SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72737

DETEKCIA ŠPORTOVÝCH AKTIVÍT Z VIDEO SEKVENCIE

DIPLOMOVÁ PRÁCA

2018 **Bc. Marek Hrebík**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72737

DETEKCIA ŠPORTOVÝCH AKTIVÍT Z VIDEO SEKVENCIE

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Dominik Sopiak
Konzultant:	Ing. Jozef Gerát

Bratislava 2018 **Bc. Marek Hrebík**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Marek Hrebík
Diplomová práca:	Detekcia športových aktivít z video sekvencie
Vedúci záverečnej práce:	Ing. Dominik Sopiak
Konzultant:	Ing. Jozef Gerát
Miesto a rok predloženia práce:	Bratislava 2018

Práca sa zaoberá detekciou a rozpoznávaním pohybov z videosekvencie. Oboznamuje s jednotlivými technológiami, ktoré boli použité v implementácii, taktiež vysvetľuje princípy riešenia, spôsoby spracovania videosekvencie. Zároveň sa v práci nachádza popis jednotlivých databáz videí, postup a metódy na spracovanie obrazu. Zároveň predstavuje navrhnuté riešenie s popisom a postupom, ktorý bol zvolený. Toto riešenie je v práci testované a porovnávané s inými výsledkami vypracovanými na tej istej databáze videí. Obsahuje zhodnotenie týchto výsledkov a víziu ďalšieho postupu.

Kľúčové slová: detekcia, rozpoznávanie, histogram orientovaných gradientov, strojové učenie, analýza hlavných komponentov

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Marek Hrebík
Master's thesis:	Detection of sport activities from video sequency
Supervisor:	Ing. Dominik Sopiak
Consultant:	Ing. Jozef Gerát
Place and year of submission:	Bratislava 2018

On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.

Keywords: detection, recognition, histogram of oriented gradients, machine learning, principal components analysis

Poďakovanie

I would like to express a gratitude to my thesis supervisor.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

HOG	Histogram of oriented gradients
MHI	Motion history image
OpenCV	Open Source Computer Vision Library
PCA	Principal Component Analysis
SVM	Support vector machine
t-SNE	t-Distributed Stochastic Neighbor Embedding

Zoznam výpisov

Úvod

Detekcia, rozpoznávanie objektov a počítačové videnie všeobecne je smer IT sféry, ktorý sa veľmi rýchlym krokom rozvíja. Poskytuje ľuďom nové možnosti, ako uľahčiť život, zaistiť bezpečnosť, zabávať sa alebo v rámci podnikovej sféry zrýchliť výrobu, zvýšiť kvalitu výrobkov bez navýšeného úsilia.

Téma rozpoznávania a detekcie pohybov je aj predmetom tejto práce, v ktorej sa pokúsime postupne popísať konkrétne metódy tohto predmetu. Zároveň sa sústredíme aj na konkrétne riešenie a implementáciu.

V práci postupne prejdeme od popísania použitých technológií a vývojového prostredia určených na implementáciu k jednotlivým princípom riešenia. Objasníme si, akým spôsobom spracovávať videosekvenciu. Prejdeme si existujúce databázy ľudského pohybu, ich vlastnosti. Ďalej vysvetlíme možnosti a algoritmy spracovávania obrazu, kde detailnejšie popíšeme jednotlivé metódy od prvotných úprav až po klasifikáciu.

Návrh riešenia odprezentujeme postupne od výberu konkrétnej databázy, cez algoritmus na predspracovanie a spracovanie príznakov, až po klasifikáciu, nástroje na testovanie a samotné porovnanie výsledkov. Na záver zhodnotíme výsledky nášho riešenia a zamyslíme sa nad jeho ďalšími možnosťami rozšírenia a úpravy.

1 Použité technológie a prostredie

1.1 OpenCV

Open Source Computer Vision Library (OpenCV) bola pôvodne vydávaná firmou Intel, neskôr výskumným centrom Willow Garage, ktoré vyvíja open source softvér pre aplikácie na poli robotiky. Podporuje frameworky TensorFlow, Torch/PyTorch, Caffe. Knižnica je vydávaná pod BSD licenciou, čo umožňuje jej komerčné používanie. Táto knižnica obsahuje množstvo nástrojov a algoritmov na detegovanie gest, identifikáciu objektov, segmentáciu a rozpoznávanie, spájanie obrazov, sledovanie pohybu, taktiež je vhodná pre implementáciu rozšírenej reality a strojového učenia. OpenCV je podporované v jazykoch C++, Python, Java, C a Matlab. Vývoj je umožnený na operačných systémov Windows, Android, Linux a Mac. [1]

1.2 FFmpeg

Táto knižnica je voľný softvérový projekt určený na manipulovanie (kódovanie, dekódovanie, multiplexovanie, prehrávanie a streamovanie) s multimediálnymi súbormi. Má voľnú licenciu GNU (General Public License). Výhodou práce s knižnicou FFmpeg je jej rýchlosť a kvalita výstupných dát. Knižnicu využíva aj OpenCV pre spracovanie a distribuovanie obrazovej sekvencie. Obsahuje nástroje na zmenu kvality videa, strihanie, čo napomáha pri kontextovej analýze obrazu.[2]

1.3 Python

Programovací jazyk Python je interpretovaný vysokoúrovňový jazyk, vytvorený v roku 1991. Jeho využitie je široké, od vývoja webových aplikácii, backendu, frontendu cez vedecké a numerické výpočty, tvorbu grafických rozhraní, vývoj softvéru, výučbu a tvorbu biznis aplikácií. V súčasnosti má širokú podporu u programátorov, čo sa odráža aj na prehľadnom zdokumentovaní. Je to open source softvér s množstvom štandardných knižníc, ktoré majú široké využitie v automatizácii, multimédiách, spracovaní obrazu, textu a ďalších sférach IT.[3]

1.4 Scikit a Numpy

1.4.1 Scikit

Scikit-learn je knižnica s otvoreným zdrojovým kódom pre strojové učenie pre programovací jazyk Python. Scikit je napísaný v jazyku Python s niektorými algoritmami písanými v Cythone pre zvýšenie výkonu tejto knižnice. Obsahuje rôzne klasifikačné, regresné a zhlukovacie algoritmy, taktiež algoritmy na modelovanie dát (krížová validácia,

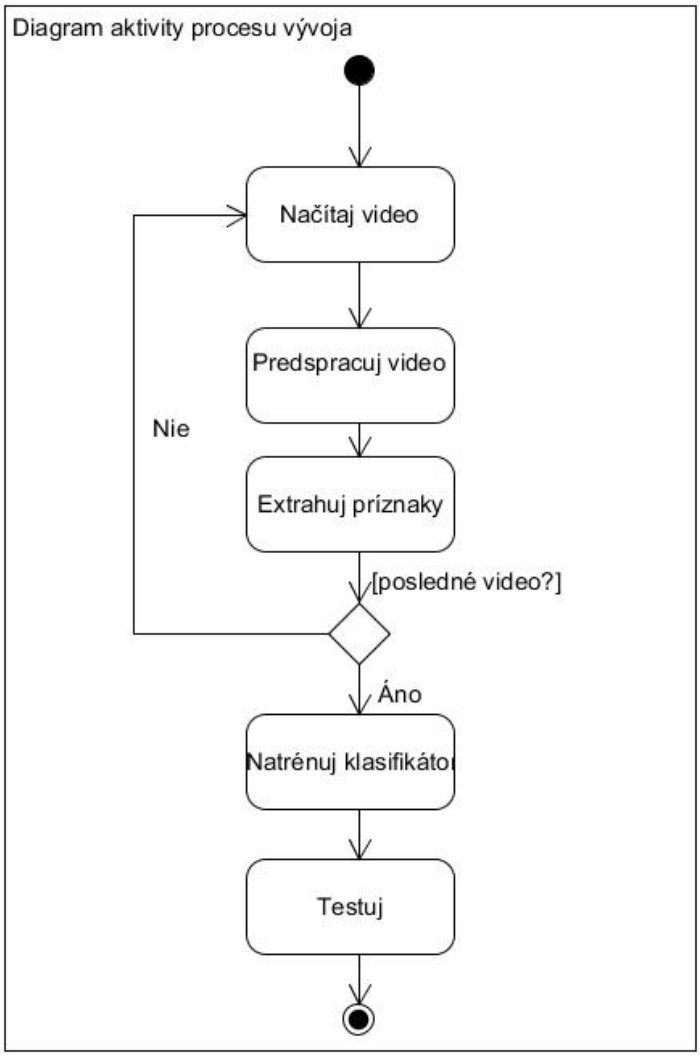
výber príznakov, clustering a iné). Je navrhnutá na spoluprácu s numerickou knižnicou NumPy a vedeckou knižnicou SciPy. [4]

1.4.2 NumPy

NumPy je rozšírenie do Pythonu, ktoré nám umožňuje rýchle a efektívne vykonávanie operácií nad poľami homogénnych dát. NumPy pole je multidimenzionálne pole objektov ktoré majú všetky rovnaký typ. V pamäti sa nachádza ako objekt, ktorý smeruje na blok pamäte, ktorý drží informáciu o type údajov v pamäti, počet dimenzií pamäte, veľkosť každej dimenzie a taktiež aj vzdialenosti medzi jednotlivými prvkami pozdĺž každej z osí. Narozdiel od klasického zoznamu, ktorý je implementovaný ako zoznam smerníkov je prehľadávanie NumPy poľa rýchlejšie. klasický zoznam potrebuje na iteráciu poľa prístup cez dva smerníky, čo predĺžuje čas prehľadávania.[5]

2 Princípy riešenia

V prvej časti tejto práce si predstavíme metódy, ktoré sme zvažovali pri riešení nášho problému. Takisto si zhrnieme aké databázy pohybov máme k dispozícii pre potreby tejto práce, typy databáz, dĺžky jednotlivých videí, vlastnosti a aké druhy aktivít sa v týchto videách nachádzajú. Taktiež je potrebné predstaviť metodológiu spracovania údajov pre zachytenie vhodných vlastností videa na rozpoznanie. Na obrázku 1 je znázornená predstava riešenia problému detekcie a rozpoznávania pohybu. Základom je načítanie videí, následne ich predspracovanie na extrakciu príznakov a samotná extrakcia. Konečným krokom v procese by malo byť testovanie a vyhodnotenie aplikácie.



Obr. 1: Diagram plánu vývoja riešenia

2.1 Spôsob spracovania sekvencie

Zo zadania vyplýva, že pre získanie príznakov z videa budeme potrebovať videosekvenciu rozdeliť na menšie časti, respektíve ju spracovať do formátu obrázku (.jpg, .png. alebo iné). Jednotlivé snímky z videa sú potrebné na hlbšiu analýzu. Podľa výberu datasetu budeme môcť zvoliť či budeme používať celú videosekvenciu, alebo iba jej časť, v ktorej je zrejmý druh pohybu.

V jednotlivých triedach videí bude potrebné hľadať spoločné a rozdielne znaky na to, aby sme následne vybrali vhodný postup na spracovanie a získanie vhodných príznakov. Knižnica OpenCV neobsahuje metódy ktoré zabezpečia čítanie a prenos obrazu, avšak jej metódy využívajú funkcionalitu ďalšej knižnice FFmpeg, ktorá distribuuje prostredníctvom rozhrania jednotlivé obrazové snímky.

2.1.1 Získanie časovo-priestorových vzťahov

Bude potrebné získať tieto vzťahy na porovnávanie s ďalšími videami, teda určenie vhodného spôsobu na získanie týchto vzťahov. Jedným zo spôsobov je napríklad algoritmus MHI (Motion History Image), ktorý nám môže zabezpečiť tieto príznaky z videa uložiť a ďalej spracovávať. Podľa tohto algoritmu by sme vedeli určiť odkiaľ kam sa aktér z videa pohyboval, akým spôsobom a rýchlosťou, čo môžme považovať za prvé vhodné príznaky pre určenie typu pohybu.[6] Tento algoritmus si bližšie popíšeme ďalej v tejto práci.

2.1.2 Extrakcia príznakov

Extrakcia príznakov je dôležitou časťou pre natrénovanie klasifikátora. Preto je potrebné vybrať príznaky, ktoré nám s čo najvyššou presnosťou zadefinujú konkrétny pohyb, ktorý sa bude vykonávať vo videu. Mali by to byť príznaky, ktoré sú jednoznačné a jedinečné pre každý z druhov pohybu. Preto musíme uvažovať vhodnú metódu extrahovania príznakov, aby sme zaistili diverzitu medzi triedami jednotlivých typov pohybu. K tomu nám bude nápomocná aj kombinácia viacerých spôsobov extrakcie.

Tieto príznaky budeme extrahovať z predspracovaného zdroja a ukladať na disk pre natrénovanie klasifikátora, prípadne viacnásobné trénovanie a testovanie bude vhodné jednotlivé výsledky ukladať pre neskoršie porovnanie.

2.1.3 Návrh klasifikátora

Po úspešnom získaní príznakov z každej videosekvencie bude potrebné natrénovať klasifikátor tak, aby nám každá skupina spracovaných videozáznamov spadala do jednej triedy. Takto natrénovaný klasifikátor budeme potrebovať uložiť pre potreby testovania a porovnávania výsledkov a diverzity jednotlivých skupín. Zároveň budeme potrebovať

zvoliť vhodný pomer dát na testovanie a tréning na základe databáz, ktoré budeme mať k dispozícii. Z tohto dôvodu budeme potrebovať čo najviac videí, aby bol návrh klasifikačnej metódy čo najpresnejší.

