VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Lokalizace chodců v dopravě pomocí obrazů

Pedestrian Localization Using Images

Jan Pitala

Bakalářská práce

Vedoucí práce: Ing. Radovan Fusek, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/079090
vsboee220261DA

Zadání bakalářské práce

Student: **Jan Pitala**

Studijní program: B0613A140014 Informatika

Téma: Lokalizace chodců v dopravě pomocí obrazů
Pedestrian Localization Using Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Lokalizace chodců pomocí obrazů je v posledních letech hodně rozvíjené téma. Aplikace, která by řešila tento problém může být použita například v oblasti samořiditelných vozidel.

1. Popište základní pojmy a metody v oblasti lokalizace chodců pomocí obrazů.
2. Seznamte se s volně dostupnými knihovnami a popište jaké možnosti nabízí v této oblasti (například OpenCV, Dlib, TensorFlow, PyTorch).
3. S pomocí knihoven vytvořte vybraný detektor chodců.
4. Experimentálně ověřte funkčnost, přesnost a rychlost navrženého řešení na dostupných datových sadách (případně vytvořte vlastní sadu testovacích a trénovacích dat).
5. Své závěry řádně zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627-1645, 2010
[2] Ren, Shaoqing, He, Kaiming, Girshick, Ross B. and Sun, Jian. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.." Paper presented at the meeting of the NIPS, 2015

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radovan Fusek, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 07.11.2022 12:25:53

Abstrakt

Cílem této práce je seznámení s metodami pro lokalizaci chodců, které jsou již v dnešní době všude okolo nás a pomáhají lidem zlepšovat život. V budoucnu budou takové metody nedílnou součástí každodenního života. Práce se bude zejména zaměřovat na konkrétní metody, srovnání přesnosti a časové náročnosti mezi sebou. Implementace detektorů pro detekci jednotlivých chodců bude v různých knihovnách pro jazyk Python. Trénink modelů bude z množiny snímků z datasetů ECP, Cityscapes a PRW. Testování jejich funkčnosti bude následně otestováno na množinách snímků ECP datasetu a Cityscapes datasetu.

Klíčová slova

Cityscapes dataset; detekce chodců; ECP dataset; Faster R-CNN; HOG; příznaky Haar; PRW dataset; Python; SSDMobileNet; YOLO

Abstract

The goal of this work is to get acquainted with methods for locating pedestrians which are already all around us today and help people improve their lives. In the future, such methods will be an integral part of everyday life. The work will mainly focus on specific methods, comparison of accuracy and time requirements among themselves. The implementation of detectors for the detection of individual pedestrians will be in different libraries for the Python language. The training of the models will be from a set of images from the ECP, Cityscapes and PRW datasets. Testing their functionality will subsequently be tested on sets of images of the ECP dataset and the Cityscapes dataset.

Keywords

Cityscapes dataset; ECP dataset; Faster R-CNN; Haar-like features; HOG; Pedestrian Detection; PRW dataset; Python; SSDMobileNet; YOLO

Poděkování

Rád bych poděkoval mému vedoucímu bakalářské práce Ing. Radovanu Fuskovi, Ph.D. za ochotu a spolupráci. Dále bych chtěl poděkovat své rodině za podporu a motivaci k dokončení této práce.

Obsah

Seznam použitých zkratek a symbolů

ACF	–	Aggregated Channels Network
AdaBoost	–	Adaptive Boosting
API	–	Application Programming Interface
BSD	–	Berkeley Software Distribution
cls	–	classification layer
CNN	–	Convolutional Neural Network
COCO	–	Common Objects in Context
Deep SORT	–	Simple Online and Realtime Tracking with a Deep Association Metric
ECP	–	EuroCity Persons dataset
E-ELAN	–	Extended Efficient Layer Aggregation Network
ELAN	–	Efficient Layer Aggregation Network
FN	–	false negative
FP	–	false positive
HOG	–	Histogram of Oriented Gradients
IoU	–	Intersection over Union
LBP	–	Local Binary Pattern
mAP	–	Mean Average Precision
PRW	–	Person Re-identification in the Wild dataset
R-CNN	–	Region-based Convolutional Neural Network
reg	–	regression layer
RPN	–	Region Proposal Network
SSD	–	Single Shot Detector
SVM	–	Support Vector Machine
TN	–	true negative
TP	–	true positive
VGG	–	Visual Geometry Group
YOLO	–	You Only Look Once

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tématem této bakalářské práce je detekce chodců v dopravě pomocí obrazů. Ta se věnuje způsobům, jak detekovat chodce nebo obecně osoby v dopravě. Využívá k tomu strojové učení a počítačové vidění k lokalizaci určitého objektu na obrázcích nebo videích a přiřazují jim určitou polohu, kde se tyto objekty vůči obrazu nacházejí.

