

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5212-79749

Martin Staňo

Rozpoznávanie objektov metódami počítačového videnia

Bakalárska práca

Študijný program: Informatika

Študijný odbor: 9.2.1 Informatika

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky

Supervisor: doc. Ing. Vanda Benešová, PhD.

Máj 2018

ZADANIE BAKALÁRSKEHO PROJEKTU

Meno študenta: **Staňo Martin**

Študijný odbor: Informatika

Študijný program: Informatika

Názov projektu: **Rozpoznávanie objektov metódami počítačového videnia**

Zadanie:

Vizuálne rozpoznávanie objektov je v súčasnosti intenzívne skúmaná oblasť výskumu s mnohými aplikáčnymi oblastami od automatickej inšpekcie, bezpečnosti, robotiky až po obohatenú realitu (angl. augmented reality).

Analyzujte súčasný stav problematiky vizuálneho rozpoznávania objektov. Zamerajte sa predovšetkým na metódy rozpoznávania objektov vo videosekvenciach s použitím neurónových sietí a metód hlbokého učenia.

Navrhnite vlastnú metódu detekcie a rozpoznávania objektov. Vami navrhnutú metódu implementujte s použitím knižnice OpenCv a vyhodnotte na štatisticky reprezentatívnom súbore testovacích dát. Pri vyhodnotení tiež porovnajte úspešnosť rozpoznávania Vami navrhнутej metódy so známymi doteraz publikovanými metódami. Vyhodnoťte i výpočtovú náročnosť implementovanej metódy.

Práca musí obsahovať:

Anotáciu v slovenskom a anglickom jazyku

Analýzu problému

Opis riešenia

Zhodnotenie

Technickú dokumentáciu

Zoznam použitej literatúry

Elektronické médium obsahujúce vytvorený produkt spolu s dokumentáciou

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava
Vedúci projektu: doc. Ing. Vanda Benešová, PhD.

Termín odovzdania práce v zimnom semestri: 12.12.2017

Termín odovzdania práce v letnom semestri: 10.5.2018

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta Informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
1

Bratislava 18.9.2017

prof. Ing. Pavol Návrat, PhD.

riaditeľ ÚSI

ČESTNÉ PREHLÁSENIE

Čestne vyhlasujem, že bakalársku prácu s názvom: Rozpoznávanie objektov metódami počítačového videnia som vypracoval samostatne, na základe konzultácií a štúdia odbornej literatúry. Neporušil som autorský zákon a zoznam použitej literatúry som uviedol na príslušnom mieste.

.....

Martin Staňo

POĎAKOVANIE

Tento cestou vyslovujem poďakovanie pani doc. Ing. Vanda Benešová, PhD. za pomoc, odborné vedenie, cenné rady a prípomienky pri vypracovaní mojej bakalárskej práce.

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program: Informatika

Autor: Martin Staňo

Bakalárská práca: Rozpoznávanie objektov metódami počítačového videnia

Vedúci práce: doc. Ing. Vanda Benešová, PhD.

Máj 2018

Počítačové videnie a jeho aplikácie na rozpoznávanie objektov a osôb prinášajú skvelé výsledky v mnohých problémových doménach, od bezpečnosti v automobilovom priemysle (detekcia chodcov a dopravných značiek), až po kontrolu kvality v priemyselnej výrobe. Intenzívne skúmanou aplikačnou oblasťou je rozpoznávanie osôb v kamerových sledovacích systémoch, pozostávajúcich z viacerých kamier. Tento problém je známy, ako problém re-identifikácie osôb. Správna identifikácia podozrievajúcej osoby na ktorejkoľvek kamere v systéme a analýza jej pohybu po verejném priestranstve majú dôležitý dopad na verejnú bezpečnosť a vyšetrovanie zločinov.

V tejto práci sa najskôr zameriame na analýzu problému re-identifikácie osôb. Predstavíme si prostriedky počítačového videnia využívané pri detekcii osôb a párovaní podobných obrazov. Následne sa zameriame na metódy hlbokého učenia a neurónové siete a opíšeme výhody ich použitia v kontexte re-identifikácie osôb.

Cieľom práce je implementovať vlastný systém na re-identifikáciu osôb s využitím knižníc Tensorflow a OpenCV a analyzovať vlastnosti vytvoreného modelu, vzhľadom na jeho rôzne konfigurácie a napokon porovnať jeho výsledky s výsledkami podobných publikácií.

Výsledky práce môžu pomôcť lepšie pochopiť vplyv hyper-parametrov modelu na jeho celkovú výkonnosť a môžu položiť základ pre vývoj komplexného systému na re-identifikáciu osôb, ktorý by bolo možné nasadiť v praxi.

Annotation

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree Course: Informatics

Author: Martin Staňo

Bachelor thesis: Object recognition using computer vision methods

Supervisor: doc. Ing. Vanda Benešová, PhD.

Máj 2018

Computer vision and its applications for object and person detection are showing great results in many problem domains ranging from automotive industry (pedestrian and traffic sign detection) to quality assurance in industrial production. One of intensively researched application domains is also person detection in camera surveillance system composed from multiple cameras. This problem is known as person re-identification problem. Correct identification of suspicious person anywhere in the system and analysis of its movement, might have crucial impact on public safety.

In this thesis we first focus on analyzing the person re-identification problem. We will describe techniques of computer vision used for person detection and matching. Next, we will focus on deep learning methods and artificial neural networks, describing its advantages in context of person re-identification.

The goal of this thesis is to implement person re-identification system using Tensorflow and OpenCV libraries and to analyze its properties based on variety of configurations. Lastly, we will compare our results with results of similar publications.

Outcomes of this thesis might help to better understand the impact of model hyper-parameters on its performance and might also provide base for further development of complex person re-identification system usable in real production environment.

Obsah

1	Úvod	1
2	Analýza	3
2.1	Re-identifikácia osôb	3
2.2	Metódy re-identifikácie osôb	3
2.2.1	Detekcia a sledovanie osôb	4
2.2.2	Získavanie príznakových vektorov - Lokálne detektory . .	6
2.2.3	Získavanie príznakových vektorov - Lokálne deskriptory .	10
2.2.4	Metriky podobnosti - Párovanie príznakov	11
2.3	Hlboké učenie	12
2.3.1	Neurónová sieť s dopredným šírením	13
2.3.2	Konvolučná neurónová sieť	14
2.3.3	Rekurentná neurónová sieť	16
2.3.4	Siamské neurónové siete	19
2.3.5	Contrastive loss funkcia	19
2.3.6	Cross entropy loss funkcia	20
2.4	Doterajšia práca v oblasti re-identifikácie osôb	20
2.4.1	Part Appearance Mixture [16]	21
2.4.2	Attentive Spatial-Temporal Pooling Networks for Video-based Person Re-Identification [27]	24
2.5	Dostupné softvérové rámce	27
2.5.1	Tensorflow	28
2.5.2	Používanie knižnice Tensorflow	29
2.5.3	Vyššie úrovne Tensorflow API	31
2.5.4	Ukážka tvorby modelu	33
3	Návrh	37
3.1	Koncept riešenia	37
3.2	Príprava vstupných dát	39
3.3	Tréning	41
3.4	Navrhnuté modely neurónových sietí	42
3.4.1	Modely sietí pre re-identifikáciu osôb na základe obrazov .	42

3.4.2	Modely pre re-identifikáciu osôb na základe video-sekvencí	44
3.5	Metriky vyhodnocovania modelov	48
3.6	Technické prostriedky	50
4	Výsledky	53
4.1	Výsledky modelov sietí pre re-identifikáciu osôb na základe obrazov	53
4.1.1	Tréning modelov	54
4.1.2	Vizualizácia aktivácií konvolučných vrstiev	55
4.1.3	Model c32_c64_d512_o32	56
4.1.4	Model c64_c128_d512_o32	57
4.1.5	Modely c64_c128_dx_ox	58
4.1.6	Modely fx_c64_c128_d4096_o256	59
4.1.7	Model c128_c256_d4096_o256	60
4.1.8	Testovanie klasifikácie	62
4.2	Výsledky modelov sietí pre re-identifikáciu osôb na základe video-sekvencí	63
4.2.1	Modely c64_c128_rnn_d512_o128	64
4.2.2	Modely cx_cx_cx_ssp_rnn_atpl_ce	65
4.2.3	Testovanie klasifikácie	67
4.3	Zhodnotenie	73
5	Záver	75
Literatúra		77
A	Plán práce a jeho plnenie	A-1
A.1	Zimný semester	A-1
A.2	Letný semester	A-2
B	Technická dokumentácia	B-3
B.1	Príprava dát	B-3
B.1.1	Organizácia súboru dát i-LIDS-VID	B-4
B.1.2	Moduly pre prípravu dát	B-4
B.2	Modely neurónových sietí	B-8

B.3	Tréning	B-11
B.4	Metriky	B-12
C	Inštalačná príručka	C-15
C.1	Python 3.6	C-15
C.2	Správca Python balíkov - pip	C-16
C.3	Virtuálne prostredie Anaconda	C-16
C.4	Inštalácia potrebných Python balíkov	C-17
C.5	CUDA a cuDNN	C-18
C.6	Stiahnutie projektu	C-18
D	Obsah elektronického média	D-19

Zoznam obrázkov

1	Vnútorná štruktúra LSTM bunky	18
2	Diagram komponentov navrhovaného systému	39
3	Ukážka vygenerovanej videosekvencie	40
4	Ukážka vygenerovanej augmentovanej dávky o veľkosti 10 párov zloženej zo samostatných obrazov	41
5	Základná siamská konvolučná sieť	44
6	Detail konvolučnej siete	45
7	Topológia modelu s rekurentnou a plne prepojenou vrstvou	46
8	Topológia modelu s SPP a ATPL vrstvou	47
9	Vývoj hodnôt loss funkcie v závislosti od počtu tréningových iterácií	54
10	Príznakové mapy prvej konvolučnej vrstvy modelu c32_c64_d512_o32	55
11	Príznakové mapy druhej konvolučnej vrstvy modelu c32_c64_d512_o32	55
12	ROC krivka pre model c32_c64_d512_o32	56
13	ROC krivka pre model c64_c128_d512_o32	57
14	ROC krivka pre modely c64_c128_dx_ox	58
15	ROC krivka pre modely fx_c64_c128_d4096_o256	60
16	ROC krivka pre model c128_c256_d4096_o256	61
17	Správna identifikácia zhodného (pozitívneho) páru	62
18	Správna identifikácia nezhodného (negatívneho) páru	62
19	Páry nesprávne označené ako nezhodné	63
20	Páry nesprávne označené ako zhodné	63
21	ROC krivka pre model c64_c128_rnn_d512_o128	65
22	ROC krivka pre modely cx_cx_cx_ssp_rnn_atpl_ce	66
23	Správna identifikácia zhodného (pozitívneho) páru sekvencií	68
24	Správna identifikácia nezhodného (negatívneho) páru sekvencií	69
25	Páry sekvencií nesprávne označené ako nezhodné	70
26	Páry sekvencií nesprávne označené ako zhodné	71
27	Príklady výstupnej príznakovej mapy pre veľkosť filtrov: 5×5 , 11×11 a 3×3 (zľava doprava)	73

Zoznam tabuliek

1	Stavy binárnej klasifikácie	48
2	Konfigurácia hyper-parametrov vrstiev modelu c32_c64_d512_o32	56
3	Výsledky metrík modelu c32_c64_d512_o32	56
4	Konfigurácia hyper-parametrov vrstiev modelu c64_c128_d512_o32	57
5	Výsledky metrík modelu c64_c128_d512_o32	57
6	Konfigurácia hyper-parametrov vrstiev modelov c64_c128_dx_ox .	58
7	Výsledky metrík modelov c64_c128_dx_ox	58
8	N ranking pre model c64_c128_d4096_o128	59
9	Konfigurácia hyper-parametrov vrstiev modelov fx_c64_c128_d4096_o256	59
10	Výsledky metrík modelov fx_c64_c128_d4096_o256	60
11	Konfigurácia hyper-parametrov vrstiev modelu c128_c256_d4096_o256	61
12	Výsledky metrík modelu c128_c256_d4096_o256	61
13	Konfigurácia hyper-parametrov vrstiev modelu c64_c128_rnn_d512_o128	64
14	Výsledky metrík modelu c64_c128_rnn_d512_o128	64
15	Konfigurácia hyper-parametrov vrstiev modelov cx_cx_cx_ssp_rnn_atpl_ce	66
16	Výsledky metrík modelu cx_cx_cx_ssp_rnn_atpl_ce	66
17	N ranking pre model c16_c32_c64_ssp_rnn_atpl_ce	66

Zoznam ukážok

1	Ukážka tvorby dvojvrstvovej konvolučnej siete na klasifikáciu obrazov	33
2	Ukážka tvorby dávky statických obrazov	B-6
3	Ukážka znovupoužitia premenných	B-8
4	Model dvojvrstvovej siamskej konvolučnej siete	B-9
5	Procedúry pre tvorbu rekurentnej , SPP a ATPL vrstvy	B-10
6	Tréningový cyklus	B-12
7	Výpočet metrík pre všetky hraničné vzdialenosťi	B-13
8	Výpočet n-rank metriky	B-14

1 Úvod

Re-identifikácia osôb, ako jeden z problémov v oblasti počítačového videnia, získava v poslednom desaťročí čoraz viac pozornosť, keďže multi-kamerové sledovacie systémy sú čoraz dostupnejšie a potreba sledovania verejných priestranstiev je neutíchajúca, vzhľadom na čoraz rozšírenejšie bezpečnostné hrozby.

Problém re-identifikácie sa dá dekomponovať na problém detektie osoby, jej sledovania, a napokon overenia či a kde bola daná osoba zachytená v minulosti. Zatiaľ čo klasické prístupy počítačového videnia poskytujú uspokojivé výsledky pre prvé dva zmienené čiastkové problémy, v prípade posledného sú vzhľadom na zvyšujúci sa počet ľudí na verejných priestranstvách kladené požiadavky nielen na presnosť algoritmov na párovanie osôb, ale aj na ich schopnosť poskytovať požadované výsledky v reálnom čase. Táto požiadavka bola ľahšie dosiahnuteľná pri systémoch, využívajúcich klasické prístupy počítačového videnia založené na lokálnych detektoroch a deskriptoroch a ich párovaní.

Neskorší rozvoj strojového učenia, konkrétnie hlbokých neurónových sietí, konvolučných a rekurentných neurónových sietí, zapríčinil, že mnohé z algoritmov klasického počítačového videnia boli prekonané a zvlášť to platilo pre oblasti rozpoznávania osôb a objektov. To ďalej spôsobilo aj prenikanie metód hlbokého učenia do systémov na re-identifikáciu osôb. Najväčšie úspechy zaznamenali siamské architektúry modelov využívajúce konvolučné neurónové siete, poprípade ich kombináciu s rekurentnými neurónovými sieťami.

Ťažisko súčasného výskumu je zamerané na vplyv hyper-parametrov modelu, na jeho presnosť a výkonnosť, ako aj vývoj nových druhov neurónových vrstiev, ktoré majú zabezpečiť filtráciu redundantných dát, nepotrebných pre učenie sa modelu. Presnosť publikovaných riešení re-identifikácie osôb je porovnávaná na viacerých unifikovaných súboroch dát (*ang. datasets*). Cieľom tejto práce je implementovať vlastný systém na re-identifikáciu osôb, využívajúci hlboké učenie, ktorý bude inšpirovaný state-of-the-art publikáciami v problémovej doméne a porovnať jeho výsledky (pre rôzne konfigurácie) s danými publikáciami.

2 Analýza

2.1 Re-identifikácia osôb

Pokrok vo výskume a aplikácií systémov rozpoznávania a sledovania objektov, resp. osôb, vybranými metódami počítačového videnia dovoľuje s uspokojivou presnosťou vykonávať zmienené činnosti, dokonca i v prostredí s výrazným vizuálnym šumom. Typickými predstaviteľmi takýchto prostredí sú rušné verejné priestranstvá, ako napríklad vlakové stanice a letiskové terminály. Aplikovateľnosť týchto metód je však limitovaná iba na systémy sledovania, pozostávajúce z jednej kamery, poprípade viacerých kamer s prekrývajúcimi sa zornými poliami. Pri konfigurácii kamer s prienikom zorných polí vieme v nami zvolenej metóde sledovania osôb zohľadniť vzájomnú polohu kamer v monitorovanom priestore, čo môže zvýšiť úspešnosť danej metódy.

Na novodobé, komerčne používané kamerové sledovacie systémy je však často kladená požiadavka pokryť veľmi rozsiahle a členité sledované priestory, v ktorých je použitie konfigurácie kamer s prekrývajúcimi sa zornými poľami nemožné, či už z finančného, alebo technického hľadiska. Vznik častí sledovaného priestoru, ktoré nie sú snímané nijakou kamerou v systéme, problém sledovania a rozpoznávania (identifikácie) osôb komplikuje a rozsiruje o nové čiastkové problémy.

Jedným z týchto čiastkových problémov je aj problém **re-identifikácie osôb**. Jedná sa o problém určenia, či osoba rozpoznaná na zázname z konkrétnej kamery je zhodná s osobou identifikovanou na zázname z inej kamery v systéme. Túto úlohu značne komplikujú viaceré faktory, ako napríklad rozdiely v osvetlení, postoji, poprípade aj farbe oblečenia osoby, ktorú chceme medzi kamerovými stopami re-identifikovať. Dôležité je preto využívať metódy, ktoré sú invariantné vzhľadom na hore uvedené faktory.

2.2 Metódy re-identifikácie osôb

Komplexné riešenie re-identifikácie osôb sa skladá z viacerých čiastkových metód, ktoré na seba sekvenčne navádzajú:

1. Metóda na detekciu osoby na obraze

2. Metóda na sledovanie osoby (v prípade video-sekvencií)
3. Metóda na získanie vektoru príznakov (*ang. feature vector*) sledovanej osoby, pričom príznaky získavame iba z výseku obrazu určeného algoritmom na detekciu osôb (*ang. bounding box*)
4. Metrika určujúca podobnosť dvoch príznakových vektorov, ktorej výsledok určí konečný verdikt, či je osoba zachytená na rôznych kamerách tá istá

Cieľom výskumu v predmetnej oblasti je preto buď vylepšovanie existujúcich metód na každej z úrovni, výskum nových, alebo ich novátorská kombinácia. V tejto kapitole si priblížime najznámejšie prístupy ku každej časti re-identifikácie osôb, pričom budeme klásť dôraz na metódy získavania príznakov a na metriky podobnosti, ktoré budú tvoriť ťažisko neskoršej práce.

2.2.1 Detekcia a sledovanie osôb

Cieľom algoritmu detektie osôb je jednoznačne určiť či a kde na obraze sa osoba, resp. osoby nachádzajú. Výstupom algoritmu sú zvyčajne dva body $A[x_1, y_1]$ a $B[x_2, y_2]$ na uhlopriečke obdĺžnikového výseku obrazu, ktorý kompletne obsahuje detekovanú osobu.

Najznámejšou metódou na riešenie tejto problematiky je využitie **lokálneho deskriptora obrazu HOG** (*ang. histogram of oriented gradients*) a SVM (*ang. support vector machine*) [9].

Lokálny deskriptor môžeme všeobecne formálne opísť ako:

$$\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_r(\mathbf{x})] = \Phi(\mathbf{I}, \mathbf{x})$$

kde $\phi(\mathbf{x})$ je vektor reprezentujúci samotný lokálny deskriptor obrazu \mathbf{I} , počítaný pre každý jeho pixel \mathbf{x} . Funkcia Φ môže reprezentovať jeden alebo viac lineárnych konvolučných filtrov, poprípade aj komplexnejšiu nelineárnu operáciu.

Funkciu Φ pre lokálny deskriptor *HOG* môžeme potom zapísť ako:

$$\Phi(\mathbf{I}, \mathbf{x}) = [g(\mathbf{x}), \Theta(\mathbf{x})]$$

jedná sa o dvojicu veľkosti g a smeru Θ gradientu z bodu x pričom:

$$g(\mathbf{x}) = \sqrt{g_x^2 + g_y^2}$$

$$\Theta(\mathbf{x}) = \arctan \frac{g_y}{g_x}$$

kde g_x a g_y sú gradienty v smere osi x , resp. y rátané ako konvolúcia okolia bodu \mathbf{x} s filtrovami K_x a K_y :

$$K_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$K_y = K_x^T$$

Výpočet veľkosti a smeru gradientu sa vykoná pre každý pixel x z regiónu obrazu \mathbb{R} o veľkosti $n * m$ pixelov, kde n a m je volené podľa veľkosti objektov, ktoré sa snažíme zachytiť a spravidla platí $n, m = 2^k; k \in N$. Tento región sa ďalej rozdelí na menšie regióny (bloky) o veľkosti 8×8 alebo 16×16 . Z výsledných hodnôt smeru a veľkosti gradientov v každom bloku sa následne vytvorí histogram s K triedami T , kde K volíme podľa počtu klasifikovaných gradientov M jedným zo zaužívaných spôsobov (napr. $K = \sqrt{M}$ resp. $K = \log_2 M + 1$). Trieda $T_i; i \in (0, 1, \dots, K - 1)$ predstavuje uhol $\Gamma = i * (\phi/K)$ kde ϕ je buď π , alebo 2π . V prípade, že uhol histogramu $\Theta(\mathbf{x})$ spadá medzi dve triedy histogramu T_i a T_{i+1} , tak je veľkosť gradientu $g(\mathbf{x})$ lineárne interpolovaná medzi triedy histogramu T_i a T_{i+1} . Aby bol HOG invariantný vzhľadom na zmenu intenzity pixelov, je vhodné ho normalizovať. Pri normalizácii sa k HOG správame ako ku K rozmernému vektoru, teda každú jeho hodnotu predelíme jeho veľkosťou S vyjadrenou ako:

$$S = \sqrt{\sum_{i=0}^{K-1} T_i^2}$$

Posledným krokom je výpočet vektoru príznakov pre celý región o veľkosti $n \times m$ pixelov. Tento vektor predstavuje zrežazenie HOG pre všetky bloky v danom regióne. Veľkosť výsledného HOG pre celý región je teda:

$$\|\text{HOG}\| = (m/b) * (n/b) * K$$

Horeuvedeným spôsobom sa **HOG** vypočíta vždy, keď sa región \mathbb{R} v obraze posunie (algoritmus funguje ako kľúčavé okienko - *ang.* sliding window). Vektor **HOG** je následne poslaný ako vstup do pred-trénovaného SVM (popr. iného lineárneho resp. nelineárneho klasifikačného modelu), ktorý určí či daný región obsahuje, alebo neobsahuje hľadanú osobu. Klasifikačný model musí byť samozrejme natrénovaný na doménových príkladoch, v našom prípade osobách.

V prípade detektie osôb vo video-sekvencii môžeme so získaným ohraničením osôb na každom obraze v sekvencii pristúpiť ku sledovaniu osôb. To znamená priradiť každej detektovanej osobe, na každom obraze v sekvencii, unikátny identifikátor, ktorý by mal zostať pre tú istú osobu nezmenený naprieč celou sekvenciou. Tento proces je známy ako **asociácia dát** (*ang.* data association). Rozšíreným spôsobom je **JPDA** (*ang.* Joint Probabilistic Data Association), ktorý problém asociácie dát adresuje ako pravdepodobnostný model. Výpočet presných pravdepodobností asociácií v JPDA je NP-ťažký problém, preto je ich výpočet aproximovaný Monte Carlo algoritmom **MCMCDA** (Markov Chain Monte Carlo Data Association) [23]. Presná matematická definícia je nad rámec tejto práce.

2.2.2 Získavanie príznakových vektorov - Lokálne detektory

Ak je v našom riešení re-identifikácie osôb vyriešený problém detektie a prípadného sledovania osôb na jednej kamere, môžeme pristúpiť k extrakcii príznakov pre ohraničenú časť obrazu obsahujúcu detektovanú osobu. Slovom obraz budeme odteraz označovať iba región, ktorý bol algoritmom detektie označený ako región obsahujúci osobu.

Oproti problému detektie osôb, kde sme zisťovali len či a kde je osoba prítomná, je problém získavania príznakov vhodných na porovnávanie pri re-identifikácii oveľa zložitejší. Jediný deskriptor (napr. HOG) vyrátaný pre celý obraz nedokáže dostatočne a jednoznačne opísť kompletný výzor osoby. Z toho dôvodu sa príznaky vhodné na porovnávanie nezískavajú z celého obrazu, ale iba z okolia salientných bodov nazývaných **kľúčové body** (*ang.* key points). Zásadnou vlastnosťou kľúčových bodov využiteľnou pri re-identifikácii osôb je ich invariancia vzhľadom na affiné transformácie obrazu. Kľúčové body sú získavané pomocou algoritmov známych ako **lokálne detektory**.

Harrisov detektor rohov

[14] je jedným z najznámejších lokálnych detektorov. Funguje na princípe výpočtu tzv. autokorelačnej matice pre okolie bodu (x, y) definovanej nasledovne:

$$M(x, y) = \sum_{p,q} w(p, q) \begin{bmatrix} I_x^2(p, q) & I_x I_y(p, q) \\ I_x I_y(p, q) & I_y^2(p, q) \end{bmatrix}$$

kde I_x a I_y sú derivácie (diferencie) intenzity pixelu obrazu v bode (p, q) (patriaceho do okolia bodu (x, y)) v smere osi x , resp. y . Funkcia $w(p, q)$ je vážiaca funkcia, typicky dvojrozmerné normálne rozdelenie, čo znamená, že body bližšie stredu majú vyššiu váhu. Testovaný bod je považovaný za kľúčový, ak má autokorelačná matica vypočítaná pre jeho okolie dve veľké vlastné hodnoty (*ang.* eigenvalues). Vlastná hodnota matice A je také číslo λ , pre ktoré platí $\det(A - I\lambda) = 0$, kde I je matica identity. V praxi to znamená, že gradienty smerujúce z okolia kľúčového bodu majú dva dominantné smery. Harris taktiež zaviedol vlastnú metriku na zisťovanie, či daná autokorelačná matica reprezentuje kľúčový bod. Metrika je založená na výpočte determinantu a stopy autokorelačnej matice M a má tvar:

$$R = \det(M) - \alpha * \text{trace}^2(M)$$

kde α je váhový koeficient (konštanta). Ak $R > t$, kde t je nami zvolená hranica, tak je príslušný bod považovaný za kľúčový.

Detekcia kľúčových bodov na základe rohov, tak ako ich definoval Harris, je základom aj pre dnes veľmi používaný detektor rohov: **Shi–Tomasi algoritmus na detekciu rohov**.

FAST detektor

FAST (*ang.* Features from Accelerated Segment Test) [24] je novší algoritmus na detekciu kľúčový bodov. Motiváciou pre jeho výskum bola potreba nájsť kľúčové body na obrazoch vo video-sekvenciach v reálnom čase, pričom sekvencia môže mať niekoľko desiatok obrazov za sekundu.

Princípom FAST algoritmu je získanie postupnosti pixelov P (spravidla šestnástich) na obvode pomyselného kruhu so stredom v bode C , ktorý vyhodnocujeme

ako kľúčový. Skúmaný bod je vyhodnotený ako kľúčový, ak v postupnosti P nájdeme aspoň η po sebe nasledujúcich pixelov p spĺňajúcich podmienku:

$$I_p > I_C + \tau \vee I_p < I_C - \tau; p \in P$$

kde τ a η sú nami zvolené parametre, ovplyvňujúce citlivosť detektora.