2.2 Databázy videí

Na internete je veľké množstvo video databáz s rôznymi pohybmi objektov, ľudí a zvierat. Výber databázy realizujeme na základe vlastností jednotlivých videí. Videá môžu byť snímané staticky z jedného miesta bez pohybu kamery, staticky s otáčaním kamery, taktiež dynamicky s rôznym pohybom a otáčaním kamery. Výber databázy bude najdôležitejšou súčasťou tejto práce, keďže od nej sa bude odvíjať celý ďalší postup.

2.2.1 Vlastnosti videa

Dôležitými vlastnosťami videí z databáz sú:

- **veľkosť**
- **rozlíšenie**
- **dĺžka**
- **počet videí**
- **spôsob snímania**
- **formát videa**
- **počet objektov**

Veľkosť videa ovplyvňuje vo vysokej miere časový úsek určený na spracovanie videa. Z tohto dôvodu je vhodné hľadať databázy videí, ktorých videá majú menšiu veľkosť, čo priamo súvisí s ich rozlíšením, počtom snímkov za sekundu a dĺžkou. Nesmieme ale zabudnúť na to, že video musí byť zreteľné a pohyb detegovateľný.

Rozlíšenie videa je dôležitým faktorom pre spracovnie z hľadiska kvality príznakov a snímkov. Zároveň jeho odporúčaná veľkosť sa líši od konkrétneho spôsobu extrakcie príznakov. Dôležité je, aby obraz konkrétnej aktivity na videu bol jasný a voľným okom rozpoznateľný.

Dĺžka sekvencie sa líši od rozmanitosti jednotlivých pohybov osoby. Niektoré databázy obsahujú videosekvencie s opakovaním pohybov, teda dĺžka videí v týchto databázach je väčšia ako tam, kde je daná aktivita alebo pohyb zachovaný bez opakovania.

Počet videí môže ovplyvniť výsledok a efektivitu rozpoznávania aktivít. Niektoré databázy obsahujú malé množstvo videí pre jeden konkrétny pohyb (okolo 10 až 20), iné zdroje videosekvencií ich majú aj viac ako 50 pre jeden typ pohybu.

Spôsob snímania v rozpoznávaní pohybu je dôležitou vlastnosťou, keďže nie všetky spôsoby riešenia je možné aplikovať na videá s pohybujúcou sa kamerou a opačne.

Formát videa súvisí s jeho veľkosťou, avšak formát je možné meniť a konvertovať na nami potrebný rozmer.

Počet objektov, ktoré sa na videu pohybujú a vyhodnocujú vplýva na výkon a rýchlosť rozpoznávania. Tu taktiež nie je možné aplikovať niektoré metódy rozpoznávania.

2.2.2 Databázy ľudského pohybu

Pre potrebu rozpoznávania pohybu osoby existuje viacero databáz, ktoré sme v našej práci uvažovali a testovali:

- **HMDB51**
- **KTH**
- **UCF-Sports**
- **Hollywood**
- **MSR Action I, II**
- **IXMAS**
- **WEIZMANN**

HMDB51 je databáza pohybov v ktorej sa nachádza 51 tried pohybu a gestikulácie ako napríklad česanie vlasov, žuvanie, lezenie, potápanie, chôdza atď. Celkovo obsahuje 6849 videí s rozlíšením 320 x 240 pixelov. Videá sú so statickou aj dynamickou kamerou. Zdrojom videí je YouTube ako aj iné verejne dostupné databázy videí. Videá sú špecifické vysokou diverzitou kvality ako aj pohybu.[7]

KTH pohybová databáza so šiestimi triedami - chôdza, pomalý beh, beh, boxovanie, kývanie rukou, tlieskanie. V každej z týchto tried sa nachádza 100 čiernobielych videí s rozlíšením 160 x 120 pixelov so statickou kamerou. Videá sú natočené v interiéri ako aj exterieri. [7]

UCF-Sports obsahuje 9 tried pohybu so 182 videami so statickou aj dynamickou kamerou. Rozlíšenie videí je 720 x 480 pixelov. Videá sú z rôznych športových podujatí vysielaných v televízii - skok do vody, vzpieranie, jazda na koni a iné.[7]

Hollywood databáza obsahuje 8 tried - vystúpenie z vozidla, bozkávanie, postavenie sa, sadnutie si a iné. Nachádza sa tam 430 videí s rozlíšením 300 - 400 x 200 - 300 pixelov s dynamickou kamerou a dynamickým pozadím.[7]

MSR Action I, II obsahujú 3 triedy pohybu so statickou kamerou a 16-timi videami s rozlíšením 320 x 240 pixelov. Namodelované videá sú určené na detekciu pohybu, nie jeho rozpoznávanie. Vo videách sa nachádza viac ako jeden pohyb. V pozadí sa objavujú aj iné osoby, prípadne objekty ako auto. [7]

IXMAS alebo *INRIA Xmas Motion Acquisition Sequences* obsahuje 13 tried s rozlíšením 390 x 291. Pohyb je zaznamenávaný z viacerých statických kamier. [7]

WEIZMANN obsahuje 10 tried pohybu, celkový počet videí je 90. Obsahujú triedy pohybu ako chôdza, beh, kývanie jednou alebo dvomi rukami, skok na mieste a iné. Videá sú natočené so statickou kamerou a pozadím v interiéri s rozlíšením 180 x 144. [7]

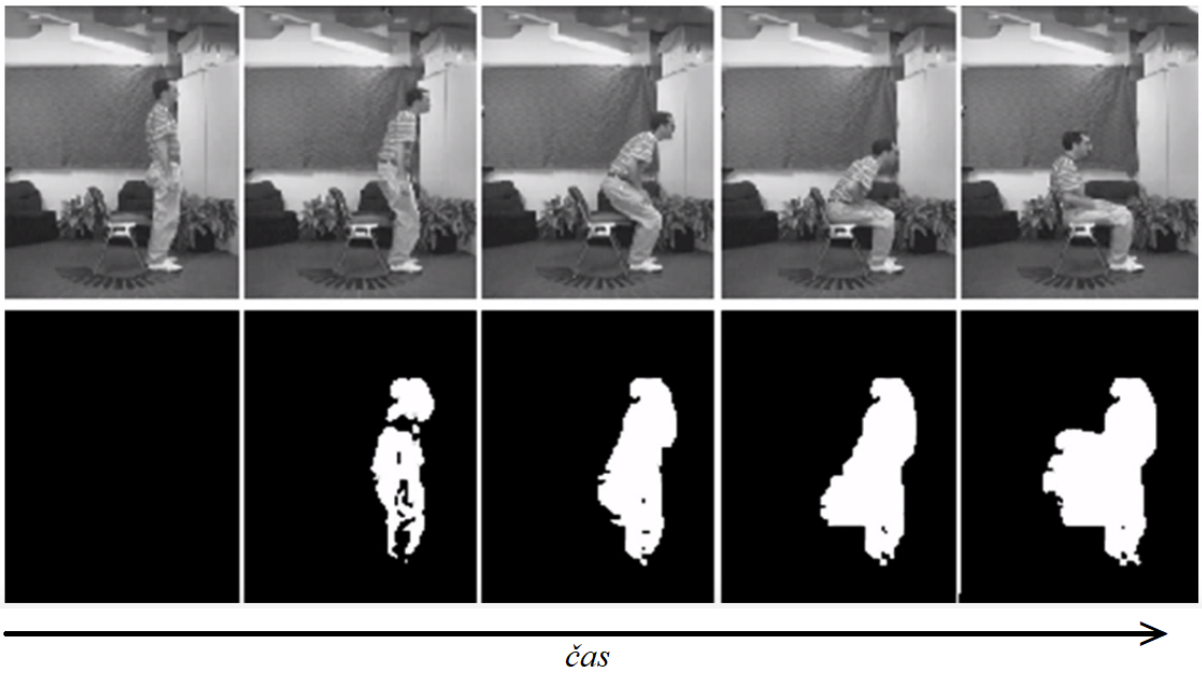
2.3 Postup spracovania obrazu

Na získanie príznakov a potrebu klasifikácie pohybu analyzujeme a zvažujeme konkrétne spôsoby spracovania obrazu z videa. Pre tento účel berieme v úvahu algoritmus Motion history image (MHI) na extrakciu prvotných príznakov a následné spracovanie pomocou Histogram of oriented gradients (HOG).

Pre zosilnenie našich príznakov a zaručenie vyššej úspešnosti klasifikácie uvažujeme použitie Principal Component Analysis (PCA), ktoré taktiež urýchli trénovanie klasifikátora. Takto získané príznaky by sme následne vedeli natrénovať pomocou Support vector machine (SVM). Z natrénovanej SVM vieme otestovať úspešnosť klasifikátora, ktorý sme získali a tým sa dostať ku výsledkom a porovnávaniam.

2.3.1 Motion History Image

MHI je metódou, ktorá je založená na časovej šablóne. Je to jednoduchá metóda avšak robustná, vhodná na reprezentovanie a analýzu pohybu.[6] Táto metóda bola prvýkrát popísaná v dokumente *"An appearance-based representation of action"* od Aarona Bobicka a Jamesa W. Davisa.[8] Rozpoznávanie akcie pomocou MHI patrí do skupiny prispôsobovania šablóny. V MHI sa informácia pohybu v čase spája do jedného obrázku, kde intenzita pixelov je funkciou histórie pohybu na danej pozícii pixelu. MHI sa vypočítava pomocou aktualizačnej funkcie.3



Obr. 2: Priebeh videa a tvorba MHI

Z obrázku 2 môžme jednoduchšie pochopiť akým spôsobom MHI pracuje. Jednotlivé snímky sú zoradené v čase akom boli nasnímané. V hornom riadku obrázkov vidíme neupravené snímky. Pod nimi je ekvivalent MHI obrázkov, ktoré sa kumulatívne spájali *(Motion Energy Image*, teda nenesú informáciu o histórii daného pohybu, iba o tom, ktoré pixely sa menili a kde sa pohyb konal.

Z tohto riešenia je možné jednoducho odvodiť riešenie MHI, kde pri kumulovaní jednotlivých snímkov do jedného je uberaná intenzita všetkých pixelov v každej iterácii. Tento proces je zapísaný rovnicou na obrázku 3.

$$H_\tau(x,y,t) = \begin{cases} \tau & ak\ \Psi(x,y,t) = 1 \\ \max(0, H_\tau(x,y,t-1) - \delta & inak \end{cases}$$

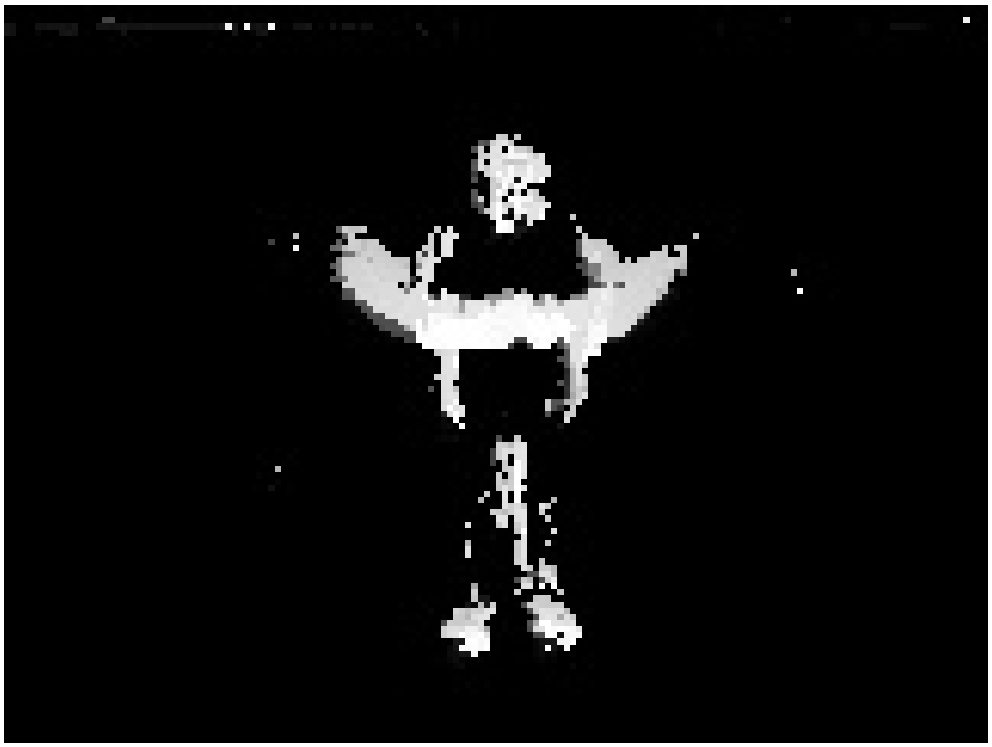
$(x,y,t) - pozícia\ pixelu\ a\ čas$
$\Psi(x,y,t) - prítomnosť\ (alebo\ pohyb)\ objektu\ v\ poslednom\ snímku$
$\tau - dĺžka\ určujúca\ časový\ rozsah\ pohybu\ (jednotlivých\ snímkov)$
$\delta\ - parameter\ stmavnutia\ pixelu$

Obr. 3: Funkcia zmeny pixelov v MHI

Táto aktualizačná funkcia je volaná pre každý ďalší snímok, ktorý analyzujeme z videosekvencie. Výsledkom tohto výpočtu je skalárny snímok, v ktorom svetlejšie časti

obrázku označujú najnovšie menené pixely a tmavšie sú tie, ktoré boli menené dávnejšie z pohľadu pohybu na videu.

