

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

DETEKCIA CHODCOV
DIPLOMOVÁ PRÁCA

2020

Bc. PATRÍCIA PAGÁČOVÁ

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

DETEKCIA CHODCOV
DIPLOMOVÁ PRÁCA

Študijný program: Počítačová grafika a geometria
Študijný odbor: 1113 Matematika
Školiace pracovisko: Katedra aplikovej informatiky
Školiteľ: RNDr. Zuzana Černeková, PhD.



ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Patrícia Pagáčová

Študijný program: počítačová grafika a geometria (Jednoodborové štúdium, magisterský II. st., denná forma)

Študijný odbor: matematika

Typ záverečnej práce: diplomová

Jazyk záverečnej práce: slovenský

Sekundárny jazyk: anglický

Názov: Detektia chodcov
Pedestrian detection

Anotácia: Naštudovať problematiku sledovania cestnej premávky. Zamerať sa na detekciu a segmentáciu účastníkov cestnej premávky, najmä chodcov. Analyzovať existujúce riešenia publikované v dostupnej odbornej literatúre. Navrhnúť a implementovať metódu na detekciu chodcov na video záberoch nasnímaných z automobilu. Preskúmať prínosy pri využití hľbkovej informácie. Vyhodnotiť dosiahnuté výsledky.

Ciel: Naštudovať problematiku sledovania cestnej premávky. Zamerať sa na detekciu a segmentáciu účastníkov cestnej premávky, najmä chodcov. Analyzovať existujúce riešenia publikované v dostupnej odbornej literatúre. Navrhnúť a implementovať metódu na detekciu chodcov na video záberoch nasnímaných z automobilu. Preskúmať prínosy pri využití hľbkovej informácie. Vyhodnotiť dosiahnuté výsledky.

Vedúci: RNDr. Zuzana Černeková, PhD.

Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 13.10.2017

Dátum schválenia: 14.12.2018

prof. RNDr. Július Korbaš, CSc.

garant študijného programu

.....
študent

.....
vedúci práce

Podakovanie: Chcela by som sa podakovať RNDr. Zuzane Černekovej, PhD. za vynikajúce vedenie diplomovej práce, za jej cenné pripomienky, čas a odbornú pomoc pri tejto diplomovej práci.

Abstrakt

Táto práca sa zaobrá detekciou chodcov v obraze, založenej na strojovom učení, za využitia hlbokých neurónových sietí. Na tento účel sa v práci využíva Object Detection API poskytované rámcom TensorFlow. Tento rámec je vyvíjaný spoločnosťou Google a poskytuje podporu pre riešenie širokej škály úloh z oblasti umelej inteligencie. Práca popisuje postupnosť krokov potrebných na spustenie procesu trénovania detekčného modelu v prostredí TensorFlow. Za pomoc tohto postupu vykonáva viacero trénovaní s použitím rôznych kombinácií vstupných parametrov. Priebehy týchto trénovaní sú zanalyzované nástrojom TensorBoard. Google poskytuje niekoľko predtrénovaných modelov, ktoré umožňujú rýchlosť a presnosť detektie chodcov všetkých troch modelov experimentálne vyhodnotíme na pripravenej sade obrazových vstupov, zachytávajúcej široké spektrum podmienok, v ktorých sa chodci môžu vyskytovať.

Kľúčové slová: detekcia chodcov, neurónové siete, TensorFlow

Abstract

This work deals with the detection of pedestrians in the image, based on machine learning using deep neural networks. For this purpose, the work uses the Object Detection API provided by TensorFlow framework. This framework is developed by Google and provides support for a wide range of artificial intelligence tasks. The work describes the sequence of steps needed to start the process of training the detection model in the TensorFlow environment. Using this procedure, it performs several trainings using different combinations of input parameters. Runs of these trainings are analyzed by TensorBoard tool. Google provides several pre-trained models that allow rapid acceleration of training. We will use three of them in our work. We will train two non-detecting pedestrians and the third one, supporting pedestrian detection, we will use unchanged. The speed and accuracy of pedestrian detection of all three models will be experimentally evaluated on a prepared set of images inputs, capturing a wide range of conditions in which pedestrians may occur.

Keywords: pedestrian detection, neural network, TensorFlow

Obsah

Úvod	1
1 Detekcia chodcov a jej využitie	2
1.1 Proces detektie objektov	3
1.1.1 Hĺbka obrázka	3
1.2 Metódy detektie chodcov	4
1.2.1 Metódy založené na príznakoch	5
1.2.2 Hlboké metódy	6
1.2.3 Klasifikátory	7
1.2.4 Detekcia chodcov pomocou YOLO	8
1.2.5 Detekcia chodcov pomocou Faster R-CNN	8
1.2.6 Detekcia chodcov s RGBD obrázkami	9
1.3 Existujúce architektúry neurónových sietí	9
1.3.1 VGG	9
1.3.2 GoogLeNet	10
1.3.3 ResNet	10
1.3.4 SENet	11
1.3.5 YOLO (You Only Look Once)	11
1.3.6 RetinaNet	13
1.3.7 SSD (Single Shot Detector)	13
1.3.8 Rodina R-CNN	14
1.3.9 Porovnanie architektúr	17
2 Neurónové siete a práca s nimi	20
2.1 Neurónové siete	20
2.1.1 Model neurónu	21
2.1.2 Učenie	22
2.1.3 Aktivačná funkcia	22
2.1.4 Stratová funkcia (Loss function)	23
2.1.5 Pretrénovanie a podtrénovanie	23
2.2 Konvolučné neurónové siete	24

2.3	Softvérové rámce pre hlboké učenie	24
2.4	Datasetsy	26
2.4.1	KITTI dataset	26
2.4.2	COCO dataset	26
2.4.3	GRAZ-01 dataset	27
3	Praktická časť	28
3.1	Transfer learning	28
3.2	TensorFlow Object Detection API	29
3.3	Inštalácie a konfigurácie	29
3.4	Príprava na trénovanie	30
3.4.1	Príprava trénovacích a validačných dát	30
3.4.2	Vytvorenie zoznamu tried	32
3.5	Trénovanie	32
3.5.1	Výber modelu pre trénovanie	33
3.5.2	Nastavenie krokov trénovania	34
3.5.3	Nastavenie konfiguračných súborov pre rôzne modely	34
3.5.4	Trénovanie modelu	35
3.5.5	Grafy celkovej straty počas trénovania	36
3.6	Exportovanie inferenčného grafu	40
4	Vyhodnotenie a porovnanie modelov	41
4.1	Metriky	41
4.1.1	Intersection Over Union - IoU	42
4.1.2	Presnosť (Precision)	43
4.1.3	Úplnosť (Recall)	43
4.1.4	Precision-Recall krivky	43
4.1.5	Average precision - AP	43
4.2	Porovnanie modelov	44
4.2.1	Testovací dataset	44
4.2.2	Modely s rôznymi konfiguračnými nastaveniami	44
4.2.3	Porovnanie modelov s modelom Kittti	47
4.2.4	Porovnanie modelov z hľadiska rýchlosťi	47
4.2.5	Celkové vyhodnotenie	48
4.2.6	Vizualizácia detekcií	48
Záver		50
Príloha		57

Zoznam obrázkov

1.1	Základný refazec operácií systému detekcie objektov	3
1.2	Príklad hĺbkovej mapy	4
1.3	Architektúra vylepšenej Faster R-CNN	9
1.4	Architektúra VGG-16	10
1.5	Reziduálny blok	11
1.6	Základný princíp YOLO	12
1.7	Architektúra YOLO	12
1.8	Architektúra RetinaNet	13
1.9	Architektúra SSD	14
1.10	Architektúra R-CNN	15
1.11	Architektúra Fast R-CNN	16
1.12	Architektúra Faster R-CNN	16
1.13	Schéma R-CNN	17
1.14	Presnosť klasifikačných modelov konvolučných neurónových sietí na datasete ImageNet	17
1.15	Porovnanie modelov na detekciu objektov	18
1.16	Porovnanie modelov z hľadiska presnosti na datasete PASCAL VOC . .	19
1.17	Porovnanie modelov z hľadiska rýchlosťi	19
1.18	Porovnanie modelov z hľadiska presnosti na datasete COCO	19
2.1	Topológia viacvrstvovej neurónovej siete	20
2.2	Matematický model neurónu	21
2.3	Porovnanie distribúcií inštancií tried datasetov COCO a PASCAL VOC	27
3.1	Screenshot počas používania programu LabelImg	31
3.2	Zoznam tried	32
3.3	Vybrané predtrénované modely	33
3.4	Príklady grafov v Tensorboarde	35
3.5	Príklad modelu, ktorý sa neučí	36
3.6	Graf stratovej funkcie v originálnom nastavení	37
3.7	Graf celkovej straty po zmene kroku učenia	37

3.8	Graf celkovej straty po zmene kroku učenia a pridaní augmentácie dát	37
3.9	Graf celkovej straty s optimalizátorom Adam	38
3.10	Graf celkovej straty v originálnom nastavení	39
3.11	Graf celkovej straty po zmene kroku učenia	39
3.12	Graf celkovej straty po zmene kroku učenia a pridaní augmentácie dát	39
3.13	Graf celkovej straty s optimalizátorom Adam	40
4.1	Vzorec Intersection over union	42
4.2	Príklad detekcie chodca	42
4.3	Tabuľka výsledkov presnosti pri rôznych nastaveniach modelov	45
4.4	Tabuľka výsledkov presnosti pri rôznych optimalizátoroch	46
4.5	Tabuľka výsledkov presnosti pri rôznych nastaveniach modelov	46
4.6	Tabuľka výsledkov presnosti pri rôznych optimalizátoroch	47
4.7	Tabuľka výsledkov presnosti	47
4.8	Porovnanie rýchlosťí modelov	47
4.9	Výsledná detekcia jednotlivých modelov. Reálny ohraničujúci box - červená a predikovaný ohraničujúci box - zelená, žltá	49

Úvod

Automobilový priemysel sa v dnešnej dobe zaraďuje medzi najdynamickejšie sa rozvíjajúce odvetvie. Stále viac a viac do neho prenikajú pokročilé technológie ako umelá inteligencia a počítačové videnie. Ich hlavnou úlohou je zvyšovanie úrovne bezpečnosti cestnej dopravy a postupný prechod k autonómnym vozidlám. Práve otázka bezpečnosti je jedným z hnciacich motorov rozvoja širokého spektra algoritmov na detekciu rôznych objektov, medzi ktoré samozrejme patria aj najzraniteľnejší účastníci cestnej premávky - chodci. Tento fakt radí detekciu chodcov medzi najatraktívnejšie témy súčasnej doby.

Táto práca v úvode opisuje rôzne metódy, ktoré sa v súčasnosti využívajú na detekciu chodcov. Najpoužívanejšou z nich sú neurónové siete, a práve preto detailne rozoberá a porovnáva práve jej existujúce architektúry. Je tu uvedených aj niekoľko ďalších prác, zaobrajúcich sa detekciou chodcov pomocou týchto architektúr.

Druhá kapitola sa už venuje samotnému princípu neurónových sietí. Rozoberajú sa v nej viaceré najznámejšie softvérové rámce pre hlboké učenie. Z nich je zvolený rámc Tensorflow, vyvíjaný spoločnosťou Google, ktorý sa následne využije na implementáciu finálneho riešenia. Na účely experimentovania a overovania vytvoreného riešenia sú nevyhnutné vhodné datasety, ktoré sú detailne popísané práve v tejto časti.

Posledné dve kapitoly tvoria jadro tejto práce. Prvá z nich sa venuje postupu trénovania modelov. Popisuje metódu trénovania, rámc, ktorý sa pri trénovaní využíva a postupnosť krokov, ktoré je treba vykonať pre spustenie trénovania. Kapitola sa ďalej venuje procesu predprípravy dát a samotnému trénovaniu modelov. Priebehy trénovania jednotlivých modelov sú zanalyzované a znázornené na grafoch, automaticky vygenerovaných počas procesu trénovania.

Záverečná kapitola práce obsahuje vyhodnotenie a porovnanie jednotlivých modelov. Sú tu popísané metriky a testovací dataset, ktoré boli na toto vyhodnotenie použité. Nachádzajú sa tu výsledky porovnania modelov z hľadiska rýchlosť a presnosti detegovania chodcov. Kapitola taktiež obsahuje sadu obrázkov, ktoré graficky demonštrujú schopnosti detekcie natrénovaných modelov.

Kapitola 1

Detekcia chodcov a jej využitie

Detekcia objektov je počítačová technológia z oblasti počítačového videnia a spracovania obrazu. Rozumieme pod ňou určenie hraníc a polohy objektu v obraze. Zaoberá sa rozpoznávaním objektov v digitálnych fotografiách či videách a ich následným zaradením do vopred definovaných kategórií.

Detekcia objektov zažíva v poslednej dobe veľký rozmach a stala sa dôležitou časťou v oblasti výskumu počítačového videnia. Dá sa nájsť veľké množstvo konkrétnych využití detekcie objektov. V oblasti medicíny je to napríklad detekcia tumorov, v priemyselných aplikáciach vizuálna inšpekcia výrobkov. Na sociálnych sietach je implementovaná detekcia tvári používateľov na fotografiách.

Ďalšia oblasť využitia detekcie objektov je sledovanie cestnej premávky. Detekcia účastníkov cestnej premávky je užitočná v mnohých aplikáciach, ako sú inteligentné mestá či autonómna doprava. Napríklad vozidlá s vlastným riadením musia zdetectovať objekty, ako sú ľudia a iní, aby mohli bezpečne jazdiť. Bezpečnosť je jednou z hlavných motivácií pri implementácii vozidla s vlastným riadením, pretože autonómne vozidlá predstavujú vysoké riziko pre každú osobu, ktorá môže byť v ich dosahu. Dnešné moderné autá majú zabudovaný systém detekcie chodcov pred vozidlom s výstražným znamením alebo s autonómnym brzdením, čo výrazne zvyšuje bezpečnosť chodcov v premávke. Z dôvodu potenciálu zlepšenia bezpečnostných systémov a autonómnych systémov riadenia predstavuje detekcia chodcov v posledných rokoch predmet intenzívneho rozvoja a výrazného pokroku.

Detekcia chodcov je problémom s dlhou históriaou. Už od roku 1990 sa ukazuje stále väčší záujem o detekciu chodcov. Vďaka rozsiahlemu výskumu v oblasti detekcie chodcov, nedávne štúdie stále ukazujú významný pokrok, avšak aby prekonali úroveň, ktorú dosahuje človek je potrebné dosiahnuť ešte veľmi veľké zlepšenie. Človek sa dokáže naučiť omnoho viac z omnoho menšieho počtu príkladov, pretože využíva model, ktorý si buduje od narodenia.

1.1 Proces detekcie objektov

Proces detekcie objektov sa bežne delí do niekoľko fáz. Za prvý krok môže byť považované získanie signálu obrazu. Teda zaobstaranie vstupných digitálnych obrazových farebných alebo čiernobielych dát fotoaparátom alebo videokamerou. Získané dáta sú následne predspracované úpravami. Napríklad zvýraznenie hrán objektov, zmena rozlíšenia, úprava jasu, kontrastu a farby alebo odstránenie nežiadúcich javov, ako je napríklad šum.

Cieľom ďalšieho kroku je získanie príznakov. Sú to vybrané miesta, kde by sa potenciálne mohol vyskytovať hľadaný objekt. Najjednoduchšie metódy pracujú s tzv. klízavým oknom, ktorý postupne prechádza celým obrazom. PomySELNÉ výrezy obrazu sú odovzdávané klasifikátoru a ten má rozhodnúť či sa tam daný objekt nachádza alebo nie. Klasifikátor sa rozhoduje na základe vstupného vektora príznakov. Príznaky slúžia ako opis samotného objektu.

Posledným krokom je klasifikácia objektu. V tomto kroku klasifikátor rozhodne na základe vektora príznakov, či sa jedná o objekt z vopred známych kategórií (tried).

Vo všeobecnosti je možné proces detekcie objektov znázorniť na základnom reťazci operácií, ktorý je zobrazený na obrázku 1.1.



Obr. 1.1: Základný reťazec operácií systému detekcie objektov [36]

1.1.1 Hĺbka obrázka

Vstupné dáta pre spracovanie obrazu nemusia byť striktne iba farebné či čiernobiele. K štandardným RGB dátam sa môže pridať aj hĺbkový kanál, ktorý reprezentuje, akú vzdialenosť majú jednotlivé objekty od roviny obrázka. Hĺbkový kanál obrázka sa dá vizualizovať pomocou šedotónového či farebného zobrazenia, kde každá intenzita alebo farba znázorňuje rozdielnú vzdialenosť. Príklad takéhoto zobrazenia je obrázok 1.2 (b). V literatúre sa tiež označuje ako tzv. hĺbková mapa.

Pri detekcii objektov môže byť hĺbková mapa veľkým prínosom. Jedným z nich môže byť zlepšenie presnosti detektora pri rôznych svetelných podmienkach. Nakoľko sa hĺbkové dáta vzhľadom na rôzne osvetlenie nemenia, detektoru umožňujú využiť priestorové rozdelenie medzi objektami. Taktiež detektoru poskytuje jednoduchšie zobrazenie prostredia pomocou základných siluet predmetov. Prínosmi hĺbkovej informácie pri detekcii chodcov sa zaoberajú aj autori článku [27], ktorý rozoberáme v časti 1.2.6.



Obr. 1.2: Príklad hĺbkovej mapy. (a) pôvodný obrázok, (b) hĺbková mapa [41]

1.2 Metódy detekcie chodcov

V posledných rokoch bolo vyvinutých a otestovaných veľké množstvo algoritmov na detekciu chodcov. Počiatočný prístup na detekciu pohybujúcich sa objektov spočíval v odčítaní pozadia (Background subtraction). Táto technika umožňuje detekciu objektov vo vnútri scény pomocou vhodného modelu pozadia. Algoritmy založené na odčítaní pozadia sa implementujú pomerne jednoducho, avšak tento prístup nie je odolný voči dynamickému pozadiu, variabilite osvetlenia alebo šumu, čo obmedzuje jeho využitie.

V roku 2001 prišli Paul Viola a Michael Jones s algoritmom Viola-Jones [39]. Algoritmus bol primárne navrhnutý pre detekciu tvári osôb v reálnom čase, možno ho však upraviť pre detekciu rôznych objektov. V roku 2005 Navneet Dalal a Bill Triggs predstavili algoritmus Histogram orientovaných gradientov [7]. Použili ho pre detekciu chodcov v statických obrazoch. V roku 2006 sa Gándhí a Trivedi riadili rovnakým prístupom, ale zamerali sa na aspekty predikcie zrážok a analýzy správania chodcov [12]. Metódu Integral Channel Features, známu ako ChnFtrs, využil vo svojej práci Piotr Dollár [8]. Prvýkrát bola opísaná v roku 2009. Veľmi vela algoritmov na detekciu chodcov bolo uverejnených v roku 2013. Sú to napríklad Multi-Order Contextual co-Occurrence (MOCO) [6], Roerei [5] a mnoho ďalších.

Väčšina existujúcich prístupov na detekciu chodcov je založená na metódach Viola-Jones, na Histogramoch orientovaných gradientov (HOG), na Haarových príznakoch či na metóde lokálneho binárneho vzoru. Sú to metódy založené na príznakoch. V sekcií 1.2.1 je metóda Viola-Jones a metóda lokálneho binárneho vzoru opísaná detailnejšie.

Z hľadiska klasifikátorov existuje niekoľko klasifikačných algoritmov používaných na detekciu chodcov. Mnohé práce používajú jednoduché klasifikátory. Väčšina klasifikačných algoritmov spočíva v kontrolovanom prístupe, ako sú napríklad Support Vector Machine (SVM) (viď. sekcia 1.2.3) alebo AdaBoost (viď. sekcia 1.2.3).