Blob detektory

Narozdiel od detektorov rohov, ktoré sa zameriavalí na detekciu bodov, pri ktorých sa intenzita pixelov menila v rôznych smeroch (gradienty intenzity pre body v okolí kľúčového bodu mali dva dominantné smery), sú detektory blobov zamerané na detekciu regiónov obrazu, kde sú jeho pixely navzájom podobné (veľkosť gradientov intenzity bodov vo vnútri regiónu je veľmi malá).

Ťažiskom blob detektorov je konvolúcia s Gaussovým filtrom. Následne sa môžeme rozhodnúť pre jeden z nasledujúcich prístupov [19]

1. Laplacian of Gaussian (LoG)
2. Difference of Gaussians (DoG)

Laplacian of Gaussians

Prvý z vymenovaných spôsobov, ako z názvu vyplýva, aplikuje Lapacov operátor (viď. nižšie) na obraz, ktorý vznikol konvolúciou pôvodného obrazu $I(x, y)$, kde x a y sú súradnice v obraze s Gaussovým filtrom, ktorého spojité definícia je:

$$g(x, y, t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

pričom parameter t určuje veľkosť blobov, ktoré filter najviac zvýrazní. Na výsledný obraz L po konvolúcií sa aplikuje Lapacov operátor:

$$\Delta L = \nabla^2 L = L_{xx} + L_{yy} = \frac{\delta L}{\delta x^2} + \frac{\delta L}{\delta y^2}$$

Parciálne derivácie druhého stupňa L_{xx} a L_{yy} obrazu $L(i, j)$ v smere osi x , resp. y v bode $[i, j]$ vieme vyjadriť ako operácie so susednými pixlami nasledovne:

$$L_x x = L(x+1, y) + L(x-1, y) - 2L(x, y)$$

$$L_y y = L(x, y+1) + L(x, y-1) - 2L(x, y)$$

Z vyššie uvedeného zápisu (sčítaním parciálnych derivácií) vyplýva, že aplikovanie Laplacovho operátora na obraz L sa dá realizovať operáciou konvolúcie s filtrom:

$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Výrazným obmedzením tejto základnej metódy je silná závislosť výsledkov od parametru t , pretože správne sú detekované iba regióny s polomerom $r = \sqrt{2t}$. Riešenie priniesol dr. Tony Lindeberg, ktorý navrhol škálovateľnú LoG detekciu vo svojej práci [20]. Riešenie sa zakladá na báze normalizácie Laplacovho operátora podľa parametra t .

Difference of Gaussians

Táto metóda je iba aproximáciou metódy LoG. Vychádzajúc z rovnice šírenia tepla:

$$\frac{\delta G}{\delta \sigma} = \sigma \nabla^2 G$$

kde $G(x, y, \sigma)$ je obraz, ktorý vznikol konvolúciou s Gaussovým filtrom s parametrom σ . Z definície derivácie môžeme odvodíť:

$$\sigma \nabla^2 G = \frac{\delta G}{\delta \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G$$

Z posledného kroku vyplýva, že nami hľadaný Lapacián $\nabla^2 G$ je možné aproximovať pomocou per-pixel rozdielu medzi obrazmi získanými dvoma Gaussovými filtromi s parametrami σ , resp. $k\sigma$. Metóda Difference of Gaussians je používaná

na detekciu kľúčových bodov aj v jednom z najpoužívanejších systémov na detekciu a deskripciu kľúčových bodov, SIFT (*ang.* scale-invariant feature transform) [5] [21]. V detektore SIFT sa pomocou DoG navrstvia dvoj-rozmerné obrazy na seba a vznikne škálový priestor (*ang.* scale space), v ktorom je tretí rozmer σ . Tým vznikne tzv. pyramída laplaciánov (*ang.* Pyramid of Laplacians). Bod $P[x, y, \sigma]$ v obraze G sa považuje za kľúčový ak:

$$P[x, y, \sigma] > p \vee P[x, y, \sigma] < p; \forall p \in \left(\bigcup_{i,j,\tau \in \{-1,0,1\}} G[x+i, y+j, \sigma+\tau] \right) - \{P\}$$

, čo reprezentuje porovnávanie kandidáta na kľúčový bod s 26 okolitými bodmi v škálovom priestore (okolie $3x3x3$ skúmaného bodu, pričom samotný skúmaný bod neuvažujeme). Ak je skúmaný bod menší alebo väčší ako všetky body vo vyššie definovanom okolí, považujeme ho za kľúčový. Detekcia kľúčových bodov pomocou pyramídy laplaciánov predstavuje robustný lokálny detektor, invariantný voči afinným transformáciám.

2.2.3 Získavanie príznakových vektorov - Lokálne deskriptory

Kľúčové body v obraze nám nedokážu poskytnúť potrebný objem informácií na opis a prípadnú identifikáciu objektu, ktorý obraz obsahuje (jedná sa iba o bod v obraze). Tento problém adresujú algoritmy známe ako **lokálne deskriptory**. Tie operujú na bodoch obrazu v blízkosti detekovaných kľúčových bodov a ich výstupom sú dátá opisujúce daný región obrazu. Lokálne deskriptory podľa typu dát, ktoré sú ich výstupmi delíme na dve skupiny:

- Float deskriptory
- Binárne deskriptory

Ako už názvy napovedajú, výstupom float deskriptorov je vektor desatinných čísel. Pri binárnych deskriptoroch je to vektor binárnych hodnôt. Je triviálne dokázať, že pri rovnakej dĺžke výstupného vektora je možné float deskriptorom zakódovať väčší objem dát, než deskriptorom binárnym.

Najznámejším príkladom lokálneho deskriptora (použitého aj v systéme SIFT) je Histogram orientovaných gradientov (HoG), ktorý je formálne opísaný vyššie v 2.2.1. I keď väčšina metód deskripcie je založená na algoritme HOG, existuje viacero spôsobov ako ho modifikovať tak, aby sme na špecifickej problémovej doméne vylepšili jeho výsledky. Modifikácie môžu zahŕňať zmenu veľkosti blokov pixlov, z ktorých získavame histogramy alebo zmenu počtu tried v histograme. Veľkosti a smery gradientov sa taktiež môžu meniť v závislosti od použitého farebného modelu popisovaného obrazu (HSV, RGB, Greyscale).

V roku 2006 bol predstavený systém lokálneho detektora a deskriptora s názvom Speeded-Up Robust Features (SURF), ktorý je podľa autorov výrazne rýchlejší než SIFT [2]. Neskôr bol v USA patentovaný. Na detekciu kľúčových bodov používa **determinat hesseho matice** s úpravou, ktorá pri výpočte využíva iba celé čísla a menej operácií, než je tomu pri pyramíde lapaciánov v prípade detektora v systéme SIFT. SURF využíva na urýchlenie výpočtu detektora dátovú štruktúru **integrálny obraz** (*ang.* integral image), taktiež nazývanú tabuľka plošných súčtov (*ang.* summed-area table) [8]. Integrálny obraz je pred každým použitím detektora a deskriptora na novom obraze najskôr dopredu vypočítaný. Jeho princípom je efektívne počítanie súčtov hodnôt v dvojrozmernom poli (v našom prípade obrazu). Výhody tejto dátovej štruktúry využíva taktiež SURF deskriptor, ktorý je založený na deskriptore zvanom **Haar Features** prezentovanom v roku 2001. [25]

2.2.4 Metriky podobnosti - Párovanie príznakov

Tým, že deskriptory operujú nad časťami obrazu v okolí kľúčových bodov, ktoré sú invariantné vzhľadom na afinné transformácie, sú aj príznakové vektory vypočítané danými deskriptormi rovnako invariantné. Porovnaním podobnosti príznakových vektorov, popisujúcich okolia kľúčových bodov medzi dvoma obrazmi (originál a transformácia obrazu), vieme identifikovať najviac podobné páry kľúčových bodov medzi obrazmi. Najznámejšou metrikou na porovnanie podobnosti (tiež nazývanej vzdialenosť) dvoch príznakových vektorov je **Euklidovská vzdialenosť (metrika)**. Pre dva vektory \vec{a} a \vec{b} , ktoré majú rovnaký počet prvkov n je Euklidovská vzdialenosť definovaná nasledovne:

$$M_e(\vec{a}, \vec{b}) = \sqrt{\sum_{i=0}^n (a_i - b_i)^2}$$

Aby sme však našli páry kľúčových bodov, ktoré majú medzi sebou minimálnu vzdialenosť (tj. sú si najviac podobné), museli by sme porovnať medzi sebou každú možnú dvojicu kľúčových bodov, čo môže byť v určitých situáciach (aplikácie v reálnom čase) výpočtovo náročné (asymptotická zložitosť $O(n^2)$, kde n je počet kľúčových bodov). Jedným z výsledkov výskumu v snahe nájsť efektívnejší porovací algoritmus je zbierka algoritmov zhrnutá v knižnici FLAAN (*ang.* Fast Library for Approximate Nearest Neighbors) [22]. Táto knižnica je integrovaná aj v známom softvérovom rámci OpenCV.

Predmetom výskumu tiež stali algoritmy, ktoré by dokázali získať maticu podobnosti (*ang.* homography matrix). Príkladom takéhoto algoritmu je aj RANSAC (*ang.* Random Sample Consensus) [12]. Jedná sa o nedeterministický algoritmus, ktorého výsledky sa zlepšujú s počtom iterácií, ktoré vykoná. Jedna iterácia po- zostáva z výberu k náhodných kľúčových bodov (a ich príznakových vektorov), kde k je minimálny počet bodov na approximáciu matice podobnosti. Následne sa na všetky zvyšné kľúčové body v pôvodnom obraze aplikuje approximovaná matica podobnosti a porovnajú sa s kľúčovými bodmi v transformovanom obraze. Vzdialenosť medzi porovnávanými bodmi sa akumuluje a reprezentuje chybovosť danej matice podobnosti. Algoritmus sa opakuje n iterácií a za výslednú maticu homografie sa považuje tá s najnižšou chybovosťou.

2.3 Hlboké učenie

S exponenciálnym nárastom výpočtového výkonu v posledných desaťročiach a s neustále napredujúcim vývojom paralelizácie výpočtových úloh sa do popredia v rámci najpoužívanejších algoritmov v doméne počítačového videnia a spracovania obrazu dostávajú algoritmy a metódy, založené na hlbokom strojovom učení a neurónových sieťach. Aby sme si dokázali priblížiť zložitejšie modely hlbokého učenia využívané pri problematikách identifikácie a klasifikácie objektov, či osôb, ktoré priamo súvisia i s re-identifikáciou osôb je nutné najskôr zadefinovať, čo je to neurónová sieť a aké sú jej úlohy.

2.3.1 Neurónová sieť s dopredným šírením

Najzákladnejším nelineárny modelom hlbokého učenia je hlboká sieť s dopredným šírením, známejšia pod názvom **viacvrstvový perceptrón** (*ang.* MLP - Multi Layer Perceptron). Táto sieť sa nazýva hlboká (resp. viacvrstvová), pretože okrem vstupnej a výstupnej vrstvy má ešte minimálne jednu **skrytú vrstvu**.

Vrstvy neurónovej siete sa skladajú s neurónov, ktoré v prípade perceptrónových sietí nazývame **perceptróny**. Stav perceptrónu môžeme opísť rovnicou:

$$\xi = \sum_{i=0}^n w_i x_i - \Theta$$

kde ξ je vnútorný potenciál neurónu, počítaný ako vážený priemer vstupných hodnôt (z predchádzajúcej vrstvy siete) x a Θ je tzv. prahová hodnota (*ang.* bias). Váha w_i predstavuje váhu vstupu x_i . Výstup perceptrónu je určený funkciou $f(\xi)$, ktorú nazývame **aktivačná funkcia**. Nutnými vlastnosťami aktivačnej funkcie sú: spojitosť, ohraničenosť a monotónnosť. Najpoužívanejšou aktivačnou funkciou je **sigmoida**:

$$f(\xi) = \frac{1}{1 + e^{-\lambda\xi}}$$

kde parameter λ určuje strmosť sigmoidy.

Úlohou viacvrstvového perceptrónu je aproximovať funkciu $y = f^*(\mathbf{x})$, ktorá mapuje vstupné hodnoty x na výstupné hodnoty (resp. kategórie v prípade klasifikácie) y . Perceptrón funkciu approximuje tak, že definuje vlastnú mapovaciu funkciu $y = f(\mathbf{x}, \Theta)$, kde Θ sú parametre, ktoré sa sieť pri procese tréningu naučí. Medzi parametre patria váhy spojení w a prahové hodnoty Θ .

Učenie neurónovej siete

Ak jednotlivé vrstvy siete zdefinujeme ako funkcie f_1, f_2, \dots, f_n pričom f_n je posledná (výstupná) vrstva siete a f_1 je prvá (vstupná) vstupná, potom celú sieť so vstupom \mathbf{x} a výstupom y vieme zdefinovať ako zloženú funkciu $y = f_n(f_{n-1}(\dots(f_1(\mathbf{x}))\dots))$. Učenie siete prebieha tzv. spätným šírením chyby. Pri tomto procese sa na základe chyby získanej na výstupe siete tzv. **loss (cost) funkciou** (ktorá je funkciu výstupu siete y) aktualizujú váhy všetkých spojení v sieti. Veľkosť

zmeny konkrétnej váhy je daná veľkosťou derivácie loss funkcie podľa danej váhy. Táto metóda sa nazýva **gradient descent** podľa toho, že rátame parciálne derivácie (gradient) loss funkcie. Váhu spojenia w_i v čase (epoch) $t + 1$ môžeme na základe metódy gradient descent vypočítať nasledovne:

$$w_i(t + 1) = w_i(t) + \eta \frac{\delta C}{\delta w_i} + \xi(t)$$

2.3.2 Konvolučná neurónová sieť

Konvolučné neurónové siete (často skrátene označované ako **CNN** z ang. convolutional neural networks) sú špecializovaný druh neurónových sietí uspôsobený na spracovávanie dát, ktorých štruktúra je podobná mriežke (v princípe n -rozmernej). Tejto špecifikácií vstupných dát presne zodpovedajú obrazy (2-D mriežka pixlov) a videá (3-D mriežka, kde je tretí rozmer čas). Ako už názov napovedá, konvolučné neurónové siete využívajú matematickú operáciu **konvolúcie** namiesto klasického maticového súčinu, aspoň na jednej zo svojich vrstiev. Konvolúcia v CNN prebieha ako diskrétna operácia medzi dvoma maticami I a K , kde I je vstup (napr. R-kanál RGB obrazu) a K je jadro (ang. kernel). Výstupom je tretia matica S , často nazývaná aj **mapa príznakov**.

Pre hodnotu vstupnej matice I na pozícii (i, j) a kernel K o veľkosti $m \times n$ môžeme operáciu konvolúcie (označenú $*$), ktorej výstupom je hodnota mapy príznakov S na pozícii (i, j) , zadefinovať nasledovne:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

Hlavnou motiváciou pre používanie konvolučných neurónových sietí v aplikáciach spracovávajúcich obraz oproti jednoduchej hlbokej sieti (MLP) je vyššia výpočtová efektivita (pamäťová aj časová). Pre jednoduchosť uvažujme plne prepojený (ang. fully-connected, resp. dense) viacvrstvový perceptrón pozostávajúci z dvoch vrstiev: vstupnej a výstupnej. Ak je veľkosť vstupnej vrstvy m neurónov a veľkosť výstupnej vrstvy n neurónov je počet prepojení (a teda aj počet váh, ktorý sa sieť musí naučiť) rovný $m \times n$ a počet násobení pri násobení matíc s daným počtom parametrov je $O(m \times n)$.

Naproti tomu v konvolučnej sieti je spojenie medzi neurónmi redšie (ang.

sparse connection). Pre konvolučný kernel obsahujúci k prvkov je každý vstupný neurón spojený iba s k neurónmi na nasledujúcej vrstve. Počet spojení (a operácií v jednom maticovom násobení) v jednoduchej konvolučnej sieti je teda rovný $k \times m$. Avšak počet parametrov, ktoré je nutné si pamätať, je rovný iba počtu prvkov kernelu k . Je to spôsobené technikou používanou v konvolučných neurónových sieťach zvanou **zdieľanie parametrov** (*ang.* parameter sharing). Je to umožnené tým, že hodnoty použitého konvolučného kernelu sú konštantné nezávisle na pozícii (pixeli), na ktorej konvolúciu vykonávame. Pri väčšine aplikácií býva k rádovo menšie než veľkosť vstupu (počet vstupných neurónov). Napríklad na detekciu hrán v obraze pozostávajúceho z tisícov pixlov môže poslúžiť kernel pozostávajúci z deviatich hodnôt. To má za následok masívnu úsporu pamäte potrebnej na ukladanie parametrov a umožňuje nám vytvárať hlbšie i širšie topológie neurónových sietí.

Aktivačná funkcia a pooling vrstva

Typicky však konvolučná vrstva nepozostáva iba zo získania mapy príznakov, ale aj z jej následného spracovania. Prvým krokom spracovania prvkov v mape príznakov je prerátanie ich hodnot pomocou nami zvolenej **nelineárnej aktivačnej funkcie**. V kontexte konvolučných neurónových sietí sa najčastejšie používa funkcia **rectified linear** definovaná ako:

$$f(x) = x^+ = \max(0, x)$$

kde x je vstup do neurónu. Táto aktivačná funkcia vracia kladný vstup nezmenený a záporný vstup oreže na nulu. Vrstvu konvolučnej siete pozostávajúcu z neurónov, ktoré majú túto funkciu ako svoju aktivačnú funkciu nazývame **ReLU** (*ang.* rectified linear unit).

Výstup z ReLU vrstvy je následne spracovaný tzv. **pooling vrstvou**. Neuróny tejto vrstvy majú na vstupe niekoľko prvkov z mapy príznakov (po vykonaní aktivačnej funkcie), spravidla prvak na určitej pozícii a prvky z jeho obdĺžnikového (štvorcového) okolia. Výstupom je určitý štatistický sumár vstupných hodnôt. Najčastejšie používané je maximum vstupných hodnôt. Takáto vrstva sa nazýva **max-pooling** vrstva. Medzi iné funkcie, ktoré pooling vrstvy využívajú, patria napríklad: priemer prvkov vstupného okolia, resp. ich vážený priemer podľa vzdialenosť od

stredného prvku.

Využitie max-pooling vrstvy v konvolučnej neurónovej sieti pri extrakcii príznakov z obrazu zabezpečí invariantnosť týchto príznakov vzhľadom na affinné transformácie (najmä posun a rotáciu) v určitom rozsahu. Využitím hlbšej topológie konvolučnej neurónovej siete (viacero na seba nadväzujúcich kombinácií: konvolučná vrstva + ReLU + max-pooling) môžeme robustnosť nášho modelu ešte viac zvýšiť.

2.3.3 Rekurentná neurónová sieť

Podobne ako konvolučné neurónové siete aj rekurentné neurónové siete (označované skrátene ako RNN - *ang. recurrent neural network*) sú navrhnuté na spracovávanie konkrétné štrukturovaných dát, v tomto prípade **sekvencií**. Sekvencia môže predstavovať dátu usporiadane v čase (snímky videa, akustické vlny zodpovedajúce reči), ale aj v priestore (slová písanej vety, pixely obrazu). Formálne môže povedať, že rekurentná sieť spracováva sekvenciu (vektor) dát $\mathbf{x} = (x^{(1)}, x^{(2)}, \dots, x^{(\tau)})$ podľa vzťahu:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \Theta)$$

, kde $x^{(t)}$ je prvok sekvencie v čase t a h označuje stav siete v príslušnom čase. Stav h nazývame aj **skrytý stav**. Váhy spojení v sieti sú označené Θ . Konkrétnie sa delia na dve váhové matice a jednu prahovú hodnotu, ktoré definujú stav $h^{(t)}$ nasledovne:

$$h^{(t)} = \tanh(\mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t)} + \mathbf{b})$$

Ako je zo vzťahu zrejmé, \mathbf{W} určuje váhu predchádzajúceho stavu, \mathbf{U} váhu aktuálneho vstupu a \mathbf{b} je prahová hodnota. Funkcia hyperbolického tangensu je tu použitá ako aktivačná funkcia. Uvedené parametre sú zdielané medzi časovými krokmi. To nám umožňuje spracovať sekvencie ľubovoľnej dĺžky bez zvyšovania pamäťových nárokov na ukladanie parametrov. Takto zadefinovanej rekurentnej neurónovej sieti však chýba výstup. Jeho definícia sa lísi podľa použitej topológie rekurentnej siete. Najznámejšie topológie RNN sú nasledovné:

- Sieť s rekurentnými spojeniami medzi skrytými stavmi h , produkujúca výstup o v každom časovom kroku t : $o^{(t)} = g(h^{(t)})$
- Sieť s rekurentnými spojeniami medzi výstupom v čase $t - 1$ a skrytým stavom v čase t : $h^{(t)} = f(o^{(t-1)}, x^{(t)}, \Theta)$. Výstup je produkovaný v každom časovom kroku: $o^{(t)} = g(h^{(t)})$.
- Sieť s rekurentnými spojeniami medzi skrytými stavmi h , produkujúca výstup o až po spracovaní celej sekvencie dĺžky τ : $o^\tau = g(h^\tau)$

Tréning rekurentných neurónových sietí prebieha tak, že sieť sa "rozvinie" v čase. Tento proces si vieme predstaviť ako vytvorenie n kópií tej istej siete, kde každá sieť posila svoj výstup na vstup nasledujúcej.

Jedným z problémov klasických rekurentných sietí (zadefinovaných vyššie) je ich neschopnosť zapamätať si dlhodobé súvislosti (*ang. long-term term dependencies*). V praxi to znamená, že vplyv vstupu $x^{(t-n)}$ na stav $h^{(t)}$ je nepriamo úmerný rozdielu času n . Tento problém je taktiež známy ako problém miznúceho gradientu (*ang. vanishing gradient problem*) [3].

Long short term memory (LSTM) [15]

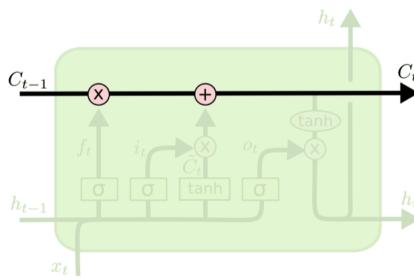
Long short term memory (LSTM) sú špeciálny druh rekurentných neurónových sietí, schopných učenia sa dlhodobých súvislostí. Ich používanie je v dnešnej dobe veľmi rozšírené a vynikajú úspešnosťou pri riešení problémov z rôznych domén (rozpoznávanie textu, hlasu i obrazových sekvencií). Základným princípom LSTM siete (niekedy nazývanou aj LSTM bunka) je tok informácií v čase cez takzvaný **stav bunky** (*ang. cell state*), označovaný ako $C^{(t)}$. Informácie v stave bunky sa menia na základe výstupu **hradiel**, ktoré sú súčasťou štruktúry LSTM bunky. Hradlo môžeme vnímať ako jednoduchú vrstvu perceptrónov (jedna váhová matica a jedna prahová hodnota), na ktorej výstup je aplikovaná sigmoidová aktivačná funkcia σ . Výstup hradla je potom násobený s funkciou, ktorú reguluje. Hradlá sú v LSTM bunke celkovo 3, pričom každé z nich má špecifickú funkciu:

- **Forget gate** (zabúdajúce hradlo) - reguluje aká časť informácií má byť zo stavu $C^{(t-1)}$ má byť zachovaná. Vstupom pre hradlo sú skrytý stav $h^{(t-1)}$

a vstupné dátá x^t . Výstup hradla $f^{(t)}$ je potom určený vzťahom: $f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f)$, kde W_f je matica váh a b je prahová hodnota, ktoré predstavujú parametre, ktoré sú predmetom učenia siete. Výstupom aktivačnej funkcie σ sú hodnoty z intervalu $(0, 1)$, kde 0 znamená stratu (zabudnutie) všetkých informácií zo stavu $C^{(t+1)}$ a 1 naopak, ponechanie informácií.

- **Input gate** (vstupné hradlo) - reguluje, aká časť nových informácií $C_n^{(t)}$ sa pridá k stavu $C^{(t-1)}$. Vstupy aj aktivačná funkcia sú rovnaké ako pri forget hradle: $i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}] + b_i)$. Nové informácie sú získané z rovnakých vstupov ale pomocou aktivačnej funkcie hyperbolického tangensu \tanh : $C_n^{(t)} = \tanh(W_c[h^{(t-1)}, x^{(t)}] + b_c)$. Stav $C^{(t)}$ môžeme teda zadefinovať ako: $C^{(t)} = f^{(t)} * C^{(t-1)} + i^{(t)} * C_n^{(t)}$
- **Output gate** (výstupné hradlo) - reguluje, čo bude výstupom (skrytým stavom) $h^{(t)}$. Výstup hradla $o^{(t)}$ je zadefinovaný obdobne ako pri input i forget hradle: $o^{(t)} = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$. Nový skrytý stav $h^{(t)}$ je závislý od stavu bunky $C^{(t)}$ a filtrovaný výstupom hradla $o^{(t)}$: $h^{(t)} = o^{(t)} * \tanh(C^{(t)})$.

Štruktúru bunky, kde sú vizualizované popísané hradlá možno vidieť na obrázku č. 1.



Obr. 1: Vnútorná štruktúra LSTM bunky

V porovnaní s klasickou rekurentnou neurónovou sieťou, kde sa sieť učila len jednu sadu váh, sa sieť založená na LSTM bunkách učí až štyri sady váh, čo nám poskytuje oveľa väčšiu štatistickú robustnosť. Rekurentné neurónové siete sú stále predmetom intenzívneho výskumu, ktorého výsledkom sú nové druhy

rekurentných buniek založených na princípe hradiel. Najznámejšia je GRU (*ang. gated recurrent unit*), ktorá dosahuje porovnateľné výsledky ako LSTM bunky [6].

2.3.4 Siamské neurónové siete

Siamská sieť predstavuje dve kópie (vetvy) tej istej siete, kde okrem topológie zdieľajú obe vetvy aj váhy vo všetkých vrstvách. Prvýkrát bola táto architektúra neurónovej siete navrhnutá a aplikovaná na verifikáciu podpisov [4]. Siamská architektúra je špeciálne navrhnutá na spracovávanie dát organizovaných do dvojíc (párov) tak, že každá vetva spracuje jeden prvok z páru a vygeneruje jeho príznakový vektor. V prípade, že je siamská sieť použitá na zisťovanie podobnosti páru dát, tak sú porovávané výstupné príznakové vektory z oboch vetiev, napríklad metódou euklidovskej vzdialenosťi. Pre tréning siamských sietí sa využívajú loss funkcie založené na vzdialosti porovnávaných párov. V prípade problémovej domény re-identifikácie osôb je vstupom pre siamskú sieť dvojica obrazov, resp. dvojica sekvenčí obrazov (v prípade re-identifikácie osôb vo video-sekvenčiach) a výstup siete je miera podobnosti osôb zachytených na vstupných obrazoch.