To nám umožňuje zaznamenať pohyb z videa do jediného obrázku, v ktorom je vystihnutý celý priebeh videa alebo jeho časti. Táto technika je vhodná na použitie pri videách so statickou kamerou a pozadím.[9]



Obr. 4: Motion History image tlieskania

Na obrázku 4 je príklad spracovaného pohybu z videa do MHI, na ktorom môžme vidieť pohyb tela pri akcii tlieskania.

2.3.2 Histogram of Oriented Gradients

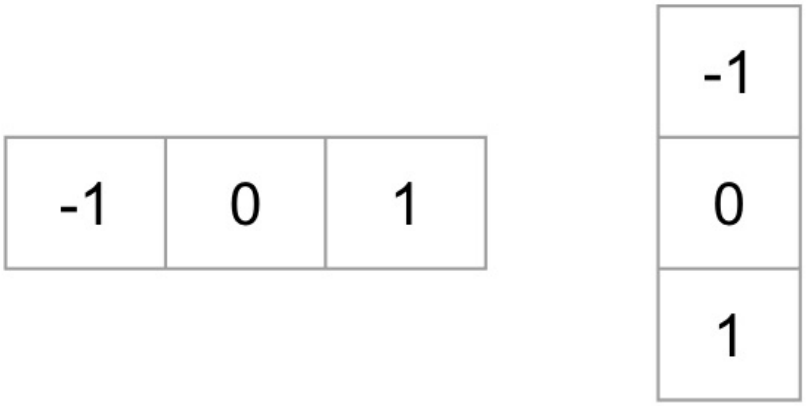
Deskriptor vlastností je forma reprezentácie obrazu alebo jeho časti, ktorá zjednodušuje obraz extrakciou dôležitých informácií. Tento deskriptor premieňa obraz na pole, resp. vektor vlastností, v ktorom sa nachádzajú informácie o obraze. Vyberá informácie o obraze, ktoré sú potrebné a tie nepotrebné preskočí.

Tento deskriptor je užitočný pre potreby detekcie a rozpoznávania obrazu, kde jeho vektor je vstupom pre klasifikačné algoritmy ako napríklad SVM.

HOG deskriptor (Histogram of oriented gradients) používa ako vektor príznakov rozdelenie smerov orientovaných gradientov. Tieto príznaky sú vešmi dobré na detekovanie hrán v obraze a poskytujú vhodné informácie o jeho vlastnostiach.

Predpríprava a výpočet gradientu Vo väčšine prípadov je potrebné orezať obraz na časť, ktorú potrebujeme analyzovať s tým, že pomer jeho strán musí byť počas zmenšovania a orezávania zachovaný.

Na výpočet HOG deskriptora je najprv nutné vypočítať horizontálne a vertikálne gradienty, následne je možné vypočítať histogram gradientov. Tento krok je jednoduché dosiahnuť pomocou filtrovania obrazu nasledovnými jadrami.



Obr. 5: Jadrá pre výpočet gradientov [10]

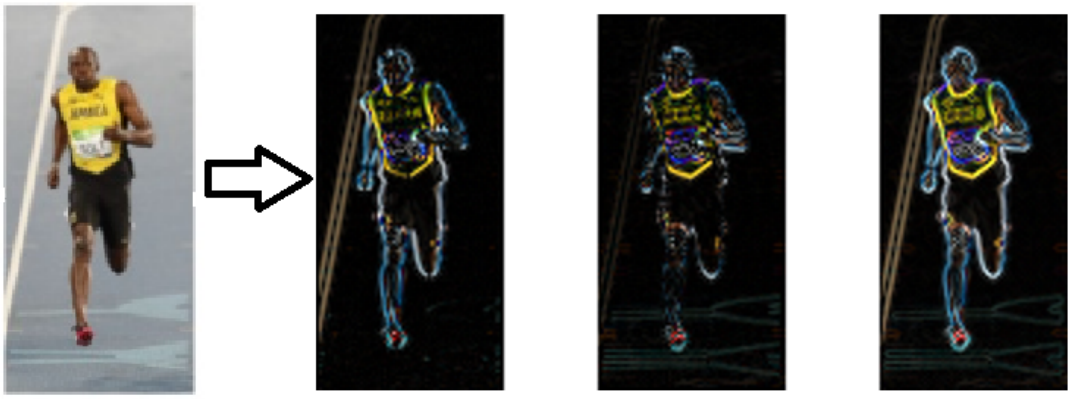
Ďalej potrebujeme nájsť veľkosť a smer gradientov pomocou nasledovných vzťahov:

$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = \arctan \frac{g_y}{g_x}$$

Obr. 6: Výpočet veľkosti a smeru gradientu [10]

Kde g je veľkosť gradientu a θ je jeho smer. x, y sú osi jednotlivých gradientov, od ktorých sa odvíjajú.

Na obrázku bežca si ukážeme, ako takéto gradienty po výpočte vyzerajú.



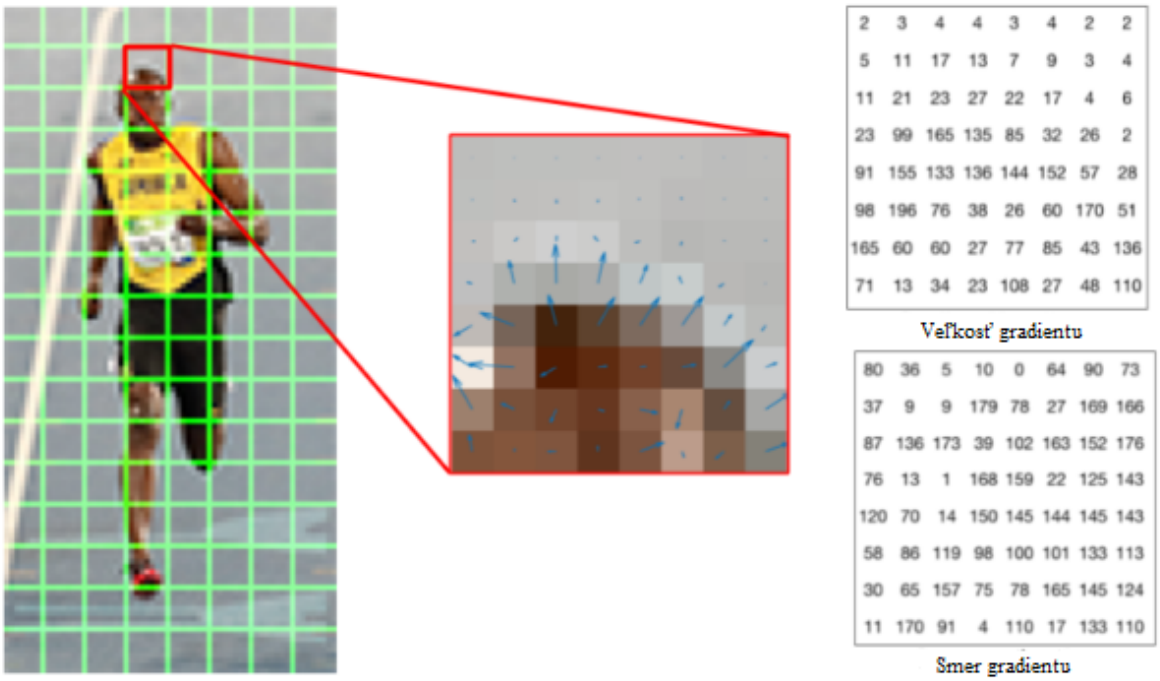
Obr. 7: Originálny obraz transformovaný na gradienty [10]

Na obrázku 7 môžme vidieť transformáciu originálneho obrazu na absolútnu hodnotu gradientu osi x, vedľa nej absolútnu hodnotu gradientu y a napravo veľkosť gradientu. Gradient osi x na obrázku zaznamenal vertikálne čiary bežeckej trate, gradient osi y zase horizontálne hodnoty. Veľkosť gradientu zaznamenáva každú ostrú zmenu intenzity jednotlivých pixelov. Pokiaľ sú časti obrazu plynulé, žiadne dôležité informácie nie sú zachytené. Obraz gradientu eliminuje nedôležité informácie v obraze, čo môžme demonštrovať tým, že aj na takomto obraze gradientov môžme rozpoznať, že ide o bežiacu osobu.

Výpočet gradientov po bunkách V tomto kroku môžme obraz rozdeliť do buniek o veľkosti 8 x 8 pixelov, pre ktoré bude histogram gradientov počítaný osobitne pre každú bunku.

Výpočet po bunkách je dôležitým krokom na redukovanie objemu dát, pretože pre každý pixel v bunke máme dokopy *8 pixelov x 8 pixelov x 3 kanály obrazu hodnôt* (192) a zároveň *8 pixelov x 8 pixelov x 2 hodnoty (veľkosť a smer gradientu)* (128). Ukážeme si, že pomocou redukcie uhlov do 9tich smerových kategórií vieme týchto 128 hodnôt uložiť ako 9 hodnôt. Reprezentácia takýmto spôsobom je kompaktnejšia, zároveň je odolnejšia voči šumu v obraze.

Histogram gradientu je vlastne vektor deviatich smerových kategórií, ktoré zoskupuje podľa uhla, ktorý gradient má.

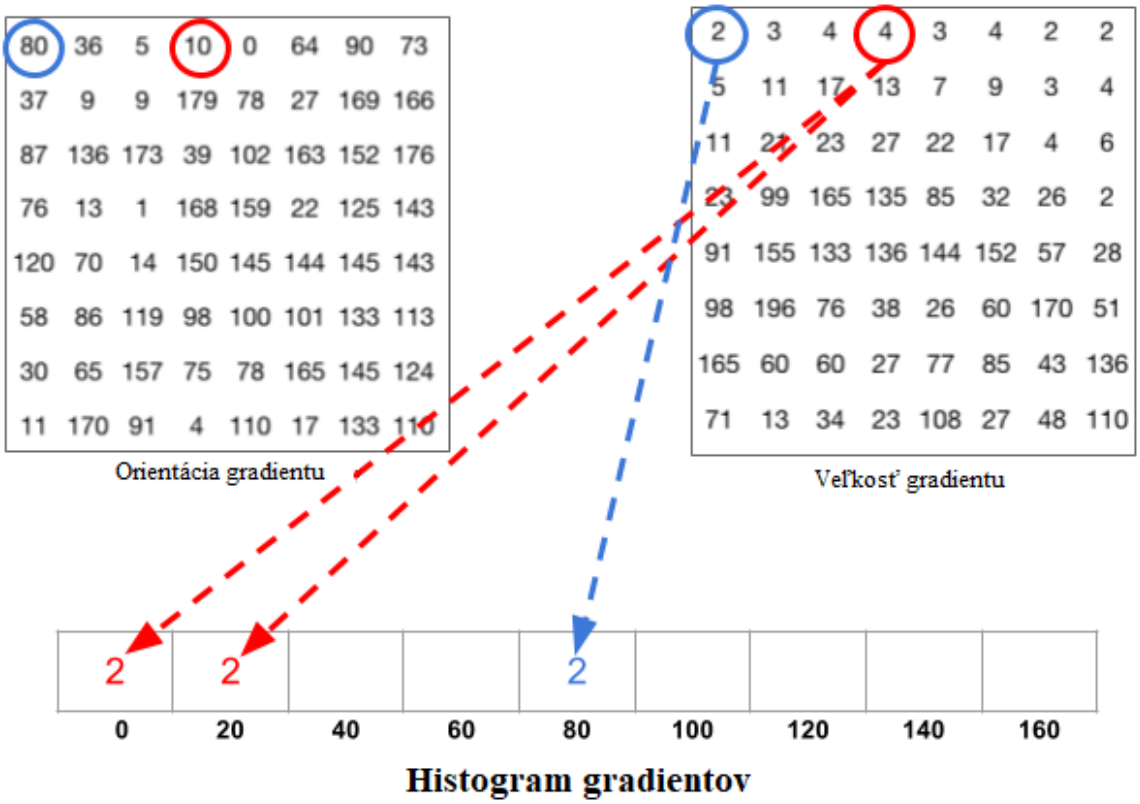


Obr. 8: Príklad veľkostí a smerov gradientov [10]

Na obrázku 8 môžme vidieť znázornenie vypočítaných hodnôt gradientu bunky *8 x 8*. Na obrázku v strede môžme vidieť jednotlivé pixely obrázku a gradienty, ktoré majú svoju veľkosť a smer. Na pravej strane obrázku hore sú veľkosti gradientov jednotlivých pixelov a dole sú ich smery v stupňoch. Stupne sú určené v rozmedzí 0 - 180 (bezznamienkový), aj keď je možné využiť aj interval 0 - 360 stupňov (znamienkový) . To všetko závisí od toho, či používame gradienty so znamienkom alebo bez. Výhodou používania 180 stupňového intervalu je, že bezznamienkové gradienty sú úspešnejšie pre potreby detekcie pohybu osoby.

Ďalším krokom je vytvorenie HOG v konkrétnych bunkách napríklad o rozmere *8 x 8*. Histogram bude obsahovať 9 skupín pre každý 20-ty stupeň z rozmedzia 0 - 180 stupňov *(0, 20, 40, 60 ... 160)*.

Následne sa rozdeľovanie do jednotlivých skupín prebieha ako na obrázku 9.