1.2.1 Metódy založené na príznakoch

Metóda Viola-Jones

Aplikácia, ktorú Viola a Jones predstavili, bola schopná v reálnom čase detegovať tváre na videu z webkamery. Algoritmus využíva to, že ľudské tváre majú podobné rysy. Napríklad oblast okolo očí je tmavšia ako líce a nosná prepážka je svetlejšia ako oblast očí. Pre detekciu týchto znakov ľudskej tváre používa Haarove príznaky. Vo vstupnom obrazze pomocou aplikácie týchto príznakov algoritmus deteguje polohu očí, nosa a úst. Následne spočíta ich vzájomnú polohu a vzdialenosť. Potom ich porovná zo vzorovými metrikami. Pod vzorovými metrikami myslíme, ako sú na tvári priemerne vzdialené oči, nos a ústa.

V roku 2003 Viola, Jones a kol. použili tento svoj algoritmus na úlohu detektie chodcov [40]. Tento systém detektie chodcov integruje informácie o intenzite obrazu s informáciami o pohybe. Autori kombinovali najmä Haarove príznaky s pohybovými informáciami, ktoré boli vypočítané vzhľadom na dva po sebe idúce snímky vo video sekvencií. Klasifikácia bola uskutočnená vzhľadom na sekvenčiu klasifikátorov AdaBoost.

Metóda lokálneho binárneho vzoru

Metóda LBP (Local Binary Pattern) bola navrhnutá pre klasifikáciu textúr v obrazoch v roku 1990. Jej základný princíp spočíva v zistení intenzity jasu v okolí vytýčenom okolo stredového pixelu. Hodnota jasu stredového pixelu slúži na prahovanie okolitých hodnôt.

Hodnota stredového pixelu je porovnaná s hodnotami každých jeho ôsmich susedov. Pokiaľ je hodnota stredového pixelu rovná alebo väčšia, napíše sa na túto pozícii jednotka, v opačnom prípade sa napíše nula. Tieto hodnoty sa zoradia v smere chodu hodinových ručičiek alebo naopak. Takto sa získa 8-ciferné binárne číslo, ktoré sa pre zjednodušenie prevedie do dekadickej sústavy. Následne z čísel, ktoré sme získali kombináciou pixelov v bunkách, vypočítame histogram. Zretezením všetkých histogramov buniek získame $2^8 = 256$ -rozmerný vektor príznakov. Matematicky sa hodnota LBP kódu pixelu zo súradnicami na obrázku (x_c, y_c) môže vyjadriť ako:

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c) 2^p, \quad (1.1)$$

$$s(x) = \begin{cases} 1, & \text{ak } x \geq 0 \\ 0, & \text{ak inak,} \end{cases} \quad (1.2)$$

kde P znamená počet vzorkovaných bodov kruhového okolia s polomerom R , g_p je hodnota bodu kruhového okolia a g_c je hodnota bodu na súradničiach x_c, y_c a $s(x)$ je

funkcia prahovania.

Prístup LBP sa stal veľmi populárnym vďaka svojej odolnosti voči rôznym osvetleniam, vďaka rýchlosťi a jednoduchému výpočtu. Príznakové vektory LBP sa veľmi často používajú v kombinácii s funkciami HOG na dosiahnutie vyššieho výkonu pri detekcii chodcov.

1.2.2 Hlboké metódy

Najnovšie výskumy v oblasti umelej inteligencie viedli k rozšíreniu moderných techník strojového učenia založených na hlbkových štruktúrach. Hlboké učenie je súčasťou širokej skupiny používaných pre strojové učenie.

Hlboké učenie je trieda algoritmov strojového učenia, ktoré používa viac vrstiev na postupné získavanie prvkov vyššej úrovne z prvotného vstupu. Napríklad pri spracovaní obrazu môžu spodné vrstvy identifikovať hrany, zatiaľ čo vyššie vrstvy môžu identifikovať pojmy relevantné pre človeka, ako sú čísllice, písmená alebo tváre. Pri hlbokom učení sa počítačový model učí vykonávať klasifikačné úlohy priamo z obrázkov, textu alebo zvuku.

Taktiež umožňuje odstránenie ručnej identifikácie príznakov z dát. Získanie príznakov však nie je úplne odstránené z pracovného postupu, ale je to automatický postup vykonávaný hlbkovým klasifikátorom. Začlenenie získania prvkov do klasifikačného procesu, umožňuje rýchlejšie vykonávanie za behu programu.

Modely sú trénované pomocou veľkej sady označených dát a architektúr neurónových sietí, ktoré obsahujú veľa vrstiev.

Neurónové siete

Neurónové siete (ANN – Artificial Neural Network), sú inšpirované ľudským mozgom, ktorý je zložený z rôznych vzájomne prepojených neurónov (uzlov). Medzi týmito uzlami sa posielajú informácie. Každý z uzlov príma informáciu z predchádzajúceho, spracováva ju a následne ju vysle do ďalšieho uzlu, pokial nie je priyatý konečný výstup.

Nápad algoritmu vznikol už v 40. rokoch 20. storočia a bol založený na biologickom neuróne. Nápad sa však zrealizovať v 70. a 80. rokoch nepodarilo, pretože v týchto rokoch boli technológie na trénovanie neurónových sietí nedostačujúce. V 90. rokoch prišli prvé úspechy neurónových sietí, keď Yann LeCun prišiel s rozpoznávaním ručne napísaných číslíc [21]. V roku 1958 Frank Rosenblatt [32] ukázal, že neurónové siete sa dajú natrénovať tak, aby vedeli rozpoznávať a klasifikovať objekty.

Neurónové siete v oblasti umelej začali dosahovať výsledky, ktoré boli pred pár rokmi ešte nepredstaviteľné a to vďaka vývoju výpočtového výkonu počítačov či dostupnosti veľkého množstva dát. V oblasti umelej inteligencie ponúkajú najlepšie riešenie.

1.2.3 Klasifikátory

Algoritmy pre detekciu chodcov možno rozdeliť podľa toho, akým spôsobom sú im nastavené príznaky. Algoritmy sa snažia v dátach hľadať tieto príznaky a na ich základe vykonávať klasifikáciu. Prvým typom sú klasifikátory, ktoré majú príznaky predprogramované. Druhým typom sú klasifikátory, ktorých príznaky sú dynamicky upravované trénovaním. Po naprogramovaní ešte nie sú schopné klasifikovať žiadne dátu. Túto schopnosť získajú až po procese natrénovania na trénovacích dátach.

Klasifikátor SVM

Support Vector Machine je metóda statického rozpoznávania vzorov. Cieľom je nájsť nadrovinu, ktorá oddeľuje s najväčšou vzdialenosťou skupinu bodov reprezentujúcich rozdielne vzdialenosťi. Inými slovami hľadá nadrovinu čo najviac vzdialenú od trénovacích dát a tým zaistuje malý počet chybne rozpoznaných objektov. Trénovacie dáta sa skladajú z pozitívnej a negatívnej trénovacej sady. Pozitívna sada obsahuje obrázky s objektami, ktoré chceme rozpoznať a negatívna obsahuje obrázky, kde nie sú objekty, ktoré chceme rozpoznať.

Support vector machine je v preklade mechanizmus podporných vektorov. Vzorky každej triedy, ktoré sú najbližšie k rozdeľujúcej nadrovine sú potom podpornými vektormi. Rozpäťím nazývame vzdialosť medzi podpornými vektormi rôznych tried. Rozdeľujúca nadrovinu je lineárna funkcia v priestore príznakov. Rovnica nadroviny má tvar:

$$w^T x + b = 0. \quad (1.3)$$

Výstupom tejto metódy je klasifikačný model, ktorý v sebe obsahuje váhy jednotlivých podporných vektorov. Vo fáze detekcie je následne tento model aplikovaný pre ohodnotenie vstupného obrazu.

Nevýhodou tejto základnej metódy je, že funguje iba v prípadoch, kedy je množina dát lineárne separovateľná. Tento problém rieši nelineárny SVM klasifikátor a SVM s voľným rozpäťím.

Klasifikátor AdaBoost

AdaBoost, teda Adaptive Boosting, bol predstavený v práci od Freund a Schapire [11]. Tento algoritmus dokáže lineárnu kombináciou jednoduchých klasifikátorov vytvoriť nelineárny klasifikátor. Algoritmus vytvorí slabý klasifikátor, ktorý minimalizuje chybu na trénovacích dátach. Tieto dáta sú vážené pomocou váhovej funkcie. Pridelením správnej váhy na konci trénovania docielime ku klasifikátoru s dobrou presnosťou. Hodnota tejto váhy je závislá na chybe klasifikátora cez trénovaciu množinu.

Klasifikátory, ktoré majú klasifikačnú presnosť menšiu ako 50 percent, sú ohodnotené, zápornou váhou. Prínosné sú len tie klasifikátory, ktoré majú presnosť väčšiu než 50 percent. Vtedy môžme hovoriť o zosilnení klasifikácie.

1.2.4 Detekcia chodcov pomocou YOLO

Autori článku [20] riešia problém detektie chodcov pomocou modifikovanej architektúry YOLOv2. Túto novo navrhnutú sietovú štruktúru nazvali YOLO-R.

Modifikácia spočíva v tom, že do pôvodnej siete YOLOv2 sa pridali tri prechodové vrstvy. Ich úlohou je prepojiť plytkú vrstvu (shallow layer) pre typické charakteristiky chodcov s hlbokou vrstvou (deep layer) pre charakteristiky chodcov. Na zlepšenie schopnosti siete extrahovať informácie o plytkých charakteristikách chodcov, sa mení poradie prechodovej vrstvy v pôvodnom YOLOv2 zo 12. vrstvy na 12. vrstvu.

V tomto článku autori použili dataset INRA, kde tréningová sada obsahuje 614 vzoriek vrátane 2416 chodcov a trénovacia sada obsahuje 288 vzoriek vrátane 1126 chodcov.

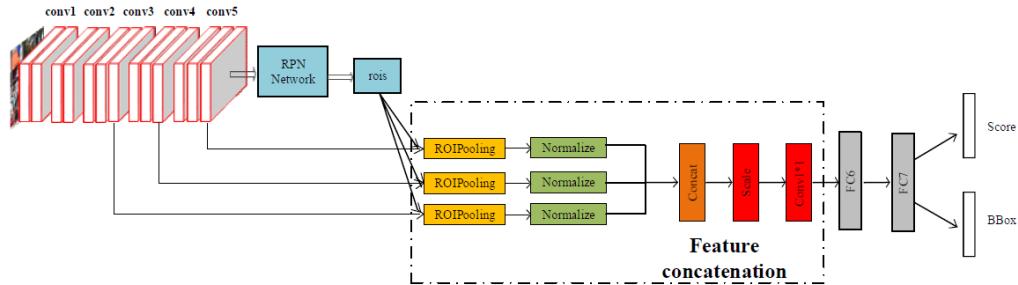
Výsledky porovnávali s pôvodnou YOLOv2. Ukázalo sa, že táto metóda môže zlepšiť presnosť detektie chodcov a zároveň znížiť mieru nesprávnej detektie tam, kde sa žiadnen chodec nenachádza. YOLO-R spĺňa požiadavky na výkon v reálnom čase, pretože počet zdetegovaných snímok môže dosiahnuť až 25 snímok za sekundu.

1.2.5 Detekcia chodcov pomocou Faster R-CNN

V článku [43] sa autori zaoberajú detekciou chodcov v komplikovaných scénach. Na úlohu detektie chodcov navrhujú novú schému pomocou vylepšenia architektúry Faster R-CNN. Na zvýšenie výkonnosti v zložitých scénach vylepšili pôvodnú Faster R-CNN kombináciou zretazenia určitých prvkov a zložitých negatívnych stratégií.

Autori zistili že pôvodná architektúra Fast R-CNN môže vynechávať niektoré dôležité znaky obrázku. Taktiež postrehli, že plytká sieť extrahuje prvky textúry a detailov, ktoré sú schopné extrahovať klúčové prvky, zatiaľ čo hlboká sieť extrahuje obrys tvarov a výrazné prvky, ktoré dodávajú reprezentatívnosť. Podľa charakteristík jednotlivých prvkov kombinujú nízkoúrovňové a vysokoúrovňové prvky do viacerých konvolučných vrstiev. Po združení a normalizovaní sa výsledné prvky zretazia. Na obrázku 1.3 je zobrazená podrobná architektúra, ktorú autori navrhli. Myšlienka zložitých negatívnych stratégií spočíva v tom, že klasifikujú vzorky pomocou klasifikátorov a pri ďalšom trénovaní klasifikátora použijú nesprávne klasifikované vzorky ako negatívne vzorky.

Navrhovaný model bol trénovaný na datasete Daimer a testovaný na datasetoch Caltech a INRA. Výsledky naznačujú, že navrhovaná vylepšená Faster RCNN prekonáva najmodernejsie metódy.



Obr. 1.3: Architektúra vylepšenej Faster R-CNN [43]

1.2.6 Detekcia chodcov s RGBD obrázkami

Autori článku [27] skúmajú využitie hĺbkovej informácie obrázka pri detekcii chodcov. K pôvodným RGB informáciám pridali aj informáciu o hĺbke a tak im vznikli 4-kanálové obrázky, ktoré nazývajú ako RGBD obrázky. Tie potom použili ako vstupné dátá pri ich práci. Skúmali aj spojenie RGB a hĺbkovej informácie pri rôznych vrstvách siete YOLOv2.

Ich zistenie bolo, že hĺbka pridáva cenné informácie pre detekčnú sieť a v porovnaní s klasickými RGB detektormi zvyšuje presnosť detekcie. Taktiež zlepšuje detekciu jednotlivcov v dave, čo znamená, že presnejšie deteguje aj čiastočne prekrytých chodcov.

1.3 Existujúce architektúry neurónových sietí

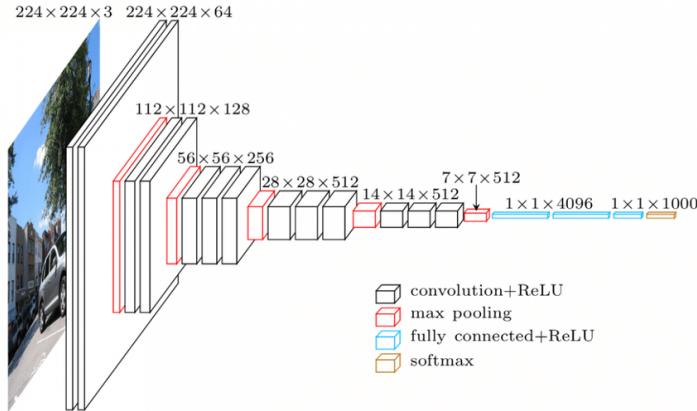
Táto podkapitola sa zaobráva rôznymi architektúrami konvolučných neurónových sietí (CNN), ktorých je v súčasnej dobe veľmi veľké množstvo. Preto budú opísané iba najznámejšie a najzaujímavejšie z nich. Koniec podkapitoly sa venuje porovnávaniu presnosťí týchto architektúr.

Existujúce architektúry neurónových sietí sa dajú rozdeliť do dvoch skupín. Prvou skupinou sú modely na klasifikáciu obrázkov a druhou sú modely na detekciu objektov. Klasifikačné modely sa používajú v modeloch na detekciu objektov ako extraktory príznakov. Najznámejšie sú modely VGG, GoogleNet, ResNet, Inception a mnoho ďalších. Medzi modely na detekciu objektov patria YOLO, RetinaNet, SDD či rodina sietí R-CNN.

1.3.1 VGG

VGG je model konvolučnej neurónovej siete, ktorý navrhli K. Simonyan a A. Zisserman z Oxfordskej univerzity [37]. V ich práci skúmali, aký vplyv má hĺbka modelu na jeho výslednú presnosť. Výsledkom toho bolo zistenie, že výrazné zlepšenie prichádza s hĺbkou 16 a 19 vrstiev. Inovatívne bolo však aj to, že mali neobvykle jednoduchú

architektúru, ktorú tvorili výhradne konvolučné filtre s veľkosťou 3×3 . VGG používa malé filtre kvôli menším parametrom a namiesto použitia väčších filtrov si ukladá viac parametrov. Ukázalo sa, že sekvencia menších filtrov môže efektívne nahradieť väčšie filtre. Touto architektúrou sa inšpirovala aj sieť ResNet. Architektúra VGG so 16 vrstvami je znázornená na obrázku 1.4.



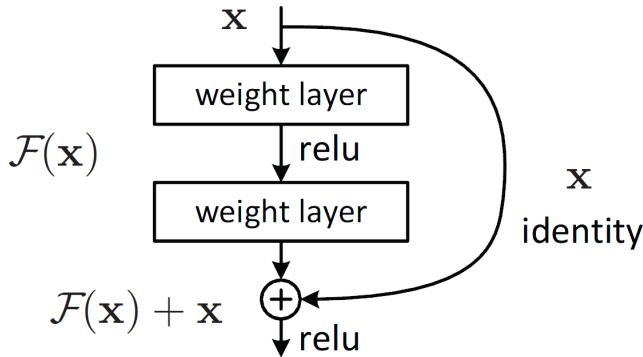
Obr. 1.4: Architektúra VGG-16 [25]

1.3.2 GoogLeNet

GoogLeNet je hlboká konvolučná neurónová sieť, ktorá má až 22 vrstiev. Článok [38] publikoval Christian Szegedy a kol. Hlavná myšlienka architektúry je založená na zistení, ako možno aproximovať optimálnu lokálnu riedku štruktúru v konvolučnej sieti a pokryť ju ľahko dostupnými hustými komponentmi. Táto architektúra prináša inovačný prvok nazývaný počiatočný modul, ktorý je založený na niekoľkých veľmi malých zvinutiach, aby sa znížil počet parametrov. Tieto počiatočné moduly sú usporiadane nad sebou. To znamená, že vlastnosti vyššej abstrakcie sú zachytávané vyššími vrstvami, preto sa očakáva, že sa zníži ich priestorová koncentrácia.

1.3.3 ResNet

Architektúru reziduálnych neurónových sietí, skrátene ResNet, navrhli autori Kaiming He a kol. z výskumného tímu Microsoftu [16]. Od roku 2015 sa stala veľmi populárnoch, keď zvíťazila na súťaži ILSVRC [33]. Táto architektúra má rôzne variácie, ktoré sa líšia počtom vrstiev a to 34, 50, 101 alebo až 152. Avšak pri zvyšovaní vrstiev vznikajú problémy, ako degradácia presnosti pri tréningu či strata schopnosti učiť sa. Tieto problémy rieši architektúra ResNet prostredníctvom tzv. reziduálnych blokov (residual block). Reziduálny blok, ktorý je znázornený na obrázku 1.5, je jej základným stavebným prvkom.



Obr. 1.5: Reziduálny blok [16]

1.3.4 SENet

Squeeze and Excitation Network v skratke SENet má chybovost len 2.251% a preto ju možno považovať za najmodernejšiu architektúru v rámci klasifikácie obrazov. Táto architektúra je podobná sieti ResNet, ale autori článku [17] ju rozširujú o tzv. Squeeze-and-Excitation blok. Účelom tohto bloku je ohodnotiť každý kanál príznakových máp podľa toho, ako veľmi je prínosný. Taktiež tieto bloky prinášajú významne zlepšenia výkonu už existujúcich súčasných CNN pri nepatrnych dodatočných výpočtových nákladoch.

Squeeze and Excitation Network zvíťazila v roku 2017 na súťaži ILSVRC, pričom prekonala víťazný model z roku 2016 relatívnym zlepšením až o približne 25%.

