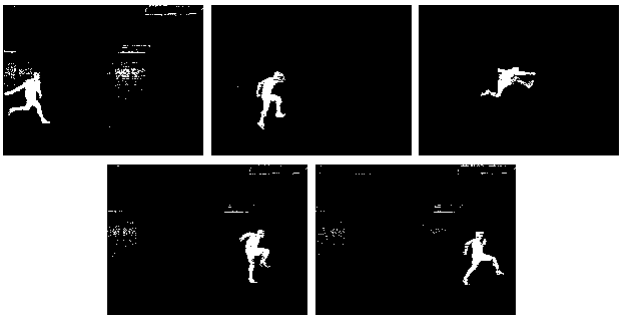
W wyniku działania programu tworzony jest plik o nazwie zdefiniowanej podczas uruchamiania. W pracy zastosowana została walidacja dziesięciokrotna dzięki czemu w każdej z iteracji z klas wydzielane jest kolejnych 10% danych, które zapisywane są jako dane testowe. W pliku zapisywana jest liczba danych testowych, numer kroku, skuteczność klasyfikatorów oraz przynależność obiektów danych testowych do każdej z klas. Dodatkowo na końcu każdego z plików zapisywana jest średnia wartość wyniku klasyfikacji dla każdej klasy (zob. listing 5.8).

Listing 5.8. Fragment utworzonego pliku z wynikami

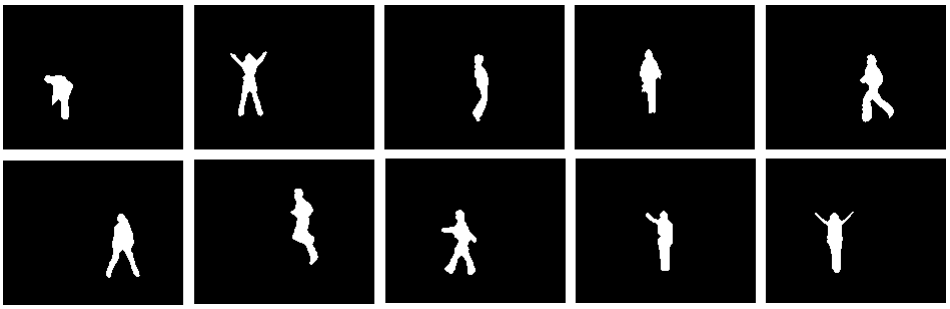
```
numer iteracji - 10
indeks sprawdzanej pozy - 5
liczba danych testowych - 16
k-NN
P1 - 3
P2 - 3
P3 - 0
P4 - 1
P5 - 9
skuteczność k-NN: 0.5625
ANN
P1 - 0
P2 - 2
P3 - 2
P4 - 0
P5 - 12
skuteczność ANN: 0.75
SVM
P1 - 8
P2 - 0
P3 - 1
P4 - 4
P5 - 3
skuteczność SVM: 0.1875


średnia skuteczność klasy  1
SVM - 0.459239
k-NN - 0.641304
ANN - 0.563225
```

6. WYNIKI

Głównym zadaniem przygotowanego programu była klasyfikacja obrazów. Jako dane uczące oraz testowe wykorzystano obrazy należące do baz danych "płotkarze" [22] (zob. rys. 6.1) oraz "Actions as Space-Time Shapes" [23] (zob. rys. 6.2).



Rys. 6.1. Klasy bazy danych "płotkarze"



Rys. 6.2. Klasy bazy danych "Actions as Space-Time Shapes"

W celu zbadania skuteczności działania programu przeprowadzono szereg testów, których wyniki zostały zaprezentowane w tabelach 6.1 oraz 6.3.

Pierwsza z baz danych zbudowana jest z 360 plików, na które składają się obiekty pięciu klas. Każda z klas złożona jest z:

- P1 - 48 plików;
- P2 - 66 plików;
- P3 - 101 plików;
- P4 - 59 plików;
- P5 - 86 plików.

Elementy tej bazy danych wymagały wykonania na nich przekształceń openingu i closingu (zob. rozdział 2) w celu usunięcia zakłóceń z obrazów.

W skład drugiej bazy danych wchodzi 2698 plików, które podzielone zostały na dziesięć klas. Każda z klas zbudowana jest z:

- P1 - 193 plików;
- P2 - 353 plików;
- P3 - 219 plików;
- P4 - 236 plików;
- P5 - 199 plików;
- P6 - 199 plików;
- P7 - 234 plików;
- P8 - 324 plików;
- P9 - 354 plików;
- P10 - 387 plików.