Obr. 9: Rozdeľovanie veľkostí gradientov podľa orientácie [10]

Môžme vidieť, že skupina kam ide veľkosť gradientu sa vyberá podľa jeho orientácie. Gradient v modrom je zaradený do skupiny 80 pretože má orientáciu 80 stupňov a je priradená hodnota 2, pretože je to jeho dĺžka. Pri gradiente v červenom kruhu je hodnota rozdelená medzi skupiny 0 a 20, keďže orientácia gradientu je 10, rozdelí sa veľkosť na dve rovnaké časti. V prípade, že by sme mali orientáciu gradientu 170 stupňov s veľkosťou 85, rozdelila by sa jeho veľkosť medzi skupinu 0 a 160 taktiež v pomere 1:1. Takto máme vytvorený prvý histogram.

Normalizácia blokov Normalizovanie blokov je potrebné na to, aby bol deskriptor nezávislý na zmene svetelnosti obrazu. Takáto normalizácia sa vypočíta následovne

$$norm = \sqrt{r^2 g^2 b^2}$$

$$r, g, b \ - zložky\ farieb\ RGB\ pixelu$$

Obr. 10: Výpočet L2 normy[10]

Takto vypočítaná hodnota sa nazýva *L2-norma*. Vydelené zložky RGB pixelu touto normou sú normalizované, teda z intervalu *<0,1>*. Teda to nám eliminuje zmeny hodnôt pre pixely, ktoré boli napríklad dvojnásobne zosvetlené.

Tak ako sme normalizovali každú zložku, je potrebné normalizovať aj zložky celého histogramu. Na to sa používa normalizácia väčších blokov aj s prekrytím [1].

Z takto získaných príznakov vieme získať finálny HOG vektor príznakov tým, že spojíme čiastkové vektory. [11][10]

2.3.3 Principal Component Analysis

Analýza hlavných komponentov (angl. Principal Components Analysis) je metóda štatistiky, ktorá využíva ortogonálnu transformáciu za účelom zmeny prvkov, v ktorých sa vyskytuje korelácia na prvky, kde sa takáto lineárna korelácia nebude nachádzať. PCA teda spracováva a upravuje jednotlivé komponenty tak, aby sa ich dimenzia zmenšila, alebo bola nanajvýš rovnaká a zároveň sa snaží zachovať čo najväčšie množstvo informácií o pôvodných premenných. Tento prístup pomáha analyzovať dané dáta v podpriestore, čo nám umožňuje ďalšie spracovanie.

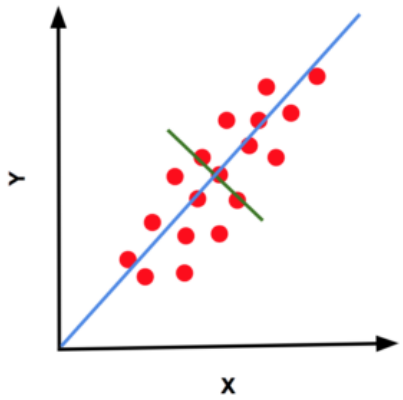
Dôležitou súčasťou v PCA sú vlastné vektory. Sú to nenulové vektory, ktorých smer sa po transformácii pomocou lineárneho operátora nemení, pojem vlastný vektor v PCA obmieňame za hlavný komponent.

V počítačovom videní má táto metóda veľmi dobrý vplyv na dáta charakterizujúce rôzne špecifické tvary, objekty, keďže nám umožňuje zosilniť význam príznakov, ktoré sú pre daný problém dôležité, a tie, ktoré nie sú potrebné, dokáže eliminovať. Táto transformácia teda zvyšuje varianciu hlavných komponentov naprieč všetkými lineárnymi kombináciami vektora, čo napomáha zvyšovať úspešnosť klasifikátora pri riešení rozpoznávania na extrakciu príznakov. Túto metódu navrhol v roku 1901 matematik anglického pôvodu Karl Pearson. Jej rôzne obmenené a vylepšené verzie sa využívajú dodnes v štatistike, *počítačovom videní*, robotike a ďalších smeroch.

Variancia Variancia kóduje informácie, ktoré obsahujú jednotlivé dáta. V dvojrozmernom priestore je potrebné pre *n* bodov mať *2n* čísel, ktoré nám reprezentujú tieto dáta. Na obrázku 11 môžme vidieť dva smery maximálnej variancie. Modrá čiara reprezentuje smer, kde je najviac informácií, tento smer je prvý hlavný komponent. Druhý hlavný komponent, ktorý je označený zelenou farbou, je smer variancie kolmý na smer prvého hlavného komponentu, kde je v tomto smere najväčší rozptyl dát. V 2D priestore existujú iba dva hlavné komponenty. Koľko dimnezií má priestor, toľko môže byť maximálne

[1] napríklad veľkosť bloku je 16 a posun bude po 8 pixelov

množstvo hlavných komponentov.



Obr. 11: HLavné komponenty v grafe [12]

Redukcia dimenzie Hlavným cieľom PCA je redukcia dimenzie. TO znamená, že dáta z 3D priestoru chceme reprezentovať v dvojrozmernom priestore. Princíp spočíva v tom, že ak máme 3D priestor s tromi hlavnými komponentami, môžme zarovnať osi tohto priestoru podľa hlavných komponentov, čím reprezentujeme tie isté dáta v inom súradnicovom systéme. Ak odstránime dimenziu, kde je tretí a teda najmenší hlavný komponent, stále nám ostanú dva hlavné komponenty s relevantnými dátami na ich reprezentáciu. Tento prístup napomáha pri vysokom počte dimenzií. Znížením dimenzie je možné filtrovať dátový šum a tak získať presnejší výsledok.

Vlastné vektory a hodnoty matice Vlastný vektor matice je vektor, ktorého smer sa pri násobení nemení. Teda platí $Av = \lambda v$ kde A je matica, v je vektor a λ je skalárna veličina (číslo).

Pri takomto násobení sa mení veľkosť vektora v a smer ostáva zachovaný.

Výpočet PCA Pre výpočet PCA je potrebné vložiť vytvorené dátové body do matice, v ktorej každý stĺpec reprezentuje jeden dátový bod. Teda pre trojrozmerný priestor, kde je n bodov bude mať matica 3 stĺpce a n riadkov ako na obrázku 12.

$$\mathbf{D} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ z_1 & z_2 & \cdots & z_n \end{bmatrix}$$

Obr. 12: 3D matica dát [12]

Po vytvorení takejto matice je potrebné vypočítať priemer všetkých dátových bodov,

teda koľko rozmerov, toľko priemerov. Tie sa vypočítajú ako súčet všetkých hodnôt v rozmere vydelený ich počtom. 13

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i$$
$$\mu_z = \frac{1}{n}\sum_{i=1}^{n} z_i$$

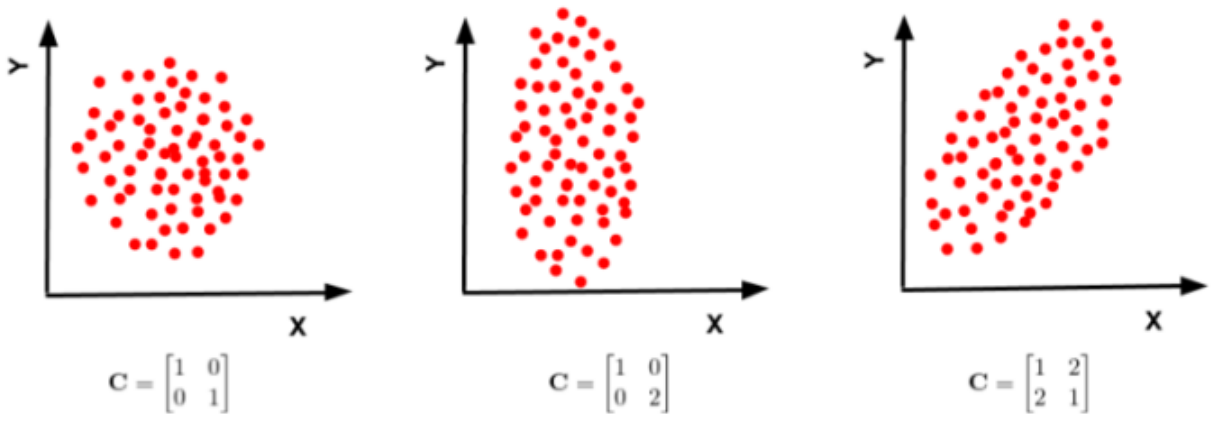
Obr. 13: Výpočet priemerov dátových bodov[12]

Na základe priemeru sa vytvorí matica, v ktorej hodnoty sa vypočítajú ako hodnota v matici D zmenšená o hodnotu priemeru 14. Po získaní tejto matice môžme získať jednoducho kovariančnú maticu, z ktorej vieme získať roztyl dát. Diagonála kovariančnej matice obsahuje variancie osí X, Y a Z, hodnoty mimo diagonály určujú kovarianciu medzi dvomi dimenziami (X, Y alebo Y, Z alebo Z, X).

$$\mathbf{M} = \begin{bmatrix} (x_1 - \mu_x) & (x_2 - \mu_x) & \cdots & (x_n - \mu_x) \\ (y_1 - \mu_y) & (y_2 - \mu_y) & \cdots & (y_n - \mu_y) \\ (z_1 - \mu_z) & (z_2 - \mu_z) & \cdots & (z_n - \mu_z) \end{bmatrix}$$

Obr. 14: 3D matica priemerov[12]

Kovariančná matica sa vypočíta ako $C = MM^T$, kde M^T je transponovan8 matica M. Rozmer matice C je m * m, kde m je počet dimenzií.



Obr. 15: Porovnanie kovariančných matíc pre rôzne rozptyly dát[12]

Na obrázku 15 môžme vidieť ako sa mení kovariančná matica v závislosti od rozptylu dát. Na prvej časti obrázku sú dáta rozptýlené rovnako všetkými smermi, teda kovariančná matica má na diagonále rovnaké hodnoty. Na strednom obrázku môžme vidieť, že hodnoty sú natiahnuté v smere osi y, diagonála kovariančnej matice nemá rovnaké hodnoty. Na poslednej časti obrázku sú hodnoty rozptýlené v 45-stupňovom uhle, diagonálne hodnoty kovariančnej matice sú rovnaké ako aj hodnoty mimo diagonály.[12]

2.3.4 Support Vector Machine

SVM je metóda strojového učenia s učiteľom, je vhodná na používanie pri klasifikácii alebo regresnej analýze. Táto metóda na základe trénovacích vzoriek, ktoré sú priradené do jednej z dvoch tried vytvorí model. Tento model je následne schopný priraďovať nové vzorky do jednej z týchto tried, čo z neho robí nepravdepodobnostný binárne lineárny klasifikátor. SVM model reprezentuje vzorky ako body v priestore, tak, že mapuje jednotlivé vstupy do kategórií, ktoré sú v tomto priestore jasne oddelené. Klasifikátor priraďuje nové vzorky podľa toho, ku ktorej z tried majú bližšie v rámci tohto priestoru.

Okrem lineárnej klasifikácie je SVM schopné efektívne riešiť aj nelineárne separovateľné problémy pomocou takzvaného jadrového triku *(angl. Kernel trick)* tým, že rozšíri dimenzionalitu dát. Pokiaľ dáta nie sú pri trénovaní označené, do ktorej kategórie patria, učenie s učiteľom nie je možné, avšak je možné učenie bez učiteľa tak, že algoritmus sa snaží nájsť prirodzené zoskupenie dát a tak ich rozdeliť do kategórií.

Medzi najčastejšie používané jadrové funkcie patria:

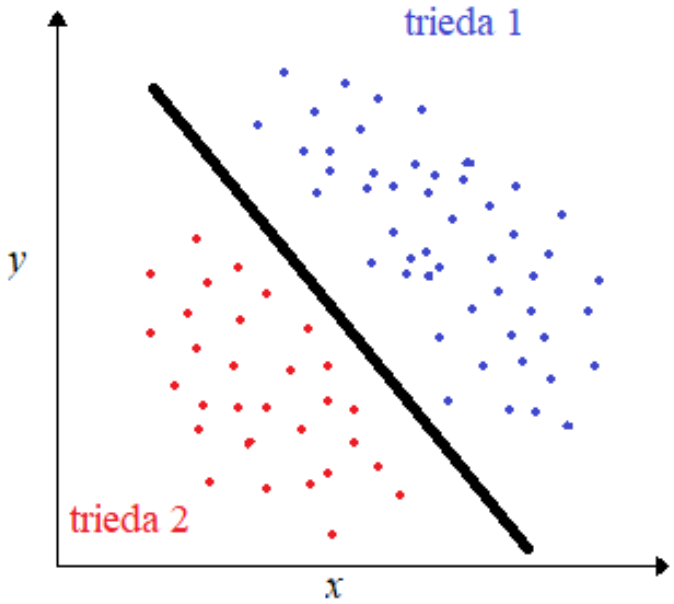
- **Polynóm stupňa p**
- $K(x, y) = (x.y + 1)^p$
- **Radiálne bázové funkcie**
- $K(x, y) = e^{(-\|x-y\|^2/2\sigma^2)}$
- **Dvojvrstvová neurónová sieť**
- $K(x, y) = \tanh(\kappa x.y - \delta)$

SVM algoritmus vytvorili Hava Siegelmann a Vladimir Vapnik. Tento algoritmus má široké využitie a je používaný v priemyselných aplikáciách, počítačovom videní, umelej inteligencii. [13]

Definícia algoritmu SVM vytvára hyperrovinu [2], alebo viacero hyperrovín, vo vysokorozmernom priestore, ktorý môže byť použitý na klasifikáciu, regresiu alebo iné úlohy. Dobrá separácia sa dosiahne vtedy, keď hyperrovina má veľkú vzdialenosť od najbližšieho trénovacieho bodu akejkoľvek triedy. Teda platí, čím je väčšia vzdialenosť hyperroviny od trénovacích údajov, tým je nižšia chyba klasifikátora pri určovaní triedy testovacej vzorky.[14]

Keďže nie všetky problémy je možné lineárne separovať, bolo navrhnuté, aby sa priestor, ktorý SVM spracováva rozvrhol do viacrozmerného priestoru, čo uľahčí rozdelenie jednotlivých dát do tried. Pre zanechanie výpočtovej rýchlosti je mapovanie v SVM navrhnuté v pôvodnom priestore, kde sa následne zvolí vhodná jadrová funkcia $k(x, y)$. Hyperroviny vo viacrozmernom priestore sú definované ako súbory bodov, kde ich vektory majú konštantnú dĺžku. Následne body z priestoru príznakov sú mapované do hyperroviny.