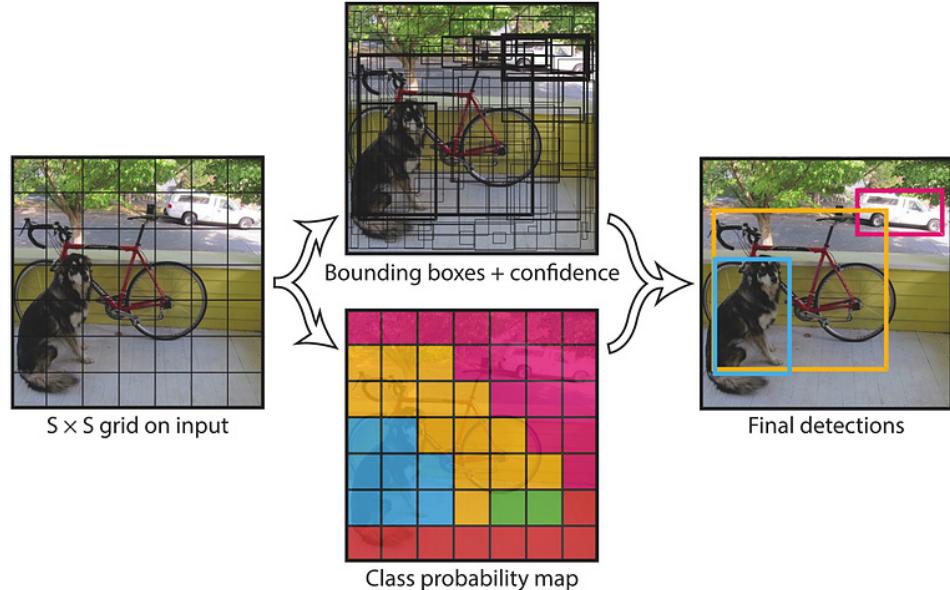
1.3.5 YOLO (You Only Look Once)

YOLO je jeden z najmodernejších systémov na detekciu objektov. Tento detektor sa stále inovuje. V súčasnosti existujú 3 verzie – YOLO, YOLOv2 a najnovšia YOLOv3. Tento detektor pôvodne publikovali Joseph Redmon, Santosh Divvala, Ross Girshick a Ali Farhadi.

Predchádzajúce detekčné systémy opäťovne používajú klasifikátory na vykonanie detekcie. Model aplikujú na obrázok na viacerých miestach a v rôznych mierkach. Potom za detekciu považujú oblasti obrazu s vysokým skóre.

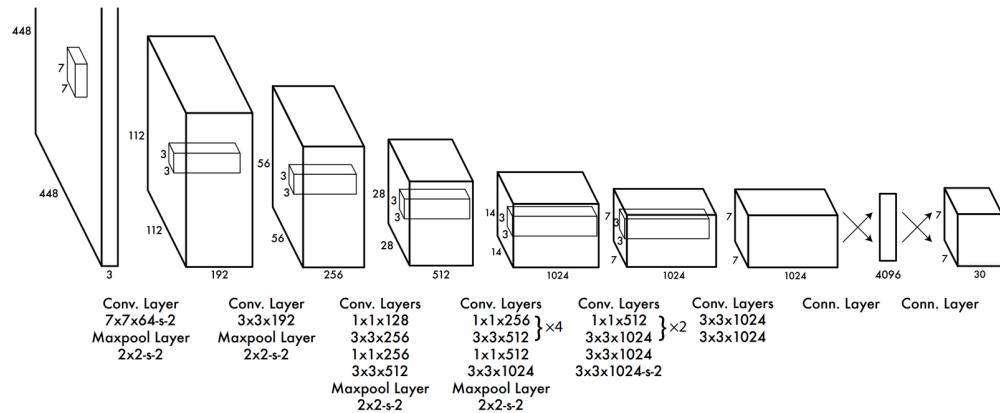
Autori predstavili v [29] úplne nový prístup k detekcii chodcov. Na rozdiel od predchádzajúcich prístupov na celý obrázok aplikovali jednu neurónovú sieť. Táto sieť rozdeľuje obraz na oblasti. Ak stred objektu spadne do určitej oblasti, tak táto oblast zodpovedá za detekciu tohto objektu. Každá oblasť definuje ohraničovacie boxy a skóre pre každú triedu. Ak sa v oblasti nenachádza žiadny objekt skóre spoľahlivosti bude nula. Každá oblasť obrázka tiež predpovedá pravdepodobnosť triedy. Bez ohľadu na to, kolko ohraničujúcich boxov sa nachádza v jednej oblasti, pre každú oblasť predpovedáme iba jednu triedu. Nakoniec môžeme detekcie prahovať určitou hodnotou, aby

sme videli iba detekcie s vysokým skóre. Základný princíp YOLO je znázornený na obrázku 1.6.



Obr. 1.6: Základný princíp YOLO [29]

Sietová architektúra YOLO je inšpirovaná GoogLeNet modelom pre klasifikáciu obrázkov [38]. Architektúra má 24 konvolučných vrstiev, po ktorých nasledujú dve plne prepojené vrstvy. Architektúra YOLO je znázornená na obrázku 1.7.



Obr. 1.7: Architektúra YOLO [29]

Nakoľko každá oblast, ktorá predpokladá viacero ohraničujúcich boxov môže mať iba jednu triedu, vznikajú v YOLO detektore silné priestorové obmedzenia. Tie obmedzujú počet blízkych objektov a taktiež detekciu viacerých malých predmetov blízko seba.

YOLO sa vyznačuje mimoriadne rýchlosťou architektúrou. Základný model spracováva obrázky v reálnom čase rýchlosťou 45 snímok za sekundu. V porovnaní s najmodernejšími detekčnými systémami, YOLO robí viac lokalizačných chýb, ale je oveľa menej

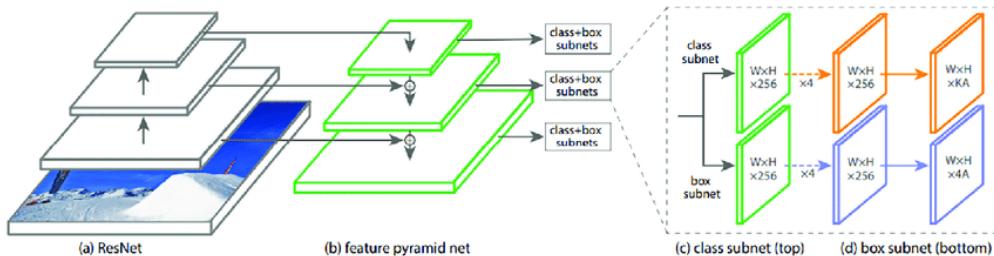
pravdepodobné, že predpovedá nesprávne detekcie tam, kde sa žiadnen objekt nenachádza.

YOLOv2 prináša výrazné zlepšenie rýchlosťi a presnosti pomocou dávkovej normalizácie v konvolučných vrstvách. Taktiež mení fázu tréningu, predpovede ohraničenia boxu. Najnovšia verzia YOLOv3 prináša ešte viac vylepšení. V roku 2018 ju publikovali [30] Joseph Redmon a Ali Farhadi. YOLOv3 predpovedá desaťkrát viac oblastí ako YOLOv2. YOLOv3 predpovedá ohraničujúce boxy v troch rôznych mierkach, čo výrazne uľahčuje detekciu malých objektov.

1.3.6 RetinaNet

RetinaNet publikovali Tsung-Yi Lin a kol. v [22]. RetinaNet je najmodernejší jedno-stupňový detektor. Autori prišli s novou ohniskovou stratou (focal loss), ktorá pomáha znížiť relatívnu stratu v prípade dobre klasifikovaných príkladov a viac sa zameriava na tažké, nesprávne klasifikované príklady.

RetinaNet pozostáva z chrbticovej (backbone) siete a dvoch podsietí špecifických pre jednotlivé úlohy. Ako chrbticovú siet používa architektúru ResNet CNN a na generovanie bohatej konvolučnej pyramídy funkcií používa siet Feature Pyramid Network (FPN), ktorá je nad sietou ResNet. Prvá podsieť siete RetinaNet vykonáva konvolučnú klasifikáciu objektov na výstupe chrbtice, zatiaľ čo druhá vykonáva regresiu konvolučného ohraničujúceho boxu. Architektúra ResNet je znázornená na obrázku 1.8.



Obr. 1.8: Architektúra RetinaNet [22]

Výsledky ukazujú, že RetinaNet natrénovaná s ohniskovou stratou, má porovnatelnú rýchlosť ako predchádzajúce jednostupňové detektory (napr. SSD alebo YOLO) a presahuje presnosť všetkých súčasných najmodernejších dvojstupňových detektorov.

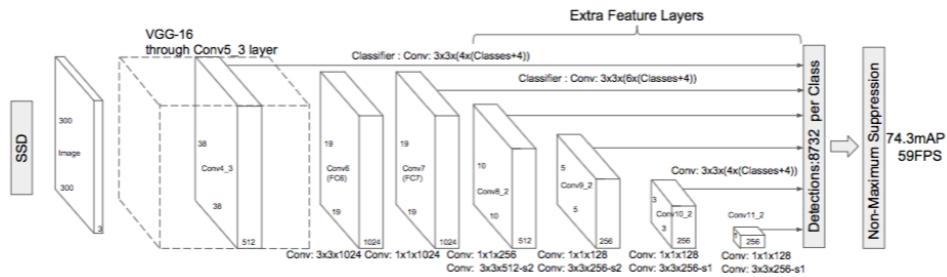
1.3.7 SSD (Single Shot Detector)

SSD je ďalšou jednostupňovou architektúrou na detekciu objektov prostredníctvom CNN. V článku [24] ju publikovali Wei Liu a kol..

Vysoká presnosť detekcie v SSD sa dosahuje použitím viacerých predvolených polí alebo filtrov s rôznymi veľkosťami a pomerom strán na detekciu objektov. V čase pred-

povede sieť generuje skóre pre prítomnosť každej kategórie objektov v každom predvolenom poli a vytvára úpravy pola tak, aby lepšie zodpovedali tvaru objektu. Aby siet detegovala objekty rôznych veľkostí, kombinuje predpovede z viacerých funkčných máp. V porovnaní s inými jednostupňovými metódami má SSD oveľa lepšiu presnosť aj pri menšej veľkosti vstupného obrázka.

SSD ako chranticovú siet používa skrátenú VGG16 CNN. Architektúra SSD je znázornená na obrázku 1.9.



Obr. 1.9: Architektúra SSD [24]

1.3.8 Rodina R-CNN

V "rodine" R-CNN sú architektúry konvolučných sietí, ktoré sú založené na R-CNN. Každá siet v tejto rodine sa vyvíjala pomocou rozšírení, ktoré časom nahradzali predchádzajúce riešenia. Postupne budú uvedené R-CNN, Fast R-CNN, Faster R-CNN. Nakoniec bude uvedená Mask R-CNN, ktorá je už dostatočne odlišná od predchádzajúcich, a preto ju budeme považovať za iné riešenie.

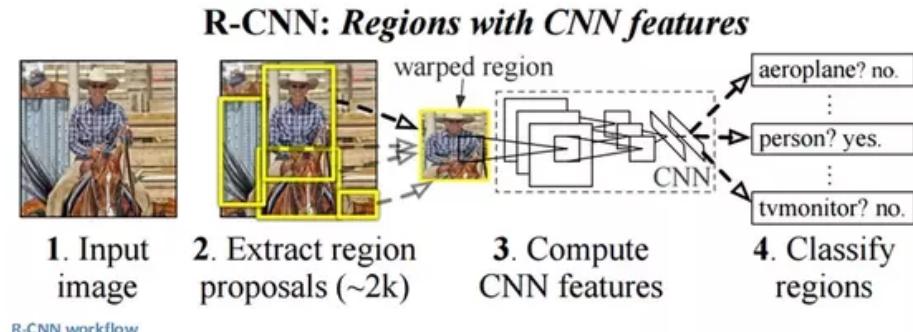
R-CNN

Regionálne konvolučné neurónové siete (Region-based CNN – RCNN) v roku 2014 publikovali Ross Girshick a kol. v článku [14]. Tieto siete viedli k vzniku rodiny R-CNN.

Detektor R-CNN pozostáva z troch modulov. Prvý z nich generuje návrhy regiónov nezávisle od kategórie. Tieto návrhy definujú súbor detekcií kandidátov, ktoré má detektor k dispozícii. Druhým modulom je konvolučná neurónová siet, ktorej výstupom je vektor funkcií s pevnou veľkosťou. Tretí modul je sada lineárnych SVM, trénovaných pre každú triedu nezávisle.

Prvý modul v R-CNN vytvára návrhy regiónov pomocou procesu, ktorý má názov "selektívne vyhľadávanie". Toto selektívne vyhľadávanie obrázkov postupne prehľadáva oknami s rôznymi veľkosťami a pre každú veľkosť sa snaží zoskupiť susediace pixely na základe textúry, farby alebo intenzity, tak aby identifikovali objekty. Takto nájde približne 2000 návrhov regiónov. Tu sa dostávame k problému, že tréning siete bude

kvôli veľkému počtu regiónov trvať veľmi dlho. Ďalšou nevýhodou je, že selektívne vyhľadávanie je fixný algoritmus. Teda v tejto fáze nenastáva učenie, čiže sa algoritmus nemôže zlepšovať. Architektúra R-CNN je znázornená na obrázku 1.10.



Obr. 1.10: Architektúra R-CNN [14]

Fast R-CNN

Ross Girshick, autor predchádzajúceho článku, vyriešil niektoré z nevýhod R-CNN. Algoritmus na rýchlejšiu detekciu nazval Fast R-CNN a publikoval ho v článku [13].

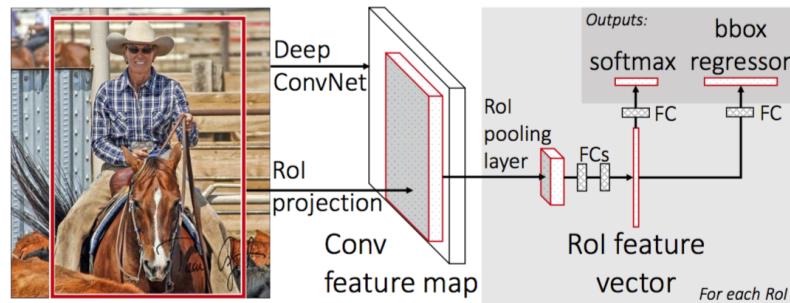
Autor si uvedomil, že v R-CNN sa pre každý obrázok veľa navrhovaných regiónov prekrývalo. Toto viedlo k opakovanému rovnakému výpočtu CNN. Preto navrhol, aby sa CNN spúšťala iba jedenkrát na jeden obrázok a aby sa potom našiel spôsob ako tento výpočet zdielať medzi návrhami.

Vo Fast R-CNN teda vstupný obrázok prichádza do CNN, aby sa vygenerovala konvolučná mapa vlastností. Z tejto mapy sa nájdú návrhy regiónov, pomocou selektívneho vyhľadávania, ktoré sa deformujú do štvorcov. Následne pomocou vrstvy regiónu záujmu (Region of Interest- RoI) sa jednotlivé návrhy spoja do vektoru s pevnou veľkosťou. Tento vektor je vstupom pre plne prepojenú vrstvu, kde sa nakoniec pomocou dvoch výstupných vrstiev vytvárajú odhady pravdepodobnosti a štyri hodnoty pre ohraničujúci box pre každú triedu. Architektúru Fast R-CNN je možné vidieť na obrázku 1.11.

Faster R-CNN

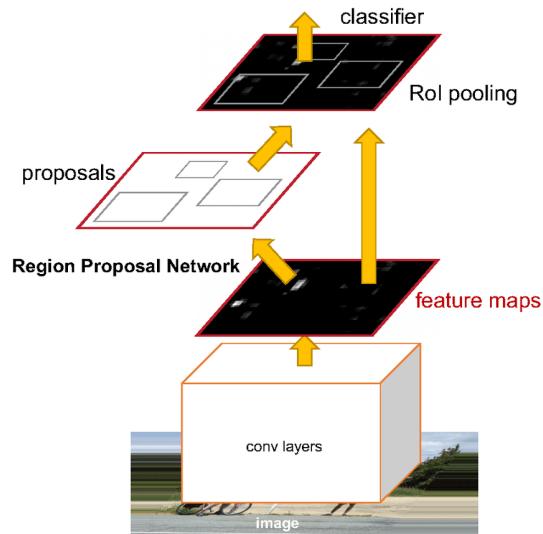
Prechádzajúce siete používali algoritmus selektívneho vyhľadávania. Selektívne vyhľadávanie je časovo náročný proces, preto sa ho autori Faster R-CNN rozhodli zmeniť. Tento detektor publikovali Shaoqing Ren a kol. v článku [31].

Faster R-CNN je v súčasnosti jedným z najpoužívanejších detektorov, pretože umožňuje detekciu v reálnom čase a jeho presnosť je porovnatelná s inými najmodernejšími detektormi.



Obr. 1.11: Architektúra Fast R-CNN [13]

Hlavným rozdielom oproti predchádzajúcim detektorom je, že namiesto selektívneho vyhľadávania sa na identifikáciu návrhov regiónov používa samostatná siet. Faster R-CNN sa skladá z dvoch modulov. Prvým modulom je hlboká konvolučná siet, ktorá navrhuje regióny a druhým modulom je rýchly R-CNN detektor, ktorý následne využíva navrhované regióny. Celý systém je jednotná, plne prepojená siet. Architektúra Faster R-CNN je znázornená na obrázku 1.12.

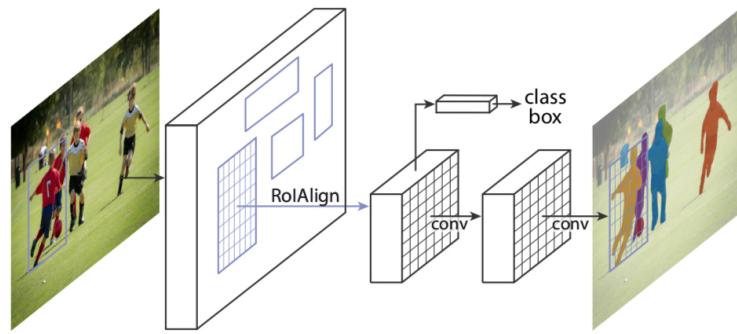


Obr. 1.12: Architektúra Faster R-CNN [31]

Mask R-CNN

Mask R-CNN, publikoval tím Facebook AI Research v článku [15]. Táto metóda neurčuje iba ohraničujúce hranice, ale aj konkrétnu masku.

Mask R-CNN je rozšírením Faster R-CNN. Pridáva vetvu k už existujúcej vetve na rozpoznávanie ohraničujúceho boxu. Výstupom tejto pridanej vetvy je binárna maska pre každý regiónu záujmu (RoI), ktorá presne určuje či daný pixel je alebo nie je súčasťou objektu. Táto metóda preto vytvára veľmi presné segmentácie obrázkov.



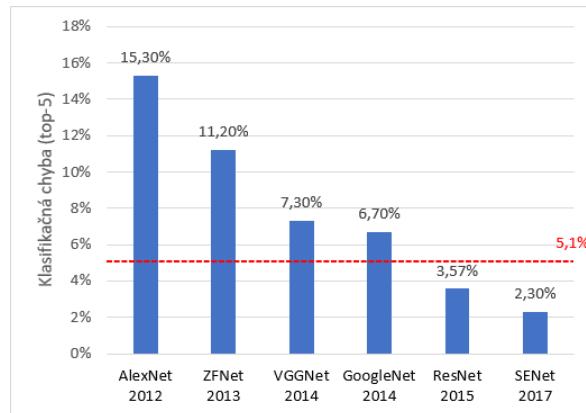
Obr. 1.13: Schéma R-CNN [15]

1.3.9 Porovnanie architektúr

Každým rokom vznikajú nové architektúry konvolučných neurónových sietí, ktoré sa snažia znižovať chybu klasifikácie obrazu. Či už úplne nové alebo vylepšené existujúce architektúry sa následne snažia preukázať svoju presnosť v rôznych súťažiach.