Wykorzystując klasyfikator k-NN, przed rozpoczęciem działania programu, należało ustalić wartość parametru k. Dla przeprowadzonych testów wyniosła ona 7. Perceptron wielowarstwowy z kolei wymagał ustalenia wielkości warstwy ukrytej, której rozmiar w pracy ustalono na 7. W celu wybrania odpowiedniej liczby neuronów w wspomnianej warstwie przeprowadzono szereg testów (zob. wykres 6.3 oraz wykres 6.4), które zostały opisane w dalszej części rozdziału.

W celu jak najbardziej wiarygodnej oceny wyników pomiarów zastosowano walidację dziesięciokrotną.

Wyniki klasyfikacji zaprezentowane w tabeli 6.1 pokazują, że najskuteczniejszym deskryptorem dla pierwszej z baz danych okazał się histogram zorientowanych gradientów, uzyskał on zdecydowanie najlepszą średnią skuteczność dla wszystkich użytych klasyfikatorów. Można również zauważyć, że efektywność współczynników kształtu jest do siebie zbliżona. Dla histogramu zorientowanych gradientów oraz klasyfikatora MLP stworzona została macierz pomyłek (zob. tabela 6.2).

Dużo gorzej od HOG oraz shape context wypadła klasyfikacja oparta o współczynniki kształtu. Żadna z użytych kombinacji nie uzyskała zbliżonych wyników do dwóch pozostałych metod. Spowodowane jest to tym, że współczynniki kształtu sprawdzają się dobrze w przypadku opisu uproszczonych, nie złożonych obliczeniowo obiek-

tów. W klasyfikacji tej wszystkie wyniki są podobne, jednak zauważyć można, że najsłabiej prezentuje się algorytm SVM.

Tabela 6.1. Wyniki dla bazy danych "płotkarze"

klasyfikator / metoda opisu obiektu	k-NN	MLP	SVM	średnia
HOG	95,03%	96,78%	96,63%	96,15%
shape context	72,12%	91,69%	78,16%	80,66%
W1,W3,W5,W6,W7	25,93%	49,08%	19,03%	31,35%
W3,W5,W7,W9	25,86%	32,43%	19,98%	26,09%
W5,W6	25,53%	45,97%	19,98%	30,49%

Tabela 6.2. Macierz pomyłek dla wyników deskryptora HOG oraz klasyfikatora MLP bazy danych "płotkarze"

przewidywana klasa / właściwa klasa	P1	P2	P3	P4	P5	suma
P1	46	2	0	0	0	48
P2	1	65	0	0	0	66
P3	0	0	101	0	0	101
P4	0	0	0	54	5	59
P5	0	0	0	2	84	86

Tabela 6.3 zawiera wyniki skuteczności klasyfikacji dla drugiej z wykorzystywanych baz danych (zob. rys. 6.2). Najsłabiej przedstawia się skuteczność klasyfikatora SVM dla współczynników kształtu gdzie skuteczność nieznacznie przekracza wartość 0%. Dla współczynników kształtu zauważalny jest również spory wzrost skuteczności perceptronu wielowarstwowego w porównaniu do wyników dla poprzednio omawianej bazy danych. Dla tej bazy najskuteczniejszym algorytmem okazał się shape context, którego uśredniona wartość skuteczności wykorzystanych klasyfikatorów jest niewiele wyższa od wyników uzyskanych przy użyciu histogramu zorientowanych gradientów.

Najwyższa cząstkową skutecznością ze wszystkich uzyskanych wyników wykazał się klasyfikator k-NN dla shape context. Tabela 6.4 przedstawia macierz pomyłek dla shape context oraz klasyfikatora k-Najbliższych Sąsiadów.