Lineárny klasifikátor Najjednoduchšou možnosťou použitia SVM je prípad lineárneho klasifikátora pre dáta, ktoré je možné lineárne rozdeliť do dvoch tried pomocou hyperroviny. Príklad takéhoto problému je na obrázku 16.

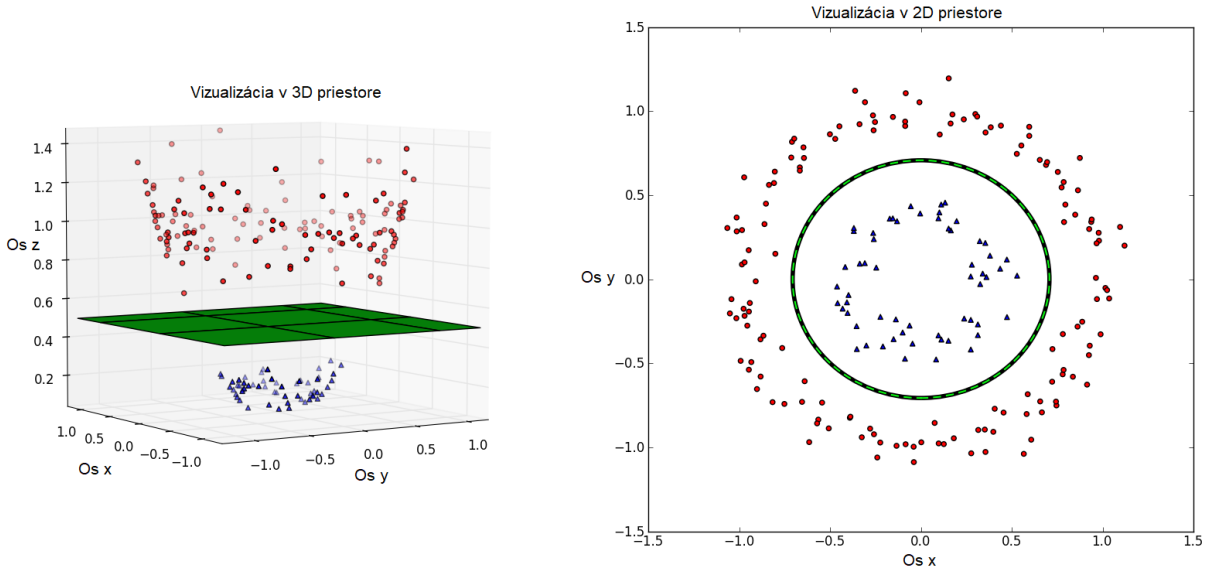


Obr. 16: Príklad lineárne separovateľného problému

Nelineárny klasifikátor Na obrázku č. 17 môžme vidieť lineárne neseparovateľný problém dvoch množín, ktorý sa po pridaní tretej dimenzie stáva lineárne separovateľný. Ta-

[2]Podpriestor, ktorý má o jednu dimenziu menej ako priestor, v ktorom sa nachádza.

kýmto spôsobom dokážeme s SVM zatriediť jednotlivé videá z množín do správnych tried. Pomocou jadrového triku vieme lineárne separovať aj lineárne neseparovateľné problémy.[14]



Obr. 17: Rozšírenie dimenzie v SVM

3 Návrh riešenia

Po oboznámení s teoretickými piliermi tejto práce prejdeme ku konkrétnemu riešeniu témy a implementácii, ktorú sa nám podarilo vytvoriť. Postupne prejdeme jednotlivé kroky vytvárania programu od analýzy datasetov, načítavania a ukladania videosekvencií do obrazových formátov. Ďalej si ukážeme metódy a spôsoby, ktorými sme následne vzniknuté dáta spracovali pre potrebu výberu príznakov a zatriedenia do jednotlivých tried. Fungovanie programu priblížime aj časťami kódu s popisom, na čo slúži a ako pracuje. Súčasťou tejto kapitoly bude aj krátky popis zmien, ktoré sme počas návrhu tejto implementácie vykonali a samozrejme aj ich dopad na finálne riešenie.

3.1 KTH Databáza

Pre naše riešenie sme vyberali z viacerých datasetov, ktoré boli popísané v kapitole 2.2.2. Nakoniec sme sa rozhodli pre databázu KTH, v ktorej máme 6 tried pohybu a nachádza sa pri každom z nich 100 videí. Naše rozhodnutie sme učinili takto z viacerých dôvodov. Potrebovali sme dataset, v ktorom sa nachádza veľké množstvo videí, zároveň vo videách sme potrebovali statickú kameru so statickým pozadím, aby sme predišli zvýšenému šumu. V tejto databáze sú pohyby:

- Boxovanie
- Mávanie rukami
- Tlieskanie rukami
- Beh
- Pomalý beh
- Chôdza

Pri analýze snímkov sme zistili, že dĺžka videí je od ***8 do 59 sekúnd***. Čo pre dané typy pohybov je postačujúci čas na to, aby sme z nich dokázali extrahovať dôležité informácie, ktoré sa tohto pohybu týkajú. Počet snímkov za sekundu v každom z videí je 25, z tohto dôvodu sme usúdili, že aj v prípade krátkeho videa s dĺžkou 8 sekúnd budeme mať k dispozícii minimálne ***200*** snímkov. Ďalším dôležitým faktom je, že sa budeme musieť popasovať s podobnosťou pohybov behu, pomalého behu a chôdze, keďže sú veľmi podobné, a ich rozdiel je iba v rýchlosti vykonávania. Je nutné dodať, že v každom z videí sa jednotlivé typy pohybov opakovali viacnásobne, z čoho sme už vopred usudzovali, že

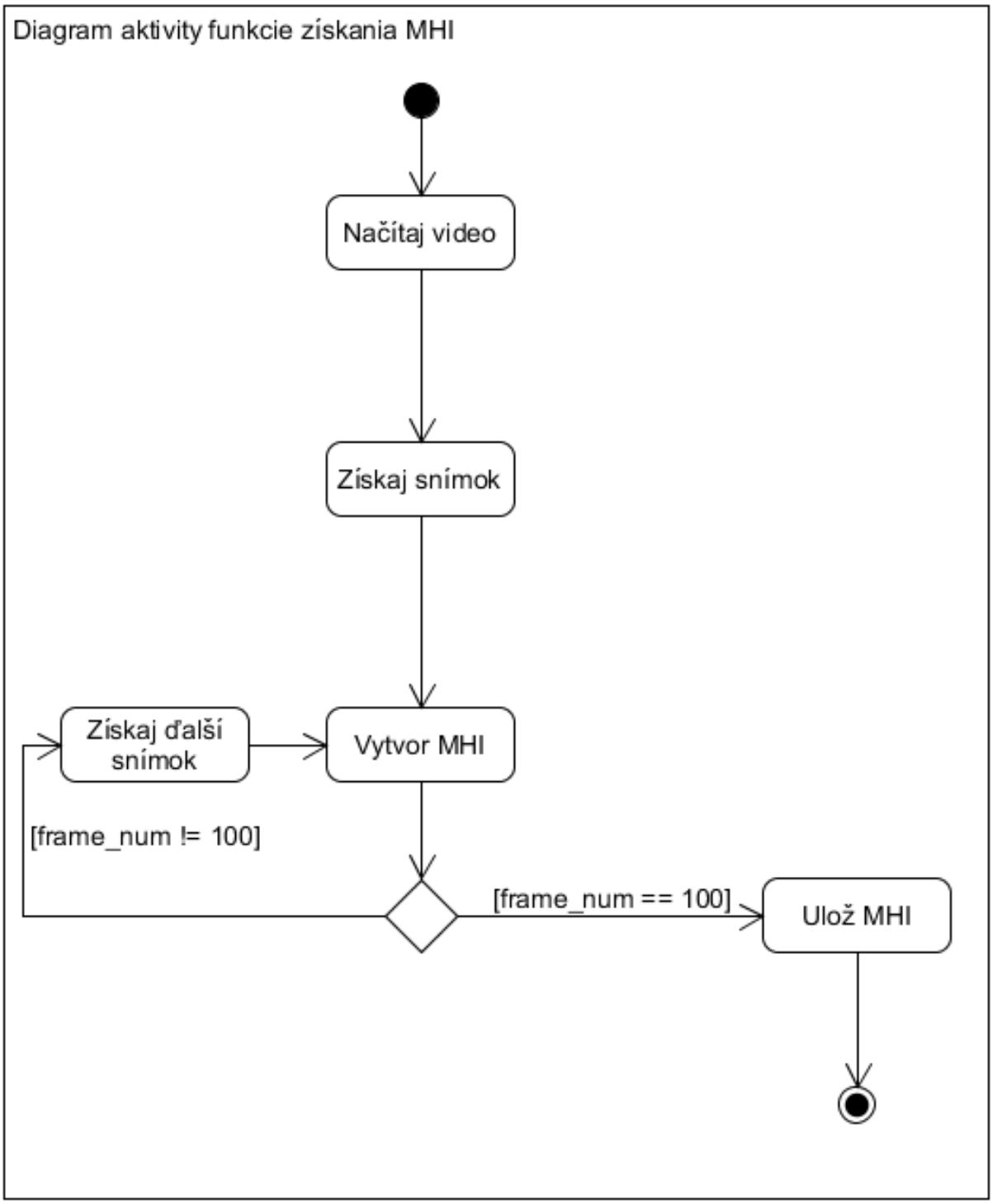
na spracovanie nám bude stačiť iba časť videa, čo nám ušetrí čas spracovania dát počas behu programu.

Obr. 18: Jednotlivé pohyby v datasete KTH [15]

Na obrázku 18 vidíme všetky kategórie pohybov, jednotlivé pohyby vykonáva viacero osôb rôznymi spôsobmi, teda je medzi týmito pohybmi aj diverzita, ktorá je určite vhodná na trénovanie pohybu.

3.2 Predspracovanie videa

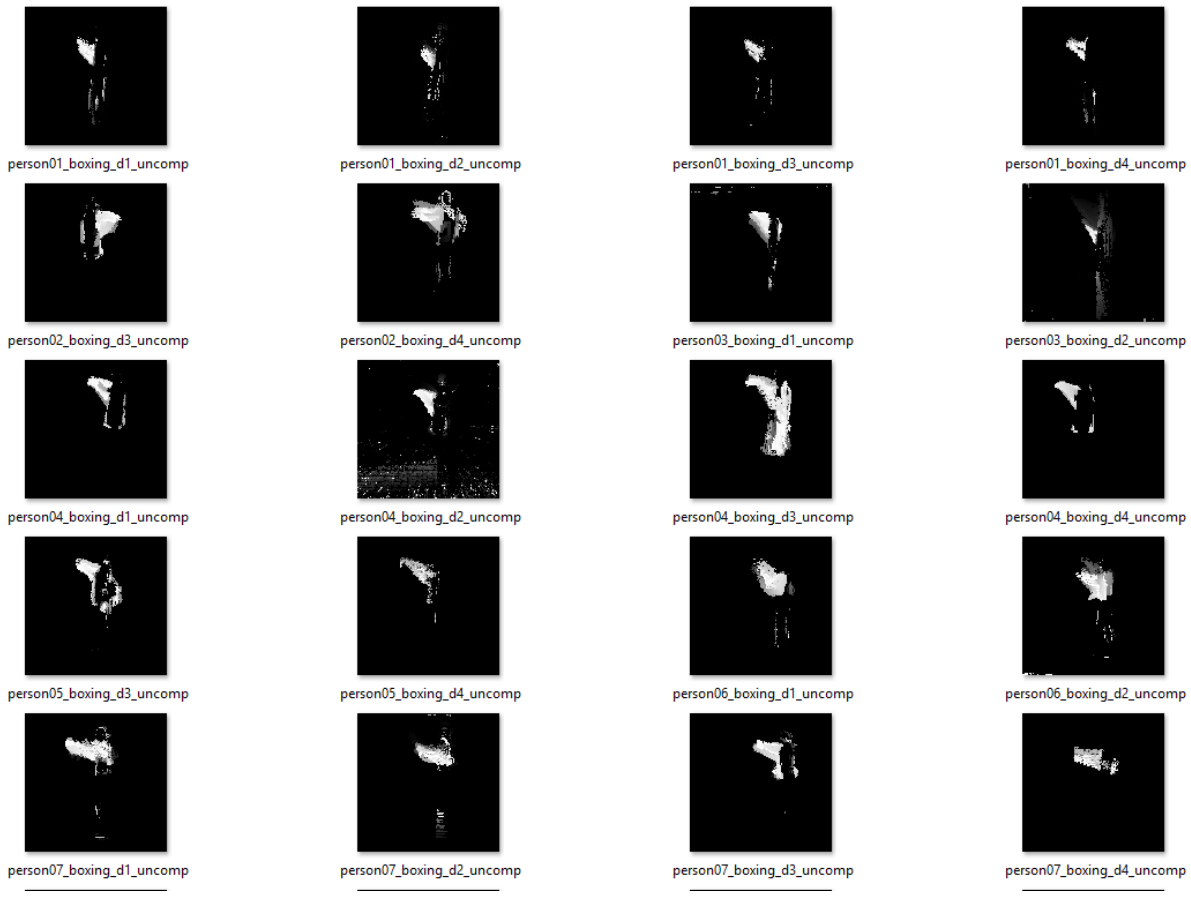
Na predspracovanie videa a vytiahnutie prvotných príznakov sme sa rozhodli z našich videí vytvoriť obrázky, na ktorých sa bude nachádzať história pohybu. Algoritmus MHI, ktorý sme pri tom použili sme popísali v časti 2.3.1. Tento algoritmus sme zvolili na základe toho, že máme databázu videí bez pohyblivého pozadia, teda predídeme šumom na pozadí a budeme sa môcť venovať iba pixelom, ktoré menia svoju pozíciu v rámci videa. Presne to je postačujúca podmienka na to, aby sme bez väčších problémov mohli implementovať funkcionalitu algoritmu MHI na nami zvolený dátový rozsah. Vytvoriť MHI sme sa rozhodli zo všetkých videí, aby sme následne videli, akým spôsobom prebehlo spracovanie, či je možné z MHI vyčítať pohyb a jeho základné znaky.