2.3.5 Contrastive loss funkcia

Na porovnanie výstupných príznakových vektorov zo siamskej siete sa používa *contrastive loss* funkcia [13]. Táto loss funkcia je založená na vzdialosti príznakových vektorov v n -rozmernom priestore, do ktorého boli sietou zasadené. Na určenie vzdialosti môže poslúžiť akákoľvek na to vhodná funkcia. V našom prípade bola zvolená funkcia *Euklidovskej vzdialenosťi*. Presná definícia contrastive loss funkcie je nasledovná:

$$L(W, Y, X_1, X_2) = (1 - Y) \frac{1}{2} (D_w)^2 + (Y) \frac{1}{2} \{ \max(0, m - D_w) \}^2$$

kde D_w je funkcia na zistenie vzdialenosťi bodov v n -rozmernom priestore. Súradnice bodov sú určené vektormi X_1 a X_2 . Podobnosť vektorov vyjadruje hodnota Y . Minimalizáciou tejto funkcie počas tréningu docielime to, že vektorov označené ako podobné budú v priestore reprezentovať body bližšie pri sebe, a

naopak, nepodobné vektory budú v priestore zasadzované ďalej od seba.

2.3.6 Cross entropy loss funkcia

Cross entropy vyjadruje mieru podobnosti dvoch pravdepodobnostných rozdelení p a q , kde p je skutočné pravdepodobnostné rozdelenie a q je rozdelenie aproximované neurónovou sieťou. Obe rozdelenia by mali byť vo forme softmax vektorov. Prvok vektoru na pozícii i predstavuje pravdepodobnosť identifikácie objektu i -tej triedy a súčet prvkov vektora je rovný jednej. Vektor p je najčastejšie vo forme *one-hot*, kde je jednotka iba na i -tom mieste ($p = [0, \dots, 1, \dots, 0]$). Rozdelenie q je softmax aktivácia výstupnej plne prepojenej vrstvy siete a je definované nasledovne:

$$q = \frac{e^{f_y i}}{\sum_j e^{f_j}}$$

,kde f je výstupný vektor plne prepojenej vrstvy. Samotná cross-entropy funkcia je potom zadefinovaná ako:

$$H(p, q) = - \sum_x p(x) \log q(x)$$

,kde x sú prvky hore definovaných diskrétnych pravdepodobnostných rozdelení.

2.4 Doterajšia práca v oblasti re-identifikácie osôb

Behom uplynulých rokov bolo publikovaných množstvo vedeckých prác, ktoré navrhovali odlišné prístupy k problematike re-identifikácie osôb. Aby bolo možné objektívne porovnanie týchto metód, je nutná ich evaluácia na rovnakom súbore dát. Tieto súbory dát nazývame **datasety**. Datasety pre re-identifikáciu osôb sú zhromažďované rôznymi akademickými inštitúciami a sú združené na prehľadovej web stránke¹. Daná stránka poskytuje informácie o datasete a odkazy na webové stránky konkrétnych datasetov. Súbory dát môžeme klasifikovať podľa viacerých kritérií. Najdôležitejšia je rozsiahlosť dát (počet osôb, počet kamier v systéme,

¹robustsystems.coe.neu.edu

celkový počet obrazov, počet obrazov na jednu osobu). Ďalším dôležitým faktorom je veľkosť obrazov a to, či zobrazujú už zdetekovanú a vysegmentovanú osobu alebo poskytujú celý záber kamery. V prípade, že dataset poskytuje obrazy kompletného záberu kamery, môžeme na danom dataseite evaluovať komplexné metódy re-identifikácie, zahrňujúce i vyššie zmienenú detekciu a segmentáciu.

Pre opis a porovnanie doteraz publikovaných metód re-identifikácie osôb ako aj pre prvotné prototypovanie nášho riešenia sme zvolili dataset **iLIDS-VID** [26]. Obrazy v tomto dataseite zachytávajú ľudí v príletovej hale letiska a obsahujú prvky, ktoré predstavujú najdôležitejšie úskalia pri re-identifikácii osôb (zmeny osvetlenia scény, postoja osoby, uhlu snímania kamery). Dáta pozostávajú so 42495 obrazov rozdelených na dve časti. Každá časť obsahuje zábery z jednej kamery. Obrazy z každej kamery sú ďalej delené do 300 adresárov, pričom každý adresár je označený číslom a obsahuje 23 až 192 obrazov (predstavujúcich video-sekvenciu) tej istej osoby. Každý obraz má veľkosť 128x64 pixelov a obsahuje už vysegmentovanú osobu.

Výber tohto datasetu bol motivovaný viacerými faktormi. Na prvom mieste je dostatočná veľkosť umožňujúca na dátach vyhodnocovať klasické (*ang. appearance driven*) prístupy k re-identifikácií osôb ako aj metódy využívajúce hlboké učenie a neurónové siete, ktoré vyžadujú rozsiahli súbor tréningových a testovacích dát. Ďalšou výhodou je veľký počet vedeckých prác, v ktorých autori vyhodnocovali nimi navrhnuté metódy na tomto dataseite. To nám dovolilo dané metódy porovnať a vybrať z nich tie najlepšie k hlbšej analýze.

2.4.1 Part Appearance Mixture [16]

Tento prístup k re-identifikácií osôb bol publikovaný v roku 2017 dvojicou autorov Furqan M. Khan a Francois Brémond z francúzskeho inštitútu pre výskum v informatike v článku pod názvom *Multi-shot Person Re-identification using Part Appearance Mixture*. Táto publikácia predstavuje jednu zo súčasných state-of-the-art metód re-identifikácie osôb. Pri evaluácii na iLIDS-VID dataseite dosiahla táto metóda najlepšie výsledky spomedzi dovtedy publikovaných a na danom dataseite testovaných metód.

Part appearance mixture (PAM) nevyužíva nijaké metódy hlbokého učenia. Jej

základom je opis výzoru osoby pravdepodobnostným modelom. Konkrétnie sa pri PAM využíva **Gaussian Mixture Model** (GMM). Tento pravdepodobostný model aproximuje vektor náhodných premenných ako kombináciu K Gaussovských pravdepodobostných rozdelení. Aproximácia prebieha na základe predom známych dátových bodov pomocou **EM (Expectation–maximization) algoritmu**. Dátovými bodmi v prípade re-identifikácie osôb sú príznakové vektory získané ľubovoľným lokálnym deskriptorom (v prípade metódy PAM boli použité deskriptory HOG a LOMO, ktorý bude popísaný nižšie).

Celkový výzorový (appearance) model Q pre danú sledovanú osobu q je formálne definovaný ako trojica zmiešaných Gaussovských modelov M_p^q , kde p predstavuje časť osoby (obrazu), z ktorej bol model vytvorený. Jeden model je vytvorený pre celý obraz a po jednom pre jeho hornú a spodnú časť. Horná a spodná časť obrazu predstavuje 60% obrazu od vrchného resp. spodného okraja. Súhrnný model Q môžeme teda označiť ako množinu:

$$Q = \left\{ M_p^q \mid p \in \{\text{celé, horná, spodná}\} \right\}$$

,kde model M_p^q je zložený z K d -rozmerných Gaussovských rozdelení $N(x|\mu, \Sigma)$:

$$N(X|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^{-1} (X - \mu)\right)$$

,kde X je náhodný vektor rozmeru d , $\mu = (\mu_1, \mu_2, \dots, \mu_d)$ je stredná (očakávaná) hodnota vektora X a Σ je jeho kovariančná matica. Kovariančná matica má veľkosť $d \times d$ a platí $\Sigma_{i,j} = cov(X_i, X_j)$. Rozmer d náhodného vektora X je v tomto prípade zhodný s rozmerom príznakových vektorov získaných zvoleným lokálnym deskriptorom.

Expectation–maximization algoritmus je iteratívny algoritmus založený na opakovaní dvoch krokov:

- **E (expectation) - krok** - výpočet funkcie $Q(\Theta, \Theta_t)$ ako strednej (očakávanej) hodnoty logaritmov pravdepodobností $p_\Theta(X, Z|x = X)$, kde x sú naše známe dátové body (príznakové vektory). Formálne:

$$Q(\Theta, \Theta_t) = E_{\Theta_t}(\log p_\Theta(X, Z|x = X))$$

- **M (maximization) - krok** - nájdenie parametrov Θ_{t+1} , ktoré maximalizujú hodnotu funkcie $Q(\Theta, \Theta_t)$. Formálne:

$$\Theta_{t+1} = \operatorname{argmax}_{\Theta} Q(\Theta, \Theta_t)$$

V prípade aplikácie EM algoritmu na nájdenie parametrov zmiešaného Gaussovského modelu, sú oba kroky algoritmu efektívne analyticky vypočítateľné. EM algoritmus je v tomto prípade použitý na nájdenie parametrov:

$$\Theta_{i=0..K} = \{\mu_{i=0..K}, \Sigma_{0..K}\}$$

takých, že:

$$\Theta^* = \operatorname{argmax}_{\Theta} p(X|\Theta) = \operatorname{argmax}_{\Theta} \prod_{i=0}^N p(X_i|\Theta)$$

,kde N je počet známych dátových bodov (v našom prípade príznakových vektorov) a $p(X_i|\Theta)$ je pravdepodobnosť i -teho dátového bodu, daná zmiešaným Gaussovským modelom s parametrami Θ . Maximalizujeme súčin pravdepodobností pre všetky známe dátové body.

Pravdepodobnosť jedného dátového bodu x daného zmiešaným modelom s K komponentami (individuálnymi Gaussovskými rozdeleniami) môžeme vyjadriť nasledovne:

$$p(x) = \sum_{i=0}^K w_i N(x|\mu_i, \Sigma_i)$$

Po nájdení zmiešaných Gaussovských modelov pre všetky osoby v datasete, resp. v aktuálnych záberoch kamier (v prípade reálnej aplikácie), je nutné vedieť určiť podobnosť dvoch osôb (metriku podobnosti). Oproti klasickému prístupu, kedy párujeme kľúčové body obrazu a porovnávame ich príznakové vektory priamo (napríklad Euklidovskou vzdialenosťou), je metrika podobnosti pri metóde PAM zložitejšia, keďže porovnávame pravdepodobnostné modely namiesto konkrétnych príznakov.

V publikácií je navrhnutá metrika podobnosti dvoch zmiešaných Gaussovských modelov M_1 a M_2 ako funkcia $d(M_1, M_2)$ definovaná nasledovne:

$$d(M_1, M_2) = \sum_{i,j=0}^{K_1, K_2} \pi_{1i} \pi_{2j} d(G_{1i}, G_{2j})$$

kde π_{nk} je konjugát k -teho komponentu n -tého zmiešaného modelu a funkcia $d(G_{1i}, G_{2j})$ reprezentuje vzdialenosť dvoch konkrétnych komponentov (Gaussovských rozdelení) skúmaných zložených modelov M_1 a M_2 .

V publikácií sú získané výsledky s použitím rôznych implementácií funkcie vzdialenosť $d(G_{1i}, G_{2j})$. Funkcia **Joffreys divergence** ($JDiv(G_{1i}, G_{2j})$), dosahovala podľa testov autorov najlepšie výsledky. Definovaná je nasledovne:

$$JDiv(G_i, G_j) = \frac{1}{2}(\mu_i - \mu_j)^T (\Sigma_i^{-1} + \Sigma_j^{-1})(\mu_i - \mu_j) + \frac{1}{2}tr(\Sigma_i^{-1}\Sigma_j + \Sigma_j^{-1}\Sigma_i - 2\mathbf{I})$$

kde Σ_k je konvariančná matica k -teho komponentu zloženého modelu a μ_k je jeho vektor stredných hodnôt. Matica identity je označená ako I .

Celková úspešnosť metódy PAM na datasete iLIDS-VID je **79,5 %** (rank 1). To z nej robí najúspešnejšiu metódu re-identifikácie osôb na danom datasete s relatívne veľkým náskokom (úspešnosť druhej metódy v poradí je 62% (rank1)).

2.4.2 Attentive Spatial-Temporal Pooling Networks for Video-based Person Re-Identification [27]

Táto metóda bola publikovaná v septembri roku 2017 kolektívom autorov z univerzity v Huazhong, Northwestern University a IBM T.J. Watson Research Center. Jedná sa o metódu využívajúcu najnovšie poznatky z oblasti hlbokých (konvolučných a rekurentných) neurónových sietí.

Základom modelu navrhnutého v publikácii je siamská architektúra siete tak ako je opísaná v stati 2.3.4. Vstupom siamskej siete sú páry sekvenčí obrazov.

Topológia vetiev siete ASTPN (ang. Attentive Spatial-Temporal Pooling Networks) je založená na kombinácii konvolučnej a rekurentnej neurónovej siete, obohatenej o **Spatial Pooling Layer** a **Temporal Pooling Layer**. Vrstva TPL (Temporal Pooling Layer) je úplne špecifická pre túto metódu a bola navrhnutá autormi publikácie.

Konvolučná sieť je zložená z troch po sebe nasledujúcich konvolučných vrstiev:

tiev. V prvých dvoch konvolučných vrstvách je využitá aktivačná funkcia $tanh$ a klasická max-pooling vrstva. Výstup z tretej vrstvy je iba po aktivačnej funkcií $tanh$ a namiesto max-pooling vrstvy je použitá špeciálna **Spatial Pooling Layer** (SPL).

Funkciou tejto konvolučnej siete je získanie príznakovéj mapy pre každý obraz zo vstupnej video-sekvencie. Aplikáciou SPL vrstvy sa príznaková mapa transformuje na príznakový vektor, ktorý ďalej slúži ako vstup pre rekurentnú sieť v konkrétnom čase t .

Rekurentná sieť je v modeli popísanom v publikácii implementovaná ako klasická rekurentná neurónová sieť popísaná v stati 2.3.3. Vstupom je vektor príznakov jedného obrazu video-sekvencie v čase t , získaný konvolučnou sieťou. Rekurentná sieť poskytuje ako výstup taktiež vektor v každom čase t . Po spracovaní celej sekvenčie vstupných vektorov sa jednotlivé výstupné vektory spoja do jednej matice P , resp. G , podľa toho, o ktorú vetvu siamskej siete sa jedná. Dáta v tejto matici charakterizujú už celú spracovávanú video-sekvenciu. Po získaní matíc P a G nasleduje aplikácia **Temporal Pooling Layer** (TPL). Jednou z možných modifikácií a prípadným vylepšením toho modelu je využitie LSTM buniek namiesto klasickej rekurentnej siete.

Spatial Pooling Layer

Narozdiel od obyčajnej max-pooling vrstvy, kde je veľkosť okienka (okolia z ktorého vyberáme maximálnu hodnotu) a jeho posun po obraze konštantný, je v spatial pooling vrstve vytvorených n výstupných príznakových vektorov, ktoré sú na konci zreťazené do konečnej reprezentácie i -teho obrazu r^i . Predpokladajme množinu veľkostí $\{(m_w^j, m_h^j) \mid j = 1, 2, \dots, n\}$. Veľkosti okienka a jeho posun sú potom dané vzťahmi:

$$win^j = \left(\left\lceil \frac{w}{m_w^j} \right\rceil, \left\lceil \frac{h}{m_h^j} \right\rceil \right)$$

$$str^j = \left(\left\lfloor \frac{w}{m_w^j} \right\rfloor, \left\lfloor \frac{h}{m_h^j} \right\rfloor \right)$$

,kde win je výsledná veľkosť okienka a str je jeho posun (ang. stride). Šírka

a výška vstupnej príznakovej mapy C je označená w a h . Výsledná množina príznakových vektorov je daná vzťahom:

$$b^j = f_R\{f_p(C, win^j, str^j)\}$$

Funkcia f_p reprezentuje funkciu max-pooling, vykonávanú na príznakovej mape C s danými parametrami. Funkcia f_R predstavuje operáciu transformácie (zmeny tvaru) matice na vektor. Výslednú reprezentáciu obrazu r získame zrefazením vektorov b :

$$r = b^1 \oplus b^2 \oplus \dots \oplus b^n$$

Takto reprezentovaný obraz vstupuje ďalej na spracovanie do rekurentnej neurónovej siete.

Temporal Pooling Layer

Kedže výstupné matice P a G rekurentnej neurónovej siete obsahujú množstvo redundantných informácií (napríklad o nemennej farbe oblečenia osoby alebo svetelných podmienkach), ich priame porovnávanie podobnostnou metrikou (euklidovská vzdialenosť) by mohlo výrazne znížiť celkovú presnosť modelu. Preto autori publikácie navrhli vrstvu Temporal Pooling Layer (TPL), ktorá má sieti pomôcť sústrediť sa na efektívne informácie z matíc P a G a ignorovať informácie redundantné. Matice P a G sú definované ako $P, G \in \mathbb{R}^{T \times N}$. T predstavuje počet časových krokov v rekurentnej sieti a N dĺžku vektora b vystupujúceho z SPL.

Princípom TPL je výpočet matice $A \in \mathbb{R}^{T \times T}$:

$$A = \tanh(PUG^T)$$

,kde $U \in \mathbb{R}^{N \times N}$ je matica váh, ktorú sa bude sieť učiť. Aktivačná funkcia tejto vrstvy je \tanh . Následne sú z každého stĺpca matice A vybrané maximálne hodnoty a sú uložené do vektora $t_p \in \mathbb{R}^T$ (ang. column-wise max-pooling). Podobne sú vybrané aj maximálne hodnoty v riadkoch a uložené do vektora $t_g \in \mathbb{R}^T$ (ang. row-wise max-pooling). Prvok na i -tej pozícii v vektore t_p , resp. t_g predstavuje váhu i -teho obrazu video-sekvencie, z ktorej bola konvolučno-rekurentnou sieťou

vytvorená matica P , resp. G . Na vektory t_p a t_g je potom aplikovaná soft-max funkcia a vzniknú vektorov pomerov $a_p = softmax(t_p)$ a $a_g = softmax(t_g)$. Funkcia soft-max upraví vektory tak, že jeho hodnoty budú v intervale $\langle 0, 1 \rangle$ a súčet všetkých hodnôt vo vektore bude rovný 1. V praxi to znamená, že i -ty prvok vektora a_p je pomer i -tej váhy vo vektore t_p k súčtu všetkých váh vo vektore:

$$[a_p]_i = \frac{[t_p]_i}{\sum_{j=0}^T [t_p]_j}$$

Finálne reprezentácie video-sekvencií $v_p, v_g \in \mathbb{R}^N$ sú potom vytvorené ako násobok medzi maticami P , resp. G a ich príslušným pomerovým vektorom a .

$$v_p = a_p P^T$$

$$v_g = a_g G^T$$

Podobnostná metrika využíva vyššie zmenenú euklidovskú vzdialenosť a Hinge loss funkciu a je zadefinovaná nasledovne:

$$E(v_p, v_g) = \begin{cases} \|v_p - v_g\|^2 & p = g \\ \max(0, m - \|v_p - v_g\|^2) & p \neq q \end{cases}$$

,kde m je parameter Hinge loss funkcie určujúci mieru, o ktorú chceme oddeliť príznaky rozdielnych osôb.

Celková úspešnosť metódy ASTPN na datasete iLIDS-VID je **62 %** (rank 1), čo z nej robí druhú najúspešnejšiu metódu na danom datasete.

2.5 Dostupné softvérové rámce

Pre vývoj aplikácií využívajúcich hlboké učenie, ako aj pre výskum v tejto oblasti, je dostupných viacero rámcov (*ang. frameworks*), ktoré majú dané činnosti podporovať. Nižšie je uvedený zoznam a krátky popis dostupných open-source rámcov:

- **Tensorflow** - rámc použitý vrámci tejto práce. Tento rámc bol vyvinutý v spoločnosti Google. Využíva metódy výpočtových grafov a je použiteľný na rôzne odvetvia hlbokého učenia (rozpoznávanie reči, obrazov, textu).

- **Caffe** - vytvorený špeciálne na prácu s konvolučnými sieťami pre klasifikáciu obrazov. Chýbajú mu prvky pre spracovanie časových sekvencií, textu a podobne, avšak poskytuje možnosť ľahko využívať pred-trénované konvolučné modely.
- **CNTK** - vytvorený spoločnosťou Microsoft. Popularitu získal najmä vrámci spracovania reči, avšak nie je obmedzený len na túto oblasť. Podporuje prácu v jazykoch C++ a Python a taktiež poskytuje pred-trénované modely.
- **Theano** - nízko-úrovňový rámec disponujúci flexibilitou a rýchlosťou (optimizáciou). Často je používaný s vysoko-úrovňovými nadstavbami, ako napr. Keras.

2.5.1 Tensorflow

Tensorflow je open-source knižnica (rámc) používaná na numerické výpočty. Je založená na výpočtových (*ang. data-flow*) grafoch, kde vrcholy grafu reprezentujú matematické operácie (násobenie, sčítanie, aktivačné funkcie, a iné) a hrany grafu predstavujú viacrozmerné polia dát (tenzory), ktoré sú danými operáciami spracovávané. Systém bol v začiatkoch vyvíjaný tímom Google Brain za účelom výskumu hlbokého učenia a neurónových sietí. Zdrojové súbory knižnice boli ne-skôr zverejnené pod licenciou Apache 2.0 a sú dostupné na². K dnešnému dňu projekt viac než 1200 prispievateľov.

Aj keď hlavnou doménou knižnice zostalo hlboké učenie, jej koncepcia je dostatočne abstraktná aj na aplikáciu v iných odvetviach. Hlavnou prednosťou knižnice je jej schopnosť výpočty paraleлизovať medzi viaceré CPU, resp. GPU alebo dokonca medzi distribuované (clustrové) systémy. Paralelizáciu na GPU umožňujú knižnice **CUDA** a **cuDNN** od spoločnosti NVIDIA.

Knižnica poskytuje svoje API pre viaceré jazyky ako napríklad C++, Java, Haskell a Go a dokonca sú vývojármí tretích strán vyvíjané aj API pre C# ale aj iné menej známe jazyky. Najväčšej obľube a podpore zo strany vývojárov sa však teší jazyk Python, ktorý bude použitý aj v našom prototype. Tensorflow API

²<https://github.com/tensorflow/tensorflow>

pre Python vyniká vynikajúcou interoperabilitou s inými balíčkami ako napríklad **numpy** pre prácu s viacrozmernými poliami alebo **matplotlib** na vizualizáciu dát.

2.5.2 Používanie knižnice Tensorflow

Všeobecný postup práce pri používaní tensorflow-u na úlohy spojené so hlbokým učením je možné rozdeliť do nasledujúcich krokov.

Príprava dát

Rozdelenie vstupných dát na tréningové a testovacie. Tréningové sú použité na trénovanie siete, testovacie na evaluáciu jej výsledkov. Keďže sa vo väčšine prípadov jedná úlohy strojového učenia s učiteľom (s pomocou), akými sú napríklad klasifikácia alebo regresia, pozostávajú vstupné dáta z párov: objekt a požadovaná výstupná hodnota. Objekt je typicky vektor (v tensorflow-e tenzor ľubovoľných rozmerov) dát, napríklad pixely obrazu alebo znaky vety. Požadovaná výstupná hodnota môže byť vyjadrená ako jedno číslo, ktoré kóduje triedu do ktorej objekt patrí (v prípade klasifikácie) alebo je to taktiež vektor (napríklad vektor pravdepodobností alebo pomerov).

Vytvorenie výpočtového grafu

Výpočtový graf sa skladá z uzlov predstavujúcich operácie a z hrán predstavujúcich vstupné a výstupné dáta týchto operácií. Dôležité je, aby graf na vstupe prijímal a na výstupe poskytoval dáta v rovnakého tvaru ako sú tvary objektov a požadovaných v tréningových, resp. testovacích dátach. Pojem tvar budeme ďalej používať v zmysle rozmeru tenzorov, s ktorými pracujeme. V tensorflow-e sú tri druhy dátových tenzorov:

- **Placeholders** - do tohto tenzoru sa vkladajú dáta manuálne, inak zostáva prázdný. Najčastejšie slúži na vkladanie vstupov do siete. Ak sa nenaplnený tenzor použije pri výpočte, vyvolá sa výnimka.
- **Variables** - dáta v tenzore sú inicializované už pri jeho vytvorení a druh inicializácie dát si môže používateľ sám zvoliť. Na výber je inicializácia

náhodnými hodnotami, inicializácia hodnotami z rovnomerného alebo normálneho rozdelenia, alebo inicializácia konštantnými hodnotami. Tenzory tohto druhu sú používané na uchovávanie váh v našom modeli a hodnoty v nich uložené sú perzistentné medzi iteráciami tréningového procesu.

- **Constants** - dátá v tomto tenzore sú určené pri inicializácii a ďalej zostávajú nemenné. Perzistencia je rovnaká ako pri tenzoroch typ *Variable*. Použitie konštantných tenzorov je zriedkavé avšak pri niektorých úlohách nutné.

Pri inicializácii akéhokoľvek tenzoru používateľ sám určuje jeho tvar, prípadne iné parametre ako názov a dátový typ.

Operácií nad tenzormi je v Tensorflow-e implementovaných mnoho. Od jednoduchých unárnych a binárnych aritmetických a logických operácií až po zložitejšie špeciálne operácie, ako napríklad maticové, goniometrické a štatistické operácie.

Loss funkcia, optimizér a vyhodnocovacie metriky

Voľba správneho druhu loss funkcie a optimizéra k typu riešenej úlohy je kľúčové pre časovú efektivitu tréningu siete. Pre úlohy klasifikácie a regresie sa najčastejšie používa cross-entropy loss funkcia. Tensorflow poskytuje aj iné loss funkcie ako napríklad Hinge loss alebo strednú kvadratickú odchýlku (*ang. MSE - mean squared error*).

Všetky optimizéry v Tensorflow-e sú založené na gradientovej optimalizácii. Implementovaný je klasický stochastic gradient descent algoritmus ako aj jeho viaceré rozšírenia, ktoré boli v posledných rokoch publikované. Medzi známe algoritmy rozširujúce gradient descent patria: Adam, AdaMax [17] a AdaGrad [10]. Prefix *Ada-* v názvoch týchto modifikovaných algoritmov znamená *adaptive* a odkazuje na to, že v daných algoritnoch je miera zmeny parametrov (taktiež nazývaná *learning rate*) adaptívna (mení sa s veľkosťou a smerom gradientu), zatiaľ, čo pri jednoduchom stochasticom gradient descent je veľkosť zmeny parametrov v jednom kroku optimalizácie konštantná.

Vyhodnocovacie metriky pomáhajú pri ladení a hodnotení vlastností modelu. Medzi základné evaluačné metriky patrí **presnosť** (*ang. accuracy*). V prípade klasifikácie sa dá presnosť vyjadriť ako pomer počtu správnych klasifikácií k počtu

všetkých klasifikácií vykonaných na testovacích dátach. Existujú však aj rôzne ukazovatele, ktoré nám dávajú hlbší náhľad do procesu tréningu. Medzi najhlavnejšie patrí vývoj hodnoty loss funkcie v čase. Ten nám poskytuje informácie či a k akej hodnote loss funkcia konverguje a či má ďalší tréning zmysel. Podobné informácie vedia poskytnúť aj rôzne histogramy váh alebo vývoj priemeru váh v konkrétnej vrstve siete v čase (taktiež by mal konvergovať). Tieto a podobné metriky si môže užívateľ vypočítať a vizualizovať sám (napríklad v knižnici matplotlib), avšak Tensorflow poskytuje zabudovaný nástroj na logovanie a prezentáciu tohto druhu dát s názvom **Tensorboard**.