Tabela 6.3. Wyniki dla bazy danych "Actions as Space-Time Shapes"

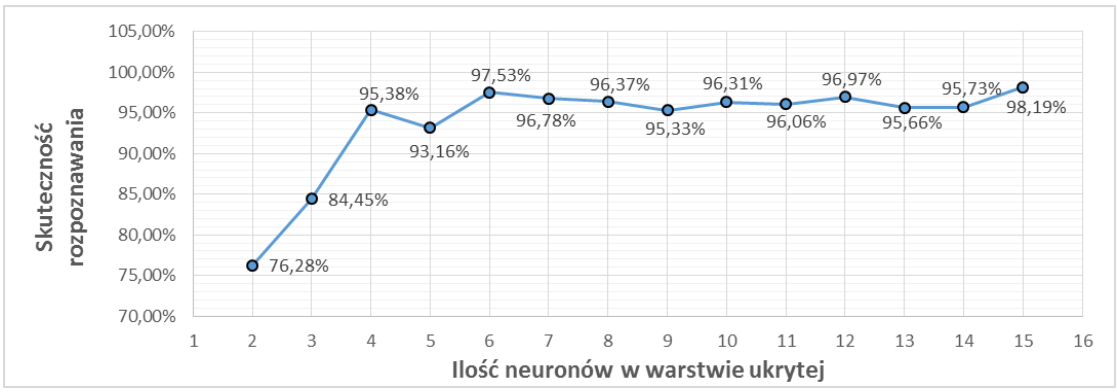
klasyfikator / **metoda opisu obiektu**	**k-NN**	**MLP**	**SVM**	**średnia**
HOG	72,21%	65,06%	77,50%	71,59%
shape context	82,47%	77,04%	73,63%	77,71%
W1,W3,W5,W6,W7	49,36%	59,23%	1,01%	36,53%
W3,W5,W7,W9	39,82%	48,88%	1,11%	29,94%
W5,W6	33,60%	50,67%	0,79%	28,35%

Tabela 6.4. Macierz pomyłek dla wyników deskryptora shape context oraz klasyfikatora k-NN bazy danych "Actions as Space-Time Shapes"

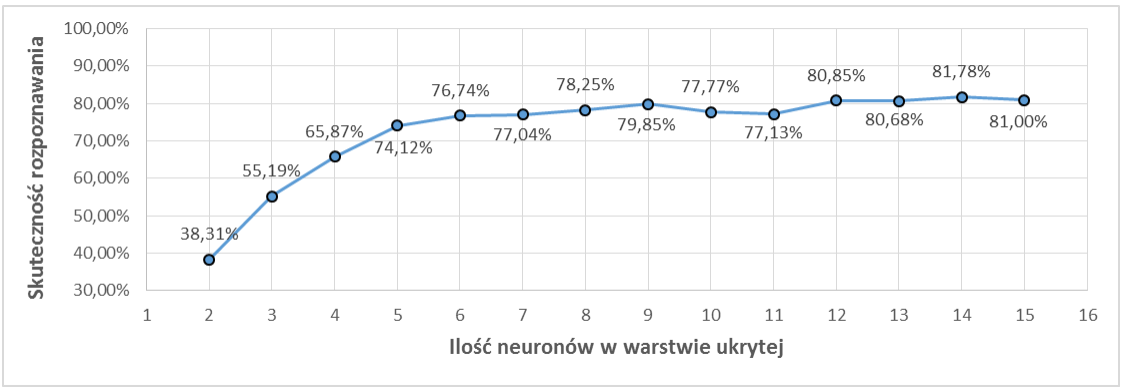
przewidywana klasa / **właściwa klasa**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**suma**
P1	179	7	0	0	0	0	0	0	3	4	193
P2	3	300	0	1	6	7	0	16	4	16	353
P3	0	0	186	12	0	0	18	0	2	1	219
P4	0	2	0	230	0	2	1	0	0	1	236
P5	0	3	0	0	122	24	7	43	0	0	199
P6	0	4	0	0	1	191	0	3	0	0	199
P7	0	0	41	16	16	0	158	2	1	0	234
P8	0	6	0	1	24	23	0	270	0	0	324
P9	0	1	0	1	0	0	0	0	281	71	354
P10	1	36	0	1	0	0	0	0	51	298	387

W celu stworzenia perceptronu wielowarstwowego (zob. listing 4.13) należało ustalić rozmiar warstwy ukrytej wykorzystywanego klasyfikatora. W tym celu sprawdzona została skuteczność klasyfikacji baz danych "płotkarze" [22] oraz "Actions as Space-Time Shapes" [23] dla różnej liczby neuronów (zob. rys. 6.3 oraz rys. 6.4).