Obr. 19: Diagram aktivity získania MHI

V tejto ukážke 19 diagramu aktivít je znázornený proces získavania MHI snímky z našej implementácie. Prvým krokom je načítanie videa, z ktorého následne vyberáme

po jednom obrázku. Z každého obrázku vytvárame MHI z predchádzajúcich snímkov. V tomto konkrétnom prípade opakujeme cyklus dovtedy, kým nezískame MHI zo snímku číslo 100, ktoré predstavuje štvrtú sekundu pohybu (25 snímkov za sekundu). V prípade, že nám príde snímok číslo 100, prejdeme na jeho uloženie na lokálny disk, kde bude pripravený na ďalšie spracovanie.



Obr. 20: Predspracované videá boxu.

Na obrázku 21 môžme vidieť príklad už predspracovaných MHI prostredníctvom nášho algoritmu. Takto sme si vytvorili všetkých 600 obrázkov z pohybov na ďalší výber príznakov, ktoré sme uložili podľa typu pohybu do jednotlivých priečinkov s ich názvami.

3.3 Výber príznakov

Keďže už máme predspracovaný obraz vo forme MHI, môžme pokračovať v extrahovaní príznakov z týchto obrazov. Na tento účel sme zvolili deskriptor HOG popísaný v časti 2.3.2. Týmto spôsobom vytvoríme vektor príznakov, ktorý budeme môcť následne spracovať prostredníctvom PCA a tak ho dať trénovať do klasifikátora pre každý jeden pohyb z trénovacej množiny. Príznaky z každého MHI obrázku by mali byť v tvare vektora

konštantnej dĺžky.

3.3.1 Príznaky HOG

V našom algoritme na získanie HOG príznakov vstupuje do funkcie *hogCreate* parameter *image* je vlastne obrázkom, ktorý je výstupom funkcie *imread* z knižnice OpenCV. Pre dobré vybratie príznakov je potrebné nastaviť a vytvoriť *HOGDescriptor*, ktorý obsahuje viacero parametrov:

- winSize
- blockSize
- blockStride
- cellSize
- nbins
- gammaCorrection
- nlevels

winSize parameter určuje veľkosť okna na detekciu, musí byť zarovnaný podľa *blockStride* a *blockSize*.

blockSize je veľkosť bloku v pixeloch. Má byť zarovnaný na veľkosť bunky *cellSize*.

blockStride musí byť násobkom veľkosti bunky, znamená veľkosť kroku, ktorým sa blok bude posúvať.

cellSize je veľkosť bunky.

nbins je počet rozdelení uhlov do ktorých jendotlivé gradienty zapadajú.

gammaCorrection hodnota určuje, či sa má použiť predspracovanie gamma úpravy alebo nie.

nlevels určuje maximálny počet detekčných okien, predvolená hodnota je 64.

Po takomto nastavení deskriptora HOG môžme následne spracovať vstupný obraz a vytvoriť tak histogram hodnôt, ktoré už sú reprezentovateľné príznaky, ktoré vložíme do vektora príznakov *arrVectorFinal*. Konečným výstupom je teda pole vektorov, ktoré obsahuje príznaky extrahované zo všetkých testovacích videí. Už tento vektor vieme následne použiť pre trénovanie klasifikátora, avšak pre zlepšenie výsledkov ešte tieto dáta upravíme. Neskôr si ukážeme ako sme vektor týchto príznakov zložili z troch MHI obrázkov z jedného videa, ktoré boli spracované vo štvrtej, šiestej a deviatej sekunde, čo dokáže presnejšie identifikovať pohyb jednotlivých aktérov.

3.3.2 Prídavné príznaky

Ďalšími príznakmi, ktoré sme sa rozhodli vybrať do vektora príznakov sú priemery hodnôt jednotlivých polí pixelov. Týmto algoritmom sme chceli znovu navýšiť úspešnosť, keďže pohyby v jednotlivých triedach sa vykonávajú v rôznych častiach snímaného obrazu. Týmito hodnotami vieme určiť v ktorých častiach obrazu MHI sa vykonával pohyb a v akej intenzite. Pre tento algoritmu sme potrebovali zvoliť vhodné parametre:

- Šírka okna
- Výška okna

Šírka okna nám udáva počet pixelov, ktoré budeme spracovávať v jednom rade pixelov v iterácii. **Výška okna** je potrebná na definovanie počtu pixelov spracovávaných v stĺpci pixelov v iterácii.

Tieto dva parametre je potrebné určiť ako delitele šírky (160px) a delitele výšky (120px) MHI obrázkov. V našej implementácii sme využili šírku a výšku s hodnotou 10.

V nami vtvorenom algoritme je vo funkcii na získanie prídavných príznakov na vstupe obrázok MHI a počet okien pixelov s rozmermi 10 x 10 pixelov na jeden obrázok. Obraz sme si pre zjednodušenie vložili do *NumPy* poľa a následne iterovali. Aby sme získali normalizovaný príznak *(z rozmedzia <0,1>)*, potrebovali sme vydeliť hodnotu príznaku z jedného okna hodnotou *255*, keďže každý pixel nadobúda hodnoty*<0,255>* podľa svietivosti pixelu.

$$X = \frac{\sum_{i=1}^{pocet\_pixelov} hodnota\_pixelu_i}{255}$$

$X - výsledný\ príznak\ jedného\ okna\ pixelov$

$pocet\_pixelov - počet\ pixelov\ v\ okne$

$hodnota\_pixelu_i - hodnota\ farby\ pixelu\ na\ i - tej\ pozícii$

Obr. 21: Výpočet príznaku okna pixelov

Takto vypočítané príznaky sme pridávali do vektoru príznakov na následné spracovanie a úpravu v PCA.

3.3.3 Spracovanie pomocou PCA

V algoritme 1 na extrakciu komponentov, ktorú sme si popísali v časti 2.3.3 sme použili metódy triedy ***PCA*** knižnice *sklearn*. Pre vytvorenie objektu PCA, sme použili parameter *n_components*, ktorý udáva počet komponentov, koľko chceme vybrať z vektora

príznakov *arrVectorFinal*. Tento počet komponentov musí byť nižší alebo nanajvýš rovný počtu vektorov príznakov. Natrénovaný model PCA s vektorom príznakov *arrVectorFinal* sme uložili pre ďalšie spracovanie v SVM.

```
pca = PCA(n_components)
pca.fit(arrVectorFinal)
joblib.dump(pca, 'ModelPCA.pkl')
```

Listing 1: Trénovanie PCA

Trénovacia množina obsahuje 240 videí *(6 tried po 40 videí)*, preto je aj maximálny počet hlavných komponentov, ktoré môžme získať z vektora príznakou pomocou PCA je 240. Teda výsledný vektor príznakov sa zmenšil na tie príznaky, ktoré sú najdôležitejšie. Neskôr v časti 3.5 si ukážeme spôsoby, ktorými sme sa snažili naše výsledky ešte viac zlepšiť a vektor PCA rozšíriť.

3.4 Klasifikácia

Pre natrénovanie klasifikátora sme sa rozhodli použiť SVM, ktorý sme popísali v časti 2.3.4. Na implementovanie SVM sme použili triedu *svm* z knižnice *sklearn*. K dátam, ktoré máme uložené v *arrVectorFinal* potrebujeme ešte označenie, ktorý vektor do akej kategórie patrí. Preto sme vytvorili ďalší vektor, v ktorom je pre každé video označenie kategórie, kam vektor patrí. Teraz už môžme natrénovať SVM.

```
clf = svm.SVC(decision_function_shape='ovr')
clf.fit(arrVectorFinal, classVector)
joblib.dump(clf, 'Model.pkl')
```

Listing 2: Trénovanie klasifikátora

Algoritmus 2 využíva volanie funkcií *svm.SVC*, ktoré nám vytvorí klasifikátor *(angl. Support Vector Classifier)*, rozhodovaciu funkciu sme zvolili *ovr (angl. one versus rest)*, keďže máme viac ako dve triedy pohybov. Následne prostredníctvom funkcie *fit* sme natrénovali klasifikátor *clf*, kde boli vložené vektory príznakov *arrVectorFinal* a vektor tried *classVector*. Takto natrénovaný model bolo potrebné uložiť pre ďalšie použitie a testovanie.

3.5 Priebežné testovania a ladenie

Po naprogramovaní všetkých základných častí nášho projektu sme potrebovali otestovať naše riešenie, či náš klasifikátor funguje dobre a či môžme prejsť ku finalizácii výsledkov, prípadne k ďalším vylepšeniam a zmenám.

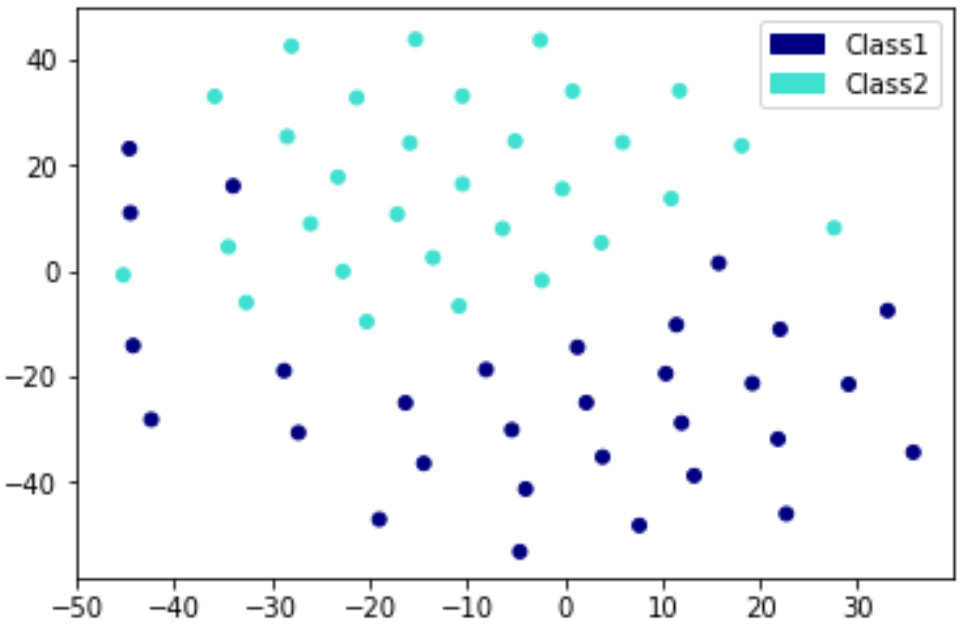
3.5.1 t-SNE

Aby sme vedeli výsledok trénovania našej SVM, potrebovali sme výsledky vizualizovať, ako prebehlo rozdelenie do tried. Na to sme využili metódu založenú na redukcii dimenzií *t-Distributed Stochastic Neighbor Embedding (t-SNE)*, ktorej algoritmus nájdete v prílohe B. Tento algoritmus umožňuje vizualizáciu viacrozmerných dát do 2D priestoru. Túto techniku je možné implementovať pomocou aproximácií, ktoré môžu byť aplikované na datasety obrovských rozmerov. Autorom tohto algoritmu je ***Laurens van der Maaten***, ktorý pôsobí ako výskumný pracovník na poli strojového učenia a počítačového videnia. t-SNE implementoval pre množstvo jazykov vrátane jazyka Python, takže sme nemali problém tento nástroj využívať. [16]

3.5.2 Prvotné testy

Ako prvý krok sme zvolili testovanie a tréning dvoch množín a teda beh a boxovanie. SVM sme natrénovali na tridsiatich videách z triedy behu a tridsiatich videách z triedy boxovania.

Výsledok trénovania bol pre nás dobrý, keďže už vizuálne bolo vidno rozdelenie týchto pohybov do dvoch tried. Výsledok je možné vidieť na obrázku 22.