Tréning a evaluácia

Po zostavení výpočtového grafu, učení tréningového cieľa a prostriedkov k nemu potrebných (loss funkcia a optimizér) je možné prejsť k samotnému procesu tréningu. Doposiaľ bol výpočtový graf neaktívny a v pamäti bola uložená len jeho topológia (typy uzlov, tvary tenzorov predstavujúcich hrany a ich prepojenia). Na to aby sme mohli s grafom interagovať, je nutné započať takzvanú **Session**. V rámci session sa typicky vykonáva tréningová alebo testovacia slučka v ktorej cez `session.run` funkciou napĺňajú vstupné placeholder tenzory modelu a vykonáva sa vybraná operácia z výpočtového grafu. Pri procese tréningu je operácia, ktorá sa vykonáva typicky operácia optimizéra, ktorý sa stará o spätné šírenie chyby (*ang. backpropagation*). V každej iterácii slučky sa do modelu posielá jedna dávka (batch) vstupných dát. Veľkosť dávky je určená tvarom vstupného placeholder tenzoru. Princíp evaluačnej slučky je rovnaký ako princíp tréningovej, avšak vykonávajú sa v nej operácie, ktoré súvisia s výpočtom nami zvolených ukazovateľov (presnosť, hodnota loss funkcie, prípadne iné).

2.5.3 Vyššie úrovne Tensorflow API

Princípy používania Tensorflow-u, tak ako sú opísané vyššie, sú univerzálne platné, avšak topológie dnešných state-of-the-art modelov, najmä pri zložitejších úlohách, akými sú rozpoznávanie objektov alebo spracovanie prirodzeného jazyka, sú natoľko komplexné, že výstavba výpočtového grafu zo základných operácií a manuálnym inicializovaním premenných by bola zdĺhavá a neprehľadná. Tensorflow preto

poskytuje funkcie na tvorbu najpoužívanejších vrstiev, ktoré zaobalaťujú deklaráciu a inicializáciu premenných ako aj operácií. Pomocou niekoľkých parametrov poskytnutých konštruktorom príslušných tried je možné vytvoriť konvolučné vrstvy pre spracovanie jedno, dvoj i troj-rozmerných dát, plne prepojené (fully-connected, dense) vrstvy poskytujúce funkciaľitu viacvrstvového perceptrónu ako aj rôzne druhy pooling vrstiev (max, average), taktiež uspôsobené pre dátá rôznych rozmerov. Všetky triedy predstavujúce vyššie uvedené vrstvy sú obsiahnuté v balíku `tf.layers`.

Ďalším dôležitým balíkom poskytujúcim vysokoúrovňové API je `tf.contrib.rnn`. Tento balík obsahuje triedy implementujúce viaceré druhy buniek rekurentných neurónových sietí: LSTM bunky opísané v časti 2.3.3, GRU bunky, ale aj niektoré najnovšie architektúry rekurentných buniek, ktoré boli publikované v nedávnej minulosti (NAS [29] a UGRNN [7] bunky). Názov pod-balíka *contrib* indikuje, že kód je z časti od prispievateľov do repozitára a stále sa môže meniť a optimalizovať predtým, než bude pridaný ako stabilná súčasť knižnice. Z tohto dôvodu sa balík `tf.contrib` neodporúča používať v systémoch nasadených v produkcií, avšak jeho používanie na výskumné účely je vhodné, ba až žiaduce, vzhľadom na možnosti odhalenia chýb.

Najvyššou úrovňou abstrakcie tvorby topológie modelu je využitie knižnice **Keras**, ktorá bola pôvodne vyvíjaná ako nadstavba pre Tensorflow a dnes je už súčasťou jeho stabilnej verzie. Keras poskytuje rovnakú úroveň abstrakcie tvorby vrstiev ako vyššie opísané balíky, ale naviac pridáva aj zjednodušenie definovania loss funkcie, konfigurácie optimizéra ako aj celého tréningového procesu (napr. deľba dát na tréningové dávky a získavanie dát pre vyhodnocovacie metriky). S väčšou abstrakciou sa však znižuje celková kontrola nad obsahom výpočtového grafu a procesom tréningu, čo môže mať za následok nižšiu presnosť modelu alebo časovú neefektivitu či tréningu, alebo aj vyhodnocovania nových dát. Preto je použitie vysokoúrovňových nástrojov vhodné hlavne na počiatok rýchle prototypovanie a neskôr je vhodné model optimalizovať s použitím nízko-úrovňového API.

2.5.4 Ukážka tvorby modelu

V nasledujúcej ukážke kódu je vyobrazená tvorba modelu "hlbokej" (viac než jednovrstvovej) konvolučnej siete. Kód obsahuje väčšinu konceptov opísaných vyššie.

Ukážka 1: Ukážka tvorby dvojvrstvovej konvolučnej siete na klasifikáciu obrazov

```
input_x = tf.placeholder(tf.float32, [batch_size, image_height * image_width * image_channels])

input_y = tf.placeholder(tf.int32, [batch_size])

input_layer = tf.reshape(input_x, [-1, image_height, image_width, image_channels])

conv1 = tf.layers.conv2d(inputs=input_layer,
                       filters=32,
                       kernel_size=[5, 5],
                       padding='same',
                       activation=tf.nn.relu)

max1 = tf.layers.max_pooling2d(inputs=conv1,
                               pool_size=[2, 2],
                               strides=2)

conv2 = tf.layers.conv2d(inputs=max1,
                       filters=64,
                       kernel_size=[5, 5],
                       padding='same',
                       activation=tf.nn.relu)

max2 = tf.layers.max_pooling2d(inputs=conv2,
                               pool_size=[2, 2],
                               strides=2)

max2 = tf.reshape(max2, [-1, 7 * 7 * 64])

dense = tf.layers.dense(inputs=max2,
                       units=1024,
                       activation=tf.nn.relu)

dropout = tf.layers.dropout(dense, rate=0.4, training=True)

logits = tf.layers.dense(dropout, units=classes)

one_hot_labels = tf.one_hot(indices=input_y, depth=classes)

loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot_labels, logits=logits)

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)

optimizer = optimizer.minimize(loss=loss, global_step=tf.train.get_global_step())
```

Kód je použiteľný na klasifikáciu obrazov. Najskôr sú inicializované dve placeholder premenné. Konštruktor placeholder premenných očakáva dva argumenty: primitívny dátový typ tenzoru a tvar tenzoru špecifikovaný ako zoznam (list).

Prvá placeholder premenná *input_x* je určená pre vstup obrazov do siete. Tvar tenzoru predpokladá, že dátá obrazu sú serializované do jednorozmerného vektoru o veľkosti $w \times h \times c$, kde w je šírka obrazu v pixloch, h je jeho výška a c je počet

farebných kanálov obrazu. V určitých klasifikačných modeloch sa ku farebným kanálom na vstup pridávajú aj kanály optického toku, poprípade iné doplnkové kanály. Druhým rozmerom tenzoru je veľkosť vstupnej dávky (*batch_size*), ktorá určuje počet obrazov (hore opísaných vektorov) vstupujúcich do siete v jednej iterácii tréningovej slučky.

Druhá placeholder premenná *input_y* slúži na vstup očakávaných výsledkov klasifikácie obrazov vo vstupnej dávke. Očakávané výsledky sú celé čísla reprezentujúce triedu, do ktorej klasifikovaný obraz patrí. K jednému obrazu prislúcha práve jedna trieda, preto je rozmer tohto tenzoru iba *batch_size*.

Nasleduje operácia *reshape*, ktorá mení tvar vstupného tenzoru *input_x* na štvorozmerný tenzor [*batch_size*, *w*, *h*, *c*]. Tento tenzor je vstupom pre prvú konvolučnú vrstvu. Konvolučnú vrstvu vytvárame volaním funkcie *tf.layers.conv2d*, ktorá očakáva argumenty: vstupný tenzor, počet filtrov (kernelov), s ktorými sa bude vstupný obraz (tenzor) konvolovať, veľkosť filtrov [*x*, *y*] a aktivačnú funkciu aplikovanú na výsledky konvolúcie. Je tiež možné argumentami špecifikovať spôsob, akým sa algoritmus vysporadúva s okrajovými pixlami obrazu a veľkosť posunu filtra po vstupnom obraze (preddefinované ako 1 pixel). V príklade vyššie je v konvolučných vrstvách použitých 32 resp. 64 filtrov o veľkosti 5×5 pixela a aktivačná funkcia *rectified linear*. Výstup z konvolučnej vrstvy je tenzor v tvaru [*batch_size*, *w*, *h*, *f*], kde *f* je počet použitých konvolučných kernelov. Môžeme teda povedať, že výstupom je *f* príznakových máp o veľkosti vstupného obrazu.

Po každej konvolučnej vrstve nasleduje max-pooling vrstva vytváraná volaním funkcie *tf.layers.max_pooling2d*. Jej argumenty sú: vstupný tenzor (výstup konvolúcie), veľkosť oblasti (okienka) [*x*, *y*], z ktorej sa vyberá maximálna hodnota a veľkosť posunu okienka (typicky rovnaké ako rozmer okienka). V našom príklade je veľkosť okienka 2×2 a posun je rovnakej veľkosti. To spôsobí zmenšenie výšky i šírky obrazu (v prípade výstupu z konvolučnej vrstvy už môžeme hovoriť o príznakovej mape) o polovicu a celkovo sa počet hodnôt (pixelov) v príznakovej mape zníži na štvrtinu pôvodného počtu.

Po poslednej max-pooling vrstve nasleduje opäť operácia *reshape*, ktorá upraví tvar výstupu max-pooling vrstvy na *batch_size* jednorozmerných vektor, pričom v každom vektore je serializovaný obsah výstupnej príznakovnej mapy. Takto upravené príznakové mapy vstupujú do plne prepojenej vrstvy vytvorenej operáciou

`tf.dense` pozostávajúcej v našom prípade z 1024 neurónov a aktivačná funkcia vrstvy bola nastavená na *rectified linear*. Oba tieto parametre je možné nastaviť v konštruktore. Výstup `dense` vrstvy je v tvare `[batch_size, units]`, kde `units` je počet neurónov vo vrstve.

V modeli ďalej nasleduje takzvaná výpadková vrstva (*ang. dropout vrstva*) . Jej účelom je zabrániť pretrénovaniu siete (*ang. overfitting*). Princípom operácie `dropout` vrstvy je ignorovanie každého neurónu vstupnej vrstvy s určitou pravdepodobnosťou p . Vytvorenie `dropout` vrstvy je realizované volaním konštruktora `tf.layers.dropout` a pravdepodobnosť p je nastavená cez jeho argument. Výstup `dropout` vrstvy má rovnaký tvar ako jej vstup.

Nasleduje výstupná plne prepojená vrstva. Počet neurónov výstupnej vrstvy je zhodný s počtom tried, do ktorých klasifikátor klasifikuje. Výstupná vrstva nemá nijakú aktivačnú funkciu. Očakávané výstupy (čísla tried) sú následne transformované na tzv. *one-hot* enkódovanie (dĺžka slova v bitoch je rovná počtu tried a nastavený je vždy len jeden bit). Výstupná vrstva a enkódované očakávané výstupy sú vstupom pre loss funkciu. V príklade je použitá *softmax cross entropy* funkcia. Tréningová funkcia bola zadefinovaná ako minimalizovanie danej loss funkcie pomocou gradient descent optimizéra.

3 Návrh

3.1 Koncept riešenia

Pri návrhu nášho riešenia re-identifikácie osôb sa zameriame na metódy založené na hlbokom učení. Využité bude učenie s učiteľom pomocou hlbokých neurónových sietí. Najskôr budú na súbore dát natrénované čisto konvolučné modely, ktoré budú spracúvať samostatné obrazy. Neskôr budú naše hlavné modely pozostávať z kombinácie konvolučnej a rekurentnej neurónovej siete a ich základná topológia bude inšpirovaná modelom opísaným v stati 2.4.2. Dôraz v práci bude kladený na porovnanie vplyvu rôznych úprav detailov tejto topológie na vplyv celkovej úspešnosti a efektivity modelu.

Medzi skúmané úpravy bude patriť hĺbka konvolučnej siete (počet konvolučných a pooling vrstiev), rozmer a počet konvolučných kernelov ako aj rozmer pooling okien. Tieto zmeny budú mať vplyv na veľkosť a vyjadrovaciu silu výsledného príznakového vektora.

Zmeny v topológií sa môžu taktiež dotknúť rekurentnej časti modelu. Tu sa ponúka možnosť experimentovať s druhom buniek, z ktorých bude rekurentná neurónová sieť zložená. Ďalšími modifikateľnými parametrami sú: počet buniek v rekurentnej vrstve, počet časových krokov použitých pri trénovaní siete a druh výstupu z rekurentnej siete (výstup v každom časovom kroku alebo iba v poslednom kroku).

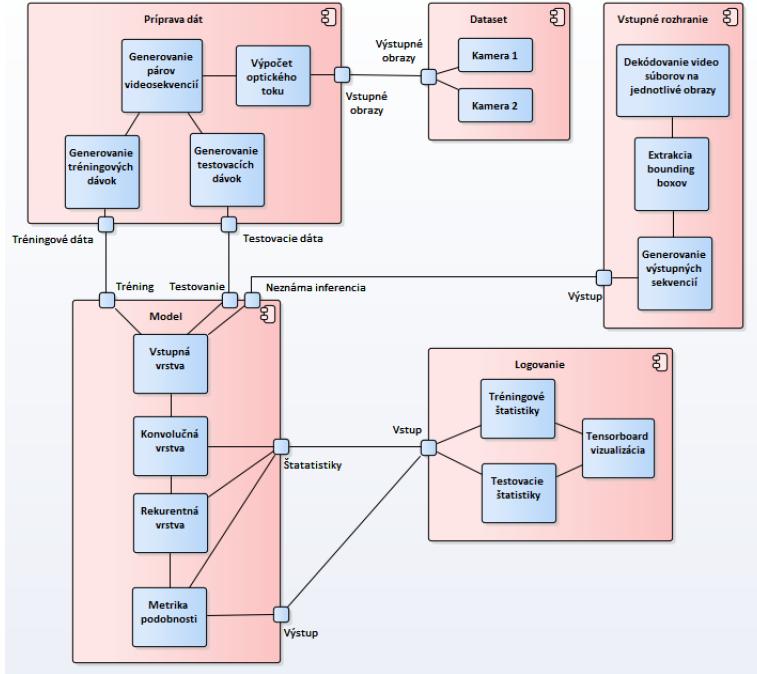
Po získaní príznakových vektorov reprezentujúcich obe porovnávané video-sekvencie môže nasledovať ich priame porovnanie podobnostnou metrikou alebo porovnaniu bude predchádzať určitý druh spracovania príznakových vektorov, ktorý z nich odstráni redundantné dáta a zameria sa len na dáta, ktoré sú pre porovnanie efektívne. V publikácii, ktorou je topológia inšpirovaná, túto úlohu plní vrstva TPL (*ang.* Temporal Pooling Layer), opísaná v 2.4.2. Príkladom iného prístupu je natrénovanie plne prepojenej podsiete, ktorá sa učí váhu jednotlivých obrazov vo video-sekvencií[28].

Kedže trénované modely, ktoré budú súčasťou riešenia, budú trénované na súbore dát i-LIDS-VID, bude ďalšou súčasťou riešenia rozdelenie tohto súboru dát na tréningové a testovacie páry video-sekvencií (poprípade obrazov). Páry video-

sekvencií budú zostavované tak, že jedna video-sekvencia je z kamery č.1 a druhá z kamery č.2. Počet párov s pozitívou zhodou (osoby na video-sekvenciách sú totožné) a párov s negatívou zhodou (osoby nie sú totožné), na ktorých sa bude model učiť, by mal byť rovnaký, aby sa zabránilo pre-trénovaniu siete na konkrétny druh zhody. Dôležitá je aj štatistická rozmanitosť pri tvorbe párov. Nesmie byť vytvorených viacerých párov zobrazujúcich tú istú osobu, za predpokladu, že pre zvyšné osoby bol vytvorený len jeden párs. To by taktiež negatívne ovplyvnilo výsledky modelu v dôsledku pre-trénovania na danú osobu. Súčasťou modulu na prípravu dát bude aj algoritmus, ktorý z viacerých párov vytvorí finálnu dávku (*ang. batch*), ktorá bude vstupom pre model.

Vo viacerých publikáciach využívajúcich model konvolučno-rekuretnnej siete sa ako vstup pre sieť okrem farebných kanálov obrazov využívajú aj kanály optického toku. Optický tok je pole dvojrozmerných vektorov, ktoré opisuje zmenu polohy ΔX a ΔY jednotlivých bodov (x, y) medzi obrazom v čase t a obrazom v čase $t + \Delta t$. V našom prípade sa optický tok vyráta pre všetky za sebou nasledujúce dvojice obrazov vo vstupnej video-sekvencii.

Výpočet optického toku môže byť realizovaný napríklad Lucas-Kanade-ovou metódou [1], ktorá je implementovaná v knižnici OpenCV ako samostatná funkcia. Inou možnosťou je použiť hustý optický tok, rátaný napríklad Fernebackovou metódou [11], ktorú knižnica OpenCV taktiež podporuje.



Obr. 2: Diagram komponentov navrhovaného systému

3.2 Príprava vstupných dát

Príprava vstupných dát bola realizovaná formou samostatných programových modulov, ktorých cieľom bolo poskytnúť rozhranie v podobe procedúr na jednoduché získavanie tréningových, resp. testovacích dávok obrazov alebo video-sekvenčí. Základnými funkciami týchto programových modulov je rozdelenie osôb z datasetu na tréningové a testovacie osoby, výber konkrétnych párov obrazov, resp. video-sekvenčí, ich prípadná augmentácia, zistenie optického toku v prípade video-sekvenčí a konverzia dávky do viacrozmerného poľa, ktorého rozmery sú kompatibilné so vstupnou vrstvou trénovaného modelu.

Vzhľadom na pomerne malú veľkosť datasetu (300 osôb), bola delba dát vykonaná v pomere 80% tréningové a 20% testovacie dát. Z tréningových, resp. testovacích dát sú následne vytvárané páry obrazov alebo video-sekvenčí, pričom jedna časť páru je z kamery č.1 a druhá z kamery č.2. Páry môžu zobrazovať pozitívnu zhodu (rovnakú osobu na oboch kamerách) alebo negatívnu zhodu (rozdielne osoby). Indexy osôb, z ktorých sa páry skladajú, sú volené náhodne v

medziach tréningových, resp. testovacích indexov, pričom osoby sú indexované abecedne vzostupne. V prípade video-sekvencií je z celej video-sekvencie danej osoby náhodne vybraná sub-sekvencia o dĺžke $n < 23$ obrazov. Toto rozhodnutie je motivované faktom, že dĺžky kompletnejších sekvencií sú variabilné vzhľadom na osobu a kameru a pohybujú sa v rozmedzí 23 až 192 obrazov.

Augmentácia vstupných dát sa lísi podľa toho či sa jedná o pári obrazov, alebo video-sekvencií. V prípade párov jednotlivých obrazov boli aplikované rôzne kombinácie operácií vertikálneho zrkadlenia, rotácie, orezania a vynulovania hodnôt všetkých troch farebných kanálov v určitých častiach obrazu (*ang. Coarse Dropout*), avšak nie na každý obraz, ale iba s určitou pravdepodobnosťou p . V prípade video-sekvencií bola aplikovaná iba operácia vertikálneho zrkadlenia, taktiež s pravdepodobnosťou p na každý obraz oboch sekvencií v páre.

V prípade video-sekvencií bol ešte následne vypočítaný hustý optický tok (*ang. dense optical flow*). Jeho výpočet bol realizovaný algoritmom, ktorý popísal Gunner Farneback [11]. Rozdiel medzi riedkym (*ang. sparse*) a hustým optickým tokom je ten, že hustý optický tok je rátaný pre každý bod (pixel) obrazu a nie len pre kľúčové body obrazu, ako je tomu v prípade riedkeho optického toku. Výstupom Farnebackovho algoritmu sú polárne súradnice (m, ϕ) , vyjadrujúce posun každého bodu medzi dvomi obrazmi. Tieto súradnice boli následne pripojené k trom existujúcim farebným kanájom (r, g, b) obrazov video-sekvencií v páre a vznikla ich výsledná päť-kanálová reprezentácia (r, g, b, m, ϕ) . Hodnoty všetkých kanálov boli normalizované min-max metódou na hodnoty z intervalu $< 0, 1 >$. Označenie (*ang. label*) toho, či je daný pári zhodný, je reprezentované číselnou hodnotou 0.0 pre negatívny pári a 1.0 pre pozitívny pári.



Obr. 3: Ukážka vygenerovanej videosekvencie



Obr. 4: Ukážka vygenerovanej augmentovanej dávky o veľkosti 10 párov zloženej zo samostatných obrazov

Používateľ si pomocou argumentov procedúry generujúcej vstupnú dávku obrazov môže zvoliť veľkosť dávky, to či sa jedná o testovaciu alebo tréningovú dávku, ako aj to, či sa má daná dávka augmentovať a či sa má pre ňu počítať optický tok. Na obrázku č. 3 je zobrazená ukážka vygenerovaného negatívneho páru video-sekvencií. Obrázok č. 4 zobrazuje ukážku pozitívnych i negatívnych párov samostatných obrazov, na ktorých vidno aj efekt augmentácie čiastočného vymazania.

3.3 Tréning

Tréningové programy boli navrhnuté tak, aby bola možná ľahká zámena modulov pripravujúcich vstupné dátá a modulov obsahujúcich samotný model. Taktiež boli navrhnuté tak, aby bolo možné tréning kedykoľvek prerušiť. Túto funkcionality zabezpečilo načítavanie váh modelu zo súboru pred začiatkom tréningu a ukladanie váh do súboru v pravidelných intervaloch počas tréningu (každých n epoch). Optimizér sa do výpočtového grafu pridal taktiež až v tréningovom module, aby bol nezávislý na trénovanom modelu. Optimalizovaná loss funkcia bola vo väčšine modelov základná contrastive loss funkcia opísaná v stati 2.3.5. Špeciálne prípady tréningového cieľa budú opísané pri návrhu modelu, kde boli zmeny aplikované.

Na zistenie konvergencie tréningu modelu bol počítaný kĺzavý priemer hodnôt loss funkcie z posledných 1000 tréningových epoch a bol pravidelne počas tréningu vypisovaný na štandardný výstup. Tréning bol vždy ukončený manuálne po zhodnotení, že sa hodnota kĺzavého priemera ustálila.

3.4 Navrhnuté modely neurónových sietí

Experimenty nad využívaným súborom dát i-LIDS-VID boli vykonávané v dvoch fázach. Modely navrhnuté v prvej fáze boli zamerané na určenie podobnosti osôb na základe párov jednotlivých (statických) obrazov. Druhá fáza experimentov pozostávala z trénoania modelov, ktorých vstupom už boli páry video-sekvencií pozostávajúce z konštantného počtu obrazov. Všetky úspešne natrénované modely pri oboch druhoch experimentov boli vyhodnocované rovnakými metrikami, aby bolo možné ich jednoduché a jasné porovnanie.

Kedže všetky navrhnuté neurónové siete spracovávali páry dát, boli architektonicky navrhnuté ako siamské siete. Tento typ sietí je opísaný v stati 2.3.4. Modely trénované na pároch samostatných obrazov boli založené na viacvrstvových konvolučných sieťach, za ktorými nasledovalo niekoľko plne prepojených vrstiev. Počet konkrétnych vrstiev bol predmetom zmien medzi jednotlivými trénovanými modelmi. Medzi ďalšie zmeny patril počet a veľkosť filtrov v konvolučných vrstvách a počet neurónov v plne prepojených vrstvách. V rámci tréningovej procedúry boli menené hodnoty learning-rate a veľkosť vstupnej dávky.

Medzi jednotlivými modelmi trénovanými na pároch video-sekvencií boli vykonávané zmeny nielen na úrovni hyper-parametrov siete, ale taktiež vrámci samotnej architektúry, tréningového cieľa i druhu použitých vrstiev. Avšak základnou koncepciou všetkých modelov v tejto kategórií bolo využitie rekurentnej vrstvy na extrahovanie príznakov, ktoré zdôrazňujú unikátnе pohybové črty osôb zachytených na sekvenciách. Zmeny boli vykonávané na rovnakých hyper-parametroch, aké boli uvedené pri modeloch trénovaných na samostatných obrazoch.

3.4.1 Modely sietí pre re-identifikáciu osôb na základe obrazov

Spoločným znakom pre všetky modely trénované na samostatných obrazoch bola konštrukcia konvolučných, plne prepojených, i max-pooling vrstiev. Váhy konvolučných vrstiev sú definované ako štvorozmerné tenzory v tvare $[x, y, in, out]$, kde x a y sú rozmery použitých konvolučných filtrov, in je počet kanálov vstupných obrazov a out je počet kanálov výstupných obrazov (počet výstupných príznakových máp). V modeloch, ktoré boli natrénované, boli použité veľkosti konvolučných filtrov 3×3 , 5×5 , 7×7 a 11×11 . Veľkosť filtrov sa líšila iba medzi jednotlivými

modelmi, vrámci jedného modelu bol použitý vždy iba jeden rozmer filtra. Počet príznakových máp (kanálov) medzi jednotlivými konvolučnými vrstvami vrámci modelu graduálne narastal. Okrem váh bol vrámci konvolučnej vrstvy zako-
novaný aj tenzor prahových hodnôt (*ang.* bias), ktorý mal tvar [*out*]. To znamená jednu prahovú hodnotu pre každú výstupnú príznakovú mapu. Pred tréningom bol tenzor váh inicializovaný hodnotami z normálneho pravdepodobnostného rozde-
lenia so strednou hodnotou $\mu = 0$ a štandardnou odchýlkou $\sigma^2 = 1.0$. Prahové hodnoty boli inicializované hodnotou 0. Výstup celej konvolučnej vrstvy je defi-
novaný ako súčet výstupu operácie konvolúcie a prahových hodnôt. Tento výstup bol aktivovaný nelineárной aktivačnou funkciou *rectified linear*.

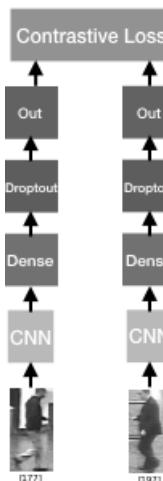
Max-pooling vrstva bola aplikovaná na výstupy každej konvolučnej vrstvy. Použitá veľkosť pooling okienka bola 2. Vzhľadom na malé rozlíšenie vstupných obrazov (128×64 pixelov) by väčšie rozmery okienka mohli znížiť kvalitu prí-
znakov nájdených konvolučnými vrstvami. Vzhľadom na rozmer pooling okienka sa rozlíšenie príznakových máp po aplikácii max-pooling operácie zmenší na po-
lovicu.