Veľmi známou súťažou je súťaž ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [33], kde sa hodnotia jednotlivé architektúry v oblasti klasifikácie obrazu alebo detektie objektov v obraze či videu. Architektúry sú v súťaži hodnotené na datasete ImageNet a to na základe chybovosti top-1 a top-5, kde chybovost top-5 sa klasifikuje ako úspech v prípade, že sa správne označenie nachádza medzi prvými piatimi predikciami modelu. Táto súťaž prebieha už od roku 2010. Veľkým pokrokom bola architektúra ResNet, ktorá v roku 2015 prekonala mieru ľudskej chybovosti. Výsledky súťaže v roku 2016 nepriniesli žiadne revolučné zlepšenia, pretože víťaz Trimpš-Soushen nepriniesol žiadne nové inovatívne technológie či novinky.

Na grafe 1.15 sú znázornené výsledky modelov, ktoré v uvedenom roku zvíťazili alebo dosiahli v súťaži veľmi dobré výsledky. Červená prerušovaná čiara znázorňuje mieru ľudskej chybovosti.



Obr. 1.14: Presnosť klasifikačných modelov konvolučných neurónových sietí na datasete ImageNet [19] [44] [37] [38] [16] [17]

Resnet a SENet patria medzi jedny z najlepších architektúr pre spracovanie obrazu a sú často používané ako defaultné.

Čo sa týka modelov na detekciu objektov, je veľmi ľahké dosiahnuť spravodlivé porovnanie medzi nimi. Na ich konečný výkon vplýva veľké množstvo iných faktorov. Jedným z nich je, aký klasifikačný model používa detektor ako extraktor príznakov. Vyššie spomenuté výsledky sa venujú práve týmto modelom samostatne. Ďalšími faktormi sú napríklad rozlíšenie vstupných obrázkov, augmentácia dát, trénovací dataset a mnoho ďalších. Neexistuje teda priama odpoveď na to, ktorý detektor je najlepší. Modely sú testované aj na rôznych datasetoch a napokia sa aj datasety medzi rokmi líšia, nie je vhodné ich uvedené presnosti v článkoch porovnávať priamo. Tabuľka 1.15 znázorňuje výsledky vybraných modelov na detekciu objektov na jednotlivých datasetoch za rôzne roky. Uvedené zobrazujú mAP skóre (Mean Average Precision) modelov.

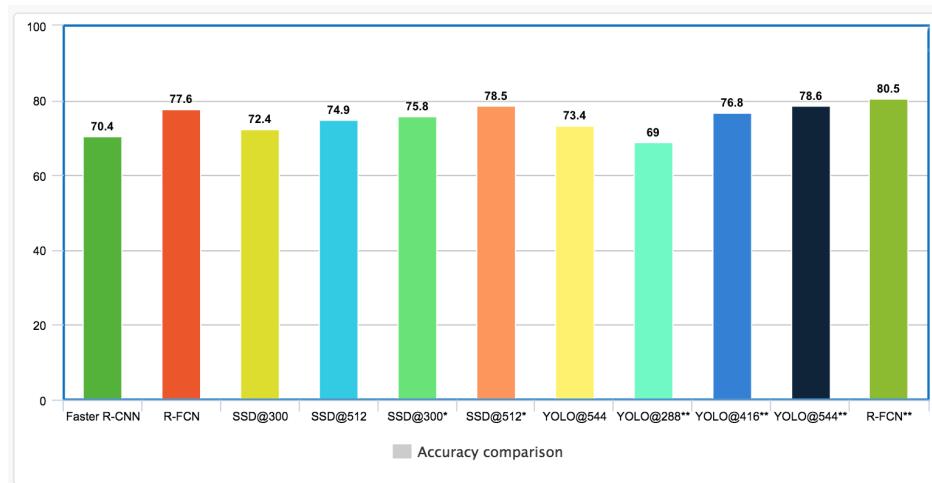
Model / Dataset	PASCAL VOC 2007	PASCAL VOC 2010	PASCAL VOC 2012	COCO 2015	COCO 2016
YOLO	63.4%	x	57.9%	x	x
YOLOv2	78.6%	x	x	21.6%	x
SSD	81.6%	x	80.0%	26.8%	x
R-CNN	58.5%	53.7%	53.3%	x	x
Fast R-CNN	70.0%	68.8%	68.4%	x	x
Faster R-CNN	78.8%	x	75.9%	21.5%	x
Mask R-CNN	x	x	x	x	39.8%

Obr. 1.15: Porovnanie modelov na detekciu objektov [29] [24] [14] [13] [31] [15]

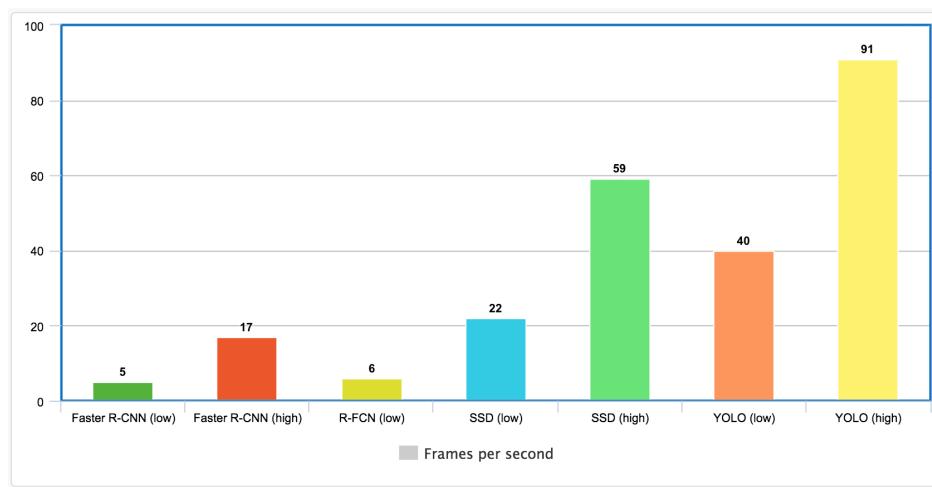
Na grafoch 1.16 a 1.17 sú výsledky presnosti a rýchlosť zosumarizované. Výsledky na grafoch zobrazujú výsledky modelov, ktoré boli trénované na dátach z PASCAL VOC 2007 a 2012 a vyhodnotené na datasete PASCAL VOC 2012. Zobrazené výsledky sú v mAP. Pri modeloch SSD a YOLO číselné označenie znamená rozlíšenie vstupných obrázkov. Ako bolo vyššie spomenuté, rozlíšenie vstupných obrázkov vplýva na celkový výkon modelu, čo je vidieť aj na výsledkoch grafu. Lepšie mAP majú modely s vyšším rozlíšením vstupných obrázkov avšak ich spracovanie trvá dlhší čas.

Na grafe 1.17 vidíme výsledky rýchlosť jednotlivých detektorov. Uvedená rýchlosť jednotlivých detektorov je určená počtom snímkov za sekundu - frame per second (fps). Výsledky na grafe však môžu byť skreslené z dôvodu merania pri rôznych mAP.

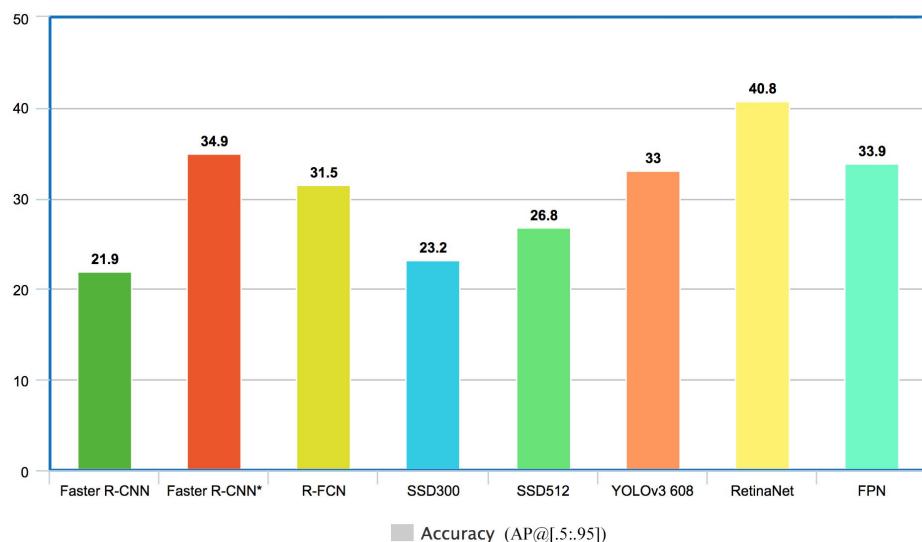
Tretí graf 1.18 zobrazuje výsledky namerané na datasete COCO. Na grafe je vidieť, že modely dosiahli oveľa nižšie hodnoty mAP ako na predošom grafe z dôvodu toho, že dataset COCO je zložitejší.



Obr. 1.16: Porovnanie modelov z hľadiska presnosti na datasete PASCAL VOC [18]



Obr. 1.17: Porovnanie modelov z hľadiska rýchlosťi [18]



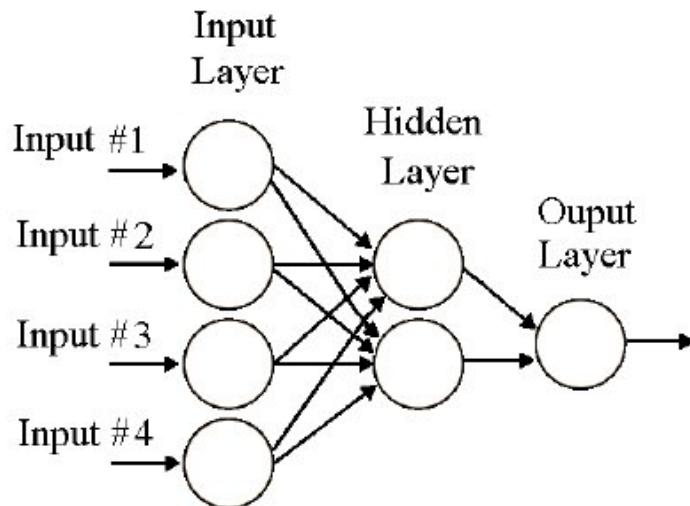
Obr. 1.18: Porovnanie modelov z hľadiska presnosti na datasete COCO [18]

Kapitola 2

Neurónové siete a práca s nimi

2.1 Neurónové siete

Typickým príkladom neurónovej siete je viacvrstvová neurónová sieť, nakoľko použitie jedného neurónu nie je príliš efektívne a dá sa využiť len na veľmi jednoduché úlohy. Viacvrstvová neurónová sieť sa skladá z minimálne troch vrstiev neurónov. Výstup jedného neurónu je vstupom jedného alebo viac neurónov ďalšej vrstvy. Spôsob, akým sú neuróny (uzly) prepojené, určuje topológiu siete. Topológia viacvrstvovej siete je znázornená na obrázku 2.1



[26]

Obr. 2.1: Topológia viacvrstvovej neurónovej siete

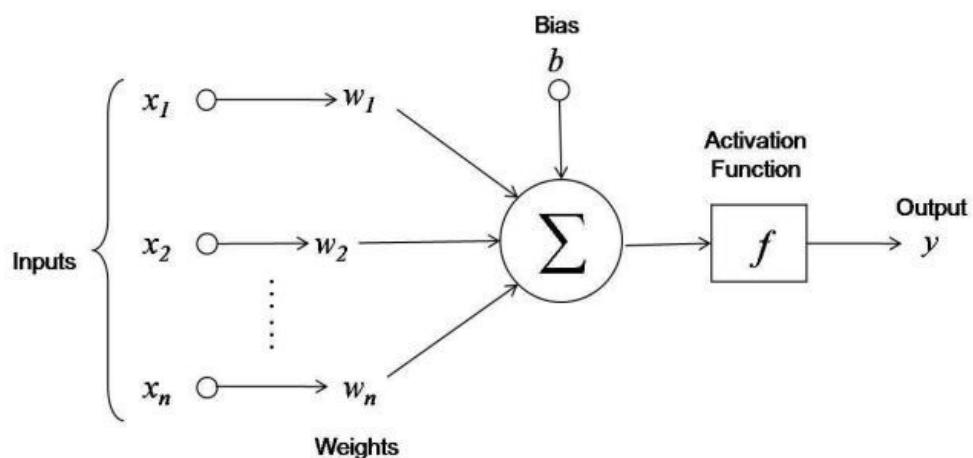
Vstupná vrstva - táto vrstva dáta nemodifikuje, ale iba ich získava z okolitého sveta a pošle ich ďalej do siete. Počet vstupných neurónov je daný počtom vstupov matematického modelu.

Skrytá vrstva - v tejto vrstve sa informácie spracovávajú a následne sú propagované do výstupnej vrstvy. Počet neurónov je určený zložitosťou problému a presnosťou, ktorú chceme pridať do siete.

Výstupná vrstva - táto vrstva musí vždy existovať v topológii siete. Počet výstupných neurónov obvykle zodpovedá počtu klasifikačných tried.

2.1.1 Model neurónu

Základným stavebným prvkom neurónovej siete je neurón, niekedy nazývaný aj ako perceptón. Ilustrácia matematického modelu neurónu je znázornená na obrázku 2.2.



Obr. 2.2: Matematický model neurónu (perceptron) [35]

Neurón má ľubovoľný počet vstupov, ale iba jeden výstup. Každému vstupu je priradený váhový koeficient. Ten určuje dôležitosť vstupu v porovnaní s ostatnými. Výstupom je potom súčet všetkých hodnôt vstupu v závislosti na ich váhovom koeficiente. S výsledkom následne pracuje aktivačná funkcia, ktorá určí výstup neurónu

$$y = f(b + \sum_{i=1}^n w_i x_i), \quad (2.1)$$

kde f je aktivačná funkcia, x_i predstavujú vstupy neurónu, w_i sú váhy vstupov a b je značené skreslenie (bias).

Myšlienka váhových koeficientov je učenlivosť, kontrola miery závislosti jedného neurónu na druhom. Koeficient môže nadobúdať pozitívne aj negatívne hodnoty. Ak je výsledný súčet vyšší ako prahová hodnota, tak je neurón prebudený a indikuje signál na svojom výstupe vo forme aktivačnej funkcie. Ďalším modifikovateľným parametrom je skreslenie b . Ten riadi vplyv neurónu ako celku.

2.1.2 Učenie

Všeobecne existujú dva základné typy strojového učenia, a to učenie sa s učiteľom a učenie sa bez učiteľa.

Učenie s učiteľom

Typickými technikami učenia s učiteľom sú neurónové siete. Pri tomto type je neurónovej sieti predložený vzor a na základe aktuálnych váhových koeficientov je vypočítaný výsledok. Ten je porovnaný s očakávaným výsledkom a následne sa spočíta chyba. Ak je chyba vyššia ako stanovená minimálna hranica, tak je spočítaná korekcia a upravia sa hodnoty váh tak, aby sa hodnota chyby znížila. Tento postup sa opakuje až pokiaľ nie je siet naučená. V tomto prípade to znamená až dokým sa nedosiahne požadovaná minimálna hranica chyby.

Učenie bez učiteľa

Pri tomto type sa na rozdiel od učenia s učiteľom s výstupom vôbec nepracuje. Algoritmus nemá k dispozícii vopred označené dátá, v dôsledku čoho je potom donútený hľadať relevantné príznaky, pomocou ktorých by mohol dátá kategorizovať. Vytvárajú sa tak zhluky vstupných dát, na základe ktorých sa algoritmus naučí reagovať na typického zástupcu zhluku. Tento druh učenia sa využíva v prípade, ak sa označenie vstupných dát nedá jednoznačne určiť.

2.1.3 Aktivačná funkcia

Aktivačná funkcia slúži na výpočet výstupnej hodnoty neurónu v závislosti od jeho vstupných hodnôt. Z rovnice 2.1 vyplýva, že hodnota y môže byť akákoľvek od mínus nekonečno až po plus nekonečno. Aktivačná funkcia slúži na zabezpečenie pomyselnej hranice, medzi ktorými môže neurón produkovať hodnoty. Voľba aktivačnej funkcie má výrazný vplyv na dobu učenia neurónovej siete. Existuje niekoľko rôznych, bežne používaných, aktivačných funkcií. Výber závisí od typu siete a tiež od typu vrstvy, v ktorej pracujú.

Jedna z najstarších a historicky najbežnejšie používaných aktivačných funkcií je logistická funkcia (pre $\alpha = 1$ nazývaná sigmoida):

$$\phi(\alpha, x) = \frac{1}{1 + e^{\alpha x}}. \quad (2.2)$$

Ďalšou aktivačnou funkciou je hyperbolický tangens:

$$\phi(\beta, x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}. \quad (2.3)$$

Ďalšou môže byť napríklad prahová funkcia, ktorá vracia 0 pre záporné hodnoty a 1 pre kladné hodnoty:

$$\phi(x) = \begin{cases} 1, & \text{ak } x \geq 0 \\ 0, & \text{inak.} \end{cases} \quad (2.4)$$

Nakoľko však nie je diferencovateľná, pri trénovaní siete robí problém. Z tohto dôvodu je táto funkcia zlou voľbou.

Najčastejšie používanou aktivačnou funkciou je v súčasnosti Rectified Linear Unit (ReLU). Veľmi často sa používa v konvolučných vrstvách. Je definovaná ako:

$$\phi(x) = \begin{cases} 0, & \text{ak } x < 0 \\ x, & \text{ak } x \geq 0. \end{cases} \quad (2.5)$$

2.1.4 Stratová funkcia (Loss function)

Na dosiahnutie schopnosti učenia sa algoritmu je potrebné odhadnúť jeho predikciu. Na naučenie parametrov modelu sa používa stratová funkcia. Tá definuje do akej miery sa predikcia líši od očakávaného výsledku. Z jej výstupov je možné určiť, akým smerom a približne o kolko je nutné zmeniť váhové koeficienty jednotlivých vstupov. Potom správny priebeh učenia neurónovej siete možno pozorovať z hodnôt stratovej funkcie.

Strata na celej databáze je suma strát na jednotlivých vzorkách:

$$L = \frac{1}{N} \sum_i L_i, \quad (2.6)$$

kde N je počet tréningových vzoriek.

2.1.5 Pretrénovanie a podtrénovanie

Na testovanie predikcie neurónových sietí si potrebujeme databázu rozdeliť na trénovaciu sadu, validačnú sadu a testovaciu sadu. Vhodný postup je zvoliť hyperparametre, ktoré najlepšie fungujú na validačných dátach a vyhodnotiť ich na testovacích dátach.

Pri učení neurónovej siete je bežné, že model sa po určitom počte iterácií dostane do stavu, kedy podáva veľmi presné výsledky na trénovacích dátach, ale na validačných už slabšie. Je to kvôli tomu, že namiesto toho aby sa sieť naučila pravidlá, sa naučila dátu. Tento stav na označuje pretrénovanie.

Trénovacie dáta sú dátu, na ktorých sa model učí. Validačné dáta zasa súžia na dozeranie pretrénovania. Po každej epoche sa spočíta chyba z validačných dát. Ak chyba na trénovacích dátach klesá, ale naopak pri validačných dátach rastie, znamená to, že model sa dostáva do stavu pretrénovania.

2.2 Konvolučné neurónové siete

Metódy spomínané v kapitole 1 sú dnes takmer úplne nahradené metódami používajúcimi konvolučné neurónové siete, ktoré sú presnejšie. Konvolučné neurónové siete (CNN – Convolution Neural Network) sú špeciálnym typom neurónových sietí (NN – Neural Network), ktoré sa pôvodne používali v aplikáciach na spracovanie obrazu.