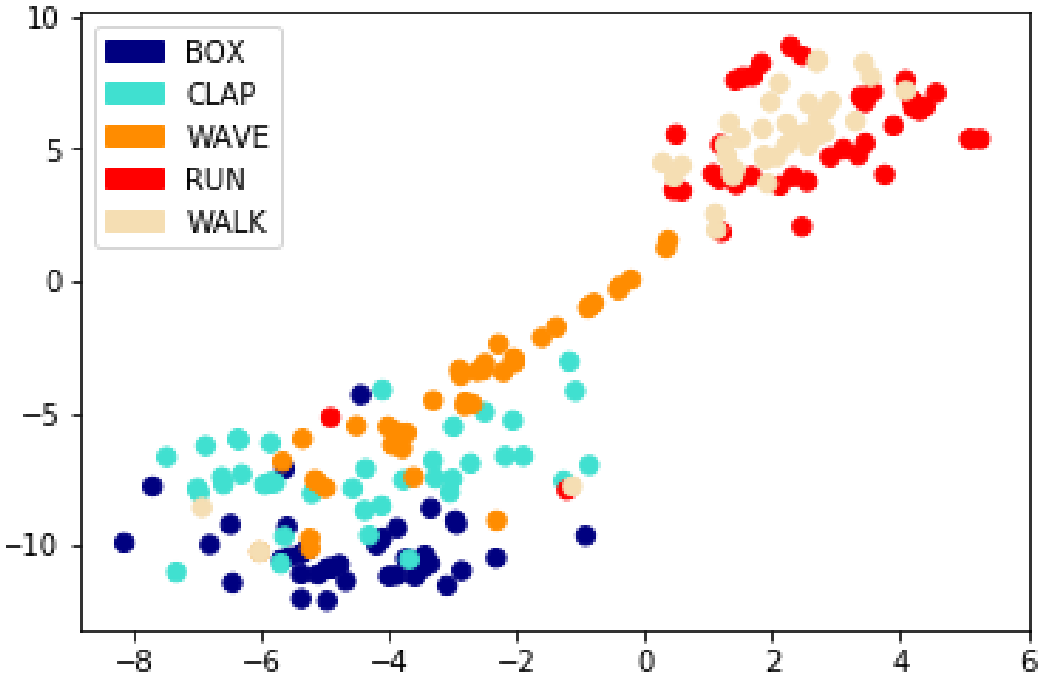


Obr. 22: Test dvoch tried pohybu

Ako príklad uvedieme taktiež pokus trénovania s jedným MHI, ktorý bol vytvorený vo 4. sekunde priebehu pohybu, avšak bez získania prídavných príznakov 3.3.2 a využitia

metódy PCA. Do tohto testu sme zapojili 5 tried pohybov, teda dataset KTH bez triedy pomalého behu, keďže sme usudzovali, že táto trieda sa dosť prelína medzi triedami behu a chôdze, čo sa ukáže aj na ďalších výsledkoch.

Takto natrénované SVM sme znovu podrobili testu *t-SNE*, kde výstup vyzeral ako na obrázku 23, výsledky síce dávali zmysel, avšak rozdelenie jednotlivých tried nebolo až tak prehľadné ako sme si predstavovali. Triedy behu a chôdze pôsobia veľmi chaoticky, tak ako aj boxovanie s tlieskaním a zasahovaním kývania.



Obr. 23: Prvý test piatich tried pohybu

Skúsili sme teda otestovať túto natrénovanú SVM, akú bude mať úspešnosť rozpoznávania. Výsledky môžte vidieť na obrázku 24.

	BOXING	CLAPPING	WAVING	RUNNING	WALKING
BOXING	62,5%	22,5%	10,0%	10,0%	10,0%
CLAPPING	10,0%	60,0%	25,0%	5,0%	0,0%
WAVING	20,0%	15,0%	52,5%	10,0%	5,0%
RUNNING	2,5%	0,0%	7,5%	62,5%	25,0%
WALKING	5,0%	2,5%	5,0%	12,5%	60,0%

Obr. 24: Úspešnosť 5 tried z 1 MHI

Z tohto dôvodu sme sa rozhodli analyzovať náš postup a pristúpiť k optimalizácii riešenia, zbieraniu a ladeniu príznakov za pomoci viacnásobného využitia MHI, prídavných príznakov 3.3.2 a aplikovania PCA, ako bolo spomenuté v častiach 3.3.1, 3.3.3, 3.3.2.

3.5.3 Rozdelenie dát a finálne testy

Dáta na tréning a testovanie sme rozdelili do dvoch skupín v pomere 40 testovacích videí a 40 trénovacích videí. Všetky testy boli vykonané na rovnakých videách, takže výsledky úspešnosti sú objektívne a porovnateľné pre každý typ konfigurácie programu. Konfigurácie sme menili na 9 rôznych nastavení pre všetky triedy pohybov databázy KTH viď obrázok 25.

Číslo testu	Počet MHI	Počet PCA	Prídavné príznaky	Počet príznakov
1.	1	0	0	3528
2.	1	0	1	3720
3.	1	1	1	240
4.	3	1	3	240
5.	3	2	3	480
6.	3	4	3	960
7.	3	8	3	1920
8.	3	12	3	2880
9.	3	16	3	3840

Obr. 25: Testovacie konfigurácie

Ako vidno na obrázku, zvolili sme obmeny počtu MHI obrázkov s násobnými počtami vektorov pre získanie hlavných komponentov ako aj počet prídavných príznakov vypočítavaných z MHI.

1. Test V prvom teste s konfiguráciou jedného obrázku MHI, bez výpočtu PCA a prídavných príznakov sme dostali prvé výsledky 26.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	22,5%	10,0%	0,0%	10,0%	10,0%
CLAPPING	10,0%	60,0%	25,0%	7,5%	5,0%	0,0%
WAVING	20,0%	15,0%	52,5%	5,0%	10,0%	5,0%
JOGGING	7,5%	0,0%	10,0%	60,0%	40,0%	35,0%
RUNNING	0,0%	0,0%	0,0%	17,5%	32,5%	5,0%
WALKING	0,0%	2,5%	2,5%	10,0%	2,5%	45,0%

Obr. 26: Test č. 1

Z výsledkov je viditeľný problém medzi triedou *Running* a *Jogging*, kde zo všetkých testovaných videí behu vyhodnotilo až 40% ako pomalý beh a 35% chôdze vyhodnotilo ako pomalý beh.

2. Test V druhom teste sme pridali ďalšie príznaky do vektora, výsledky pohybov sa zlepšili až na triedy behu, kde bol prepad až na úspešnosť 7,5% a triedy behu, kde sa tiež znížila úspešnosť. Porovnanie je na obrázku 27.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	20,0%	10,0%	2,5%	12,5%	7,5%
CLAPPING	10,0%	70,0%	22,5%	5,0%	5,0%	2,5%
WAVING	20,0%	7,5%	55,0%	5,0%	10,0%	5,0%
JOGGING	7,5%	2,5%	12,5%	80,0%	62,5%	37,5%
RUNNING	0,0%	0,0%	0,0%	0,0%	7,5%	10,0%
WALKING	0,0%	0,0%	0,0%	7,5%	2,5%	37,5%

Obr. 27: Test č. 2

3. Test V tomto teste sme sa rozhodli zistiť vplyv PCA na celkové riešenie problému. Výsledky ukázali všeobecné zlepšenie, hlavne v triede mávania rukami, kde sa úspešnosť zvýšila z 55% na 72.5%.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	15,0%	7,5%	2,5%	10,0%	7,5%
CLAPPING	10,0%	77,5%	12,5%	5,0%	5,0%	2,5%
WAVING	20,0%	5,0%	72,5%	5,0%	10,0%	10,0%
JOGGING	7,5%	2,5%	7,5%	77,5%	65,0%	30,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	7,5%	10,0%
WALKING	0,0%	0,0%	0,0%	10,0%	2,5%	40,0%

Obr. 28: Test č. 3

4. Test V nasledujúcich testoch nás zaujímal vplyv trojnásobného použitia obrázkov MHI na výsledky a úspešnosť testovania v spolupráci s rozširovaním PCA. Použili sme MHI z prvých 4, 6 a 9 sekúnd videa. Objavilo sa nepatrné zlepšenie až na prepad úspešnosti triedy *Jogging*, v ktorom úspešnoť od predchádzajúceho testu klesla o 17,5%. Finálny počet príznakov na jeden vektor bol 240.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	17,5%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	77,5%	15,0%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	75,0%	7,5%	10,0%	5,0%
JOGGING	2,5%	0,0%	2,5%	60,0%	52,5%	5,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 29: Test č. 4

5. Test V nasledujúcom teste sme iba zdvojnásobili počet vektorov príznakov videí (480 príznakov na video), aby sme mohli pomocou PCA vypočítať dvojnásobný počet príznakov ako v predchádzajúcom teste. Z výsledkov nám vyplynulo, že tento postup rozširovania počtu vektorov sa nám môže vyplatiť, keďže sa nám výsledky nezhoršili a v triede *Waving* sme dokonca dostalo výsledok lepší o 2,5%.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	17,5%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	77,5%	12,5%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	77,5%	7,5%	10,0%	5,0%
JOGGING	2,5%	0,0%	2,5%	60,0%	52,5%	5,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 30: Test č. 5

6. Test Test so štvornásobne zvýšeným počtom vektorov (960 príznakov na video) nám znovu priniesol zlepšenie v triede *Waving* a to dokonca o celých 5%. Vzhľadom na to, že sa nám výsledky stále zlepšovali, snažili sme sa ísť ešte ďalej.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	17,5%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	77,5%	10,0%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	82,5%	7,5%	10,0%	5,0%
JOGGING	2,5%	0,0%	0,0%	60,0%	52,5%	5,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 31: Test č. 6

7. Test Siedmy test s osemnásobným zväčšením počtu vektorov (1920 príznakov na video) nám znovu vylepšil výsledky o 2,5% pre triedy *Clapping* a *Waving*. Stále to nemalo žiaden negatívny vplyv na ostatné výsledky.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	15,0%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	80,0%	7,5%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	85,0%	7,5%	10,0%	5,0%
JOGGING	2,5%	0,0%	0,0%	60,0%	52,5%	5,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 32: Test č. 7

8. Test Pri teste, kde sme zvolili 12-násobné rozšírenie (2880 príznakov na video) počtu vektorov sa nám už zlepšovanie výsledkov zastavilo. Pri testoch sme namerali identické hodnoty, čo nám naznačovalo, že ďalšie zvyšovanie počtu vektorov nám už nebude naše výsledky vylepšovať. Týmto by sme mohli konfiguráciu, ktorá bola nastavená v tomto teste, respektíve v teste predchádzajúcom, považovať za najvhodnejšie zvolenú a porovnať ju s inými riešeniami.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	15,0%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	80,0%	7,5%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	85,0%	7,5%	10,0%	5,0%
JOGGING	2,5%	0,0%	0,0%	60,0%	52,5%	5,0%
RUNNING	0,0%	0,0%	0,0%	0,0%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 33: Test č. 8

9. Test Ako sme sa predpokladali, výsledky pri 16-násobnom rozšírení (3840 príznakov na video) sa vôbec nezlepšili, naopak, v triede *Jogging* sme zaznamenali pokles o 2,5% z úspešnosti.

	BOXING	CLAPPING	WAVING	JOGGING	RUNNING	WALKING
BOXING	62,5%	15,0%	7,5%	0,0%	0,0%	7,5%
CLAPPING	12,5%	80,0%	7,5%	2,5%	2,5%	2,5%
WAVING	22,5%	5,0%	85,0%	7,5%	12,5%	5,0%
JOGGING	2,5%	0,0%	0,0%	57,5%	50,0%	5,0%
RUNNING	0,0%	0,0%	0,0%	2,5%	12,5%	5,0%
WALKING	0,0%	0,0%	0,0%	30,0%	22,5%	75,0%

Obr. 34: Test č. 9

Pri tomto výsledku môžme konštatovať, že výsledky sú uspokojujúce až na veľmi nízku úspešnosť triedy *Running*, ktorá sa až na 52,5% obmieňa s triedou *Jogging* a na 22,5% s triedou *Walking*. Pre naše riešenie to nepovažujeme za veľký problém vzhľadom na to, že pohyby v ktorých sa náš algoritmus mýlil a zamieňal si ich, sú veľmi podobné, rozdiel je iba v rýchlosti jednotlivých aktérov na videu, kde niektorí pri pomalom behu pomaly kráčali a naopak.

4 Porovnanie s existujúcim riešením

Pre porovnanie nášho riešenia sme si vybrali prácu, v ktorej pracovali s tou istou databázou videí ako my v našej práci. V porovnávanom riešení využívajú lokálne časovo-priestorové príznaky. Tieto pohybové príznaky následne trénujú pomocou SVM.

Na testovanie úspešnosti SVM využili viacero scenárov:

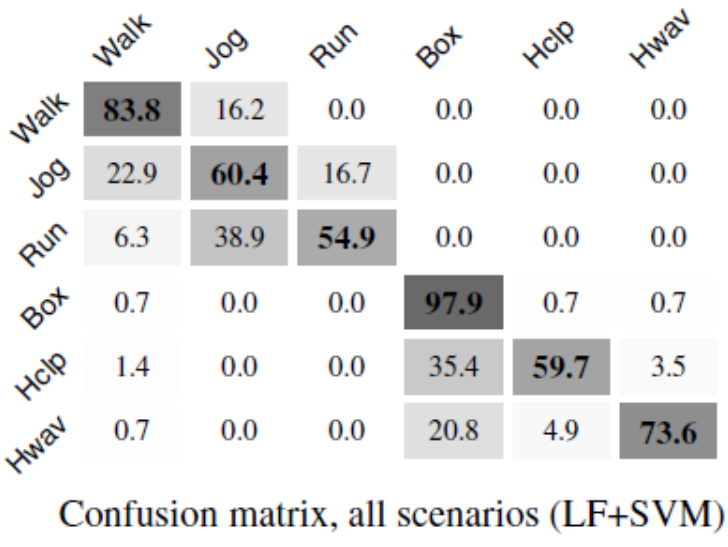
- **S1 - vonkajšie prostredie**
- **S2 - vonkajšie prostredie s variabilnou vzdialenosťou**
- **S3 - vonkajšie s rôznym oblečením**
- **S4 - vo vnútri**

4.1 Výsledky

V práci majú reprezentované dva výsledky. Priemer výsledkov pre všetky štyri scenáre a pre scenár S2.