Plne prepojené vrstvy sa tak tiež skladali z tenzorov váh a prahových hod-
nôt. Tenzor váh mal tvar [*in*, *out*], kde *in* je veľkosť vstupného jednorozmerného tenzora a *out* je počet neurónov na danej plne prepojenej vrstve. Váhy boli inicia-
lizované z orezaného normálneho rozdelenia (*ang.* truncated normal distribution) so štandardnou odchýlkou $\sigma^2 = 0.01$. Pre hodnoty x z orezaného normálneho rozdelenia platí invariant: $\mu - 2\sigma^2 < x < \mu + 2\sigma^2$. Tenzor prahových hodnôt je jednorozmerný o veľkosti *out* a bol inicializovaný hodnotou 0. Výstup plne prepo-
jenej vrstvy bol definovaný ako súčet prahových hodnôt a maticového súčinu váh a vstupného tenzora. Vstupný tenzor do plne prepojenej vrstvy má tvar [*n*, *x*], kde *n* je veľkosť dŕavy (počet vstupných párov obrazov) a *x* je výstup predchádzajúcej vrstvy, premenený do tvaru jednorozmerného tenzora.

Vzhľadom na obmedzenú veľkosť súboru dát na trénovanie boli modely ná-
chylné k pre-trénovaniu. Z tohto dôvodu sú architektúry trénovaných modelov relatívne plytké. Príliš veľký počet konvolučných vrstiev, filtrov v konvolučných vrstvách a príliš široké plne prepojené vrstvy by súčasne poskytli vyšší počet parametrov na učenie, avšak súčasne by sa jednotlivé tréningové páry naučila ako pra-
vidlá a nedokázala by na ich základe pre neznáme testovacie páry generalizovať

svoju inferenciu. Okrem obmedzenia počtu parametrov modelu bola na zamedzenie pre-trénovania využitá dropout vrstva umiestnená za prvou plne prepojenou vrstvou. Jej úlohou je deaktivácia neurónov predchádzajúcej vrstvy. Pravdepodobnosť deaktivácie každého neurónu bola stanovená empiricky na $p = 0.4$. Dropout vrstva je aktívna iba počas tréningu. Počas fázy testovania k deaktivácií neurónov nedochádza.

Postupnosť vrstiev, tak ako je opísaná vyšie, predstavuje jednu vetvu siamskej siete. Obe vetvy sú identické a váhy medzi nimi sú zdielané. Obrazy zo vstupných párov z kamery č.1 sú vstupom pre jednu vetvu a obrazy z kamery č.2 sú vstupom pre druhú vetvu. Výstupom siamskej siete sú teda dva príznakové vektory, ktoré sú výstupom z poslednej plne prepojenej vrstvy v oboch vetvach. Na tieto príznakové vektory je následne aplikovaná contrastive loss funkcia. Ukážku opísanej konvolučnej siamskej siete je možné vidieť na obrázku č. 5.

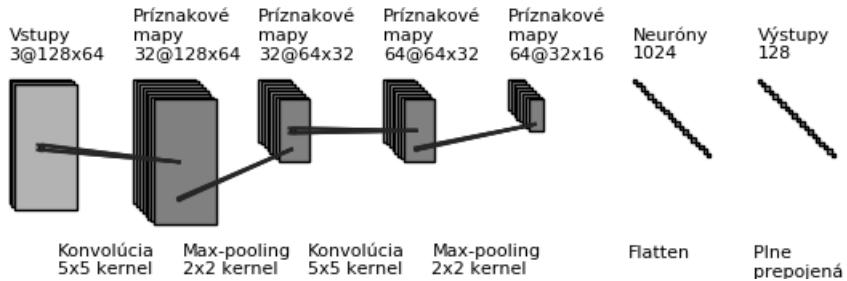


Obr. 5: Základná siamská konvolučná sieť

Na obrázku č. 6 možno vidieť detailný rozbor príkladu konvolučnej siete aj s vyznačenými počtami a veľkosťami príznakových máp vystupujúcich z daných vrstiev ako i s veľkosťami konvolučných filtrov a pooling okienok.

3.4.2 Modely pre re-identifikáciu osôb na základe video-sekvencií

Všetky natrénované modely určujúce podobnosť video-sekvencií využívajú siamskú architektúru, podobne ako modely spracúvajúce samostatné obrazy. S týmito



Obr. 6: Detail konvolučnej siete

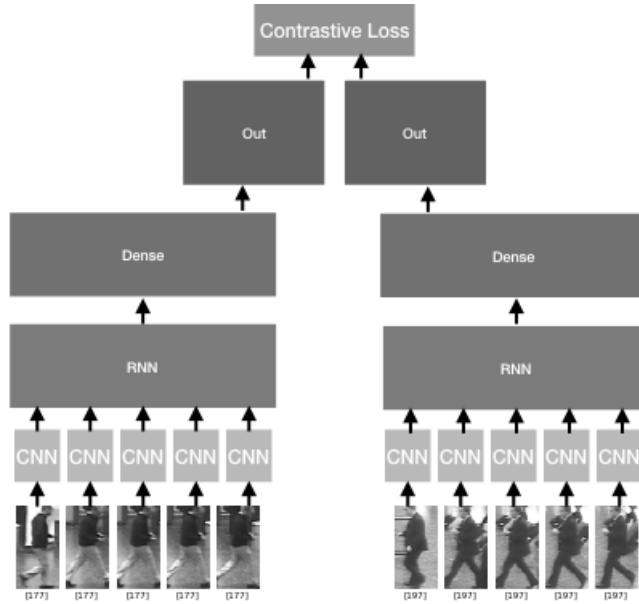
jednoduchšími modelmi majú taktiež spoločné prvé konvolučné a max-pooling vrstvy. V po nich nasledujúcich vrstvách sa topológia mierne líši medzi jednotlivými modelmi, čo je následkom ich iteratívneho zlepšovania.

Modely s rekurentnou vrstvou a výstupnou plne prepojenou vrstvou

Vrámcí jedenej rekurentnej vrstvy bolo použitých 128 až 1024 LSMT buniek (v závislosti na konkrétnom modeli). Rekurentná sieť je v modeli zadefinovaná tak, že počet buniek vo vrstve ako aj počet vrstiev je ľahko možné meniť ako hyperparameter. Vstupom do rekurentnej vrstvy je výstup z poslednej max-pooling vrstvy upravený na trojrozmerný tenzor $[b, s, x]$, kde b je veľkosť vstupnej dávky (počet párov video-sekvencií) a s je počet obrazov v sekvencii. Hodnota x je určená implicitne tak, aby sa zachoval pôvodný počet prvkov tenzoru. Výstup z rekurentnej vrstvy je ukladaný pre všetky časové úseky sekvencie (pre každý obraz v sekvencií). Tvar výstupného tenzora je $[b, s, c]$, kde c je počet LSTM buniek v poslednej rekurentnej vrstve. Tvar tohto tenzoru bol následne zmenený na $[b, s * c]$ a v tomto tvare vstupoval do nasledujúcej plne prepojenej vrstvy. Za ňou nasledovala drop-out vrstva a výstupná vrstva. Tvar výstupných príznakových vektorov reprezentujúcich dané video-sekvencie je rovnaký ako v prípade modelov založených na samostatných obrazoch. Pri tréningu je aplikovaná contrastive loss funkcia. Tento model je ilustrovaný na obrázku č. 7.

Modely s rekurentnou vrstvou a výstupnou ATPL vrstvou

Modely tohto druhu sú rovnaké ako tie opísané v odseku vyššie, až po bod výstupu z rekurentnej vrstvy. Výstup z týchto vrstiev je spracovaný v zmysle ATPL (ang.



Obr. 7: Topológia modelu s rekurentnou a plne prepojenou vrstvou

Attentive Temporal Pooling) vrstvy opísanej v stati 2.4.2. Táto vrstva rozširuje sieť o maticu trénovateľných parametrov, ktoré umožňujú sieti zamerať sa na unikátne a výrazné časové (pohybové) príznaky a odfiltrovať príznaky, ktoré sú redundantné. Tvar výstupných tenzorov, ktoré sú vstupom pre loss funkciu je $[b, c]$.

Modely s rekurentnou vrstvou LSTM buniek, SPP vrstvou a výstupnou ATPL vrstvou

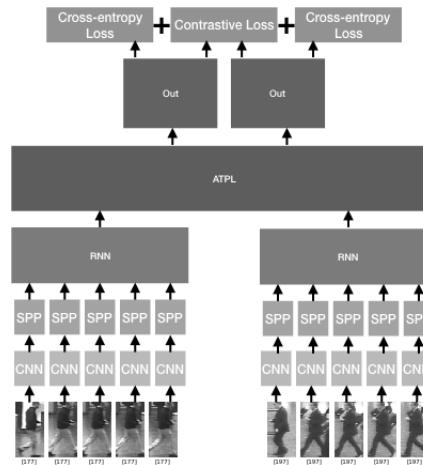
Tento druh modelu rozširuje model s ATPL vrstvou. Za poslednou konvolučnou vrstvou nasleduje v tomto type modelu SPP vrstva (*ang.* spatial pyramid pooling), tak ako je definovaná v stati 2.4.2. Množina pooling faktorov je: $j = \{8, 4, 2, 1\}$. Veľkosť príznakových máp po troch konvolučných a dvoch max-pooling vrstvách je 32×16 pixlov. Po dosadení do vzťahov definujúcich SPP vrstvu sú výsledné veľkosti pooling okienok: $\{[4, 2]; [8, 4]; [16, 8]; [32, 16]\}$.

Modely s rekurentnou vrstvou, SPP vrstvou a výstupnou ATPL vrstvou a cross-entropy loss funkciou

Jedná sa o poslednú úpravu modelu opísaného v prechádzajúcom odseku. Celá topológia siete zostáva nezmenená. Predmetom zmeny je iba tréningový cieľ neurónovej siete. Loss funkcia $L(x_1, x_2)$ bola zadefinovaná nasledovne:

$$L(x_1, x_2) = C_L(x_1, x_2) + C_E(x_1) + C_E(x_2)$$

kde C_L je contrastive loss funkcia a C_E je softmax cross-entropy loss funkcia (definovaná v stati 2.3.6), ktorá bola aplikovaná samostatne na oba výstupné príznakové vektory siamskej siete. Výstupné príznakové vektory majú v tomto prípade tvar $[p]$, kde p predstavuje počet tried softmax klasifikácie, v našom prípade počet osôb v súbore dát. Označenie správnej klasifikačnej triedy, ktorú daný vektor reprezentuje, bol vektor taktiež tvaru $[p]$, v takzvanom *one-hot* kódovaní. Vektor v tomto kódovaní je zložený zo samých núl a jednotka je iba na mieste, ktoré predstavuje číslo označenej klasifikačnej triedy. Pri využití cross-entropy loss funkcie musí byť veľkosť skrytého stavu rekurentnej vrstvy rovná počtu klasifikovaných osôb preto, aby výstupné príznakové vektory z ATPL vrstvy mali rovnaký tvar ako vektory označenia osôb v *one-hot* kódovaní. Finálna podoba tohto modelu je vyobrazená na obrázku č. 8.



Obr. 8: Topológia modelu s SPP a ATPL vrstvou

3.5 Metriky vyhodnocovania modelov

Po tréovaní sa môžeme na model pozerať ako na binárny klasifikátor. Ak si zvolíme určitú hraničnú vzdialenosť t , tak potom na základe reálnej vzdialosti $D(x_1, x_2)$ príznakových vektorov x_1 a x_2 a ich označenia podobnosti $y \in \{0, 1\}$ vieme určiť početnosť správne identifikovaných pozitívnych (TP - ang. true positive) a negatívnych zhôd (TN - ang. true negative) ako i početnosť chýb prvého (FP - ang. false positive - chybne identifikovaná pozitívna zhoda) a druhého druhu (FN - ang. false negative - chybne identifikovaná negatívna zhoda). Stavy binárnej klasifikácie páru sú vyobrazené v tabuľke č. 1.

Tabuľka 1: Stavy binárnej klasifikácie

	$y = 1$	$y = 0$	Σ
$D(x_1, x_2) < t$	TP	FP	PCP
$D(x_1, x_2) > t$	FN	TN	PCN
Σ	CP	CN	Total

V nej sú vyobrazené aj vzťahy pre súhrnné početnosti párov podľa ich skutočnej a predpovedanej zhody. Početnosť CP (ang. condition positive) je počet všetkých párov, ktoré sú označené značkou y ako zhodné. Obdobne je CN (ang. condition negative) početnosť párov označených ako nezhodné. Pre početnosti PCP (ang. predicted condition positive) a PCN (ang. predicted condition negative) je logika označovania podobná, avšak namiesto y je založená na modelom vypočítanej (predikovanej) vzdialenosťi páru $D(x_1, x_2)$ a hraničnej vzdialenosťi t .

ROC krivka

Prvou metrikou, ktorou sme natrénované modeli hodnotili, je ROC (ang. receiver operating characteristic) krivka. Body tejto krivky sú určené ako dvojice $[FPR(t), TPR(t)]$, kde $TPR(t)$ a $FPR(t)$ sú nasledujúce pomery početností:

$$TPR(t) = \frac{TP(t)}{CP(t)}$$

$$FPR(t) = \frac{FP(t)}{CN(t)}$$

rátané pre danú hraničnú vzdialenosť t . Počet bodov krivky je voliteľný a závisí od počtu n zvolených hraničných vzdialenosťí t , pre ktoré vyrátame klasifikačné početnosti. V navrhnutých vyhodnocovacích programoch bolo zvolené $n = 2000$. Samotné hraničné vzdialosti boli rovnomerne rozptýlené od minimálnej po maximálnu vzdialenosť spomedzi všetkých vypočítaných vzdialostí párov. Testovacia vzorka pozostávala z 8000 párov v prípade modelov pracujúcich so samostatnými obrazmi a 1000 párov v prípade modelov založených na video-sekvenciach.

AUC

Táto metrika úzko súvisí s ROC krivkou. Jedná sa o plochu pod touto krivkou (*ang. area under curve*). Keďže pomery TPR a FPR sa pohybujú v intervale $<0, 1>$ je aj hodnota AUC z tohto intervalu. Čím lepšia je klasifikačná schopnosť modelu tým je hodnota AUC bližšia k 1, teda FPR - chyba 1. druhu - je blízka 0. Hodna AUC vyjadruje pravdepodobnosť, s ktorou správne ohodnotí model zhodný párs.

F1 skóre

Metrika F1 skóre vyjadruje harmonický priemer dvoch pomerných veličín: precíznosti (*ang. precision*) a senzitívity (*ang. recall*).

$$precision = \frac{TP}{PCP}$$

$$recall = TPR = \frac{TP}{CP}$$

Samotné F1 skóre vieme potom zadefinovať ako:

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{2TP}{2TP + FP + FN}$$

F1 skóre, podobne ako AUC, vyjadruje celkovú klasifikačnú schopnosť modelu. Obor hodnôt je taktiež rovnaký ako pri AUC, tj. $<0, 1>$. Vo vytvorených vyhodnocovacích moduloch je F1 skóre počítané pre každú hraničnú vzdialenosť t a ako výsledné skóre je uvažované maximum z týchto hodnôt. Hraničná vzdiale-

nosť t , pre ktorú bolo F1 skóre maximálne, je tá hraničná vzdialenosť, ktorá by bola použitá pre inferenciu podobnosti neznámeho vstupného páru v prípade nasadenia modelu v praxi.

N rank (CMC)

Označme P množinu všetkých osôb v súbore dát a osobu $p \in P$ ako šablónu (*ang. template*) a vytvorime usporiadanú množinu párov $M = \{(p, x); x \in P\}$, ktorej prvky usporiadame podľa vzdialosti $D(p, x)$ zostupne. Ak sa pári $N(p, p) \in M$ nachádza vrámci prvých n prvokov usporiadanej množiny M , tak môžeme zaznačiť, že prispieva k ranku n . Ak postup zopakujeme so všetkými osobami $p \in P$, tak pomer počtu párov j , ktoré prispievajú k ranku n a počtu osôb v množine P nazývame n -rank. Napríklad 50% top 1-rank, znamená, že pre polovicu osôb zo súboru dát je modelom určený najpodobnejší pári ten, ktorý je aj v súbore dát označený ako podobný. Tento postup sa v literatúre nazýva kumulatívna párovacia charakteristika (*ang. Cumulative Matching Characteristic - CMC*).

Touto metrikou sú porovnávané všetky publikácie pracujúce nad datasetom i-LIDS-VID.

Chybová matica

Chybová matica, resp. matica zámen (*ang. confusion matrix*) je prostriedok na vizualizáciu výkonnosti a chybovosti klasifikačného systému v podobe matice (tabuľky). Riadky i stĺpce sú označené podľa tried, do ktorých klasifikátor klasifikuje. V prípade datasetu i-LIDS-VID sa jedná o maticu 300×300 (počet osôb). Matica sa vypĺňa počas výpočtu n -ranku. Do matice sa zanesie jednotka na miesto $[p, q]$, ktoré predstavuje pári s najmenšou vzdialenosťou $N(p, q)$ spomedzi usporiadanej množiny párov M , tak ako bola definovaná v odseku vyššie. Ak $p = q$ tak podobnosť páru bola určená správne. Prípady kedy $p \neq q$ sú zaujímané z hľadiska určenia vlastností párov, ktoré sú pre model mätúce (ich podobnosť bola učená nesprávne).

3.6 Technické prostriedky

Všetky moduly (príprava dát, trénovanie, modely sietí i vyhodnocovanie) boli napísané v jazyku Python 3.6 s použitím knižníc *Tensorflow* vo verzií 1.4.0, *Numpy*

a *Scipy*. Knižnica Tensorflow bola inštalovaná tak, aby podporovala paralelizáciu výpočtov na GPU, čoho predpokladom bolo nainštalovanie knižníc *CUDA 8.0* a *cuDNN*. Všetky knižnice boli nainštalované v izolovanom virtuálnom Python prostredí *Anaconda*. Ako hlavné vývojové prostredie bolo využívané prostredie PyCharm.

Tréning prebiehal na počítači umiestnenom na Fakulte informatiky a informačných technológií STU, ku ktorému bol umožnený vzdialený prístup cez protokol *ssh*. Počítač disponoval grafickou kartou *nVidia GTX 1080 ti* s 11GB GDDR5 pamäte. Na verziovanie a zálohu zdrojových kódov bol použití *git*.

4 Výsledky

Výsledky práce budú prezentované štruktúrovane podľa druhov modelov neurónových sietí tak, ako boli opísané v návrhu riešenia v stati 3.4. V rámci každého druhu trénovanej neurónovej siete budú uvedené výsledky metrík pre najzaujíma-vejšie konfigurácie hyper-parametrov, s ktorými bol daný druh modelu trénovaný. Následne budú výsledky rôznych modelov porovnané medzi sebou a na záver budú porovnané výsledky najúspešnejších modelov s výsledkami iných publikácií pracujúcich nad datasetom i-LIDS-VID.

Pre zjednodušenie označovania jednotlivých modelov v texte a nadpisoch sekcií bola zavedená notácia jednotlivých vrstiev modelov vo formáte:

$<typ><velkost>$

kde, *typ* je typ vrstvy (napr. *c* - konvolučná, *d* - plne prepojená, *r/rnn* - rekurentná, *spp* - spatial pyramid pooling, *atpl* - attentive temporal pooling layer, a pod.) a *velkost* je počet filtrov konvolučných vrstiev resp. počet neurónov plne prepojených, resp. počet buniek rekurentných vrstiev. Jednotlivé vrstvy sú v notácii spájané znakom : - a ich poradie zľava doprava je ich poradie v modeli od vstupu k výstupu. Okrem tejto vägnej notácie bude pri každom vyhodnotenom modeli i tabuľka ktorá jeho topológiu opisuje obširnejšie.

4.1 Výsledky modelov sietí pre re-identifikáciu osôb na základe obrazov

V tejto časti sú uvedené vybrané najzaujímavejšie výsledky modelov sietí, ktorých topolózia je opísaná v stati 3.4.1. Jedná sa o modely siamských konvolučných neurónových sietí pre re-identifikáciu osôb na základe párov samostatných obrazov. Medzi menené hyper-parametre patrili najmä počet filtrov v konvolučných vrstvách a počet neurónov v skrytej a výstupnej plne prepojenej vrstve.

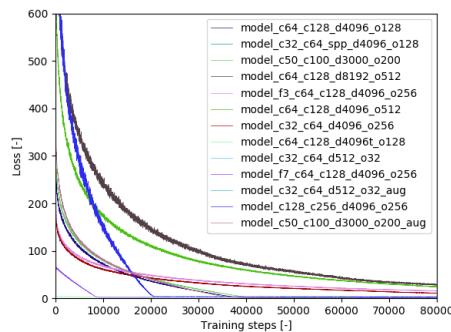
Konfigurácie hyper-parametrov sú uvádzané v tabuľkách. Typy vrstiev sú uvádzané skratkami. Konvolučná vrstva je definovaná skratkou: napr. *c + t + p*, kde *c* predstavuje operáciu konvolúcie, *t* je aktivačná funkcia (v tomto príklade *tanh*)

a p je vrstva max-pooling. Plne prepojená vrstva je označená skratkou d (*angl. dense*).

4.1.1 Tréning modelov

Všetky nižšie uvedené siamske konvolučné modely boli trénované na čo najväčšej vstupnej tréningovej dávke párov obrazov. Veľkosť tejto dávky bola limitovaná dostupnou pamäťou grafického procesoru, na ktorom tréningový proces prebiehal. Technické podrobnosti o systéme sú uvedené v sekcií 3.6. Veľký počet párov v tréningových dávkach urýchlił konvergenciu loss funkcie pri tréningu v porovnaní s prístupom malých vstupných dávok (*ang. mini batches*). Konkrétna veľkosť vstupných dávok sa líšila v závislosti od zložitosti modelu, tj. od počtu konvolučných filtrov a neurónov v skrytej a výstupnej vrstve, keďže okrem dávky dát na spracovanie sa do pamäte grafického procesora ukladali aj samotné váhy siete. Pri komplexnejších modeloch bola veľkosť tréningovej dávky menšia (cca.200 párov) než pri jednoduchších modeloch (300 a viac párov).

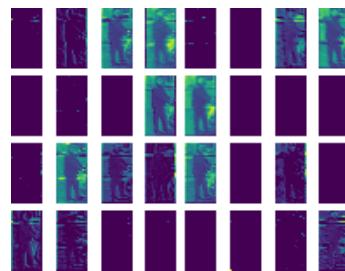
Na minimalizáciu loss funkcie bol pri každom z učených konvolučných modelov použitý *Gradient Descent* optimizér s hodnotou rýchlosťi učenia (*ang. learning rate*) $l_r = 0.00001$. Rýchlosť učenia bola stanovená empiricky jej postupným expoenciálnym znižovaním, až pokial nezačala hodnota loss-funkcie od prvých iterácií tréningového cyklu klesať. Graf vývoja hodnôt loss funkcie v závislosti od počtu tréningových iterácií pre viacero trénovaných modelov je zobrazený na obrázku č. 9.



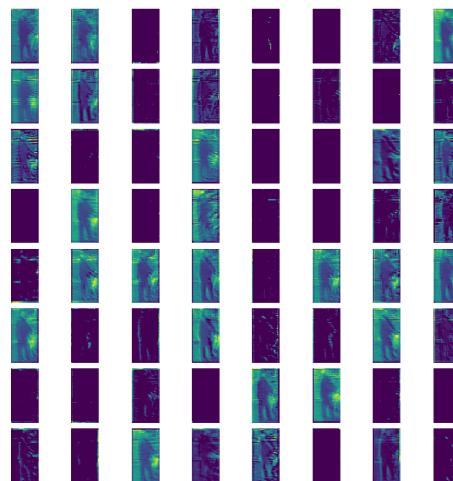
Obr. 9: Vývoj hodnôt loss funkcie v závislosti od počtu tréningových iterácií

4.1.2 Vizualizácia aktivácií konvolučných vrstiev

Na dodatočný prieskum natrénovaných modelov neurónových sietí boli taktiež získané vizualizácie aktivácií konvolučných vrstiev pri spracovaní konkrétnych vstupných obrazov. Tieto vizualizácie poskytujú informáciu o tom, ktoré výrazné príznaky obrazov sa konvolučné filtre naučili zachytávať. Vizualizácie sú vo forme jedno-kanálových obrazov jednotlivých príznakových máp, ktoré sú výstupom konvolučných vrstiev. Na obrázkoch č. 10 a 11 je ukážka aktivácií prvej, resp. druhej konvolučnej vrstvy modelu *c32_c64_d512_o32*. Ako je z ukážok zrejmé, tak konvolučné vrstvy tohto modelu sa zmerali najmä na obrysy postáv alebo na iné miesta obrazov s výrazným gradientom intenzity pixlov. Tento trend bol platný aj pre ostatné modely.



Obr. 10: Príznakové mapy prvej konvolučnej vrstvy modelu *c32_c64_d512_o32*



Obr. 11: Príznakové mapy druhej konvolučnej vrstvy modelu *c32_c64_d512_o32*

4.1.3 Model c32_c64_d512_o32

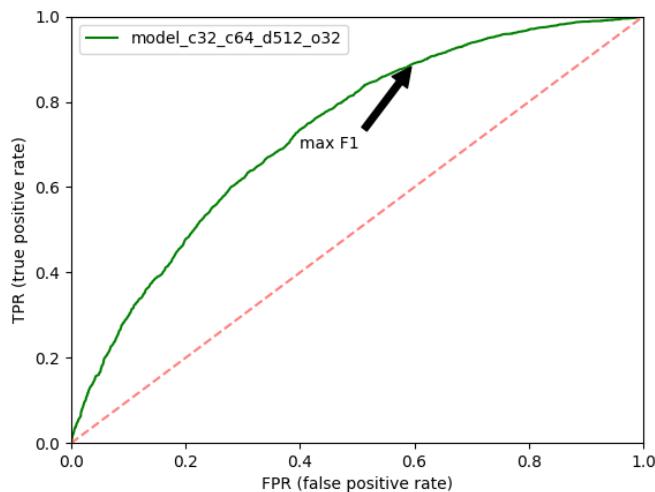
Tento model dosiahol pri relatívne malej veľkosti siete priemerné výsledky, ktoré budú pri hodnotení zvyšných modelov brané ako východzie. Zmeny výsledkov metrík oproti výsledkom tohto modelu budú hodnotené vzhľadom na zmeny konkrétnych hyper-parametrov a vplyv týchto zmien bude analyzovaný.