Zatial čo základná neurónová sieť má vstupné hodnoty vo forme vektora, tak v konvolučných neurónových sietach slúži ako vstup objem. Hlavný rozdiel medzi NN a CNN spočíva v tom, že CNN používa konvolúciu namiesto násobenia matíc aspoň v jednej zo svojich vrstiev. Hlavnou myšlienkou CNN je vytvorenie pyramídy konvolučných a pooling vrstiev, ktoré zmenšujú šírku a výšku obrazu a zvyšujú jeho hĺbku.

Konvolučné vrstvy vykonávajú konvolúciu nad vstupmi do neurónovej siete. Konvolučná operácia sa uskutočňuje pomocou malej neurónovej siete so zdieľanými váhami. Pooling vrstva zo vstupného obrazu vyextrahuje iba zaujímavé časti pomocou matematických operácií a tým sa redukuje jeho dimenzionalita.

2.3 Softvérové rámce pre hlboké učenie

Rámc pre hlboké učenie (Deep Learning Frameworks) je rozhranie, knižnica alebo nástroj, ktorý nám umožňuje ľahšie a rýchlejšie vytvárať modely hlbokého učenia bez toho, aby sme sa dostali do detailov základných algoritmov. Poskytujú jasný a výstižný spôsob definovania modelov pomocou súboru vopred zostavených a optimalizovaných komponentov.

Dobrý rámc by mal byť ľahko pochopiteľný, použiteľný a optimalizovaný z hľadiska výkonu. Mal by uľahčovať proces získavania údajov, predikcie týchto údajov a mal by mať dobrú podporu komunity.

Ďalej bude uvedený krátky prehľad niektorých open-source rámcov.

TensorFlow

TensorFlow je dnes jedným z najpopulárnejších rámcov pre hlboké učenie. Bol vyvinutý tímom Google Brain pre interné použitie Google. Google služby ako Gmail, Translator, Photos využívajú na pozadí TensorFlow. Rovnako je tento rámc použitý aj v aplikáciách firiem Uber, Airbnb, Nvidia a ďalších.

TensorFlow je bezplatná open-source softvérová knižnica, ktorá je používaná na numerické výpočty. Pre numerické výpočty využíva grafy toku údajov (data flow graphs). Tie umožňujú robiť viacero dátovo náročných výpočtov. Uzly v tomto grafe predstavujú matematické operácie, zatial čo hrany reprezentujú viacrozmerné polia dát (tenzory), ktoré zabezpečujú komunikáciu medzi nimi. TensorFlow pracuje so statickým grafom

toku údajov. To znamená, že najprv definujeme graf, potom spustíme výpočty a ak potrebujeme vykonať zmeny v architektúre, preškolíme model.

Knižnica je užitočná na vytváranie a experimentovanie s algoritmami hlbokého učenia. TensorFlow má vopred napísané kódy pre väčšinu komplexných modelov hlbokého učenia, ako sú napríklad rekurentné neurónové siete či konvolučné neurónové siete. Hlavnou výhodou knižnice je flexibilná architektúra, ktorá umožňuje vykonávať výpočty na viacerých jadrách na CPU alebo GPU paralelne.

TensorFlow je dostupný pre Window, Linux, MacOS ale aj pre mobilné zariadenia s operačným systémom Android. Knižnica má k dispozícii API vo viacerých jazykoch, ako sú napríklad Python, JavaScript, C++, Java, Go. Python API je v súčasnosti najkompletnejšie a najjednoduchšie na použitie.

Existuje veľa nástrojov, ktoré uľahčujú prácu s TensorFlow. Jedným s najlepších je TensorBoard. Je to vizualizačný nástroj pre lepšie porozumenie, debugovanie a optimalizáciu programu.

PyTorch

Rámec PyTorch bol vyvinutý pre služby Facebooku, ale pre svoje vlastné úlohy ho už používajú spoločnosti ako Twitter a Salesforce. PyTorch pracuje s dynamicky aktualizovaným grafom. Toto je veľmi užitočné, ak sa nevie, kolko pamäte bude treba na vytvorenie modelu neurónovej siete. Ďalšou výhodou je, že obsahuje mnoho vopred natrénovaných modelov. PyTorch je pravdepodobne najväčším konkurentom TensorFlow.

Caffe

Caffe je jednoduchý rámec pre hlboké učenie, ktorý poskytuje veľkú rýchlosť. Rámec je vhodný najmä pri modelovaní konvolučných neurónových sietí. Je napísaný v jazyku C++ a C s API pre Python a Matlab.

Keras

Keras je najlepší rámec pre hlboké učenie pre začiatočníkov. Je napísaný v jazyku Python a má vysokoúrovňové programové rozhranie. Obsahuje početné implementácie bežne používaných stavebných blokov neurónovej siete, ako sú vrstvy, aktivačné funkcie, optimalizátory a množstvo nástrojov na uľahčenie práce s obrázkami a textovými údajmi. Keras podporuje konvolučné neurónové siete aj rekurentné neurónové siete. Pôvodne bol vyvíjaný ako nadstavba pre TensorFlow.

2.4 Datasetsy

Neexistuje žiadny kvalitný model bez súboru kvalitnej trénovacej sady. V priebehu rokov bolo zhromaždené veľké množstvo trénovacích sád pre chodcov, ktoré sú dostupné na internete. Všetky majú rozdielne charakteristiky, slabé a silné stránky.

Medzi najstaršie trénovacie sady patrí sada INRA. Táto sada bola zhromaždená ako súčasť výskumných prác na zisťovaní vzpriamených ľudí v obrazoch a videách. Napriek tomu, že obsahuje pomerne málo vzoriek, prináša veľmi kvalitné anotácie chodcov v rôznych prostrediach. Avšak trénovacia sada musí byť dostatočne veľká, aby model zvládal zovšeobecnenie. Napríklad na detekciu osôb v aplikácii s vlastným autom je potrebné mať dostatočne rozmanitú scénu. V dnešnej dobe prevládajú Caltech a KITTI ako trénovacie sady pre chodcov. Obe sady poskytujú veľký počet vzoriek. Trénovacia sada KITTI však vyniká väčšou rozmanitosťou.

Rozhodli sme sa použiť aj trénovaciu sadu KITTI, pretože táto práca sa zaoberá detekciou chodcov z pohľadu autonómneho vozidla. Súbory údajov boli zaznamenané z automobilu, teda dokonale zodpovedajú zadaniu práce.

2.4.1 KITTI dataset

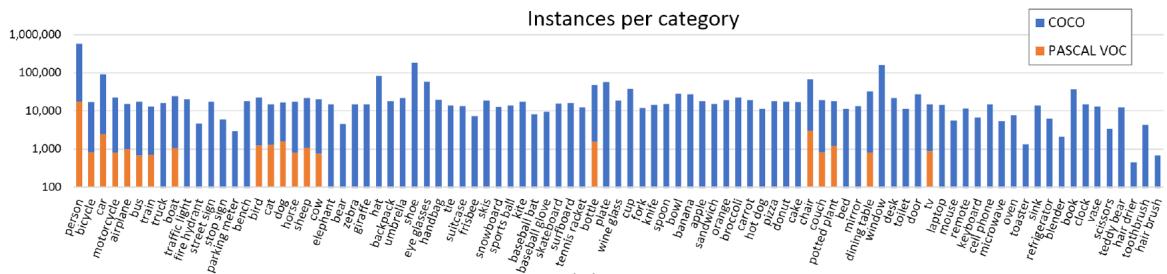
Dataset [3] bol vytvorený pre výskum počítačového videnia a strojového učenia o stereofónnom snímaní, optickom toku, vizuálnej odometrii, segmentácii ciest, odhadu hĺbky detekcií 2D a 3D objektov a sledovaní objektov.

Trénovacia sada je zaznamenaná autom, ktoré je vybavené dvoma farebnými videokamerami s vysokým rozlíšením a šedou stupnicou, laserovým skenerom Velodyne a GPS. Natáčané prostredie obsahuje vidiecke oblasti a diaľnice. Na jednej snímke je viditeľných až 15 automobilov a 30 chodcov.

Trénovaciu sadu je možné rozdeliť podľa obtiažnosti do troch častí – na ľahký, stredný a ťažký. Napríklad obtiažnosť je väčšia, ak je objekt vzdialený. Toto delenie sa vykonáva podľa výšky ohraničujúceho rámčeka, podľa úrovne oklúzie a orezania.

2.4.2 COCO dataset

V roku 2015 spoločnosť Microsoft vydala dataset COCO, celým názvom Common Objects in Context [23]. COCO je rozsiahly dataset pre detekciu objektov a segmentáciu. Ako jeho názov naznačuje ide o dataset, ktorý obsahuje obrázky zložitých každodených scén zahrňujúcich bežné objekty v ich prirodzenom kontexte. Dataset obsahuje až 330 000 obrázkov a 1,5 milióna inštancií objektov. K dispozícii je 80 tried objektov a 91 tried vecí. Distribúcia jednotlivých inštancií podľa kategórie je znázornená na obrázku 2.3.



Obr. 2.3: Porovnanie distribúcií inštancií tried datasetov COCO a PASCAL VOC [23]

Najpočetnejšou kategóriou je kategória ”person”. V tejto kategórii sú zahrnutí ľudia v rôznych pozíciach či stavoch. Sú to napríklad ľudia, ktorí sedia, jedia, lyžujú, plávajú, bicykujú, ale taktiež aj kráčajúci ľudia, ktorých možno označiť za chodcov.

2.4.3 GRAZ-01 dataset

Dataset bol vytvorený na inštitúte elektrických meraní a spracovania meracích signálov na Technickej Univerzite v Grazi. Databáza obsahuje obrázky ľudí a bicyklov. Čo sa týka obrázkov ľudí, dajú sa rozdeliť do štyroch skupín, a to na prototypy, priemerné obrázky, kde sa nachádza viacero osôb, obrázky na ktorých sú iba časti osôb alebo sú veľmi ďaleko a na skupinu ľahko rozoznateľných obrázkov.

Väčšina týchto osôb v datasete môže byť označená za chodcov, preto sme sa v praktickej časti rozhodli použiť aj tento dataset. Vyberali sme obrázky chodcov zo všetkých štyroch skupín, aby sme pri trénovaní zabezpečili väčšiu rozmanitosť obrázkov. Dataset je dostupný na [2].

Kapitola 3

Praktická časť

Pre našu prácu sme sa rozhodli použiť neurónové siete, presnejšie konvolučné neurónové siete. Pre prácu s neurónovými sietami sme si zvolili knižnicu Tensorflow. V tejto časti sú opísané inštalácie a konfigurácie potrebné na jej správne využívanie. Ďalšie odseky sa venujú predpríprave dát a súborov, ktoré sú potrebné na následné trénovanie sietí. Ďalej je opísané samotné trénovanie spolu s výberom modelu a nastaveniami krokov trénovania.

Pri trénovaní sietí sme použili rozhranie knižnice TensorFlow s názvom TensorFlow Object Detection API, ktoré je bližšie opísané v odseku 3.2 a metódu prenášaného učenia (transfer learning), ktorá je opísaná v odseku 3.1.

Trénovanie prebiehalo na notebooku s procesorom Intel Core i5-6300HQ obsahujúcim 4 fyzické jadrá s maximálnou taktovacou frekvenciou 3.2 GHz. Procesor disponuje 6 MB L1 cache a využíva 8 GB dostupnej RAM pamäte. Súčasťou notebooku je grafická karta NVIDIA GeForce GTX 960M, ktorá obsahuje 2 GB vlastnej pamäte typu GDDR5 s maximálnou frekvenciou 1253 MHz a výpočty vykonáva na svojich 640 CUDA jadrách. Ako operačný systém využíva tento notebook 64 bitový Microsoft Windows 10.

3.1 Transfer learning

Transfer learning, v doslovnom preklade prenášané učenie, je vo všeobecnosti proces, pri ktorom sa model natrénovaný na jeden problém určitým spôsobom používa na druhý problém. Prenášané učenie je populárnu metódou v počítačovom videní, pretože umožňuje vytvárať presné modely spôsobom, ktorý šetrí čas. Na to, aby bol model dostatočne výkonný, sa zvyčajne vyžaduje trénovanie na veľmi veľkom datasete. Trénovanie na takomto obsiahлом množstve dát môže trvať aj niekoľko týždňov. S prenášaným učením sa namiesto začatia procesu učenia od nuly začne s natrénovanými parametrami, ktoré boli natrénované počas riešenia iného problému.

V súčasnosti existuje mnoho modelov, ktoré sú predtrénované na veľkých datasetoch ako je napr. dataset COCO či Kitti. Tieto modely sa používajú ako základné siete pri prenášanom učení. Základné siete sa z veľkého datasetu naučia rozpoznávať triviálne tvary a malé časti rôznych objektov vo svojich počiatočných vrstvách. V prvých niekoľkých vrstvách sa siete naučia rozpoznávať farby, vodorovné a horizontálne čiary, v ďalších sa naučia rozpoznávať triviálne tvary a v nasledujúcich už aj časti objektov. Posledné vrstvy sa už zameriavajú na vlastnosti špecifické pre danú úlohu. Z tohto dôvodu stačí pri prenášanom učení trénovať už len posledných párov vrstiev, preto sa doba trénovalia výrazne skráti. Autori článku [42] skúmajú všeobecnosť a špecifickosť neurónov v každej vrstve siete.

3.2 TensorFlow Object Detection API

Spoločnosť Google vydala nový rámec s volne dostupným zdrojovým kódom, ktorý uľahčuje identifikáciu objektov v obrazu. Tento rámec sa nazýva TensorFlow Object Detection API. Uľahčuje zostavenie, učenie a nasadenie modelov na detekciu. V modelovej ZOO predstavujú niekoľko predtrénovaných modelov so špecifickými architektúrami neurónových sietí, ktorých výhodou je, že sú neustále dopĺňané a aktualizované. Pri každom modeli je uvedená jeho rýchlosť vykonávania, presnosť a typ výstupu. Niektoré modely majú architektúru, ktorá umožňuje rýchlejšiu detekciu, ale s menšou presnosťou, zatiaľ čo iné modely poskytujú presnejšiu detekciu avšak na úkor ich rýchlosťi.

Jedným z hlavných kritérií pri výbere vhodného modelu je výpočtový výkon zariadenia, na ktorom bude detekcia prebiehať. Z tohto dôvodu Google poskytuje aj predtrénované modely, ktoré je možné spúštať aj na zariadeniach s nižším výkonom, ako sú napríklad mobilné telefóny alebo tablety.

3.3 Inštalácie a konfigurácie

Existuje niekoľko metód, ktoré možno použiť na inštaláciu Tensorflowu. Prvou z nich je inštalácia pomocou pythonovského systému na správu balíkov - pip. Avšak oveľa výhodnejší spôsob inštalácie je inštalovanie v prostredí Python distribúcie Anaconda. Výhodou je, že ponúka kompletný systém správy balíkov, širokú podporu platformiem spolu s efektívnejším využívaním GPU (Graphics Processing Unit) a lepším výkonom pri behu na CPU (Central Processing Unit). Na základe týchto výhod sme sa rozhodli v našej práci zvolať práve tento postup inštalácie.

Ako bolo spomenuté v časti 2.3.1 TensorFlow umožňuje vykonávať výpočet bud na CPU alebo GPU jadrach. Rozhodli sme sa použiť verziu TensorFlow GPU, pretože väč-

šina modelov strojového učenia je optimalizovaná pre GPU. Na účely detektie chodcov sme zvolili verziu tensorflow-gpu 1.14.

Pre podporu GPU musí byť nainštalovaná kompatibilná verzia CUDA (Compute Unified Device Architecture) a cuDNN (CUDA Deep Neural Network library). Pri použití Anacondy sa tieto knižnice nainštalovali automaticky spolu s TensorFlowom. Kompatibilné verzie s tensorflow-gpu 1.14. sú cuda10.0 a cudnn 7.6.5. Nainštalovaná bola Anaconda verzie 3.7, avšak kvôli kompatibilite so zvolenou verziou TensorFlowu sme namiesto prednastaveného Pythonu 3.7 nastavili verziu Python 3.6.1.

Na konfiguráciu modelu a trénovacích parametrov používa vybrané rozhranie TensorFlow Object Detection API takzvané knižnice Protobuf, ktoré sme taktiež nainštalovali.

3.4 Príprava na trénovanie

Predtým ako začneme s trénovaním, potrebujeme si pripraviť dátá a určité súbory. V tejto podkapitole budú opísané jednotlivé kroky tejto prípravy.

3.4.1 Príprava trénovacích a validačných dát

Získanie obrázkov vhodných do nášho datasetu

Existuje mnoho datasetov, ktoré obsahujú chodcov. Pre tvorbu nášho datasetu sme vybrali obrázky z rôznych dostupných datasetov. Prevažnú časť tvorili obrázky z datasetu Kitti a datasetu GRAZ-01, ktorú vytvorili na Technickej Univerzite v Grazi a menšiu časť tvoril dataset COCO. Obrázky z Kitti datasetu majú rozlíšenie 1392 x 512 px a obrázky z GRAZ-01 a COCO majú v priemere približne rozlíšenie 480 x 640 px.

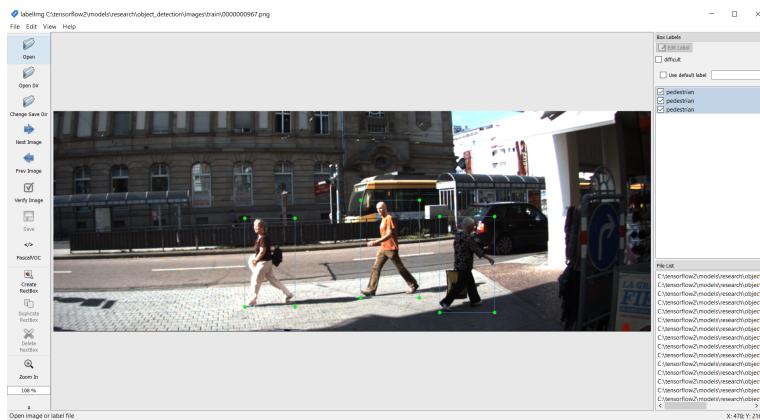
Kedže dataset COCO obsahuje obrázky s rôznymi objektami, vyberali sme len tie, ktoré obsahovali chodcov. Celkovo náš dataset obsahoval 100 obrázkov. Tvorilo ho 38 obrázkov z Kitti datasetu, 43 z datasetu GRAZ-01 a 19 z COCO datasetu. Menší dataset sme použili z dôvodu nízkeho výkonu notebooku, na ktorom sme trénovanie vykonávali. Hlavným obmedzením bola veľkosť pamäte RAM, kvôli ktorej trénovanie pri väčšom datasete zlyhalo, v dôsledku nedostatku pamäte.

Rozdelenie na trénovacie a validačné dátá

Pomer akým rozdelíme dátá na trénovacie a validačné môže hrať v detegovaní dôležitú úlohu. Malé množstvo tréningových dát môže viest k nedostatočne všeobecnému detegovaniu. Nakolko máme pomerne malý dataset rozhodli sme sa pre všeobecné pravidlo rozdelenia 90% trénovacie dát a 10% validačné dátá.

Označenie chodcov

Predtým ako sme vložili tieto dáta do siete na trénovanie, sme ich potrebovali označiť. Na túto úlohu sme použili verejne dostupný program s názvom LabelImg. Tento program umožňuje označovať jednotlivé objekty do obdĺžnikových rámčekov a následne k nim priradiť označenie. Nakolko chceme detegovať iba jeden objekt, každému vyznačenému objektu sme priradili rovnakú triedu s názvom "pedestrian". Ukážku ako sa používa program LabelImg je možné vidieť na obrázku 3.1.