Analizując oba wspomniane wykresy zauważyć można, że skuteczność poprawnej klasyfikacji, do pewnego momentu, zwiększa się wraz ze wzrostem liczby neuronów. Po przekroczeniu liczby sześciu neuronów skuteczność pozostaje na zbliżonym poziomie, pomimo że wielkość warstwy ukrytej rośnie. Można przyjąć, że wykorzystując warstwę ukrytą, złożoną z więcej niż sześciu neuronów wyniki będą porównywalne. W pracy, dla uproszczenia sieci, warstwa ta zbudowana jest z 7 neuronów.



Rys. 6.3. Skuteczność klasyfikacji bazy danych "płotkarze" wykorzystując deskryptor HOG w zależności od użytej liczby neuronów w warstwie ukrytej



Rys. 6.4. Skuteczność klasyfikacji bazy danych "Actions as Space-Time Shapes" wykorzystując deskryptor shape context w zależności od użytej liczby neuronów w warstwie ukrytej

7. PODSUMOWANIE

Celem pracy inżynierskiej było stworzenie programu, który miał za zadanie rozpoznawanie pozy postaci ludzkiej. Dzięki wykorzystaniu biblioteki OpenCV możliwe było wykonanie klasyfikacji obrazów poprzez użycie zaimplementowanych w niej funkcji, co pozwalało wyeliminować potencjalne błędy związane z samodzielnym stworzeniem klasyfikatorów. Biblioteka ta pozwala również bezproblemowo wczytywać obrazy z plików oraz dokonywać na nich dowolnych przekształceń.

W celu rozpoznania sylwetki postaci niezbędne było wydobycie wartości charakterystycznych dla każdego z obrazów. W pracy użyte zostały deskryptory obrazów takie jak: histogram zorientowanych gradientów, shape context oraz współczynniki kształtu. Każda z wymienionych metod różni się od siebie w znacznym stopniu. Współczynniki kształtu to najmniej złożone deskryptory, dlatego są one najbardziej ogólne, najmniej dokładne oraz najprostsze w implementacji. Shape context wymaga znalezienia konturu badanego obiektu przez co, w przypadku obrazów, na których znajduje się kilka obiektów o podobnym rozmiarze, możliwe są błędne wyniki. Pozwala on natomiast na manipulacje liczbą badanych punktów konturu oraz ilością binów, w których znajdują się szukane punkty, dzięki czemu możliwe jest dobranie dogodnych parametrów. Histogram zorientowanych gradientów jest najbardziej złożony obliczeniowo. Do jego obliczenia potrzebne jest wykrycie wszystkich konturów na obrazie oraz rozłożenie ich pod określonym kątem. Obraz następnie jest dzielony na mniejsze części, dla których tworzone są histogramy.

Do klasyfikacji wykorzystane zostały trzy metody: k-Najbliższych Sąsiadów, maszyna wektorów nośnych oraz perceptron wielowarstwowy. Najmniej skuteczny okazał się klasyfikator SVM. K-NN oraz perceptron wielowarstwowy wykazały się dużo większą skutecznością, a ich wyniki były do siebie zbliżone.

Autor za własny wkład pracy uważa:

- implementację oraz wykorzystanie deskryptorów obiektów takich jak: współczynniki kształtu, shape context oraz histogram zorientowanych gradientów;
- wykorzystanie maszyny wektorów nośnych, k-Najbliższych Sąsiadów oraz perceptronu wielowarstwowego do klasyfikacji póz postaci ludzkiej;
- zbadanie efektywności zastosowanych klasyfikatorów.

Literatura

[1] Gonzalez R. C., Woods R. E.: *Dgital Image Processing, 3rd Edition.* Pearson Prentice Hall, New Jersey 2008.

[2] Tadeusiewicz R., Korogoda P.: *Komputerowa analiza i przetwarzanie obrazów.* Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków 1997.

[3] Duda R. O., Hart P. E., Stork D. G.: *Pattern Classification, 2nd Edition.* Wiley-Interscience Publication, Wiley 2001.

[4] Bishop Ch.: *Pattern recognition nad machine learning.* Springer, Berlin 2006.

[5] Stąpor K.: *Metody klasyfikacji obiektów w wizji komputerowej.* PWN, Warszawa 2011.

[6] Cunningham P., Delany S. J.:*K-Nearest Neighbour Classifiers.* UCD School of Computer Science, Dublin, Irlandia 2007.

[7] Haykin S.: *Neural Networks A Comprehensive Foundation, 2nd Edition.* Pearson Prentice Hall, Ontario, Canada, 1999.

[8] Hagan M. T.: *An introduction to neural networks.* UCL Press Limited, Londyn 1997.