Tabuľka 2: Konfigurácia hyper-parametrov vrstiev modelu c32_c64_d512_o32

Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov
Conv1	$c + t + p$	32, 5×5 , 1, 2×2
Conv2	$c + t + p$	64, 5×5 , 1, 2×2
Hidden1	d	512
Out	d	32

Tabuľka 3: Výsledky metrík modelu c32_c64_d512_o32

Metrika	Hodnota
Accuracy	0.6678
F1 skóre	0.7152
AUC	0.7265
Hraničná vzdialenosť pri max. F1 skóre	4.8012



Obr. 12: ROC krivka pre model c32_c64_d512_o32

4.1.4 Model c64_c128_d512_o32

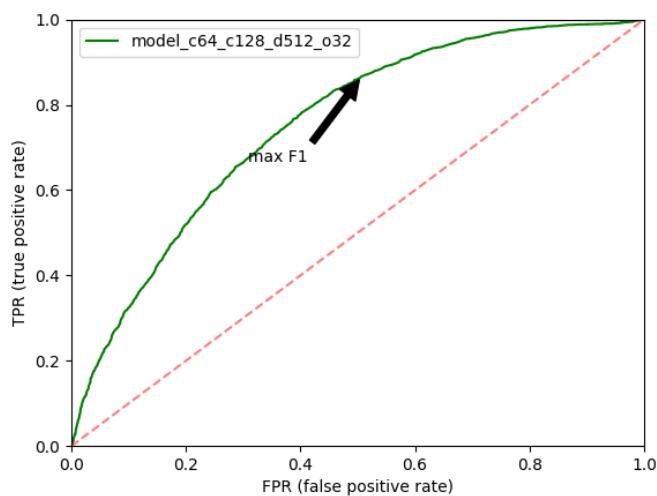
Pri tomto modeli bol dvojnásobne zvýšený počet filtrov v oboch konvolučných vrstvách oproti modelu opísanom v sekcii 4.1.3. Skrytá a výstupná vrstva zostali nezmenené. F1 skóre sa zvýšilo o 2.1% a AUC sa zvýšila o 3.62%.

Tabuľka 4: Konfigurácia hyper-parametrov vrstiev modelu c64_c128_d512_o32

Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov
Conv1	$c + t + p$	64, 5×5 , 1, 2×2
Conv2	$c + t + p$	128, 5×5 , 1, 2×2
Hidden1	d	512
Out	d	32

Tabuľka 5: Výsledky metrík modelu c64_c128_d512_o32

Metrika	Hodnota
Accuracy	0.6900
F1 skóre	0.7303
AUC	0.7528
Hraničná vzdialenosť pri max. F1 skóre	4.3882



Obr. 13: ROC krivka pre model c64_c128_d512_o32

4.1.5 Modely c64_c128_dx_ox

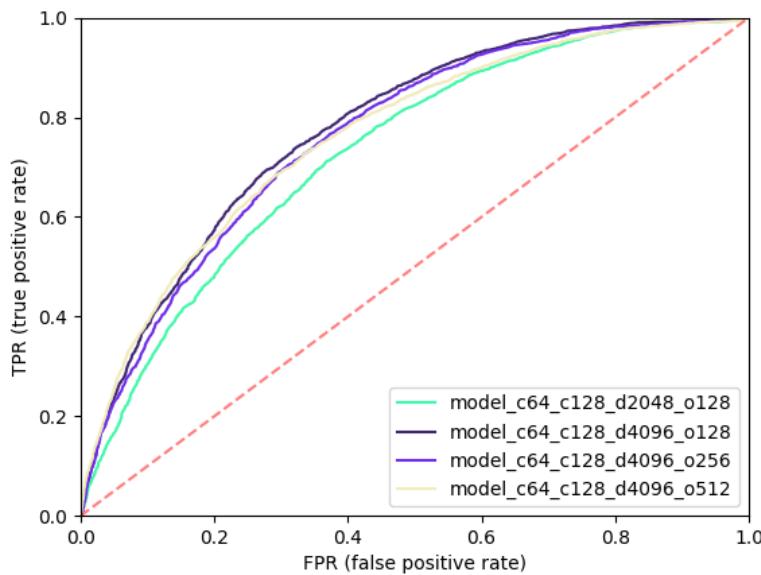
Počet konvolučných filtrov zostal oproti modelu opísanom v sekcii 4.1.4 nezmenený. Zmeny nastali v počte neurónov v skrytej a výstupnej vrstve. Keďže výsledky týchto modelov boli veľmi podobné, tak budú nižšie uvedené agregované v jednej tabuľke a jednom grafe ROC.

Tabuľka 6: Konfigurácia hyper-parametrov vrstiev modelov c64_c128_dx_ox

Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov
Conv1	$c + t + p$	64, 5×5 , 1, 2×2
Conv2	$c + t + p$	128, 5×5 , 1, 2×2
Hidden1	d	2048 resp. 4096
Out	d	128, 256 resp. 512

Tabuľka 7: Výsledky metrik modelov c64_c128_dx_ox

Metrika	d2048_o128	d4096_o128	d4096_o256	d4096_o512
Accuracy	0.67175	0.7085	0.6967	0.6962
F1 skóre	0.7176	0.7392	0.7348	0.7235
AUC	0.7306	0.7775	0.7642	0.7641



Obr. 14: ROC krivka pre modely c64_c128_dx_ox

Model c64_c128_d4096_o128, ako najúspešnejší podľa všetkých vyčíslených metrík, bol podrobnený n -rank testovaniu, ktorého výsledky sú uvedené v tabuľke č. 17. Oproti referenčnému modelu v stati 4.1.3 dosahuje tento model o 7.02% vyššie AUC a o 3.35% vyššie F1 skóre.

Tabuľka 8: N ranking pre model c64_c128_d4096_o128

Rank	%
1	46%
5	66%
10	74%
20	81%

V porovnaní s výsledkami rank-ovania riešení navrhnutými v publikáciach pracujúcich nad rovnakým súborom dát (i-LIDS-VID) uvedených v zozname na stránke ³, kde najnižší výsledok pre metriku rank 1 je 25.9% a na najvyšší 79.5% a priemer pre všetky uvedené publikácie dosahuje 47.9%, sú výsledky tu uvedeného modelu prakticky priemerné. Avšak oproti dvom state-of-the-art publikáciám opísaným v sekcií 2.4, ktoré dosahujú výsledky rank 1 62% a 79.5% sú tieto výsledky podpriemerné.

4.1.6 Modely fx_c64_c128_d4096_o256

Topológia týchto modelov je inšpirovaná skôr natrénovanými modelmi pre re-identifikáciu osôb na základe obrazov, ktoré dosahovali najvyššie hodnoty metrík AUC a F1 skóre. Konkrétnie bola zvolená topológia modelu fx_c64_c128_d4096_o256. Pri týchto modeloch bola menená veľkosť konvolučných filtrov v oboch konvolučných vrstvách. Skúmané veľkosti filtrov boli: 3×3 , 5×5 (referenčná veľkosť), 7×7 a 11×11 .

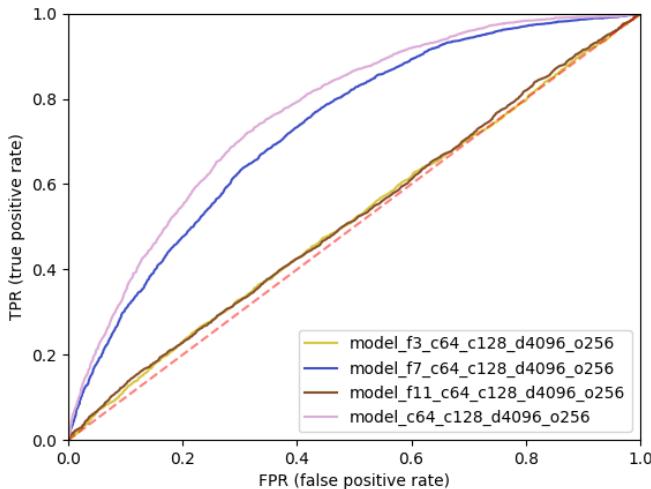
Tabuľka 9: Konfigurácia hyper-parametrov vrstiev modelov fx_c64_c128_d4096_o256

Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov
Conv1	$c + t + p$	$64, 5 \times 5, 3 \times 3, 7 \times 7, 11 \times 11, 1, 2 \times 2$
Conv2	$c + t + p$	$128, 5 \times 5, 3 \times 3, 7 \times 7, 11 \times 11, 1, 2 \times 2$
Hidden1	d	4096
Out	d	256

³http://www.eecs.qmul.ac.uk/~xiatian/downloads_qmul_iLIDS-VID_ReID_dataset.html

Tabuľka 10: Výsledky metrík modelov *fx_c64_c128_d4096_o256*

Metrika	3×3	5×5	7×7	11×11
Accuracy	0.5171	0.7036	0.6685	0.5171
F1 skóre	0.6666	0.7322	0.7190	0.6666
AUC	0.5150	0.7660	0.7289	0.5186



Obr. 15: ROC krivka pre modely *fx_c64_c128_d4096_o256*

Pri modeloch s filtrovami veľkosti 3×3 a 11×11 bol pozorovaný pokles metriky AUC až na úroveň blízku 0.5, čo zodpovedá náhodnej klasifikácií. Model sa pri tejto konfigurácii nedokázal naučiť rozpoznávať kľúčové príznaky na obrazu. Možná príčina je tá že filter 3×3 nedokázal zachytiť prvky celé a filter 11×11 zachytil až príliš veľké okolie týchto kľúčových bodov. Konfigurácia s filtrovami 5×5 a 7×7 dosahovala porovnatelné výsledky. Konfigurácia s filtrovami 5×5 však dosahovala o 5.1% vyššiu AUC a o 1.8% vyššie F1 skóre oproti konfigurácií s filtrovami 7×7 .

4.1.7 Model **c128_c256_d4096_o256**

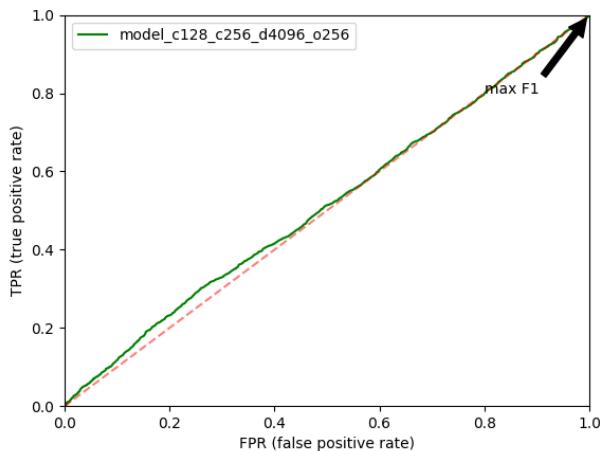
Cieľom tejto konfigurácie hyper-parametrov bolo ďalej zvýšiť počet parametrov siete tým, že zvýšime počet výstupných príznakových máp z oboch konvolučných vrstiev až na počet 128 pre prvú a 256 pre druhú konvolučnú vrstvu.

Tabuľka 11: Konfigurácia hyper-parametrov vrstiev modelu c128_c256_d4096_o256

Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov
Conv1	$c + t + p$	128, 5×5 , 1, 2×2
Conv2	$c + t + p$	256, 5×5 , 1, 2×2
Hidden1	d	4096
Out	d	256

Tabuľka 12: Výsledky metrík modelu c128_c256_d4096_o256

Metrika	Hodnoty
Accuracy	0.5209
F1 skóre	0.6666
AUC	0.5123



Obr. 16: ROC krivka pre model c128_c256_d4096_o256

Testovanie tohto modelu ukázalo, že vysoký počet parametrov mal za následok pre-trénovanie neurónovej siete. Tá na testovacích dátach ukázala neschopnosť generalizovať naučené informácie. Rovnako vykazovali pre-trénovanie i modely kde bol počet parametrov zvýšený pridaním viacerých neurónov do plne prepojených vrstiev. Príkladom takého modelu je: model_c64_c128_d8192_o512. Ďalšími príkladmi modelov s príliš vysokým počtom parametrov sú modely s tromi a viac konvolučnými a/alebo plne prepojenými vrstvami. Pre stručnosť nie sú v tejto sekcií uvedené všetky vychodnocované modely, ale iba tie s najlepšími výsledkami podľa

metrík a tie, kde zmena konkrétnych hyper-parametrov zapríčinila výraznú zmenu výsledkov. Ďalšie testované siete je možné nájsť na priloženom elektronickom médiu.

4.1.8 Testovanie klasifikácie

Na základe evaluácie modelov neurónových sietí pre re-identifikáciu osôb na základe samostatných obrazov prostredníctvom metrík F1 skóre a AUC sme identifikovali najvhodnejší model na testovanie jeho klasifikačných schopností. Ten bol vybraný na základe najvyšej hodnoty metriky AUC a jedná sa o model: *c64_c128_d4096_o128*, ktorý dosiahol $AUC = 0.7775$. Následne bola pre tento model identifikovaná hraničná vzdialenosť t , pri ktorej dosahoval tento model maximálne F1 skóre. Nájdená hraničná vzdialenosť bola $t = 4.6942$ pre hodnotu F1 skóre 0.7399, kde $TPR = 0.86425$ a $FPR = 0.469$.

Samotné testovanie klasifikácie prebiehalo na náhode generovaných pároch samostatných obrazov pričom počet pozitívnych a negatívnych párov bol rovnaký. Pomocou modelu neurónovej siete bola vypočítaná euklidovská vzdialenosť v n -rozmernom priestore medzi obrazmi páru. Táto vzdialenosť bola následne porovnávaná so zistenou optimálnou hraničnou vzdialenosťou t . Nižšie sú vyobrazené ukážky všetkých štyroch možných stavov binárnej klasifikácie podobnosti páru: správna identifikácia podobných i nepodobných párov ako aj chyby 1. a 2. druhu.



Obr. 17: Správna identifikácia zhodného (pozitívneho) páru



Obr. 18: Správna identifikácia nezhodného (negatívneho) páru



Obr. 19: Páry nesprávne označené ako nezhodné



Obr. 20: Páry nesprávne označené ako zhodné

Z pozorovania párov nesprávne označených ako nezhodných vyplýva, že hlavné faktory, ktoré vplyvajú na výskyt tohto druhu chyby, sú: veľmi odlišné natočenie alebo postoj osoby, odlišné svetelné podmienky, veľmi odlišné pozadie a oklúzia iným objektom alebo osobou.

Páry nesprávne označené ako zhodné boli najčastejšie tvorené osobami s veľmi podobnou farbou oblečenia, podobnou farbou pozadia, na ktorom boli zachytené, poprípade podobnými doplnkami (tašky, ruksaky).

Všetky klasifikované páry sú dostupné na elektronickom médiu v lokalite: /results/model_c64_c128_d4096_o128/classification.

4.2 Výsledky modelov sietí pre re-identifikáciu osôb na základe video-sekvencií

Táto podkapitola obsahuje výsledky modelov sietí, ktorých topológie sú opísané v rámci state 3.4.2. Obsiahnuté sú viaceré topológie konvolučno-rekurentných sietí. Ich výsledky budú prezentované podľa zložitosti topológií vzostupne.

Konfigurácie hyper-parametrov a typy vrstiev budú uvádzané opäť v tabuľkách. Typy vrstiev sú uvádzané skratkami. K skratkám zo sekcie 4.1 pribudli skratky označujúce rekurentné a špeciálne vrstvy:

- lstm - rekurentná vrstva zložená z LSTM buniek

- rnn - základná rekurentná vrstva
- spp - spatial pyramid pooling vrstva
- atpl - atentive temporal pooling vrstva

4.2.1 Modely c64_c128_rnn_d512_o128

Tento model reprezentuje prvotné rozšírenie konvolučných modelov na báze samostatných obrazov o rekurentnú vrstvu. Táto vrstva má za úlohu získať príznakovú reprezentáciu celej sekvencie obrazov, na základe príznakov jednotlivých obrazov sekvencie, ktoré extrahovala konvolučná vrstva. Za rekurentnou vrstvou nasleduje skrytá a výstupná vrstva.

Tabuľka 13: Konfigurácia hyper-parametrov vrstiev modelu c64_c128_rnn_d512_o128

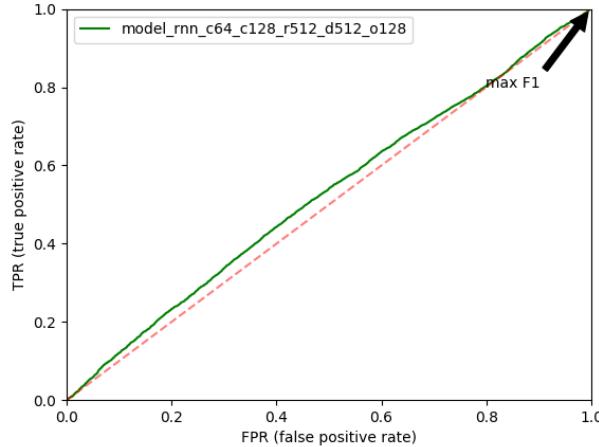
Vrstva	Typ	Počet, veľkosť filtra, posun filtrov a max-pool resp. počet neurónov (RNN buniek)
Conv1	$c + t + p$	64, 5×5 , 1, 2×2
Conv2	$c + t + p$	128, 5×5 , 1, 2×2
Rnn1	LSTM	512
Hidden1	d	512
Out	d	128

Tabuľka 14: Výsledky metrík modelu c64_c128_rnn_d512_o128

Metrika	Hodnoty
Accuracy	0.5224
F1 skóre	0.6669
AUC	0.5239

Výsledky tohto modelu nie sú uspokojivé (podobné náhodnej klasifikácií). Podobne neuspokojivé výsledky dosahovali i všetky druhy modelov, ktoré nevyužívali upravený tréningový cieľ s cross-entropy loss funkciu. Výsledky týchto modelov tu pre stručnosť nie sú uvedené, avšak sú dostupné na elektronickom médiu.

Možné dôvody neuspokojivých výsledkov zahŕňajú pre-trénovanie alebo stratu priestorovo-časových príznakov (*ang. spatial-temporal features*) z rekurentnej vrstvy.



Obr. 21: ROC krivka pre model *c64_c128_rnn_d512_o128*

Táto strata mohla byť zapríčinená druhom výstupu rekurentnej vrstvy, ktorý obsahuje množstvo redundantných informácií o sekvencií, keďže sa využívajú výstupy rekurentnej vrstvy, pre každý časový krok spracovávanej sekvencie. Stratu vplyvu dôležitých príznakov mohla ďalej umocniť aplikácia skrytej plne prepojenej vrstvy, ktorá sa kvôli vyššiemu zmienenému množstvu redundantných informácií z rekurentnej vrstvy nedokázala efektívne učiť iba na základe dôležitých príznakov. Pre nasledujúce úpravy modelov vznikla preto potreba vyselektovať iba dôležité priestorovo-časové príznaky z rekurentnej vrstvy.

4.2.2 Modely `cx(cx(cx(ssp_rnn_atpl_ce`

Modely v tejto konfigurácii predstavujú najkomplexnejšie natrénované konvolučno-rekurentné modely vrámci tejto práce. Využívajú aj špeciálne SPP a ATPL vrstvy a tréningový cieľ rozšírený o minimalizáciu cross-entropy loss funkcie z oboch výstupných príznakových vektorov. ATPL vrstva pomáha sieti extrahovať dôležité príznaky z rekurentnej vrstvy.

Hyper-parametre podliehajúce zmene boli počty konvolučných filtrov v konvolučných vrstvách.

Tabuľka 15: Konfigurácia hyper-parametrov vrstiev modelov *cx_cx_cx_ssp_rnn_atpl_ce*

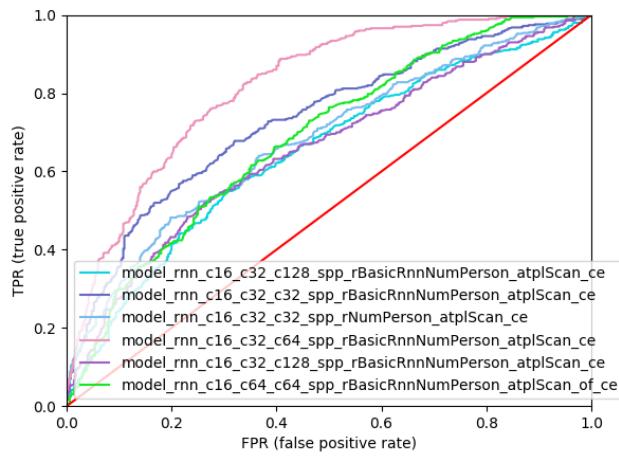
Vrstva	Typ	Počet, veľkosť filtrov, posun filtrov a max-pool resp. počet neurónov (RNN buniek)
<i>Conv1</i>	$c + t + p$	16, 5×5 , 1, 2×2
<i>Conv2</i>	$c + t + p$	32, 64, 5×5 , 1, 2×2
<i>Conv3</i>	$c + t$	32, 64, 128, 5×5 , 1, N/A
<i>SpatialPooling</i>	<i>spp</i>	N/A
<i>Rnn1</i>	<i>rnn</i>	300 (počet osôb - kvôli ATPL vrstve)
<i>TemporalPooling</i>	<i>atpl</i>	N/A

Tabuľka 16: Výsledky metrík modelu *cx_cx_cx_ssp_rnn_atpl_ce*

Metrika	<i>c16_c32_c128</i>	<i>c16_c32_c32</i>	<i>c16_c32_c32(LSTM)</i>	<i>c16_c32_c64</i>	<i>c16_c32_c128</i>	<i>c16_c64_c64</i>
Accuracy	0.621	0.679	0.642	0.743	0.628	0.634
F1 skóre	0.6703	0.6977	0.6784	0.7735	0.6729	0.6997
AUC	0.6478	0.7311	0.6790	0.8181	0.6554	0.6863

Tabuľka 17: N ranking pre model *c16_c32_c64_ssp_rnn_atpl_ce*

Rank	%
1	6%
5	28%
10	43%
20	56%



Obr. 22: ROC krivka pre modely *cx_cx_cx_ssp_rnn_atpl_ce*

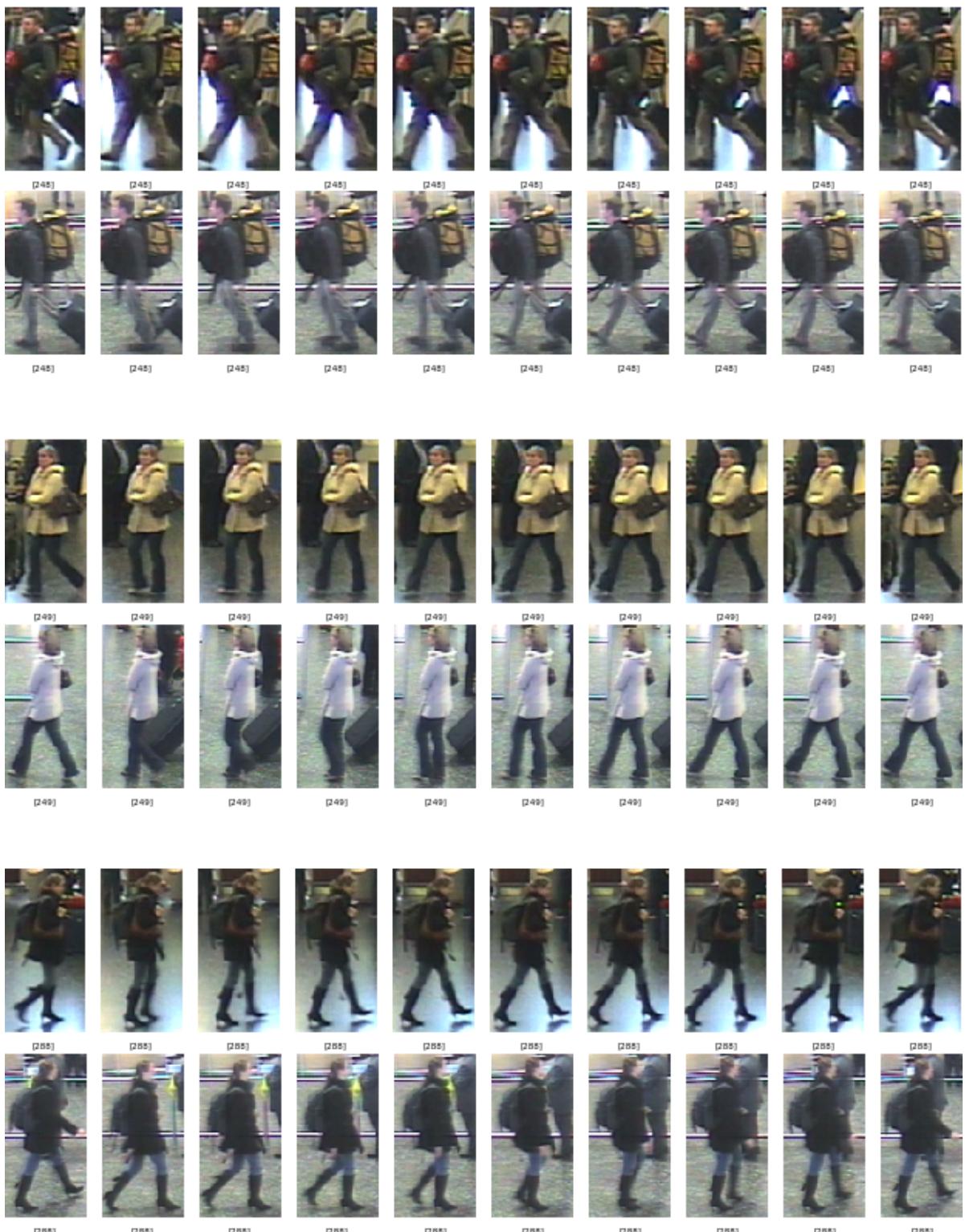
Najlepšie výsledky na základe metrík dosahoval model *c16_c32_c64_ssp_rnn_atpl_ce*.

Oproti konvolučno-rekurentnému modelu uvedenom v sekcií 4.2.1 dosahuje tento model o 56.15% vyššiu AUC a o 15.62% vyššie F1 skóre.

Napriek tomu, že tento model dosahoval pri testovaní jeho klasifikačných schopností najlepšie výsledky, tak hodnoty jeho *n*-rankovania (CMC) tomu ne-korešpondujú. Táto chyba je zapríčinená pravdepodobne spôsobom generovania párov video-sekvencií použitých pri výpočte CMC. Na základe hodnôt CMC je tento model kvalitatívne podpiemerný v porovnaní s ostatnými publikáciami, avšak pri klasifikačnom testovaní dosahoval výsledky porovnateľné s modelom c64_c128_d4096_o128, ktorý dosahuje pri CMC analýze priemerné výsledky v porovnaní s podobnými publikáciami.