Obr. 3.1: Snímka obrazovky počas používania programu LabelImg

Avšak bolo potrebné rozhodnúť ako sa pri označovaní bude postupovať. Nakolko sa na obrázkoch môžu vyskytovať ľudia v rôznych pozíciah a situáciach, ako napríklad vo veľmi veľkej vzdialenosť, čiastočne prekrytí iným človekom alebo predmetom, na tele môžu niesť objekty, ktoré narušujú ich charakteristický tvar tela, sediaci ľudia ale aj cyklisti, bolo potrebné rozhodnúť na základe akých kritérií sa budú daní ľudia označovať a či ich máme alebo nemáme zahrnúť do našej databázy.

V literatúre sme našli mnoho článkov zaobrajúcich sa problémom čiastočne prekrytých chodcov [34] [28] [9].

Pretože sa má detekcia primárne zameriavať na chodcov, osoby chodiace pešo, rozhodli sme sa sediacich ľudí a cyklistov vylúčiť. V prípade ak chodca žiadnen predmet alebo osoba neprekryvala, tak sme nakreslili rámček tesne okolo celého rozsahu tela. Ak na tele niesol nejaký predmet, označili sme len tvar tela bez tohto predmetu. V literatúre sme našli mnoho článkov zaobrajúcich sa problémom čiastočne prekrytých chodcov [34] [28] [9]. Napríklad v článku [34] sa označovali iba viditeľní chodci a čiastočne prekrytí sa ďalej označovali ako "odrezaní". My sme si na označovanie vybrali rovnakú cestu, ako použili autori tohto článku [9], ktorí pri čiastočne prekrytých chodcoch odhadovali polohy skrytých častí. Ďalej sme sa rozhodli, že nebudeme označovať veľmi malých chodcov. Určili sme preto hranice tak, že najmenšia šírka bola 10 px a najmenšia výška chodca, bola 20 px.

Podla týchto kritérií sme v každom obrázku označili každého chodca. Program

ku každému obrázku vygeneroval súbor vo formáte .xml. Tento súbor obsahuje štyri súradnice pre každý obdlížnikový rámček a príslušnú triedu objektu v ňom.

Generovanie dát pre TensorFlow

Ďalším krokom bolo vygenerovanie vstupných dát pre TensorFlow vo forme TFRecords. Táto forma dát obsahuje informácie o počte a parametroch označení (labels). Každý súbor vo formáte .xml, ktorý obsahoval informácie o trénovacích a validačných dátach, sme si najskôr preformátovali na .csv súbory. Následne sme z nich vygenerovali požadované dáta vo forme TFRecords.

3.4.2 Vytvorenie zoznamu tried

Posledným bodom prípravy na testovanie bolo vytvorenie súboru vo formáte .pbtxt. Tento súbor obsahuje zobrazenia názvov tried na celočíselné hodnoty. Tieto hodnoty sú identifikačné čísla tried. TensorFlow potrebuje tento súbor, aby vedel preložiť identifikačné číslo na názov triedy, ktorú dané číslo zastupuje.

Je ale veľmi dôležité aby tieto celočíselné hodnoty v súbore začínali od čísla 1 a nie od 0, pretože 0 je špeciálna (placeholder) hodnota. Takto je potrebné zobraziť všetkých n tried. My máme len jednu triedu, preto náš zoznam tried bude obsahovať iba jedno zobrazenie. Obsah súboru je zobrazený na obrázku 3.2.

```

labelmap.pbtxt x model_
1 item {
2   id: 1
3   name: 'pedestrian'
4 }
```

Obr. 3.2: Vytvorený zoznam tried

3.5 Trénovanie

Trénovanie najmodernejších detektorov objektov od nuly je pomerne dlhý a náročný proces. Toto trénovanie môže trvať aj niekoľko dní, dokonca aj keď sa na trénovanie použijú veľmi výkonné GPU. V záujme urýchlenia učenia sme sa rozhodli použiť už predtrénovaný model. Na inicializáciu nášho nového modelu znova použijeme niektoré z jeho parametrov.

V tejto práci porovnávame rôzne predtrénované modely, ktoré boli predtrénované na rôznych datasetoch. Taktiež porovnávame rôzne nastavenia konfiguračných súborov, ktoré by mohli zlepšiť výsledné modely.

3.5.1 Výber modelu pre trénovanie

Z modelovej ZOO [4] sme si vybrali predtrénované modely. Nakolko chceme aby naše modely vedeli detegovať chodcov aj na videu, potrebujeme vybrať predtrénované modely, ktoré sú rýchle a zároveň aj presné. Na základe týchto vlastností sme si vybrali tieto tri modely, ktoré budeme medzi sebou porovnávať:

- `faster_rcnn_resnet101_kitti`
- `faster_rcnn_resnet101_fgvc`
- `faster_rcnn_resnet101_coco`

Rýchlosť týchto modelov je uvedená v milisekundách za obrázok (ms) čo predstavuje čas na zdetegovanie jedného obrázku. Presnosť je uvedená v mean average precision (mAP), čo je priemer zo všetkých nameraných priemerných presností AP pre každú triedu. Definícia AP je uvedená v odseku 4.1.5. Avšak presnosti modelov sú uvedené v rôznych metrikách mAP. Prvý model používa metriku PASCAL VOC označovanú ako $mAP_{IoU=.5}$ a ďalšie dva používajú metriku COCO mAP, ich rozdiel je vysvetlený v odseku 4.2.2. Rýchlosť a presnosť modelov sú uvedené v tabuľke na obrázku 3.3.

Názov modelu	Rýchlosť (ms)	Presnosť (mAP)
<code>faster_rcnn_resnet101_kitti</code>	79	87
<code>faster_rcnn_resnet101_fgvc</code>	395	58
<code>faster_rcnn_resnet101_coco</code>	106	32

Obr. 3.3: Vybrané predtrénované modely

Všetky tri modely používajú neurónovú siet s názvom Faster R-CNN spolu so sieťou ResNet (Residual Neural Network), ktorá je použitá ako extraktor príznakov. Vybrali sme si modely, ktoré používajú rovnaké siete, aby sme mohli lepšie porovnávať vplyv rôznych datasetov, na ktorých boli tieto modely predtrénované.

Prvý model bol predtrénovaný na datasete Kitti s dvoma triedami. Tieto triedy sú "pedestrian" a "car". Druhý model bol predtrénovaný na datasete iNaturalist Species, ktorá obsahuje až 2854 tried rastlín a zvierat. Tento dataset neobsahuje triedu "pedestrian", čo bude zaujímavé pre následné porovnávanie modelov. Tretí model s názvom `faster_rcnn_resnet101_coco` bol natrénovaný na datasete COCO, ktorý bol opísaný v predchádzajúcej kapitole v odseku 2.4.2. Tento dataset obsahuje aj triedu "person", čo je dôvod, prečo sme si aj tento model vybrali na dotrénovanie. Očakávame, že práve táto trieda by mohla obsahovať užitočné prvky, ktoré sa už siet naučila rozpoznávať a mohli by byť podobné chodcom.

K týmto modelom sme si stiahli zodpovedajúce konfiguračné súbory z TensorFlow Object Detection API úložiska [4], ktoré obsahujú aj ďalšie potrebné súbory na ďalšie trénovanie. Model `faster_rcnn_resnet101_kitti` sme ďalej netrénovali, zmenili sme len zodpovedajúce súbory tak, aby sa pri výslednej detekcií označovali iba chodci.

Tento model budeme ďalej využívať na porovnanie výkonu ostatných modelov. V nasledujúcich častiach práce sa venujeme už len modelom `faster_rcnn_resnet101_fgvc` a `faster_rcnn_resnet101_coco`.

3.5.2 Nastavenie krokov trénovania

V TensorFlow Object Detection API sú všetky parametre definované pomocou konfiguračného súboru, ktorý sme si stiahli v predchádzajúcim bode. V tomto súbore sme zmenili niektoré parametre, aby zodpovedali nášmu trénovaciemu modelu. Počet tried sme zmenili na 1, nakoľko chceme detegovať len jednu triedu - chodcov. Taktiež sme nastavili počet testovacích dát, v našom prípade ich bolo 10. Pre správne spustenie trénovania bolo potrebné nastaviť parameter `from_detection_checkpoint` na hodnotu `true`, aby sa začal proces prenášaného učenia.

3.5.3 Nastavenie konfiguračných súborov pre rôzne modely

Pre rôzne experimenty bolo potrebné zmeniť rôzne parametre v konfiguračných súboroch pre jednotlivé modely. V tomto odseku budú opísané parametre, ktoré sme sa rozhodli zmeniť a tiež pôvodné nastavenie konfiguračných súborov.

Batch size: Veľkosť dávky definuje počet tréningových vzoriek, ktoré prechádzajú sietou naraz. Tento parameter sa vyberá hlavne na základe dostupnej RAM pamäte v počítači. Nakoľko naša RAM pamäť nemá optimálnu veľkosť, rozhodli sme sa pri každom modeli použiť veľkosť dávky 1. Použitím menšej veľkosti dávky nebudú nároky na pamäť tak vysoké.

Optimizer: TensorFlow object detection API ponúka tri optimalizátory, a to Adam, Momentum a RMSProp. Hlavným rozdielom medzi nimi je spôsob, akým aktualizujú parametre sietí. V tejto práci skúmame vplyv optimalizátora Momentum a Adam.

Learning rate: Krok učenia je jedným z najdôležitejších parametrov. Pri používaní predtrénovaných modelov je dobré nastavovať krok učenia na nižšie hodnoty, pretože vysoký krok učenia zvyšuje riziko straty predchádzajúcich znalostí siete.

Data augmentation options: Augmentácia dát je technika, ktorá zväčšuje existujúci dataset prostredníctvom informácií získaných z rovnakého datasetu. Nakoľko náš trénovací dataset nemá veľkú veľkosť, rozhodli sme sa túto techniku použiť. Použitím augmentácie dát taktiež predídeme preučeniu siete. TensorFlow object detection API poskytuje viaceré techniky, ako sú napríklad preklopenie, rotácia, orezanie a mnoho ďalších. Pri našich modeloch sme overovali prínos vertikálneho preklopenia, náhodného orezania a náhodného jasu.

Originálne nastavenie tých parametrov, ktoré sme sa rozhodli zmeniť, bolo pre oba modely rovnaké. Originálne parametre pre oba modely sú nasledovné:

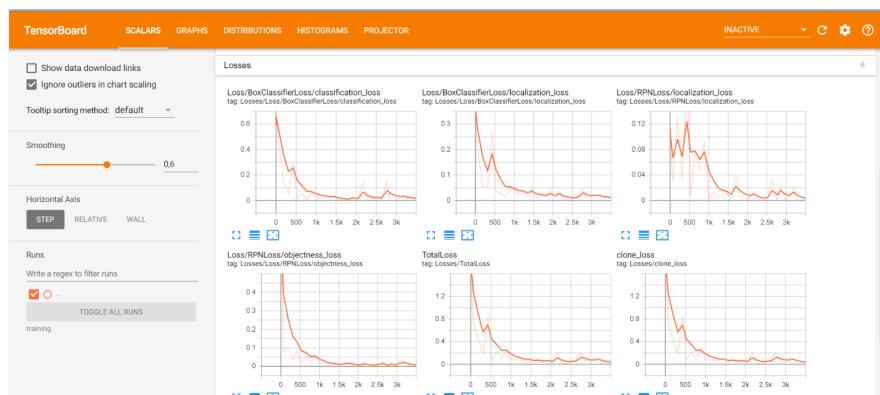
- batch size: 1
- optimizer: momentum_optimizer
- learning rate: 0,003
- data augmentation options: random_horizontal_flip

3.5.4 Trénovanie modelu

Ako bolo už spomenuté, na učenie nášho modelu využijeme už predtrénovaný model. Tento predtrénovaný model si pri trénovaní ukladal takzvané kontrolné body (checkpoints). Kontrolné body sú údaje, ktoré zachytávajú stav premenných v konkrétnom čase. Vďaka týmto kontrolným bodom môžeme vykonávať učenie vo viacerých fázach. Tieto body sú výhodou aj pri ďalšom trénovaní, pretože umožňujú spustenie trénovania od posledného kontrolného bodu a tak pri neočakávanom zrušení procesu trénovania nemusíme začať model trénovať odznova.

Pri našom trénovaní sme využili posledný kontrolný bod predtrénovaného modelu a trénovanie nášho modelu sme začali od tohto bodu. Počas trénovania vidíme v príkazovom riadku poradie kroku, ktorý práve prebieha a aktuálnu hodnotu straty (loss). Táto strata má spočiatku vysoké hodnoty a pri úspešnom trénovaní by mala s rastúcim počtom krokov postupne klesať.

Proces trénovania je možné zobraziť pomocou nástroja TensorBoard. Pre zobrazenie TensorBoardu sme príkazom spustili webový server, ktorý nám ho sprístupnil pomocou webového prehliadača. V TensorBoarde je možné vidieť rôzne grafy a informácie o tom ako učenie napreduje. Príklady grafov v Tensorboarde je možné vidieť na obrázku 3.4.



Obr. 3.4: Príklady grafov v Tensorboarde

Uzly v grafoch predstavujú operácie a orientované hrany reprezentujú odovzdanie výsledku jednej operácie na ďalšiu. Najdôležitejším grafom je stratový graf (loss graph),

ktorý ukazuje celkovú stratu klasifikátora v priebehu času. Čím je strata nižšia, tým lepší je výkon modelu.

Proces trénovania je možné ukončiť stlačením klávesov Ctrl+C v príkazovom riadku. Proces trénovania sme ukončili v momente, keď bola strata dostatočne malá a keď počas dlhšieho času nedošlo k výraznému zlepšeniu

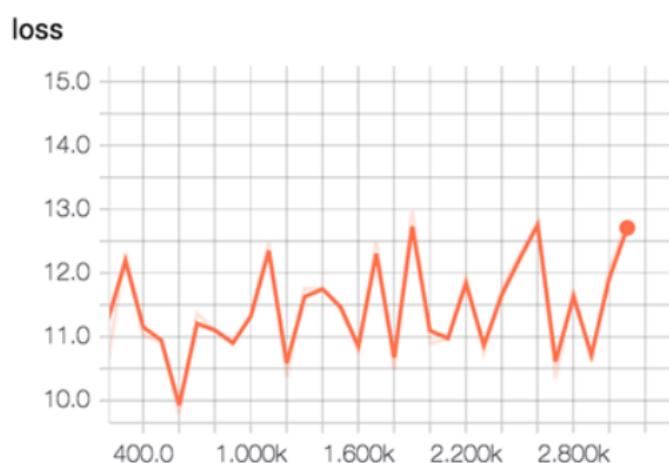
3.5.5 Grafy celkovej straty počas trénovania

Oba modely sme natrénovali so štyrmi rôznymi nastaveniami konfiguračného súboru. Takto nám vzniklo až 8 rôznych grafov celkovej straty, z ktorých možno vidieť ako modely napredujú. Na grafoch 3.5 až 3.13 sú znázornené hodnoty v dvoch rôznych farbách. Vyblednutá oranžová farba znázorňuje skutočné hodnoty celkovej straty, zatiaľ čo tmavšia farba predstavuje vyhľadenie grafu hodnotou 0,6.

Každý model bol natrénovaný na približne 13 000 až 15 000 krokoch. Pri väčšine modelov tento počet krovok stačil na to, aby sa celková strata znížila na dostatočne malú hodnotu. Každé toto trénovanie trvalo približne 4 hodiny.

Optimálny graf celkovej straty

Pomocou grafu celkovej straty môžme určiť aký dobrý je výkon nášho modelu. Ako už bolo spomenuté čím je strata nižšia, tým lepší je výkon nášho modelu. Preto je našim cieľom, pri trénovaní minimalizovať celkovú stratu. Pri optimálnom stave by mala strata v priebehu iterácie klesať, čo znamená že model našiel riešenie. Môže však nastať aj prípad, kedy strata neklesá, ale pohybuje sa hore a dole. Znamená to, že model sa neučí, ale výsledky iba odhaduje. Takýto prípad ilustruje obrázok 3.5.



Obr. 3.5: Príklad modelu, ktorý sa neučí

faster_rcnn_resnet101_coco

Na obrázkoch 3.6 - 3.8 sú znázornené grafy celkovej straty počas trénovania modelu faster_rcnn_resnet101_coco s rôznymi menšími zmenami originálneho konfiguračného súboru.

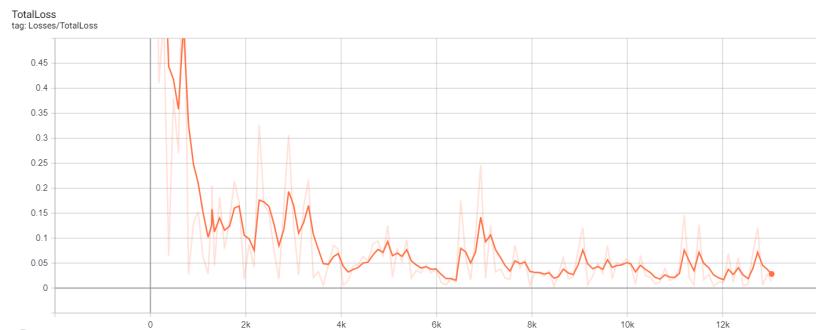
Obrázok 3.6 znázorňuje graf modelu s pôvodným nastavením parametrov.

Obrázok 3.7 zobrazuje graf po znížení hodnoty kroku učenia z 0.003 na 0.002.

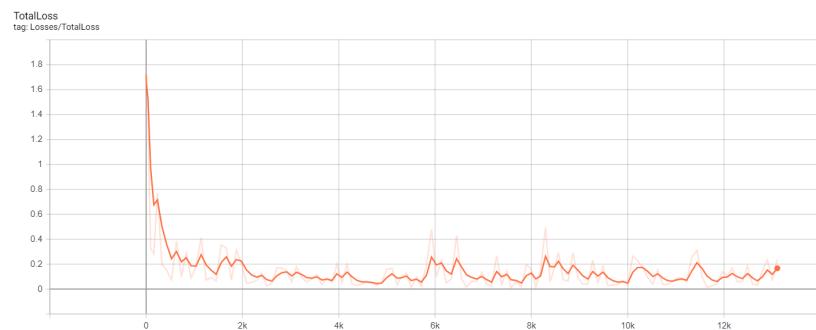
Obrázok 3.8 znázorňuje graf po pridaní ďalších techník na augmentáciu dát spolu z predchádzajúcou zmenou kroku učenia. Pridané techniky na augmentáciu dát boli náhodné orezanie (random_crop_image) a náhodný jas (random_adjust_brightness).



Obr. 3.6: Graf celkovej straty v originálnom nastavení



Obr. 3.7: Graf celkovej straty po zmene kroku učenia

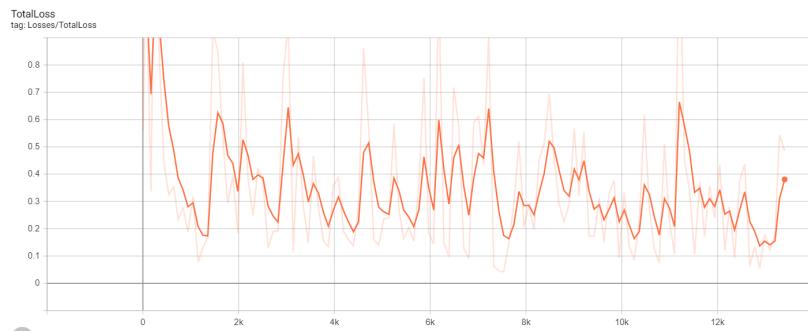


Obr. 3.8: Graf celkovej straty po zmene kroku učenia a pridaní augmentácie dát

Na všetkých grafoch je vidieť, že na začiatku trénovalia hodnota celkovej straty rapídne klesá. Dôvodom je, že sme trénovali nezačali od začiatku, ale od kontrolného bodu predtrénovaného modelu. Hodnoty celkovej straty kolíšu, pretože veľkosť dávky je 1 a strata pre každú iteráciu sa môže mierne lísiť od predchádzajúcej iterácie. Kľúčové je ale to, že celkovo sa počas tréningu tieto hodnoty znížili.