[9] Efron B.: *Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation.* Journal of the American Statistical Association, s. 316-331, 1983.

[10] Dalal N., Triggs B.: *Histograms of oriented gradients for human detection.* Proceedings of IEEE Conference Computer Vision and Pattern Recognition, San Diego, USA, 2005

[11] Belongie S., Malik J., Puzicha J.: *Shape Matching and Object Recognition Using Shape Contexts.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 2, s. 509-522, 2002.

[12] Bishop Ch.: *Neutral Networks for Pattern Recognition.* Clarendon Press, Oxford 1995.

[13] Kecman V.: *Learning and Soft Computing.* MIT Press, Cambridge, MA, 2001.

[14] Garcia-Pedrajas G., Ortiz-Boyer D.: *Boosting k-nearest neighbor classifier by means of input space projection.* University of Cordoba, Cordoba, Hiszpania 2009.

[15] Do T-T., Kijak E.: *Face recognition using co-occurrence histograms of oriented gradients.* 37th International Conference on Acoustics, Speech, and Signal Processing, s. 1301-1304, Japonia 2012.

[16] Bohg J., Kragic D.: *Learning Grasping Points with Shape Context.* Sztokholm, Szwecja 2009.

[17] Polat K., Gunes S.: *Breast cancer diagnosis using least square support vector machine.* Selcuk University, Konya, Turcja 2006.

[18] Szelski R.: *Computer Vision: Algorithms and Applications.* Springer 2010.

[19] Sonka M., Hlavac V., Boyle R.: *Image Processing Analysis, and Machine Vision.* USA 2008.

[20] `http://opencv.org/opencv-3-1.html`, Dostęp na 21.12.2016.

[21] Przednowek K., Krzeszowski T., Iskra J., Wiktorowicz K.: *Markerless Motion Tracking in Evaluation of Hurdle Clearance Parameters.* Proceedings of the 2nd Int. Congress on Sport Sciences Research and Technology Support (icSPORTS-2014), pp. 129-136, SCITEPRESS, 2014.

[22] `http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html`, Dostęp na 8.01.2017.

[23] `http://sharky93.github.io/docs/gallery/auto_examples/plot_hog.html`, Dostęp na 8.01.2017.

[24] Hsu Ch-W, Lin Ch-J: *A Comparison of Methods for Multi-class Support Vector Machines.*Department of Computer Science and Information Engineering National Taiwan University, Taipei, Tajwan 2001

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2017
Wydział Elektrotechniki i Informatyki
Katedra Informatyki i Automatyki

STRESZCZENIE PRACY DYPLOMOWEJ INŻYNIERSKA

ZASTOSOWANIE METOD UCZENIA MASZYNOWEGO DO ROZPOZNAWANIA POZY POSTACI LUDZKIEJ

Autor: Bartosz Gąsior, nr albumu: EF-DI-137405
Opiekun: Dr inż. Tomasz Krzeszowski
Słowa kluczowe: metody opisu obiektów, klasyfikacja, OpenCV, morfologia matematyczna

Praca miała na celu zaprojektowanie oraz stworzenie aplikacji służącej do rozpoznawania pozy postaci ludzkiej na obrazie. Aplikacja została napisana w języku C++ z wykorzystaniem darmowej biblioteki OpenCV. Do opisania kształtu obiektów znajdujących się na obrazie wykorzystano współczynniki kształtu, histogram zorientowanych gradientów oraz shape context. Do przeprowadzenia operacji klasyfikacji obrazów użyte zostały klasyfikatory takie jak: maszyna wektorów nośnych, k-Najbliższych Sąsiadów oraz perceptron wielowarstwowy.

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2017
Faculty of Electrical and Computer Engineering
Department of Control and Computer Engineering

BSC THESIS ABSTRACT

HUMAN POSE RECOGNITION USING METHODS OF MACHINE LEARNING

Author: Bartosz Gąsior, nr albumu: EF-DI-137405
Supervisor: Tomasz Krzeszowski, PhD, Eng.
Key words: object description methods, classification, OpenCV, mathematical morphology

The purpose of the study was to design and create an application for human pose recognition in images. The application was developed in C++ environment using open source OpenCV library. Shape coefficients, a histogram of oriented gradients and shape context were used for description of objects in images. Image classification was performed by support vector machine, k-nearest neighbours and a multi-layer perceptron.