4.2.3 Testovanie klasifikácie

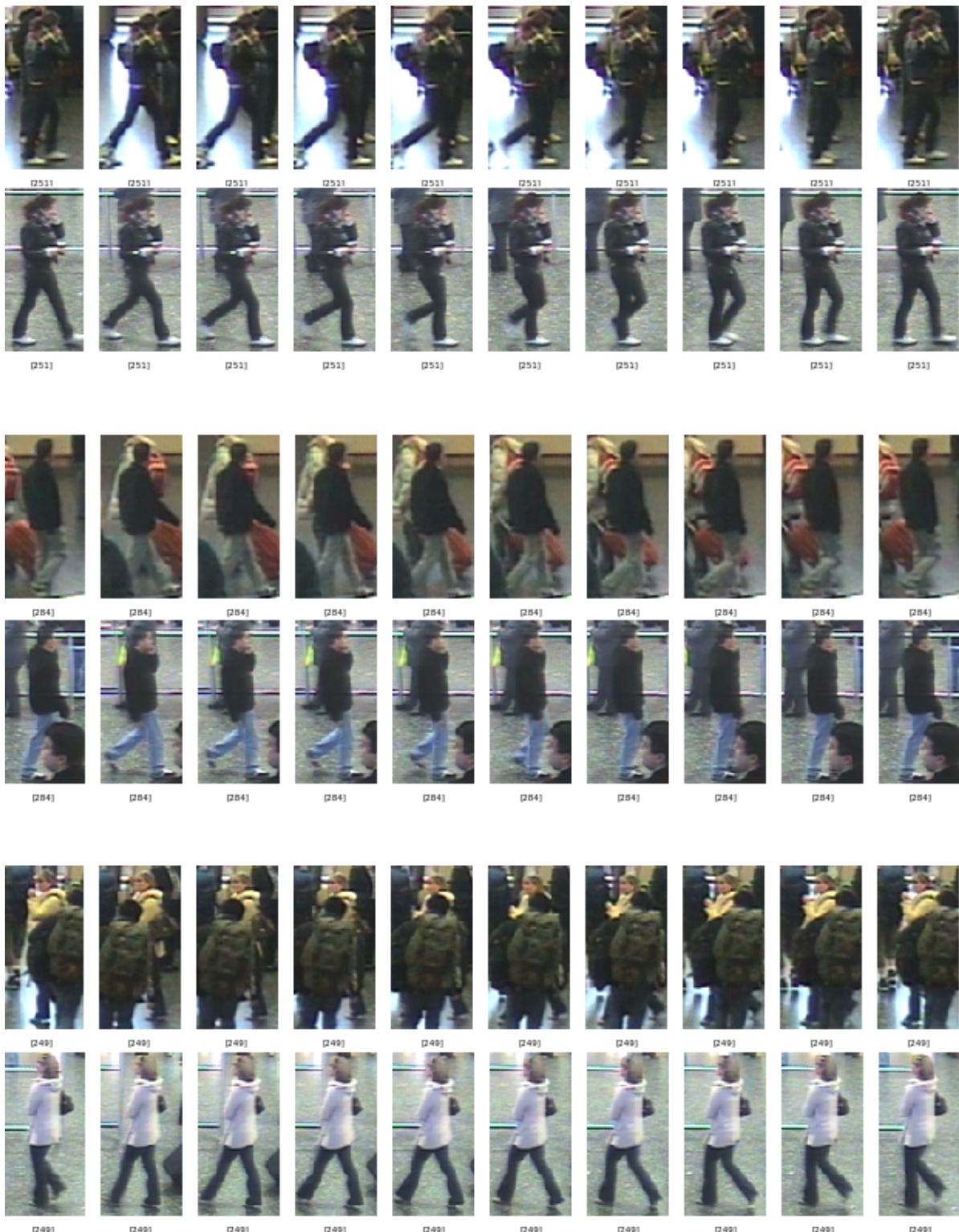
Rovnako ako pri modeloch neurónových sietí na báze samostatných obrazov, tak i pri modeloch pre re-identifikáciu osob na základe video-sekvencií bol zvolený jeden model, ktorý bol vyhodnocovaný ako klasifikátor. Kritéria pre jeho výber boli rovnaké ako pri výber rýdzo konvolučného modelu: F1 skóre a AUC. Konvolučno-rekurentným modelom s najvyššou hodnotou $AUC = 0.8181$ je model *c16_c32_c64_ssp_rnn_atpl_ce*. Tento model využíval obe špeciálne vrstvy (ATPL a SPP) a taktiež upravený tréningový cieľ s cross-entropy loss funkciou. Optimálna hraničná vzdialenosť pre klasifikáciu podobnosti párov video-sekvencií týmto modelom bola: $t = 4.4279$, kedy model dosahoval F1 skóre: 0.7778. Nižšie na obrázkoch 23, 24, 25, 26 sa nachádzajú ukážky správnej identifikácie pozitívnych a negatívnych párov, ako i chyba prvého a druhého druhu.



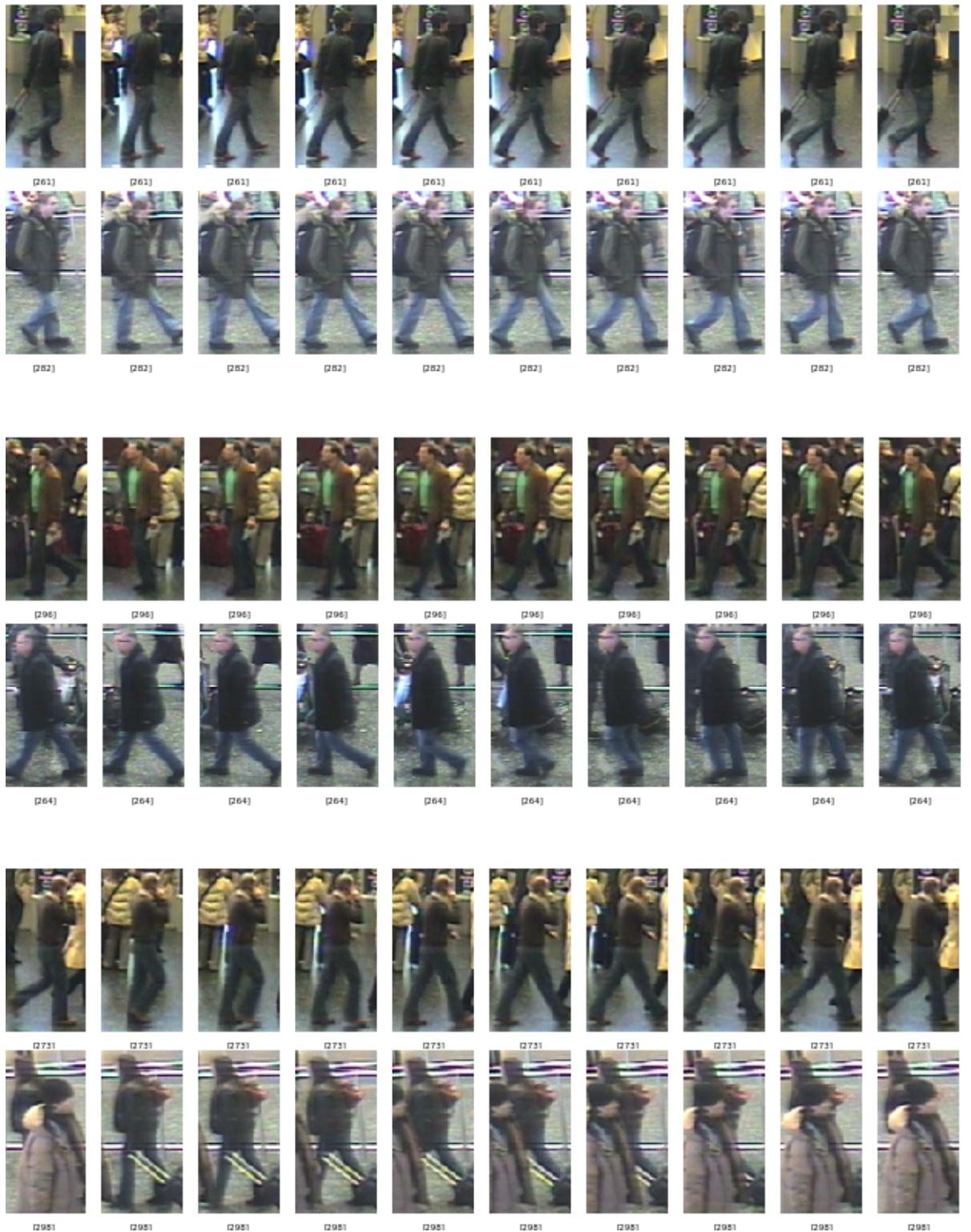
Obr. 23: Správna identifikácia zhodného (pozitívneho) páru sekvencií



Obr. 24: Správna identifikácia nezhodného (negatívneho) páru sekvencií



Obr. 25: Páry sekvencií nesprávne označené ako nezhodné



Obr. 26: Páry sekvencí nesprávne označené ako zhodné

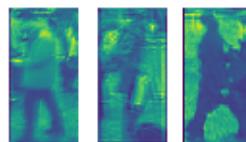
Zbežnou analýzou chybne identifikovaných párov sa ukázalo, že oba druhy chybných identifikácií sú najčastejšie zapríčinené oklúziami. Oproti modelom na báze samostatných obrazov je klasifikácia na základe video-sekvencií viac invariantná na rozdiely v svetelných podmienkach a farbách (oblečenia i pozadia) medzi sekvenciami v pár, keďže okrem priestorových príznakov na jednotlivých obrazoch sekvencie sa sieť sústredí taktiež na časový aspekt sekvencie (napr. chôdzu). Kompletná galéria výsledkov je dostupná na elektronickom médiu na lokalite:

/results/model_rnn_c16_c32_c64_ssp_rnn_atpl_ce/classification

4.3 Zhodnotenie

Aj napriek tomu že neboli natrénované všetky kombinácie menených hodnôt hyper-parametrov sietí, ako napríklad pri metóde krížovej validácie (*ang. cross-validation*) [18], tak množina evaluovaných modelov má dostatočnú veľkosť na vyvodenie záverov o tom, ako jednotlivé hyper-parametre ovplyvňovali výkonnosť konkrétnych modelov.

Správnosť voľby veľkosti filtrov v jednotlivých konvolučných vrstvách je možné overiť pomocou vizualizácie výstupných príznakových máp z týchto vrstiev. Pri voľbe príliš malej alebo naopak príliš veľkej veľkosti konvolučných filtrov nie je takmer možné z príznakových máp identifikovať obraz, ktorý sie spracovávala (napríklad kontúry osoby ako na obrázkoch v časti 4.1.2), čo znamená že filtre neboli schopné správne zachytiť príznaky obrazov. Nesprávna veľkosť filtrov sa tak tiež prejavila pre vyhodnocovanie modelu určenými metrikami (viď. sekcia 4.1.6). Porovnanie príznakových máp pre tri veľkosti filtrov je ilustrované na obrázku č. 27.



Obr. 27: Príklady výstupnej príznakovovej mapy pre veľkosť filtrov: 5×5 , 11×11 a 3×3 (zľava doprava)

Počet konvolučných filtrov ako i počet neurónov v plne prepojených vrstvách je priamo úmerný počtu trénovaných parametrov siete. Ak je počet týchto parametrov príliš vysoký a tréningový súbor dát nie je dostatočne rozsiahli, tak má sieť tendenciu k pre-trénovaniu sa, čím stratí schopnosť generalizovať svoju inferenciu na testovacom súbore dát. Dosvedčuje to i evaluácia modelu (a jemu podobných modelov) popísaná v stati 4.1.7.

Vyplývajúcou zovšeobecnenou stratégiou pri ladení hyper-parametrov neurónových sietí trénovaných na obmedzenom súbore dát je začať s menej rozsiahloou architektúrou siete a postupne ju rozširovať do hĺbky (počtom vrstiev) i do šírky (počtom neurónov v skrytých vrstvách) a empiricky zistiť, kde sa nachádza hranica, pri ktorej dôjde k pre-trénovaniu modelu.

Pri modeloch pre re-identifikáciu osôb na báze video-sekvencií, ktoré využívali kombináciu konvolučnej a rekurentnej neurónovej siete, platia pre konvolučnú časť modelu rovnaké zistenia ako pre čisto konvolučné modely. Návrh rekurentných častí týchto modelov je podmienený nutnosťou extrahovať z video-sekvencií iba dôležité príznaky a naopak potlačiť vplyv nevýrazných duplicitných informácií. Modely využívajúce vrstvu ATPL zameranú na splnenie tohto cieľa dosahovali na základe metriky AUC až o 56% lepšie výsledky, než modely bez nej.

Na základe vizuálnej analýzy výsledkov klasifikácie párov oboma druhmi modelov (na báze samostatných obrazov i video-sekvencií) sa ako najvýznamnejšie dôvody nesprávnej klasifikáciu ukázali: rozdiely v osvetlení, farebná odlišnosť (resp. podobnosť - podľa druhu chyby) a oklúzia s inými objektmi. Modely na báze video-sekvencií boli vďaka získavaniu priestorovo-časových príznakov z rekurentnej vrstvy (narozdiel od iba priestorových príznakov pri konvolučných modeloch) hore uvedenými problémami menej ovplyvnené. Čo dokazujú aj rozdiely metrík AUC a F1 skóre, kde modely na báze video-sekvencií dosahujú o 5.2% a 4.6% lepšie výsledky, než modely na báze samostatných obrazov.

5 Záver

Cieľom práce bolo analyzovať problematiku vizuálneho rozpoznávania objektov a navrhnuť vlastnú metódu rozpoznávania objektov zameranú na operáciu, najmä nad video-sekvenciami, s využitím metód hlbokého učenia a neurónových sietí. Problémová doména, pre ktorú bolo riešenie vypracované, bola špecifikovaná na problém re-identifikácie osôb vo video-sekvenciach.

V rámci analýzy problémovej domény a existujúcich riešení sme sa najskôr zamerali na priblíženie problematiky a hrubý opis procesu re-identifikácie osoby. Následne sme sa zamerali na detailnejšiu analýzu jednotlivých krokov tohto procesu, pri ktorej sme si priblížili deterministické algoritmy a postupy počítačového videnia aplikovateľné na zvolenú problémovú doménu. Najviac bol kladený dôraz na opis lokálnych detektorov a deskriptorov, ako prostriedkov na získavanie príznakov obrazov. Potom bol priblížený postup párovania príznakov medzi dvoma obrazmi, predstavujúci ľažisko porovnávania obrazov, ktoré je v rámci re-identifikácie osôb kľúčové.

Nasledujúca časť analýzy bola venovaná teórií hlbokého učenia a neurónových sietí ako alternatíve ku konvenčným algoritmom počítačového videnia. Detailne boli opísané konvolučné a rekurentné siete ako i architektonický model siamských neurónových sietí. Tieto koncepty hlbokého učenia predstavujú dnes najpoužívanejšie techniky pre spracovávanie obrazov, časových sekvencií a ich porovnávanie a tvoria piliere takmer všetkých súčasných metód pre re-identifikáciu osôb, ktoré využívajú hlboké učenie.

Po priblížení teoretických základov nasledoval opis dvoch existujúcich publikovaných riešení na re-identifikácie osôb. Analýza problematiky bola zakončená prehľadom dostupných nástrojov na implementáciu algoritmov strojového učenia a opis vybranej použitej knižnice Tensorflow.

V úvode praktickej časti bola definovaná komponentová štruktúra navrhovaného systému pozostávajúca z modulu na prípravu dát, tréning, definíciu modelu neurónovej siete a vyhodnocovanie modelov. Ďalej bola v rámci návrhu upresnená stratégia delenia súboru dát na testovacie a tréningové časti ako i spôsob pred-spracovania dát formou ich augmentácie. Následne boli popísané topológie nami navrhnutých modelov neurónových sietí pre re-identifikáciu osôb a metriky,

pomocou ktorých bola úspešnosť týchto modelov meraná. V rámci opisu topológií bolo poukázané aj na hyper-parametre, ktorých vplyv na úspešnosť modelu bol skúmaný.

Trénované boli dve skupiny modelov: modely pre re-identifikáciu osôb na základe samostatných obrazov a modely pre re-identifikáciu osôb na základe video-sekvencií. Prezentované boli výsledky vybraných modelov z každej z uvedených skupín, najmä tie, ktoré poukazovali na zaujímavé súvislosti hodnôt hyperparametrov modelu a jeho úspešnosti.

Vzhľadom na malú veľkosť súboru dát predstavovalo najväčší problém pretrénovanie modelov a návrh efektívnej topológie pre modely re-identifikácie na základe video-sekvencií. Opatrenia proti pre-trénovaniu pozostávali z využitia regularizačnej techniky drop-out, augmentácie dát a optimálnej zložitosti (počtu parametrov) modelu. Najvyššia úspešnosť modelov pre re-identifikáciu osôb vo video-sekvenciách bola dosiahnutá použitím špeciálnych vrstiev SPP a ATPL.

Najúspešnejšie z oboch kategórií modelov boli vybrané na základe hodnôt metrík AUC a F1 skóre a následne boli testované ako klasifikátory. Na základe výsledkov klasifikácií sme určili aspekty, ktoré najčastejšie zapríčňovali chybnú klasifikáciu. Medzi tieto faktory parili výrazné odlišnosti v pozadí obrazu, svetelných podmienkach a pozorovacom uhle osoby. Ďalším významným aspektom ovplyvňujúcim správnosť klasifikácie je zakrytie (oklúzia) re-identifikovanej osoby inými osobami alebo objektami, poprípade i samotná prítomnosť iných osôb v obrazze (aj keď nezapríčňujú oklúziu).

Okrem prezentovania absolútnych výsledkov jednotlivých modelov spracovaných v tejto práci boli tie najlepšie výsledky spomedzi nich porovnané i relatívne voči publikáciám, ktoré opisujú podobné riešenia. V tejto konkurencii dosahujú naše najúspešnejšie modely priemerné výsledky.

Literatúra

- [1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004.
- [2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.
- [3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.
- [4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 737–744. Morgan-Kaufmann, 1994.
- [5] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, Apr 1983.
- [6] Junyoung Chung, Çağlar Gülcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [7] J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and Trainability in Recurrent Neural Networks. *ArXiv e-prints*, November 2016.
- [8] Franklin C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212, January 1984.
- [9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.
- [10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report

UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

- [11] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.
- [12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [13] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 1735–1742, Washington, DC, USA, 2006. IEEE Computer Society.
- [14] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.
- [15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [16] F. M. Khan and F. Brèmond. Multi-shot person re-identification using part appearance mixture. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 605–614, March 2017.
- [17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [18] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
- [19] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, Norwell, MA, USA, 1994.

- [20] Tony Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, November 1998.
- [21] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.
- [22] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP’09*), pages 331–340. INSTICC Press, 2009.
- [23] S. Oh, S. Russell, and S. Sastry. Markov chain monte carlo data association for multi-target tracking. *IEEE Transactions on Automatic Control*, 54(3):481–497, March 2009.
- [24] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, Jan 2010.
- [25] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.
- [26] T. Wang, S. Gong, X. Zhu, and S. Wang. Person re-identification by discriminative selection in video ranking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(12):2501–2514, Dec 2016.
- [27] Shuangjie Xu, Yu Cheng, Kang Gu, Yang Yang, Shiyu Chang, and Pan Zhou. Jointly attentive spatial-temporal pooling networks for video-based person re-identification. *CoRR*, abs/1708.02286, 2017.
- [28] Zhen Zhou, Yan Huang, Wei Wang, Liang Wang, and Tieniu Tan. See the forest for the trees: Joint spatial and temporal recurrent neural networks for video-based person re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [29] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

A Plán práce a jeho plnenie

A.1 Zimný semester

- **1-2. týždeň** - oboznámenie sa s detailami témy, povrchové štúdium problémovej domény (počítačové videnie, rozpoznávanie objektov).
- **3-4. týždeň** - rozprava o konkretizácii témy, štúdium vybraných častí o hlbokom učení, rešerš knižníc na implementáciu hlbokého učenia.
- **4-6. týždeň** - špecifikovanie témy na doménu re-identifikácie osôb, výber knižnice Tensorflow, štúdium príkladov naprogramovaných pomocou Tensorflow-u, naprogramovanie jednoduchého OCR (*ang. optical character recognition*) pomocou hlbokej neurónovej siete.
- **7-8. týždeň** - rešerš a štúdium vybraných vedeckých publikácií z oblasti re-identifikácie osôb, prehľadové články o metódach re-identifikácie osôb ako aj niektoré konkrétné metódy opísané do hĺbky, prerobenie modelového OCR klasifikátora tak, aby využíval konvolučnú sieť.
- **9-10. týždeň** - štúdium o klasických metódach počítačového videnia (lokálne detektory, deskriptory a ich párovanie), štúdium ďalších publikácií, štúdium zložitejších mechanizmov tensorflow-u (zdieľanie premenných, rekurentné siete, atď.), spisovanie analýzy práce.
- **11-12. týždeň** - pokračovanie spisovania analýzy, štúdium detailov ohľadne problémov opisovaných v analýze (SIFT, SURF deskriptory, PAM metóda).
- **12-14. týždeň (predĺženie)** - štúdium detailov ohľadne vybraných častí hlbokého učenia (rekurentné siete, LSTM bunky), dokončenie spísania analýzy práce, spisanie konceptu riešenia, odovzdanie práce v zimnom semestri,

Plán práce bol v zimnom semestri bol splnený s minimálnymi odchýlkami.

A.2 Letný semester

- **1-2. týždeň** - implementácia siamskej architektúry v tensorflow-e, zadefinovanie loss funkcie a tréningového cieľa, prvé skúšky trénovania modelu.
- **3-4. týždeň** - implementácia delenia dát na tréningové a testovacie páry a dávkovanie, štúdium vizualizácie štatistik.
- **4-6. týždeň** - natrénovanie prvého funkčného modelu a jeho analýza, diskusia o tom ktoré hyper-parametre by bolo vhodné meniť, implementácia logovania štatistik
- **7-8. týždeň** - implementácia výpočtu optického toku pomocou OpenCV knížnice a zakomponovanie vstupu optického toku do modelu, tréning ďalších konfigurácií modelu, spisovanie návrhu riešenia a samotného riešenia.
- **9-10. týždeň** - analýza rozdielov medzi konfiguráciami modelu a porovnanie výsledkov s inými publikáciami
- **11-12. týždeň** - testovanie najúspešnejších modelov v konfigurácií reálnych klasifikátorov a analýza výsledkov týchto klasifikácií. Dokončenia dokumentácie k práci a zapracovanie pripomienok vedúceho práce

Plán práce v letom semestri bol obsahovo naplnený, avšak v niektorých prípadoch nebolo dodržané presné časové rozvrhnutie.

B Technická dokumentácia

Dátové súbory a zdrojové kódy tohto projektu sú organizované do adresárov a balíkov podľa funkcionality a významu. Projekt obsahuje štyri Python balíky:

- `./dataprep/` - pozostáva z modulov poskytujúcich funkcionalitu generovania tréningových, testovacích a rankovacích párov obrazov i video-sekvencií.
- `./models/` - obsahuje zdrojové kódy pre vytvorenie výpočtových grafov jednotlivých modelov neurónových sietí.
- `./training/` - združuje programy, ktoré vykonávajú samotný tréning neurónových sietí, ich funkcionalitou je taktiež ukladanie váh siete v pravidelných intervaloch a ich načítanie v prípade obnovenia procesu tréningu po jeho prerušení.
- `./metrics/` - tento balík agreguje moduly pre výpočet vyhodnocovacích metrík pre zvolený model neurónovej siete.

Okrem balíkov zdrojových kódov obsahuje ďalej projekt nasledujúce adresáre obsahujúce vstupné a vygenerované dátá:

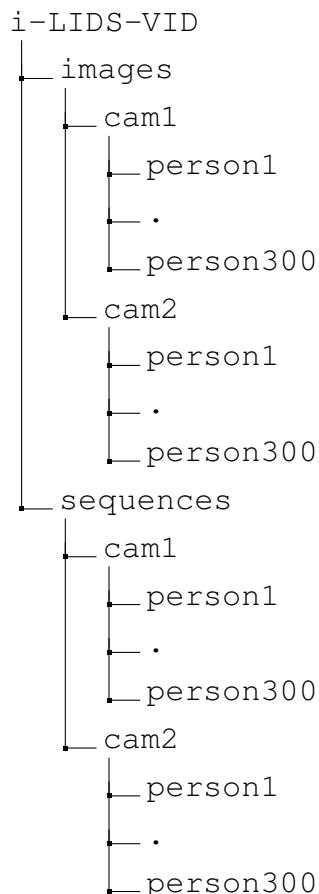
- `./i-LIDS-VID/` - adresár obsahujúci obrazy súboru dát.
- `./results/` - slúži na ukladanie vytvorených súborov obsahujúcich výsledky testovacích metrík.
- `./weights/` - obsahuje súbory s naučenými váhami všetkých trénovaných modelov neurónových sietí

B.1 Príprava dát

Príprava dát je realizovaná balíkom `dataprep/`. Procedúry modulov tohto balíka selektujú a transformujú dátá z adresára `i-LIDS-VID/` do formátu kompatibilného so vstupnými dátami pre trénovacie a vyhodnocovacie moduly.

B.1.1 Organizácia súboru dát i-LIDS-VID

Súborová štruktúra i-LIDS-VID datasetu je nasledujúca:



Adresáre `person[x]` už ďalej obsahujú konkrétnie obrazy osôb. V prípade pod-adresára `images` sa jedná o jeden obraz osoby v každom adresári. V pod-adresároch `sequences` sú to sekvencie obrazov premenlivej dĺžky. Všetky obrazy sú veľkosti 128×64 pixlov vo farebnom modeli RGB (3 farebné kanály).

B.1.2 Moduly pre prípravu dát

V balíku `dataprep/` sa nachádzajú nasledujúce moduly:

1. `ilidsvid.py` - slúži na generovanie, testovacích a tréningových párov samostatných obrazov z `i-LIDS-VID/images`
2. `ilidsvid_seq.py` - zabezpečuje vytváranie testovacích a tréningových

párov samostatných obrazov z i-LIDS-VID/sequences a poskytuje metódy pre augmentáciu tréningových dát.

3. `ilidsvid_vid.py` - zastrešuje tvorbu testovacích a tréningových párov sekvenčí obrazov z i-LIDS-VID/sequences. Okrem toho poskytuje tiež prostriedky pre augmentáciu a výpočet optického toku z dát.
4. `ilidsvid_rank.py` - poskytuje metódy pre tvorbu množiny párov $M = P_1 \times P_2$, kde P_1 a P_2 sú množiny obrazov osôb zachytených kamerou 1 resp. kamerou 2 v zložke i-LIDS-VID/images. Táto množina je využívaná na výpočet n -rank metriky.
5. `ilidsvid_rank_vid.py` - obdobne ako v prípade `ilidsvid_rank.py`, ale vytváraná je množina párov video-sekvenčí.

Pri moduloch generujúcich vstupné dávky (tréningové alebo testovacie) pre neurónové siete sú osoby pre zostavenie párov v dávke vyberané náhodne z množiny buď tréningových, alebo testovacích osôb. Konkrétnie obrazy alebo pod-sekvencie voliteľnej dĺžky z celej sekvencie sú tiež vyberané náhodne.

Postup tvorby dávky samostatných obrazov

Načítavanie obrazov zo súboru do pola v pamäti tvaru [128, 64, 3] je vykonávané metódou `misc.imread` knižnice *Scipy*. Toto pole je následne upravené na jednorozmerné tvaru [128 * 64 * 3]. Obe polia (obrazy) tvoriace párs spolu s označením ich podobnosti a označením oboch osôb sú uložené ako objekt. Dávka sa vždy skladá z rovnakého počtu vygenerovaných pozitívnych a negatívnych párov. Výsledná dávka je dvojrozmerné pole tvaru [b, 128 * 64 * 3], kde b je používateľom špecifikovaná veľkosť dávky. Keďže hodnoty intenzity pixlov sú z intervalu <0, 255> a cieľom ich je normalizovať na hodnoty z intervalu <0, 1> je nutné ich predeliť hodnotou 255. Na operáciu s poliami bola použitá knižnica *Numpy*. Úryvok kódu, ktorý demonštruje proces tvorby dávky je uvedený v ukážke 2.