Z celkového porovnania grafov je vidieť, že zmeny ktoré sme vykonali pomohli model zlepšiť. Významne sa model zlepšil po pridaní augmentácie dát, kde je z grafu vidieť, že hodnoty straty kolíšu menej.

Dalej sme sa venovali zmene optimalizátora. Predchádzajúce modely používali optimalizátor s názvom Momentum. V tomto kroku sme skúmali vplyv optimalizátora Adam. Z predchádzajúcich trénovali sme vybrali ten najlepší model a na ňom sme tieto zmeny vykonávali. Najlepším modelom bol tretí model. Ten, ktorému sme znížili krok učenia a pridali ďalšie augmentácie dát. Obrázok 3.9 zobrazuje graf celkovej straty modelu s optimalizátorom Adam.



Obr. 3.9: Graf celkovej straty s optimalizátorom Adam

Hodnoty straty na grafe kolíšu a celkovo sa počas tréningu neznižujú. Z toho vyplýva, že nastala situácia, kedy sa model neučí. Vykonaná zmena optimalizátora teda model nevylepšila. Výsledky presnosti modelu na testovacom datasete sú uvedené v odseku 4.2.2.

faster_rcnn_resnet101_fgvc

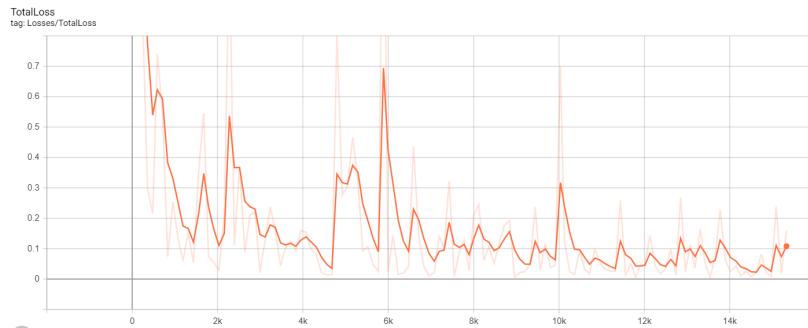
Grafy na obrázkoch 3.10 - 3.12 zobrazujú celkovú stratu počas trénovalia modelu faster_rcnn_resnet101_fgvc. Ako pri predchádzajúcim modeli aj tu sme vykonávali rôzne experimenty s parametrami pôvodného konfiguračného súboru.

Obrázok 3.10 znázorňuje graf modelu s pôvodným nastavením.

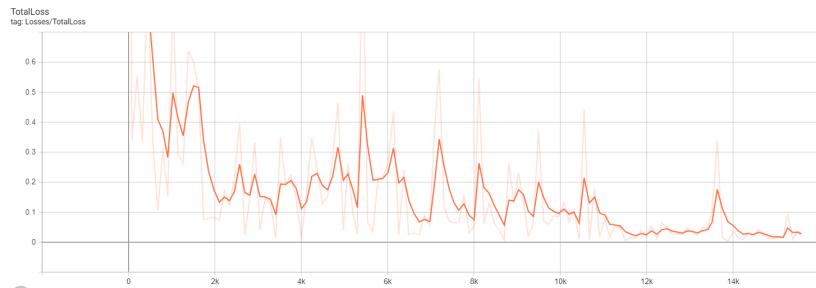
Obrázok 3.11 zobrazuje graf modelu po znížení hodnoty kroku učenia z 0,003 na 0,002.

Obrázok 3.12 zobrazuje graf po pridaní náhodného orezania a náhodného jasu.

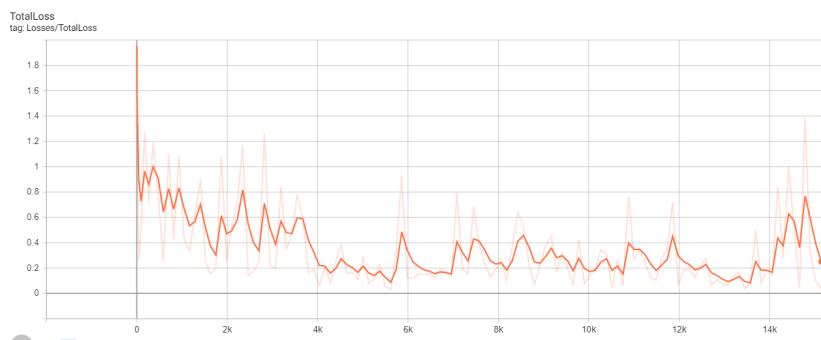
Ako aj pri prvom modeli tak aj pri modeli faster_rcnn_resnet101_fgvc hodnoty celkovej straty kolíšu, avšak celkovo sa znižujú. Pri originálnom nastavení hodnoty



Obr. 3.10: Graf celkovej straty v originálnom nastavení



Obr. 3.11: Graf celkovej straty po zmene kroku učenia

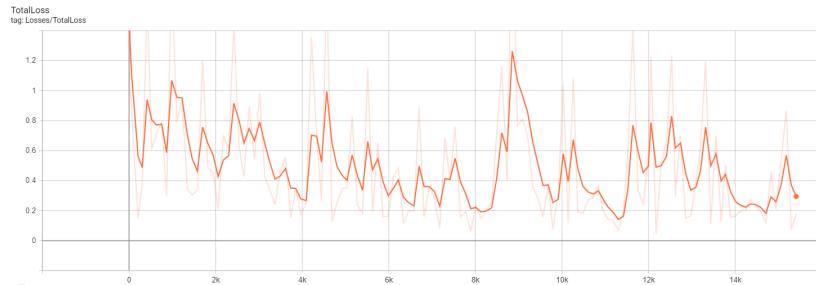


Obr. 3.12: Graf celkovej straty po zmene kroku učenia a pridaní augmentácie dát

straty kolíšu vo väčšej mieri a po 10 000 krokoch sa začínajú ustálovať. Keď sme zmenili krok učenia na menšiu hodnotu, hodnoty straty kolísali v menšom rozmedzí a okolo kroku 12 000 sa začali ustálovať. Ďalšou významnou zmenou bolo pridanie augmentácie dát. Ako je vidieť na obrázku 3.12 hodnoty straty sú mierne kolíšu, ale znižujú sa plynulejšie. Po kroku 14 000 sa začínajú hodnoty straty mierne zvyšovať, čo môže značiť preučenie modelu. Preto očakávame, že model bude mať najlepšie výsledky pri kroku 14 000, kedy je strata ešte dostatočne malá.

Taktiež sme skúmali vplyv optimalizátora Adam. Graf na obrázku 3.13 zobrazuje graf celkovej straty s týmto optimalizátorom.

Z grafu vyplýva, že pri zmene optimalizátora nastal rovnaký prípad ako pri modeli faster_rcnn_resnet101_coco. Optimalizátor Adam spôsobil, že model sa neučil a výsledky iba odhadoval.



Obr. 3.13: Graf celkovej straty s optimalizátorom Adam

3.6 Exportovanie inferenčného grafu

Po ukončení trénovalia sme extraholi zmrazený inferenčný graf, ktorý bude potrebný na vykonanie detekcie. Na exportovanie inferenčného grafu sme potrebovali vybrať najlepší kontrolný bod modelu.

Počas trénovalia modelu vždy zostáva uložených iba posledných 5 kontrolných bodov a staršie sa priebežne vymazávajú. Z toho dôvodu sme v jazyku C# vytvorili aplikáciu, ktorá porovnáva dva priečinky a ak nie sú rovnaké, aplikácia automaticky kopíruje jeden priečinok do druhého. Túto aplikáciu sme využili na to, aby sa nám automaticky uložili všetky vytvorené kontrolné body a nie len posledné. Vďaka tejto aplikácii sme na exportovanie inferenčného grafu mohli vybrať viacero kontrolných bodov modelu spomedzi všetkých. Každý názov kontrolného bodu obsahuje číslo, ktoré zodpovedá počtu vykonaných krokov uskutočnených až do uloženia príslušného bodu. Skript, ktorý sme použili na exportovanie tohto grafu sa volá `export_inference_graph.py`. Tento skript nám vygeneroval súbor formátu .pb s názvom `frozen_inference_graph`. Tento inferenčný graf bol potrebný na vizualizáciu detekcií.

Kapitola 4

Vyhodnotenie a porovnanie modelov

V tejto kapitole sa zaoberáme metrikami na vyhodnocovanie detekcie objektov. Pomocou vybraných metód potom porovnáme natréновané modely. Pri modeloch skúmame, aký vplyv mali datasety na výsledný výkon modelu. Ďalej porovnávame tieto modely s rôznymi nastaveniami konfiguračných súborov. Pre objektívne vyhodnotenie sú modely vyhodnotené a následne porovnávané na samostatnom testovacom datasete. Tento dataset je opísaný v odseku 4.2.1.

4.1 Metriky

Pri vyhodnocovaní detekcie objektov je potrebné zamerať sa na dve odlišné úlohy. Prvou z nich je klasifikácia, ktorá hovorí či detektor správne určil existujúci objekt na obrázku. Druhou je lokalizácia, určujúca ako dobre detektor predikoval umiestenie objektu. Na meranie presnosti lokalizácie sa typicky používa metrika Intersection over Union - IoU. Taktiež je potrebné zaoberať sa aj nesprávnou klasifikáciou objektu. V našom prípade môže detektor nesprávne klasifikovať pozadie ako chodca. Preto je potrebné priradiť skóre spoľahlivosti každému ohraničujúcemu boxu a vyhodnotiť model na rôznych úrovniach spoľahlivosti. Na tieto potreby sa zaviedla metrika priemernej presnosti (Average Precision - AP), ktorá je popísaná v odseku 4.1.5.

Pri klasifikácií môžu nastať štyri možné výsledky:

- Skutočne pozitívny (True positive - TP) - znamená, že objekt bol detegovaný správne
- Skutočne negatívny (True negative - TN) - znamená, že detektor správne neoznačil žiadnen objekt

- Falošne pozitívny (False positive - FP) - znamená, že objekt bol detegovaný, avšak na obrázku sa daný objekt nenachádzal
- Falošne negatívny (False negative - TN) - znamená, že objekt neboli detegovaný, avšak na obrázku sa daný objekt nachádzal

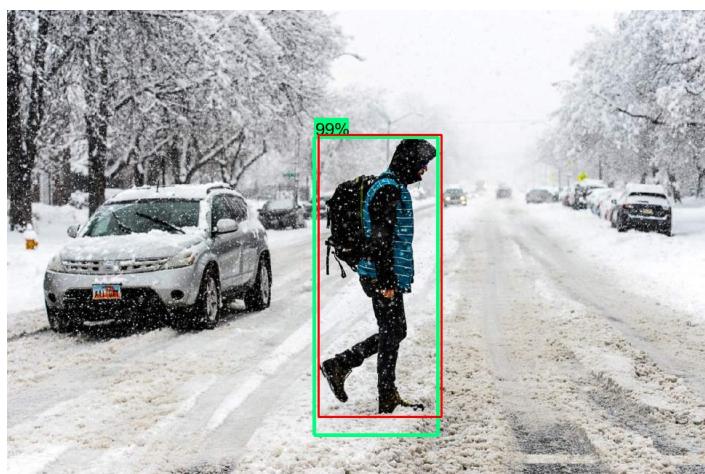
4.1.1 Intersection Over Union - IoU

Je známa metrika na meranie presnosti lokalizácie detektora. IoU určuje do akej miery sa prekrývajú reálne ohraničujúce boxy (ground-truth bounding boxes) s predikovanými ohraničujúcimi boxami. Reálne ohraničujúce boxy sú tie boxy z testovacej sady, ktoré určujú, kde sa objekt skutočne nachádza. Inými slovami IoU určuje ako dobrá je naša predikcia v porovnaní s reálnymi ohraničeniami objektov. IoU sa počíta na základe vzorca uvedeného na obrázku 4.1.

$$\text{IoU} = \frac{\text{Oblast prieniku (prekrytie)}}{\text{Oblast zjednotenia}}$$

Obr. 4.1: Vzorec na výpočet Intersection over Union

IoU sa pohybuje medzi 0 a 1. Čím väčšie je IoU tým je predikcia lepšia. Na obrázku 4.2 je zobrazený príklad, ako môžu reálne a predikované boxy vyzerat.



Obr. 4.2: Príklad detekcie chodca. Reálny ohraničujúci box – červená a predikovaný ohraničujúci box – zelená

4.1.2 Presnosť (Precision)

Presnosť opisuje do akej miery je predikcia správna. Inými slovami, je to pomer správnych detekcií objektov k celkovému počtu detegovaných objektov. Presnosť sa počíta nasledovným vzorcom:

$$\text{Presnosť} = \frac{TP}{TP + FP}. \quad (4.1)$$

Presnosť sa pohybuje medzi 0 a 1. Čím väčšie je toto číslo, tým väčšia je pravdepodobnosť, že všetko, čo detektor predikoval ako pozitívnu detekciu, je naozaj správna predikcia.

4.1.3 Úplnosť (Recall)

Úplnosť opisuje do akej miery je detektor schopný predikovať všetky objekty na obrázku. Je to pomer správnych detekcií objektov k celkovému počtu objektov, ktoré by mali byť predikované. Úplnosť sa počíta nasledovným vzorcom:

$$\text{Úplnosť} = \frac{TP}{TP + FN}. \quad (4.2)$$

Hodnota úplnosti sa pohybuje medzi 0 a 1. Ak sa hodnota bude blížiť k 1, znamená to, že detektor rozpoznal takmer všetky objekty na obrázku.

popri tom môže vynechať skutočné výskytu objektov.

4.1.4 Precision-Recall krivky

Medzi presnosťou a úplnosťou existuje inverzný vzťah. Tieto metriky závisia od nastaveného prahu skóre spoľahlivosti. Model s vyššou prahovou hodnotou generuje menej falošne pozitívnych detekcií ale popri tom môže vynechať skutočné výskytu objektov. Naopak, ak je model príliš citlivý, nájde väčšinu reálnych objektov, avšak vyprodukuje aj veľa falošne pozitívnych detekcií. Preto je potrebné nájsť rovnováhu medzi presnosťou a úplnosťou.

Existuje tzv. precision-recall krivka, ktorá sumarizuje presnosť a úplnosť. Táto metrika poskytuje vyhodnotenie pre celý model. Nastavením rôzneho prahu spoľahlivosti sa vytvoria rôzne páry presnosti a úplnosti, z ktorých vznikne táto precision-recall krivka. Prah spoľahlivosti určuje, čo sa počíta ako pozitívna detekcia.

4.1.5 Average precision - AP

Priemerná presnosť slúži ako metrika na vyhodnotenie výkonnosti detektorov objektov. Jedná sa o metriku s jediným číslom, ktorá zahŕňa presnosť (precision) aj úplnosť (recall) a sumarizuje tak precision-recall krivku. Inými slovami, priemerná presnosť je plocha pod precision-recall krivkou. Čo sa týka využiteľnosti, metrika AP je omnoho

výhodnejšia, pretože je ľahšie porovnávať dve číselné hodnoty ako dve krivky, ktoré majú tendenciu sa pretínať.

V článku PASCAL Visual Object Classes (VOC) Challenge [10] sa priemerná presnosť vypočíta spriemerovaním presnosti (precision) cez sadu jedenástich rovnomerne rozmiestnených hodnôt úplnosti (recall) t.j. $[0, 0.1, \dots, 1]$. Priemerná presnosť sa dá využadiť nasledujúcim vzorcom:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{interp}(r), \quad (4.3)$$

kde

$$p_{interp}(r) = \max_{\tilde{r} > r} p(\tilde{r}). \quad (4.4)$$

To znamená, že sa zodpovedajúca presnosť pre určitú hodnotu úplnosti r interpoluje tak, že sa zoberie maximálna presnosť, ktorej hodnota úplnosti \tilde{r} je väčšia ako r . Táto interpolovaná presnosť sa označuje ako $p_{interp}(r)$.

4.2 Porovnanie modelov

4.2.1 Testovací dataset

Pre potreby objektívneho vyhodnotenia a vzájomného porovnania modelov sme zoštavili testovací dataset. Tento dataset tvorí 50 obrázkov a bol vytvorený tak, aby zohľadňoval široké spektrum podmienok a situácií, v ktorých sa chodci môžu nachádzať a ktoré vplývajú na ich viditeľnosť. Rôznorodosť obrázkov nám tak zabezpečuje transparentnejšie porovnanie modelov.

Dataset obsahuje obrázky chodcov

- v rôznych svetelných podmienkach (slnečno, oblačno, šero, tma)
- v rôznych meteorologických podmienkach (jasno, dážď, hmla, smog)
- patriacich do rôznych rás (černosi, belosi, aziati)
- rôznej postavy (normálni, chudí, obézni, nízki, vysokí, mladí, starí)
- s postihnutím (amputovaná noha)
- s rôznymi doplnkami (okuliare, klobúk, šiltovka, rúško, taška, ruksak, kočík, barle, dáždnik, kapucňa)
- v rôznom postoji (státie, kráčanie, beh, tlačenie predmetu)

4.2.2 Modely s rôznymi konfiguračnými nastaveniami

Pre porovnanie modelov sme použili metriku AP, ktorá bola opísaná v odseku 4.1.5. Konkrétnie sme použili 3 metriky AP, ktoré sú uvedené na stránke [1]. Prvou je metrika

označovaná ako AP , ktorá sa používa pri súťaži COCO Challenge. Zodpovedá prieberu AP cez rôzne prahové hodnoty IoU a to od 0,5 do 0,95 s veľkosťou kroku 0,05. Ďalšia metrika s označením $AP_{IoU=.5}$, je metrika PASCAL VOC, kde sa detegovaný ohraničujúci box považuje za skutočne pozitívny iba ak je $IoU \geq 0,5$ vzhľadom na reálny ohraničujúci box. Posledná metrika s označením $AP_{IoU=.75}$, sa popisuje ako prísna metrika s prahovou hodnotou $IoU \geq 0,75$.

Aby sme zistili či sú modely dostatočne natrénované a ďalej sa pri trénovaní nezlepšujú, vyhodnocovali sme modely aj s rôznymi krokmi trénovania.

faster_rcnn_resnet101_coco

Pre vyhodnotenie modelu faster_rcnn_resnet101_coco sme exportovali inferenčný graf v približne 6000, 10 000 a 13 000 krokoch. Tabuľka 4.3 znázorňuje namerané hodnoty presnosti pre jednotlivé modely s rôznymi nastaveniami v rôznych krokoch trénovania. Zmeny konfiguračných súborov boli nasledovné:

- zmena kroku učenia z 0,003 na 0,002
- pridanie dvoch techník augmentácie dát, konkrétnie random_crop_image a random_adjust_brightness

faster_rcnn_resnet101_coco	6 000 krovok			12 000 krovok			13 000 krovok		
	AP	AP $IoU=.5$	AP $IoU=.75$	AP	AP $IoU=.5$	AP $IoU=.75$	AP	AP $IoU=.5$	AP $IoU=.75$
Rôzne nastavenia									
Originálne	70,2	96,2	82,8	70,8	96,4	82,9	69,5	96,5	82,9
Zmena kroku učenia	71,6	96,4	82,3	70,9	96,7	83,5	70,7	96,6	83,7
Pridanie augmentácie dát	69,4	96,2	80,3	71,1	95,8	82,1	70,3	96,5	80,9

Obr. 4.3: Tabuľka výsledkov presnosti pri rôznych nastaveniach modelov

Nakolko boli modely trénované len na 13 000 krokoch, zmeny hodnôt pri rovnakej metrike nie sú až tak viditeľné. Už po 6000 krokoch dávali všetky modely uspokojivé výsledky. Pri rôznych metrikách sa však hodnoty modelu s rovnakým nastavením medzi sebou výrazne líšia. Všetky modely dávali najlepšie výsledky pri prahu 0,5, kedy sa presnosť blížila k 100%. Najprísnejšou metrikou sa ukazuje byť metrika AP zo súťaži COCO Challenge, kde sú aj rozdiely medzi jednotlivými modelmi najviac vidieť. Preto sa pri výslednom zhodnotení, ktorý model je najlepší, budeme riadiť touto metrikou.