4.1.1 Všetky scenáre

Na obrázku 35 sú spriemerované výsledky všetkých scenárov.



Obr. 35: Výsledky všetkých scenárov

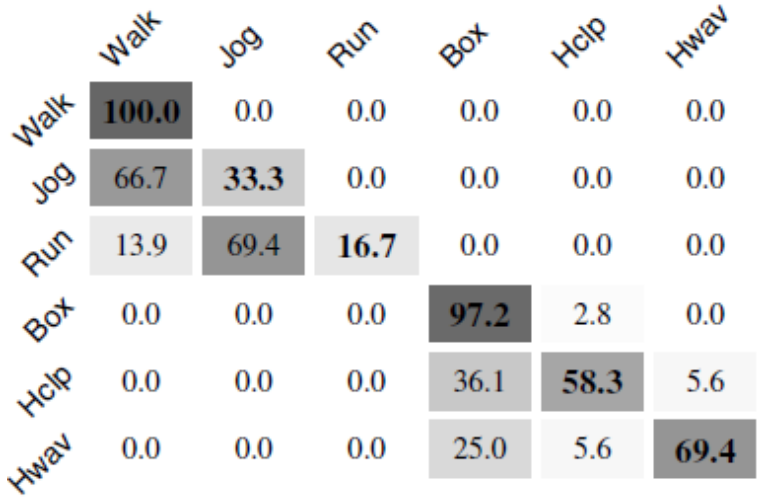
Z výsledkov môžme usúdiť, že všetky kategórie im zaradilo s viac ako 50%-tnou úspešnosťou. Oproti nášmu riešeniu sa im podarilo lepšie vyhodnotiť pohyb behu o 42,4%, boxu o 35.4%, chôdze o 8,8% a pomalý beh o zanedbateľných 0.4%.

Naopak zhoršenie oproti nášmu výsledku sme zaznamenali pri pohybe potlesku - o 20,3% a kývaniu oboma rukami o 11,4%.

Tieto výsledky pôsobia celkom slušne, vzhľadom na to, že neobsahujú vysokú chybu pri pohybe behu ako v našom riešení.

4.1.2 Scenár S2

V tomto teste testovali na jednom scenári vo vonkajšom prostredí. Výsledky sú na obrázku 36.



Obr. 36: Výsledky scenára S2

V týchto výsledkoch môžme skonštatovať, že naše výsledky boli nižšie v pohyboch boxu o 34,7%, pohybe behu o 4,2% a pohybe chôdze o 25%, kde sa im podarilo správne rozpoznať všetky testovacie videá.

Na opačnej strane v našich výsledkoch sa nám viac darilo pri pochyboch tlieskania o 21,7%, pohybe kývania o 15,6%, pohybe pomalého behu o 26,7%.

V oboch prípadoch sme zaznamenali vysoký podiel zamieňania pohybu behu s pomalým behom, respektíve chôdzou. Taktiež vysoké percento pomalého behu bolo zaregistrované ako chôdza či v našom, alebo ich riešení.

Zamieňanie týchto pohybov si môžme odôvodniť ich podobnosťou, keďže každá osoba beží, respektíve chodí inou rýchlosťou. To, čo je u niekoho pomalým behom je pre iného chôdzou. Týmito argumentami si zastávajú názor aj v porovnávanom riešení.

4.1.3 Zhodnotenie výsledkov

Výsledky, ktoré sme dosiahli sú uspokojivé, vzhľadom na metódy, ktoré sme použili môžme prehlásiť, že sa nám podarilo vytvoriť klasifikátor, ktorý dokáže rozpoznať

jednotlivé typy pohybov na základe ich časovo-priestorových vzťahov.

Vidíme aj priestor na vylepšenie riešenia, ktoré by mohlo byť predmetom ďalšieho skúmania.

Záver

Cieľom našej práce bolo vytvoriť systém, ktorý dokáže detekovať a rozpoznať pohyb osoby zo statického záznamu.

Zo začiatku sme sa snažili popísať teoretické vlastnosti a princípy rozpoznávania a detekcie ľudského pohybu, jednotlivé metódy a algoritmy, ako aj databázy videí na priblíženie tejto problematiky a našej konkrétnej implementácie.

Na naše riešenie bola zvolená databáza KTH, v ktorej sa nachádzalo 6 typov pohybov. Tieto pohyby sme následne spracovávali pomocou metódy MHI, kde pre jedno video sme využili viacnásobné spracovanie MHI v rôznych časových úsekoch. Ďalším kľúčovým bodom bolo vyňatie prídavných príznakov z MHI pre záznam a spracovanie príznakov pomocou PCA, kde sme taktiež otestovali viacero konfigurácií a možností spracovania. Takto vypočítané príznaky sme následne natrénovali pomocou SVM a testovali úspešnosť.

Ukázalo sa, že pri viacnásobnom duplikovaní vektorov príznakov nám PCA vyhodnotilo lepšie výsledky ako využitie PCA na pôvodných vektoroch príznakov. Taktiež lepšie výsledky sa prejavili po využití troch MHI z videosekvencie, ktoré boli zachytené v rôznych časových úsekoch videa.

V konečnej časti sme odprezentovali naše výsledky prostredníctvom tabuliek a percentuálnych úspešností. Tieto výsledky sme porovnali s výsledkami rozpoznávania pomocou iných metód. Na záver môžme povedať, že sa nám podarilo vytvoriť rozpoznávací systém, ktorého výsledky sú porovnateľné s inými metódami. Vidíme však priestor na ďalšie vylepšenia a aplikovanie iných a novších metód.

Zoznam použitej literatúry

1. *About - OpenCV library* [https://opencv.org/about.html]. (Accessed on 04/23/2018).
2. *Developer Documentation* [http://ffmpeg.org/developer.html]. (Accessed on 04/28/2018).
3. *Applications for Python | Python.org* [https://www.python.org/about/apps/]. (Accessed on 04/23/2018).
4. PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, roč. 12, s. 2825–2830.
5. *Frequently Asked Questions — SciPy.org* [https://www.scipy.org/scipylib/faq.html#what-is-numpy]. (Accessed on 04/28/2018).
6. AHAD, Md. Atiqur Rahman, TAN, Joo Kooi, KIM, Hyoungseop a ISHIKAWA, Seiji. Motion history image: its variants and applications. *Machine Vision and Applications*. 2010, roč. 23, s. 255–281.
7. *https://www.cs.utexas.edu/~chaoyeh/web_action_data/dataset_list.html* [https://www.cs.utexas.edu/~chaoyeh/web_action_data/dataset_list.html]. (Accessed on 04/22/2018).
8. A. BOBICK, J. Davis. *An appearance-based representation of action - IEEE Conference Publication* [https://ieeexplore.ieee.org/document/546039/?denied]. 1996. (Accessed on 04/23/2018).
9. AHAD, Md. Atiqur Rahman, TAN, Joo Kooi, KIM, Hyoungseop a ISHIKAWA, Seiji. Motion history image: its variants and applications. *Machine Vision and Applications*. 2010, roč. 23, s. 255–281.
10. MALLICK, Satya. *Histogram of Oriented Gradients | Learn OpenCV* [https://www.learnopencv.com/histogram-of-oriented-gradients/]. 2016. (Accessed on 05/10/2018).
11. DALAL, Navneet a TRIGGS, Bill. *Histograms of Oriented Gradients for Human Detection* [http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf]. 2008. (Accessed on 04/28/2018).
12. MALLICK, Satya. *Principal Component Analysis | Learn OpenCV* [https://www.learnopencv.com/principal-component-analysis/]. (Accessed on 05/11/2018).

13. SCHOLKOPF, Bernhard a SMOLA, Alexander J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001. ISBN 0262194759.

14. HASTIE, Trevor, TIBSHIRANI, Robert a FRIEDMAN, Jerome. *https://web.stanford.edu/~hast* [`https://web.stanford.edu/~hastie/Papers/ESLII.pdf`]. (Accessed on 04/28/2018).

15. YOUSEFI, B a CHU KIONG, Loo. Comparative Study on Interaction of Form and Motion Processing Streams by Applying Two Different Classifiers in Mechanism for Recognition of Biological Movement. 2014, roč. 2014, s. 723213.

16. MAATEN, Laurens van der a HINTON, Geoffrey. Visualizing Data using t-SNE. *Journal of Machine Learning Research*. 2008, roč. 9, s. 2579–2605. Dostupné tiež z: `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

Prílohy

A Štruktúra elektronického nosiča

/CHANGELOG.md

· file describing changes made to FEIstyle

/example.tex

· main example *.tex* file for diploma thesis

/example_paper.tex

· example *.tex* file for seminar paper

/Makefile

· simply Makefile – build system

/fei.sublime-project

· is project file with build in Build System for Sublime Text 3

/img

· folder with images

/includes

· files with content

/bibliography.bib

· bibliography file

/attachmentA.tex

· this very file

B Algoritmy

```
import numpy as np
from sklearn.decomposition import PCA, IncrementalPCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn import manifold
trainData = np.load('arrVectorFinal.npy')
bag_of_labels = np.load('classVector.npy')

print(trainData.shape)

n_components = 200
ipca = IncrementalPCA(n_components=n_components, batch_size=200)
X_ipca = ipca.fit_transform(trainData)

#print 'Explained variation per principal component: {}'.format(ipca.explained_variance_ratio_)
print("Sum {0}".format(np.sum(ipca.explained_variance_ratio_)))

colors = ['navy', 'turquoise', 'darkorange', 'red', 'wheat', 'yellow']
     #navy-vojnove_lodstvo,turquoise-tyrkysova(modro-zelena)


bag_of_labels = np.squeeze(bag_of_labels)

if False:
    area = np.full(X_ipca.shape, 25)
    plt.scatter(X_ipca[bag_of_labels == 1,0], X_ipca[bag_of_labels == 1,1], s=area, c=colors[0])
    plt.scatter(X_ipca[bag_of_labels == 2,0], X_ipca[bag_of_labels == 2,1], s=area, c=colors[1])
    plt.scatter(X_ipca[bag_of_labels == 3,0], X_ipca[bag_of_labels == 3,1], s=area, c=colors[2])
    plt.scatter(X_ipca[bag_of_labels == 4,0], X_ipca[bag_of_labels == 4,1], s=area, c=colors[3])
    plt.scatter(X_ipca[bag_of_labels == 5,0], X_ipca[bag_of_labels == 5,1], s=area, c=colors[4])
    plt.scatter(X_ipca[bag_of_labels == 6,0], X_ipca[bag_of_labels == 6,1], s=area, c=colors[5])

    plt.show()
    plt.gcf().clear()

tsne = manifold.TSNE(n_components=2, init='random',
                     random_state=0, perplexity=50)
Y = tsne.fit_transform(X_ipca)

area = np.full(Y.shape, 40)
plt.scatter(Y[bag_of_labels == 1,0], Y[bag_of_labels == 1,1], s=area, c=colors[0])
plt.scatter(Y[bag_of_labels == 2,0], Y[bag_of_labels == 2,1], s=area, c=colors[1])
plt.scatter(Y[bag_of_labels == 3,0], Y[bag_of_labels == 3,1], s=area, c=colors[2])
plt.scatter(Y[bag_of_labels == 4,0], Y[bag_of_labels == 4,1], s=area, c=colors[3])
plt.scatter(Y[bag_of_labels == 5,0], Y[bag_of_labels == 5,1], s=area, c=colors[4])
plt.scatter(Y[bag_of_labels == 6,0], Y[bag_of_labels == 6,1], s=area, c=colors[5])

navy_patch = mpatches.Patch(color='navy', label='BOX')
turquoise_patch = mpatches.Patch(color='turquoise', label='CLAP')
```

```
dark_orange_patch = mpatches.Patch(color='darkorange', label='WAVE')
red_patch = mpatches.Patch(color='red', label='RUN')
gold_patch = mpatches.Patch(color='wheat', label='WALK')
#yellow_patch = mpatches.Patch(color='yellow', label='JOGG')
plt.legend(handles=[navy_patch, turquoise_patch, dark_orange_patch, red_patch, gold_patch])

plt.show()
```

Listing B.1: Test TSNE

```
  HEIGHT = 10
  WIDTH = 10

  def processImg(image, boxes):
    count = 0
    rowBoxNum = 120 / HEIGHT
    colBoxNum = 160 / WIDTH
    sumOfBox = 0
    additionalVector = []
    value = 0

    data = np.asarray(image.getdata()).reshape(image.size)

    for i in range(0, rowBoxNum):
        for j in range(0, colBoxNum):
            for row in range(i*HEIGHT, i*HEIGHT + HEIGHT):
                for column in range(j*WIDTH, j*WIDTH + WIDTH):
                    sumOfBox = sumOfBox + data[column][row]
                    count = count + 1
            value = float(sumOfBox)/(WIDTH*HEIGHT)/255
            additionalVector.append(value)
            sumOfBox = 0
    return additionalVector
]
```

Listing B.2: Algoritmus získania prídavných príznakov