Ukážka 2: Ukážka tvorby dávky statických obrazov

```
from scipy import misc
import numpy as np
import random
import os

class Pair:
    def __init__(self, image1, image1_label, image2, image2_label, label):
        self.image1 = image1
        self.image1_label = image1_label
        self.image2 = image2
        self.image2_label = image2_label
        self.label = label

    def get_negative_pair(training):
        persons = get_persons()
        indices = []
        if training:
            indices = random.sample(range(0, int(len(persons) * 0.8)), 2)
        else:
            indices = random.sample(range(int(len(persons) * 0.8), len(persons)), 2)
        p1 = persons[indices[0]]
        p2 = persons[indices[1]]

        si1 = random.sample(range(len(get_person_sequence(p1, 1))), 1)
        si2 = random.sample(range(len(get_person_sequence(p2, 2))), 1)

        img1 = get_person_sequence(p1, 1)[si1[0]]
        img2 = get_person_sequence(p2, 2)[si2[0]]

        image1 = misc.imread(name="..\\i-LIDS-VID\\sequences\\cam1\\" + p1 + "/" + img1)
        image1 = np.reshape(image1, (-1))
        image2 = misc.imread(name="..\\i-LIDS-VID\\sequences\\cam2\\" + p2 + "/" + img2)
        image2 = np.reshape(image2, (-1))
        return Pair(image1, p1, image2, p2, 0.0)

    def get_pairs(batch_size, training):
        pairs = []
        for i in range(int(batch_size / 2)):
            pairs.append(get_positive_pair(training))

        for i in range(int(batch_size / 2)):
            pairs.append(get_negative_pair(training))

        random.shuffle(pairs)
        return pairs

    def get_batch(batch_size, training):
        pairs = get_pairs(batch_size, training)

        cam1_images = [pair.image1 for pair in pairs]
        cam1_images = np.asarray(cam1_images, dtype=np.float32) / 255.0
        cam2_images = [pair.image2 for pair in pairs]
        cam2_images = np.asarray(cam2_images, dtype=np.float32) / 255.0
        batch_labels = [pair.label for pair in pairs]
        batch_labels = np.asarray(batch_labels, dtype=np.float32)

        return cam1_images, cam2_images, batch_labels
```

Postup tvorby dávky video-sekvencií

Postup tvorby dávok video-sekvencií je principiálne podobný tvorbe dávok samostatných obrazov. Rozdiel je v tvaru výstupnej dávky: $[b * s, 128 * 64 * c]$, kde s je dĺžka sekvencie a c je počet kanálov obrazov (3 bez výpočtu optického toku 5 s jeho výpočtom). Výpočet optického toku bol realizovaný funkciou z knižnice *OpenCV*: *cv2.calcOpticalFlowFarneback*.

Dostupné procedúry

Nižšie je uvedený zoznam procedúr (i s popisom ich argumentov) použiteľných na vygenerovanie dávok dát:

- *ilidsvid.get_augmented_batch(batch_size, training)* - augmentovaná dávka samostatných obrazov z i-LIDS-VID/images
- *ilidsvid_seq.get_augmented_batch(batch_size, training)* - augmentovaná dávka samostatných obrazov z i-LIDS-VID/sequences
- *ilidsvid.get_batch(batch_size, training)* - neaugmentovaná dávka samostatných obrazov z i-LIDS-VID/images
- *ilidsvid_seq.get_batch(batch_size, training)* - neaugmentovaná dávka samostatných obrazov z i-LIDS-VID/sequences
- *ilidsvid_vid.get_batch(training, optical_flow, augment, batch_size, seq_len)* - dávka video-sekvencií z i-LIDS-VID/sequences
- *batch_size* - prirodzené číslo - veľkosť dávky
- *training* - boolean - testovacia/tréningová dávka
- *optical_flow* - boolean - výpočet optického toku medzi obrazmi video-sekvencie
- *augment* - boolean - augmentácia dávky
- *seq_len* - prirodzené číslo - dĺžka video-sekvencie

B.2 Modely neurónových sietí

Všetky vytvorené modely neurónových sietí sú vo forme triedy v samostatnom zdrojovom súbore v balíku `models/`. Výpočtový graf sa vytvorí zavolením konštruktora tejto triedy. Jednotlivé vrstvy sú vytvárané volaním metód triedy. Na zabezpečenie zdieľania váh medzi vетvami siamskej siete bola použitá konštrukcia znovupoužitia premenných knižnice Tensorflow, tak ako je to znázornené v ukážke č. 3.

Ukážka 3: *Ukážka znovupoužitia premenných*

```
import tensorflow as tf

with tf.variable_scope("siamese") as scope:
    self.out1 = self.convnet(self.input1, training)
    scope.reuse_variables()
    self.out2 = self.convnet(self.input2, training)
```

Contrastive loss funkcia bola implementovaná manuálne pomocou Tensorflow operácií, keďže doposiaľ nebola implementovaná ako súčasť knižnice.

Keďže model je enkapsulovaný v triede, tak navonok je ako jej rozhranie dostupných iba niekoľko dôležitých uzlov (operácií) výpočtového grafu. Medzi ne patria hlavne:

1. *input1* - vstupná premenná (placeholder) pre prvú časť páru vstupných dát
2. *input2* - vstupná premenná (placeholder) pre druhú časť páru vstupných dát
3. *labels* - vstupná premenná (placeholder) pre označenie podobnosti páru
4. *loss* - operácia contrastive loss funkcie
5. *distance* - operácia Euklidovskej vzdialenosťi páru

Komplexnejšie konvolučno-rekurentné modely (najmä tie využívajúce cross-entropy loss funkciu) poskytujú navonok viacero operácií pre interakciu s modelom a môžu vyžadovať dodatočné vstupné dátá (napr. označenie osôb). V ukážke č. 4 je vyobrazený kód pre vystavanie výpočtového grafu dvojvrstvovej siamskej konvolučnej siete.

Ukážka 4: Model dvojvrstvovej siamskej konvolučnej siete

```
import tensorflow as tf

class Siamese:

    def __init__(self, training):
        self.input1 = tf.placeholder(tf.float32, [None, 24576])
        self.input2 = tf.placeholder(tf.float32, [None, 24576])

        with tf.variable_scope("siamese") as scope:
            self.out1 = self.convnet(self.input1, training)
            scope.reuse_variables()
            self.out2 = self.convnet(self.input2, training)

        self.labels = tf.placeholder(tf.float32, [None])
        self.loss = self.contrastive_loss()
        self.distance = self.euclidian_distance()

    def convnet(self, inputx, training):
        input_reshaped = tf.reshape(inputx, [-1, 128, 64, 3])
        conv1 = self.conv_layer(input_reshaped, [5, 5, 3, 32], [32], "conv1")

        max1 = tf.layers.max_pooling2d(inputs=conv1,
                                       pool_size=[2, 2],
                                       strides=2,
                                       name="max1")

        conv2 = self.conv_layer(max1, [5, 5, 32, 64], [64], "conv2")

        max2 = tf.layers.max_pooling2d(inputs=conv2,
                                       pool_size=[2, 2],
                                       strides=2,
                                       name="max2")

        max2flat = tf.reshape(max2, [-1, 32 * 16 * 64], name="max2flat")

        dense1 = self.fc_layer(max2flat, 512, "dense1")

        dropout1 = tf.layers.dropout(inputs=dense1,
                                     rate=0.3,
                                     name="dropout1",
                                     training=training)

        out = self.fc_layer(dropout1, 32, "dense2")
        return out

    def conv_layer(self, inputx, kernel_shape, bias_shape, name):
        weights = tf.get_variable(name=name + "_weights",
                                  dtype=tf.float32,
                                  shape=kernel_shape,
                                  initializer=tf.random_normal_initializer)
        biases = tf.get_variable(name=name + "_biases",
                                 dtype=tf.float32,
                                 shape=bias_shape,
                                 initializer=tf.constant_initializer)
        conv = tf.nn.conv2d(input=inputx,
                            filter=weights,
                            strides=[1, 1, 1, 1],
                            padding='SAME')
        return tf.nn.relu(conv + biases)

    def fc_layer(self, _input, units, name):
        assert len(_input.get_shape()) == 2
        n_prev_weight = _input.get_shape()[1]
        initializer = tf.truncated_normal_initializer(stddev=0.01)
```

```

weights = tf.get_variable(name=name + '_weights',
                         dtype=tf.float32,
                         shape=[n.prev_weight, units],
                         initializer=initializer)
biases = tf.get_variable(name=name + '_biases',
                         dtype=tf.float32,
                         shape=[units],
                         initializer=tf.constant_initializer)
return tf.nn.bias_add(tf.matmul(_input, weights), biases)

def euclidian_distance(self):
    euclidian2 = tf.pow(tf.subtract(self.out1, self.out2), 2)
    euclidian2 = tf.reduce_sum(euclidian2, 1)
    return tf.sqrt(euclidian2 + 1e-6, name="distance")

def contrastive_loss(self):
    margin = 5.0
    c = tf.constant(margin)
    labels_true = self.labels
    labels_false = tf.subtract(1.0, self.labels, name="1-y")
    euclidian = self.euclidian_distance()
    pos = tf.multiply(labels_true, euclidian, name="y_x_distance")
    neg = tf.multiply(labels_false, tf.pow(tf.maximum(tf.subtract(c, euclidian), 0), 2), name="ny_x_c->_distance_2")
    losses = tf.add(pos, neg, name="losses")
    loss = tf.reduce_mean(losses, name="loss")
    return loss

```

Na ukážke č. 5 nižšie sú uvedené procedúry na vytváranie rekurentnej SPP a ATPL vrstvy. Tieto vrstvy boli implementované základnými maticovými a aritmetickými operáciami, keďže nie sú implementované ako funkcie API vyšej úrovne.

Ukážka 5: Procedúry pre tvorbu rekurentnej , SPP a ATPL vrstvy

```

def spp_layer(self, input_, levels, name):
    shape = input_.get_shape().as_list()
    with tf.variable_scope(name):
        pool_outputs = []
        for level in levels:
            kernel = [1,
                      np.ceil(shape[1] * 1.0 / level).astype(np.int32),
                      np.ceil(shape[2] * 1.0 / level).astype(np.int32),
                      1]
            stride = [1,
                      np.floor(shape[1] * 1.0 / level).astype(np.int32),
                      np.floor(shape[2] * 1.0 / level).astype(np.int32),
                      1]
            poll = tf.nn.max_pool(value=input_,
                                  ksize=kernel,
                                  strides=stride,
                                  padding='SAME',
                                  name="spp-pool")
            pool_outputs.append(tf.reshape(poll, [shape[0], -1]))
        spp_pool = tf.concat(pool_outputs, 1)
        spp_pool = tf.reshape(spp_pool, [self.seq_len, self.batch_size, -1])
        spp_pool = tf.unstack(spp_pool)

    return spp_pool

def atpl_layer(self, input1, input2):
    temp_mat = tf.get_variable(name="temp_mat",
                               shape=[self.hidden_size, self.hidden_size],
                               dtype=tf.float32,
                               initializer=tf.random_normal_initializer)

```

```

# in1_temp_mat = tf.scan(lambda a, x: tf.matmul(x, temp_mat), input1)
in1_temp_mat = tf.matmul(tf.reshape(input1, [self.batch_size * self.seq_len, self.hidden_size]), >>
    >> temp_mat)
in1_temp_mat = tf.reshape(in1_temp_mat, [self.batch_size, self.seq_len, self.hidden_size])
self.in1_temp_mat_in2 = tf.matmul(in1_temp_mat, input2, transpose_b=True)

self.atpl_mat = tf.tanh(self.in1_temp_mat_in2, name="atpl_mat")

max_col = tf.reduce_max(self.atpl_mat, axis=1, name="max_col")
max_row = tf.reduce_max(self.atpl_mat, axis=2, name="max_row")

col_softmax = tf.nn.softmax(max_col)
row_softmax = tf.nn.softmax(max_row)

out1 = tf.squeeze(tf.matmul(input1, tf.expand_dims(col_softmax, -1), transpose_a=True))
out2 = tf.squeeze(tf.matmul(input2, tf.expand_dims(row_softmax, -1), transpose_a=True))

out1 = tf.reshape(out1, [self.batch_size, -1])
out2 = tf.reshape(out2, [self.batch_size, -1])
return out1, out2

def rnn_layers(self, inputs, hidden_size):
    cell = tf.nn.rnn_cell.BasicRNNCell(num_units=hidden_size)
    outputs, _ = tf.nn.static_rnn(cell=cell,
                                 inputs=inputs,
                                 dtype=tf.float32)

    return outputs

```

B.3 Tréning

Cieľom tréningových modulov bola správa tréningového procesu neurónovej siete nezávislá na trénovanom modely alebo použitom module pre generovanie vstupných dát. Všetky tréningové moduly sú obsiahnuté v rámci balíka `./training`. Výsledná funkcia pozostáva z:

1. Inicializácia importovaného modelu siete.
2. Vytvorenia optimizéra (`tf.train.GradientDescentOptimizer`).
3. Inicializácie správcu ukladania a načítavania učených váh (`tf.train.Saver()`)
4. Načítanie existujúcich váh ak existujú.
5. Spustenie tréningového cyklu.
6. Výpočet kĺzavého priemeru hodnoty loss funkcie na zistenie konvergencie.
7. Ukladanie váh každých x iterácií tréningového cyklu

Vrámci tréningových modulu má používateľ možnosť meniť model, ktorý chce trénovať, modul pre generovanie vstupných dávok, rýchlosť učenia (ang. learning rate), použitý druh optimizéra a veľkosť vstupnej dávky. Váhy súte sú ukladané do súborov s príponou *.ckpt*. Tieto súbory sú pre človeka nečitateľné. Interakcia s modelom prebieha pomocou metódy *session.run()*, ktorá nám umožňuje vykonať konkrétnu operáciu vo výpočtovom grafe modelu a poskytnúť modelu vstupné dátu vo forme slovníka (ang. dictionary). Aby mohla byť operácia v grafe vykonaná, je nutné poskytnúť vstupné dátu všetkým vstupným (placeholder) premenným. V tréningovom cykle sa vykonáva operácia minimalizácie loss funkcie modelu, ktorú poskytuje zvolený optimizér. Príklad tréningového cyklu je vyobrazený v ukážke č. 6.

Ukážka 6: Tréningový cyklus

```
import tensorflow as tf
import numpy as np

siamese = model.Siamese(True)
train_step = tf.train.GradientDescentOptimizer(0.00001).minimize(siamese.loss)

for step in range(1000000):
    batch_x1, batch_x2, batch_y = dataset.get_augmented_batch(20, True)

    _, loss_v = sess.run([train_step, siamese.loss], feed_dict={
        siamese.input1: batch_x1,
        siamese.input2: batch_x2,
        siamese.labels: batch_y})

    losses_window.append(loss_v)

    if len(losses_window) > 1000:
        losses_window.remove(losses_window[0])

    if step % 10 == 0:
        avg_loss = np.asarray(losses_window)
        avg_loss = np.mean(avg_loss)
        print('step %d: avg_loss %.3f' % (step, avg_loss))
        losses_agg.append(loss_v)

    if step % 100 == 0 and step > 0:
        saver.save(sess, './weights/' + model_name + '.ckpt')
```

B.4 Metriky

Kostra modulov pre výpočet metrík je podobná ako u tréningových modulov. Tak tiež prebieha inicializácia evaluovaného modelu a načítanie jeho naučených váh. V cykle je následne pre n dávok testovacích dát vykonávaná metódou *session.run()*

operácia modelu **siamese.distance**, ktorá počíta vzdialenosť medzi prvkami páru po jeho zasadení sietou do n -rozmerného priestoru. Zo získanej množiny vzdialenosťí D zistíme maximálnu vzdialenosť párov d_{max} . Pre zadefinovaný počet hraničných vzdialenosťí n vieme potom určiť inkrement hraničnej vzdialenosťi s ako $s = \frac{d_{max}}{n}$. Následne je možné pristúpiť k výpočtu metrík opísaných v stati 3.5. Časť kódu ukazujúca výpočet týchto metrík je vyobrazená v ukážke č. 7

Ukážka 7: Výpočet metrík pre všetky hraničné vzdialenosťi

```

num_of_thresholds = 2000
max_dist = max(distances)
threshold_step = max_dist / num_of_thresholds

thresholds = []
for i in range(num_of_thresholds + 1):
    thresholds.append(i * threshold_step)

true_positives = []
true_negatives = []
false_positives = []
false_negatives = []
true_positive_rates = []
false_positive_rates = []

f1s = []
accs = []

for i in thresholds:
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    for j in range(len(distances)):
        if int(test_labels[j]) == 0:
            if distances[j] > i:
                tn += 1
            else:
                fp += 1
        elif int(test_labels[j]) == 1:
            if distances[j] > i:
                fn += 1
            else:
                tp += 1

    true_positives.append(tp)
    true_negatives.append(tn)
    false_positives.append(fp)
    false_negatives.append(fn)
    true_positive_rates.append(tp / (tp + fn))
    false_positive_rates.append(fp / (fp + tn))
    f1s.append((2*tp) / (2*tp + fp + fn))
    accs.append((tp + tn) / (tp + tn + fp + fn))

```

Z vypočítaných metrík je vykreslený graf zodpovedajúci ROC krivke, ktorý je vo formáte *.png* uložený do adresára *./results*. Číselné hodnoty metrík AUC, F1 a ACC sú uložené do textového súboru do rovnakého adresára. Názvy textových súborov i grafov obsahujú názov evaluovaného modelu pre ich jednoduchú

identifikáciu.

Výpočet metriky n -rank je realizovaný podľa postupu v sekcií 3.5 a využívajú sa pri tom dátá generované modulmi `iliidsvid_rank.py`, pre hodnotenie modelov pracujúcimi so samostatnými obrazmi a `iliidsvid_rank_vid.py`, pre modely pracujúce s video-sekvenciami. Kód pre výpočet n -rank metriky je uvedený v ukážke 8.

Ukážka 8: Výpočet n -rank metriky

```
persons = dataset.get_persons(test_only)

ranks_histogram = np.zeros(shape=[len(persons)], dtype=np.int32)
ranks = []
confused_pairs = []

for person in persons:
    pairs = dataset.get_person_pairs(person, test_only)

    for pair in pairs:
        image1 = np.reshape(pair.image1, (1, 24576))
        image2 = np.reshape(pair.image2, (1, 24576))
        distance = sess.run([siamese.distance], feed_dict={
            siamese.input1: image1,
            siamese.input2: image2
        })
        pair.set_distance(distance)
    pairs.sort(key=lambda x: x.distance)

    if not pairs[0].label:
        confused_pairs.append([pairs[0].image1_label, pairs[0].image2_label])

    rank = 0
    while not pairs[rank].label:
        rank += 1
    ranks_histogram[rank] += 1
    ranks.append(rank)
    print(person, rank)

for i in range(0, len(ranks_histogram) - 1):
    ranks_histogram[i + 1] += ranks_histogram[i]

ranks_percents = ranks_histogram / len(persons)
ranks = np.asarray(ranks, dtype=np.float32)
avg_rank = np.mean(ranks)
```

C Inštalačná príručka

Cieľom tejto inštalačnej príručky je previesť používateľa procesom inštalácie softvéru potrebného pre editovanie a spúšťanie zdrojových súborov projektu tejto bakalárskej práce.

C.1 Python 3.6

Kedže všetky zdrojové súbory projektu sú písané v jazyku Python 3, prvým krokom je inštalácia interpretéra tohto jazyka. Väčšina operačných systémov založený na systéme Unix už obsahuje určitú verziu jazyka Python. Napríklad Ubuntu 17.10 obsahuje verziu Python 3.6 a MacOS je dodávaný s verziou 2.7. Ak si však používateľ nemá jazyk nainštalovaný, poprípade chce aktualizovať existujúcu verziu má nasledujúce možnosti:

Ubuntu (Linux)

Od verzie Ubuntu 16.10 je možné nainštalovať Python 3 z univerzálneho Ubuntu repozitára pomocou nasledujúcich príkazov:

```
$ sudo apt-get update  
$ sudo apt-get install python3.6
```

MacOS

Kedže MacOS nie je dodávaný s najnovšou verziou jazyka Python, je nutná jej manuálna inštalácia. Je možná inštalácia cez príkazový riadok napríklad pomocou správca balíčkov Homebrew. Inštalácia správca Homebrew je možná nasledujúcim príkazom:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Inštalácia samotného jazyka Python pomocou Homebrew je možné vykonať nasledujúcim príkazom:

```
$ brew install python
```

Windows

Pre operačný systém Windows je dostupná možnosť stiahnutia spustiteľného inštalátora zo stránky: <https://www.python.org/downloads/>. Na uvedenej stránke sú dostupné inštalátory aj pre iné operačné systémy.

C.2 Správca Python balíkov - pip

Kedže zdrojové kódy projektu sú závislé od niekoľkých balíkov tretích strán (napr. i samotnej knižnice Tensorflow) je nutné tieto balíky sťahovať a inštalovať. Pre tento účel bol vytvorený správca Python balíkov *pip*. Ten už je spravidla súčasťou najnovších distribúcií jazyka Python. Pokiaľ však z nejakého dôvodu nie je v distribúcií zahrnutý, môže byť nainštalovaný manuálne nasledujúcimi príkazmi:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
$ python get-pip.py
```

C.3 Virtuálne prostredie Anaconda

Projekty zamerané na strojové učenie bývajú často závislé na veľkom množstve knižníc a softvérových rámcov, ktoré sú v neustálom vývoji. Častokrát je požiadavka mať predmetné softvérové závislosti v konkrétnych verziach, aby bola zaistená ich správna kompatibilita a interoperabilita. Virtuálne Python prostredia umožňujú používateľom vytvoriť viaceré izolované ekosystémy balíkov a knižníc v presne požadovaných verziách. Jedným z najpopulárnejších riešení je virtuálne prostredie Anaconda. Jeho inštalácia je možná buď pomocou inštalátora stiahnutého z webovej stránky: <https://www.anaconda.com/download/> alebo z príkazového riadka (príklad pre Linux verziu):

```
$ curl -O https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh  
$ bash Anaconda3-5.0.1-Linux-x86_64.sh
```

Po procese inštalácie je nutné vykonať:

```
$ source ~/.bashrc
```

Následne je možné vytvoriť virtuálne prostredie príkazom:

```
$ conda create --name my_env python=3
```

Nasledujúcim príkazom je možné do vytvoreného prostredia vstúpiť a začať s jeho konfiguráciou (inštaláciou balíkov):

```
$ source activate my_env
```

Po ukončení práce je možné uvedeným spôsobom z prostredia vystúpiť:

```
$ source deactivate
```

C.4 Inštalácia potrebných Python balíkov

Aby bolo možné spúštať tréningové a evaluačné skripty, ktoré projekt obsahuje, je nutné mať nainštalované nasledujúce Python balíky, pri ktorých je uvedená aj ich funkcia:

- Tensorflow - rámec pre hlboké strojové učenie. Je možné ho nainštalovať pre rôzne výpočtové zariadenia (CPU, GPU) a pre rôzne operačné systémy
- Numpy - výpočty nad n -rozmernými poliami
- Scipy - načítavanie obrazov do pamäte v podobe polí
- Matplotlib - vykreslovanie grafov
- Imgau - augmentácia obrazových dát
- OpenCV - knižnica implementuje algoritmy počítačového videnia. Využívaná na výpočet optického toku. Taktiež je na nej závislá knižnica Imgau

Ich inštalácia vo virtuálnom (alebo globálnom) Python prostredí je možná pomocou *pip* nasledujúcimi príkazmi:

```
#Tensorflow CPU Linux
$ pip3 install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.8.0-cp27-none-->
  >> linux_x86_64.whl

#Tensorflow GPU Linux
$ pip3 install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-gpu-1.8.0-cp27-->
  >> none-linux_x86_64.whl

#Numpy
$ pip install numpy

#Scipy
$ pip install scipy

#Matplotlib
$ pip install matplotlib

#Imgau
$ pip install imgau

#OpenCV
$ pip install opencv-python
```

C.5 CUDA a cuDNN

Aby bolo možné paralelizovať výpočty knižnice Tensorflow na grafickom procesore je nutná nielen inštalácia správnej (GPU) verzie knižnice ale tiež inštalácia dvoch softvérových rámcov od spoločnosti nVidia: **CUDA** a **cuDNN**.

Rámec CUDA je možné stiahnuť z webovej stránky spoločnosti nVidia:

<https://developer.nvidia.com/cuda-downloads>

Rámec cuDNN je tiež dostupný na stiahnutie na lokalite:

<https://developer.nvidia.com/cudnn>, kde je však nutná bezplatná registrácia.

Požadované verzie oboch rámcov sa líšia podľa verzie knižnice Tensorflow.

Pre presné požiadavky na verzie je nutné navštíviť webové stránky:

https://www.tensorflow.org/install/install_linux

Po nainštalovaní uvedených softvérových rámcov je nutné uloženie ciest k ich polohe v súborovom systéme do systémov premennej, kde ich očakáva knižnica Tensorflow:

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH:+${LD_LIBRARY_PATH}:}/usr/local/cuda/extras/CUPTI/lib64
```

C.6 Stiahnutie projektu

Samotné zdrojové súbory tohto bakalárskeho projektu sú dostupné na *GitHub* repozitári a je možné ich stiahnuť nasledujúcim príkazom:

```
$ git clone https://github.com/ms-martin/ms-fiit --bare
```

Poprípade sú taktiež dostupné na elektronickom médiu priloženom k tejto práci.

D Obsah elektronického média

```
├── datprep  
├── models  
├── training  
├── metrics  
├── results  
├── i-LIDS-VID  
└── martin_stano_79749_bc.pdf  
├── latex  
└── obsahMedia.pdf
```

Dataperp

Balík obsahujúci moduly na prípravu vstupných dát z datasetu i-LIDS-VID pre neurónové siete. Jednotlivé moduly sú vymenované a opísané v sekcií B.1.2.

Model

Tento balík obsahuje moduly na vytvorenie výpočtových grafov predstavujúcich neurónovú sieť v knižnici Tensorflow. Moduly a ich funkciaalita sú opísané v sekcií B.2.

Training

Balík, ktorý obsahuje moduly na tréning neurónových sietí a ukladanie ich váh. Moduly a ich funkciaalita sú opísané v sekcií B.3.

Metrics

Balík obsahujúci moduly pre evaluáciu netrénovaných modelov neurónových sietí. Moduly a ich funkciaalita sú opísané v sekcií B.4.

Results

Tento adresár združuje výsledky všetkých evaluačných procedúr spustených nad netrénovanými modelmi sietí. Výsledky sú buď v textovej alebo grafickej podobe a sú pomenované (poprípade sa nachádzajú v pod-adresári) na základe názvu modelu, ktorý bol vyhodnocovaný.

i-LIDS-VID

Adresár obsahujúci nemodifikovaný dataset i-LIDS-VID, tak ako je dostupný hneď po jeho stiahnutí. Z tohto adresára čerpajú dátu moduly v balíku *dataprep*. Bližšia štruktúra tohto súboru dát je opísaná v sekcií B.1.1

martin_stano_79749_bc.pdf

Dokument vo formáte *.pdf* obsahujúci text tejto práce vrátane príloh.

Latex

Adresár obsahujúci zdrojové súbory pre L^AT_EX na kompliaciu dokumentu *martin_stano_79749_bc.pdf*

obsahMedia.pdf

Obsah elektronického média vo forme PDF dokumentu.