V tabuľke 4.3 sú zvýraznené najlepšie výsledky modelov s rôznymi metrikami. Pri všetkých metrikách sa najlepším modelom ukazuje byť model so zníženou hodnotou kroku učenia. Najprísnejšia metrika AP ukazuje, že model má najlepšie výsledky pri 6000 krokoch.

Tabuľka 4.4 znázorňuje jednotlivé namerané hodnoty s pôvodným optimalizátorom Momentum a zmeneným optimalizátorom Adam. Zmena bola vykonávaná na modeli, ktorý mal zmenený krok učenia a pridané ďalšie augmentácie dát.

Ako už bolo možné vidieť aj z grafu celkovej straty modelu s optimalizátorom Adam

faster_rcnn_resnet101_coco	6 000 krokov			12 000 krokov			13 000 krokov		
	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75
Momentum	69,4	96,2	80,3	71,1	95,8	82,1	70,3	96,5	80,9
Adam	51,5	88,7	56,1	51,9	88,8	55,8	49,6	84,6	55,4

Obr. 4.4: Tabuľka výsledkov presnosti pri rôznych optimalizátoroch

na obrázku 3.9, tak aj z nameraných výsledkov je vidieť, že tento optimalizátor výrazne zhoršil presnosť modelu.

faster_rcnn_resnet101_fgvc

Podobne ako pri modeli faster_rcnn_resnet101_coco tak aj pri modeli faster_rcnn_resnet101_fgvc sme porovnávali rôzne nastavenia originálneho konfiguračného súboru pri rôznych krokoch trénovalia. Nakolko sa tieto modely trénovali na viacerých krokoch, rozhodli sme sa vynechať vyhodnotenie na 6000 krokoch a porovnávať výsledky po približne 12 000, 13 000 a 15 000 krokoch.

Tabuľka 4.5 znázorňuje namerané výsledky pre model faster_rcnn_resnet101_fgvc s nasledujúcimi zmenami konfiguračných súborov:

- zmena kroku učenia z 0,003 na 0,002
- pridanie dvoch techník augmentácie dát, konkrétnie random_crop_image a random_adjust_brightness

faster_rcnn_resnet101_fgvc	12 000 krokov			13 000 krokov			15 000 krokov		
	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75
Rôzne nastavenia									
Originálne	45	81,4	46,2	45	81,6	44	46,4	80,7	51,1
Zmena kroku učenia	47,8	80,8	52,6	47,4	81,9	50,6	47,8	81,8	54,7
Pridanie augmentácie dát	53,4	82,1	63,9	55,2	83,3	63,8	54	82,4	62

Obr. 4.5: Tabuľka výsledkov presnosti pri rôznych nastaveniach modelov

Na výsledkoch sa ukazuje, že zmeny, ktoré sme uskutočnili, originálnemu modelu výrazne zlepšili presnosť. Na nameraných hodnotách je vidieť ako sa našimi zmenami originálny model postupne zlepšoval. V tabuľke sú taktiež vyznačené najlepšie výsledky pre jednotlivé metriky. Najlepším modelom je model s pridaním augmentácie dát a zmenou kroku učenia pri kroku trénovalia 13 000. Tieto zmeny zvýšili presnosť originálneho modelu pri najprísnejšej metrike AP až o 8,8%.

Taktiež sme skúmali vplyv optimizátora Adam. Modelu so zmenením krokom učenia a s pridanou augmentáciou dát sme zmenili pôvodný optimalizátor Momentum na Adam. Tabuľka 4.6 znázorňuje výsledky po tejto zmene.

Tak ako sme očakávali, optimalizátor Adam výrazne zhoršil presnosť modelu, pretože výsledky iba odhadoval.

faster_rcnn_resnet101_fgvc	12 000 krokov			13 000 krokov			15 000 krokov		
Optimalizátor	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75	AP	AP IoU=.5	AP IoU=.75
Momentum	53,4	82,1	63,9	55,2	83,3	63,8	54	82,4	62
Adam	43,9	76,6	46,7	45,5	77,3	51,3	40	73,6	39,9

Obr. 4.6: Tabuľka výsledkov presnosti pri rôznych optimalizátoroch

4.2.3 Porovnanie modelov s modelom Kittí

Z predošlých výsledkov porovnania sme vybrali dva najlepšie modely. Jeden pre faster_rcnn_resnet101_coco a druhý pre faster_rcnn_resnet101_fgvc. Tieto sme následne porovnali s predtrénovaným modelom na datasete Kittí. Na tabuľke 4.7 sú znázornené jednotlivé výsledky.

Modely	AP	AP IoU=.5	AP IoU=.75
faster_rcnn_resnet101_kitti	49,1	92,6	44,5
faster_rcnn_resnet101_fgvc	55,2	83,3	63,8
faster_rcnn_resnet101_coco	71,6	96,4	82,3

Obr. 4.7: Tabuľka výsledkov presnosti

Ako najpresnejší sa ukázal byť model faster_rcnn_resnet101_coco, ktorý bol predtrénovaný na datasete COCO. Z výsledkov vyplýva, že dataset, na ktorom boli tieto modely predtrénované, má veľký vplyv na ich výslednú presnosť.

Zaujímavostou ale je, že aj model faster_rcnn_resnet101_fgvc predtrénovaný na datasete, ktorý neobsahoval chodcov, má po dotrénovaní pomerne uspokojivé výsledky.

4.2.4 Porovnanie modelov z hľadiska rýchlosťi

Ďalšie porovnávanie, ktoré sme uskutočnili, sa zaoberala rýchlosťou modelov. Porovnávali sme modely, ktoré dopadli najlepšie vo vyhodnotení presnosti spolu s predtrénovaným modelom faster_rcnn_resnet101_kitti. Pre toto porovnanie sme vybrali video, na ktorom sme experimentálne porovnávali natrénované modely. Toto video obsahovalo chodcov nasnímaných z automobilu. Video malo dĺžku 2 minúty a spolu obsahovalo 3597 snímkov. V tabuľke 4.8 je uvedená rýchlosť jednotlivých modelov v snímkach za sekundu (frame per second - fps).

Modely	FPS
faster_rcnn_resnet101_kitti	1,99
faster_rcnn_resnet101_fgvc	2,75
faster_rcnn_resnet101_coco	1,25

Obr. 4.8: Porovnanie rýchlosťí modelov

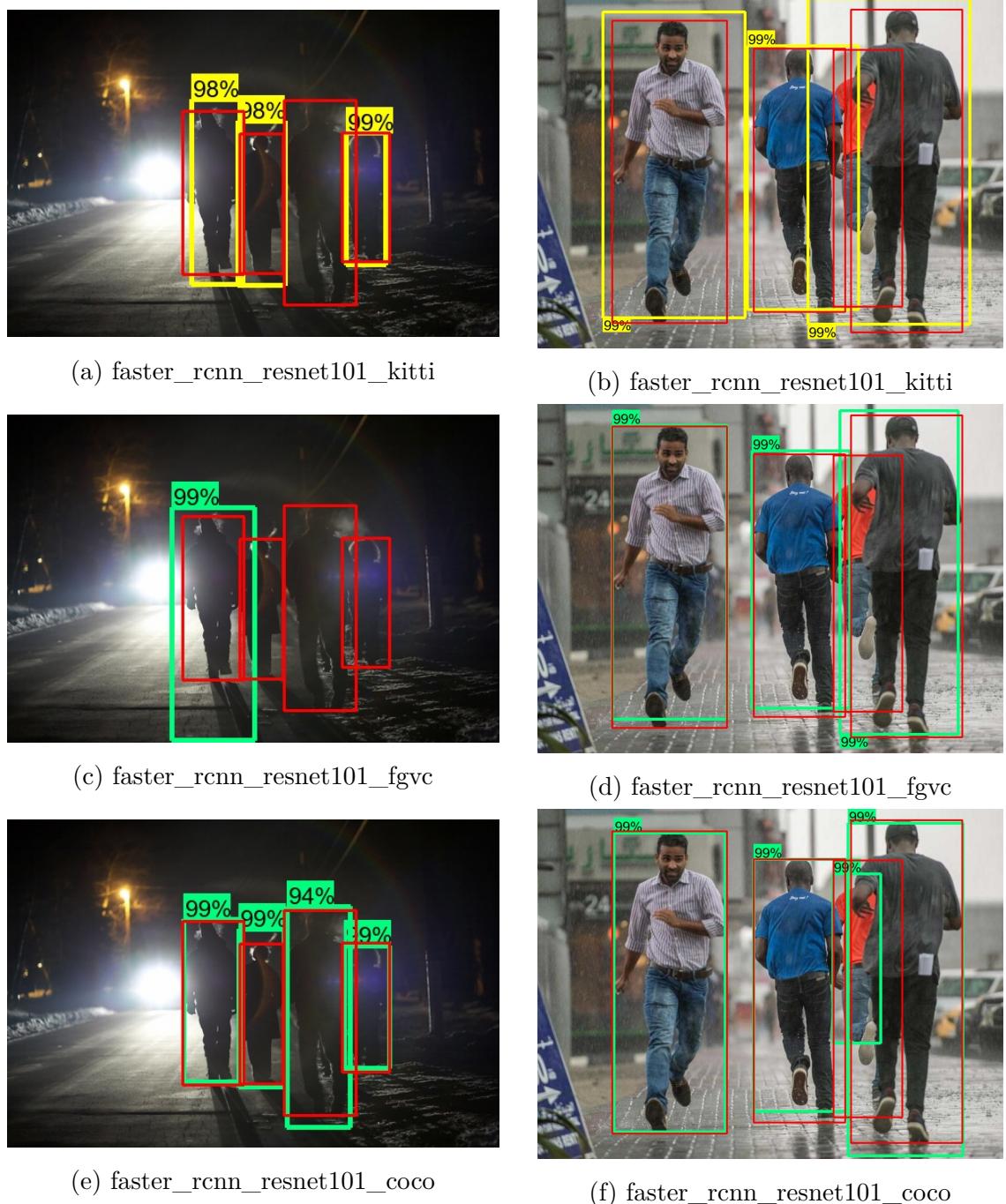
4.2.5 Celkové vyhodnotenie

Z celkového vyhodnotenia vyplýva, že model faster_rcnn_resnet101_coco bol zo všetkých modelov najpresnejší, ale aj najpomalší. Naopak model faster_rcnn_resnet101_fgvc dosahoval najvyššiu rýchlosť, avšak na úrok presnosti. Najzaujímavejším výslednom bolo, že model faster_rcnn_resnet101_fgvc, predtrénovaný na datasete, obsahujúcom iba rastliny a zvieratá, má po dotrénovaní na chodcoch, celkovo uspokojivé výsledky.

4.2.6 Vizualizácia detekcií

Pre vizualizáciu detekcií natrénovaných modelov sme vytvorili dva skripty v jazyku Python. Jeden pre vizualizáciu detekcií na obrázkoch a druhý pre vizualizáciu detekcií na videu. Pre vizuálne porovnanie s reálnymi ohraničovacími boxami sme do skriptu na detekciu obrázkov pridali aj zobrazovanie týchto boxov. Pri spustení skriptov sme potrebovali inferenčný graf, ktorý sme si predtým exportovali.

V tomto odseku budú zobrazené výsledné obrázky, na ktorých bola detekcia vykonávaná pomocou rôznych modelov. Vybrali sme dva reprezentatívne obrázky, na ktorých je vidieť, ako modely dokážu detegovať chodcov v noci a taktiež chodcov s čiastočným prekrytím.



Obr. 4.9: Výsledná detekcia jednotlivých modelov. Reálny ohraničujúci box - červená a predikovaný ohraničujúci box - zelená, žltá

Záver

Hlavným cieľom tejto práce bolo navrhnúť a implementovať metódu na detekciu chodcov na video záberoch nasnímaných z automobilu.

Prvotnou úlohou bolo zoznámenie sa s problematikou cestnej premávky so zameraním sa na detekciu chodcov a analýza existujúcich riešení v dostupnej odbornej literatúre. Problém sledovania cestnej premávky sme si podrobne naštudovali. Zoznámili sme sa s jednotlivými publikovanými metódami na detekciu chodcov, kde sme pobrobné analyzovali vybranú metódu – neurónové siete. Táto bola následne použitá v praktickej časti. V úvode sme tiež vykonali porovnania rozličných existujúcich architektúr neurónových sietí z hľadiska presnosti a rýchlosťi. Pojem a prínosy hĺbkovej informácie boli opísané práve v tejto časti. Taktiež bol rozobraný aj článok, ktorý skúmal jej využitie pri detekcii chodcov.

Praktická úloha spočívala v samotnom návrhu a implementácii metódy. Pre našu prácu sme sa rozhodli použiť neurónové siete. Nakoľko existuje veľmi veľa rámcov na detekciu chodcov, poskytujúcich veľmi dobré výsledky, považovali sme vytvorenie úplne novej neurónovej siete za zbytočný prístup. Preto náš návrh spočíval v použití metódy prenášaného učenia. V nej sme z predtrénovaného modelu využili jeho schopnosť detektovať objekty všeobecne a následne sme túto schopnosť dodatočným učením vyšpecifikovali na detekciu chodcov. Z predtrénovaných modelov, poskytovaných spoločnosťou Google, sme si vybrali tri, pričom každý z nich bol predtrénovaný na inom datasete. Prvý na datasete neobsahujúcom ľudí vôbec, druhý na datasete ľudí obsahujúcim, avšak nie v roli chodcov a tretí, ktorý chodcov už obsahoval.

Poslednou úlohou bolo vyhodnotenie dosiahnutých výsledkov. V rámci tej sme vytvorili testovací dataset, na ktorom sme uskutočňovali vyhodnotenie a porovnávanie natrénovaných modelov. Jednotlivé modely sme na tomto datasete porovnávali z hľadiska presnosti. Evaluácia presnosti modelov bola vyhodnotená pomocou niekolkých metrik, ktoré sa používajú na porovnávanie riešení v rôznych detekčných súťažiach. Na porovnávanie rýchlosťi detekcie modelov sme využili videozáZNAM z palubnej kamery automobilu pohybujúceho sa centrom mesta.

Z hľadiska presnosti sme najskôr porovnávali dotrénované modely s rôznymi zmienami konfiguračných nastavení. Zatiaľ čo na výsledkoch presnosti modelu faster_rcnn_resnet101_coco sa tieto zmeny prejavili len nepatrne, tak modelu faster_rcnn_res-

net101_fgvc tieto zmeny výrazne zlepšili jeho presnosť. Pri modeli faster_rcnn_resnet101_coco sa ukázalo byť najlepšou zmenou zníženie kroku učenia. Pri modeli faster_rcnn_resnet101_fgvc najpozitívnejšia zmena nastala pri znížení kroku učenia spolu s pridaním techník na augmentáciu dát.

Nakoniec sme vykonali porovnanie medzi nami natrénovanými modelmi s najlepšou kvalitou detekcie a predtrénovaným modelom faster_rcnn_resnet101_kitti. Pri porovnávaní sme zohľadňovali presnosť a rýchlosť detekcie. Ukázalo sa, že model faster_rcnn_resnet101_coco bol zo všetkých najpresnejší, ale zároveň aj najpomalší. Naopak model faster_rcnn_resnet101_fgvc bol menej presný, ale vynikal svojou rýchlosťou.

Vykonané experimenty ukazujú, že za pomoci metódy prenášaného učenia sa dá úspešne dotrénovať detekcia určitej triedy aj na modeli, ktorý bol predtrénovaný na datasete neobsahujúcim danú triedu objektov. Takto dotrénovaný model môže dosahovať uspokojivú presnosť a dobrú rýchlosť.

Priestor pre ďalšie možnosti rozširovania práce vidíme v návrhu a implementácií inteligentného systému riadenia dopravy na križovatkách s využitím našej práce, na získanie informácií o množstve chodcov na priechodoch.

Literatúra

- [1] COCO - Common Objects in Context. [online] Dostupné na internete: <http://cocodataset.org/#home>.
- [2] GRAZ-01 dataset. [online] Dostupné na internete: https://cvssp.org/featurespace/web/related_papers/graz1.html.
- [3] KITTI dataset. [online] Dostupné na internete: <http://www.cvlibs.net/datasets/kitti/>.
- [4] Tensorflow detection model zoo. [online] Dostupné na internete: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.
- [5] Rodrigo Benenson, Markus Mathias, Tinne Tuytelaars, and Luc Van Gool. Seeking the strongest rigid detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3666–3673, 2013.
- [6] Guang Chen, Yuanyuan Ding, Jing Xiao, and Tony X Han. Detection evolution with multi-order contextual co-occurrence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1798–1805, 2013.
- [7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, pages 886–893. IEEE, 2005.
- [8] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. 2009.
- [9] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311. IEEE, 2009.
- [10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, pages 303–338, 2010.

- [11] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [12] Tarak Gandhi and Mohan M Trivedi. Pedestrian collision avoidance systems: A survey of computer vision based recent studies. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 976–981. IEEE, 2006.
- [13] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [18] Jonathan Hui. Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). [online] Dostupné na internete: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn_-ssd-and-yolo-5425656ae359.
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [20] Wenbo Lan, Jianwu Dang, Yangping Wang, and Song Wang. Pedestrian detection based on yolo network model. In *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1547–1551. IEEE, 2018.
- [21] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition

- with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.
- [22] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
- [23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [25] Manolis Loukarakis, José Cano, and Michael O’Boyle. Accelerating deep neural networks on low power heterogeneous architectures. 2018.
- [26] Tiago M Nascimento, Davidson R Boccardo, Charles B Prado, Raphael CS Machado, and Luiz FRC Carmo. Program matching through code analysis and artificial neural networks. *International Journal of Software Engineering and Knowledge Engineering*, pages 225–241, 2012.
- [27] Tanguy Ophoff, Kristof Van Beeck, and Toon Goedemé. Improving Real-Time Pedestrian Detectors with RGB+ Depth Fusion. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2018.
- [28] Jean Ponce, Tamara L Berg, Mark Everingham, David A Forsyth, Martial Hebert, Svetlana Lazebnik, Marcin Marszalek, Cordelia Schmid, Bryan C Russell, Antonio Torralba, et al. Dataset issues in object recognition. In *Toward category-level object recognition*, pages 29–48. Springer, 2006.
- [29] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [30] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, 2018.
- [31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

- [32] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, page 386, 1958.
- [33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pages 211–252, 2015.
- [34] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, (1-3):157–173, 2008.
- [35] Shamoona Siddiqui. How would we find a better activation function than ReLU. [online] Dostupné na internete: https://medium.com/shallow-thoughts-about-deep-learning/how-would-we-find-a-better-activation-function-than-relu-_4409df217a5c.
- [36] E. Šikudová, Z. Černeková, V. Benešová, Z. Haladová, and J. Kučerová. Počítacové videnie. detekcia a rozpoznávanie objektov. *Praha: Wikina Praha*, page 397, 2013.
- [37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.
- [38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [39] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. IEEE, 2001.
- [40] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, pages 153–161, 2003.
- [41] Gokce Nur Yilmaz and Federica Battisti. Depth Perception Prediction of 3D Video for Ensuring Advanced Multimedia Services. In *2018-3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–3. IEEE, 2018.

- [42] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
- [43] Xiaoqian Yu, Yujuan Si, and Liangliang Li. Pedestrian detection based on improved faster rcnn algorithm. In *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 346–351. IEEE, 2019.
- [44] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Príloha - Obsah CD

- Datasetsy
- Testovacie video
- Skripty na detekciu
- Obrázky s vykonanou detekciou