Pokud se bavíme o počítačovém vidění, tak jde o obor, který se zabývá využíváním obrazu a videí k získání informací, jako je například detekce, klasifikace a sledování objektů nebo událostí, ze zachyceného zařízení. Metody, které se můžou k tomu používat, se zabývají nalezením vzorového objektu ve snímku nebo videosekvenci. Algoritmy se snaží vyhledat podobné prvky, které by mohly odpovídat hledanému vzoru. Poté můžeme dostat z algoritmu polohu objektu, kde se právě nachází vůči snímku. Tyto algoritmy používají detektor, který je nejdříve natrénován na množině vzorových dat a na ní se naučí detekovat konkrétní typ objektů. Poté lze detektor použít na nové snímky. Mezi detektory můžeme zařadit kaskádový detektor objektů s využitím příznaků typu Haar, *Aggregated Channels Network* (ACF) detektor, YOLO, MobileNetSSD a spoustu dalších. Počítačové vidění se používá například v oblastech průmyslových robotů, autonomních vozidel, detekci jevů, modelování objektů a spoustu dalších. [1]

Proto cíle této práce jsou seznámení se základními pojmy využívaných v analýze obrazu, metodami, které budou použity pro detekci chodců v dopravě a seznámení s datovými sadami, na kterých budou detektory natrénovány a ověřovány. U detektorů se bude ověřovat jejich funkčnost, přesnost a rychlost. Přesnost a úspěšnost těchto detektorů objektů záleží na různých faktorech, jako jsou kvalita kamerového systému, zhoršená viditelnost těchto zařízení v noci při špatném osvětlení nebo za zhoršeného počasí, kvalita natrénování určitého detektoru, který se stará o samotnou lokalizaci těchto objektů a spoustu dalších.

Rozpoznání těchto objektů, v tomto případě chodce, mohou využívat vozidla, která pomocí obrazových senzorů, jako je například kamerový systém, mohou detekovat případné hrozby. To by pomohlo k bezpečnosti na silnici a popřípadě zabránit následnému střetu vozidla s chodcem nebo alespoň minimalizovat škody, jak na lidských životech, tak i na vozidlech. Tato technologie by se pak

dala aplikovat, někde se již používá, na autonomních vozidlech nebo jako jeden z jízdních asistentů, který by mohl pomáhat samotnému řidiči s řízením vozidla. Samozřejmě by tento asistent musel být doplněn dalšími asistenty, protože na cestách se jako překážky nevyskytují pouze chodci, ale také to mohou být i jiné objekty, jako jsou například kočárky, kola, zvířata, vozíky. Dále by se tyto detektory daly využívat u kamer, které se nacházejí u semaforů s přechodem pro chodce a mohly by tak monitorovat prostor, kde se lidé nacházejí, než chtějí přejít silnici. Jejich lokalizace by na určitém místě u kraje vozovky mohla pomoct k plynulosti v dopravě, aby semafory zbytečně nezastavovaly silniční provoz, když by nebyl chodec, který by chtěl přejít přes cestu.

Detekce lidí a následné správné rozpoznání nemusí být úplně jednoduché. Jde to vidět na obrázku 1.1, který je z publikace *Is Faster R-CNN Doing Well for Pedestrian Detection?* [2] a využívá jednu z metod pro detekci lidí, která bude v následující kapitole popsána. V levé části se nachází deset detekcí, které správně detekovaly člověka. Na pravé straně jde vidět dalších deset detekcí, které již nedetekovaly správně člověka a byly za něho zaměněny. Mezi často špatně detekované objekty, které jsou zaměňovány za člověka, jsou různé značky, pouliční lampy, stromy, semafory, poštovní schránky, reklamní plochy.

(a) Small positive instances (b) Hard negatives

Obrázek 1.1: Příklady správných detekcí lidí (a) a nesprávných detekcí lidí (b) [2]

Kapitola 2

Metody pro lokalizaci objektů pomocí obrazů

Jak bylo zmíněno v úvodu, práce se zabývá lokalizací chodců v obrazových datech. Digitální obrazová data mohou být zachycena různými způsoby jako například z kamery či fotoaparátu, ultrazvuku a dalších lékařských zařízení. Na obrazová data se během lokalizační fáze může použít mnoho různých algoritmů jako je odstranění šumu, změna rozlišení, detekce hran, vylepšení obrazu a segmentace obrazu, tudíž se můžeme vyhnout problémům jako je hromadění šumu nebo zkreslení během zpracování. Kromě úprav a předzpracování snímků se také zpracování obrazu věnuje detekci objektů. Oblast využití je velmi pestrá. Můžeme se setkat s digitálním zpracováním obrazu v oblasti robotiky, počítačového vidění, lékařství a mnoho dalších. [3]

V posledních letech se obrazová data velmi často kombinují s algoritmy strojového učení. Cílem tohoto učení je, aby se počítače díky dostupným datům a algoritmům dokázaly učit a neustále se zlepšovat. Snaží se napodobit způsoby lidského mozku a jeho učení. Existují dva způsoby lidského učení. První z nich je memorování, což je učení se textu nazpaměť, a to není z pohledu strojového učení zajímavé. Strojové učení se zajímá o druhý způsob lidského učení, a to je generalizace nebo zobecnění na základě určitých zkušeností. Čím má více dat, ze kterých se může učit, tím více se zlepšuje jeho přesnost. Algoritmy jsou trénovány, aby se snažily najít korelace mezi velkým množstvím dat a také se snaží najít určité vzory. Na základě těchto informací se naučené algoritmy snaží dělat nejlepší odhady a predikce. Algoritmy lze rozdělit podle způsobu zpracování na dávkové, což znamená, že jsou všechna data požadována před začátkem výpočtu a potom máme inkrementální, kde se dokážou algoritmy „přiučit“ při dodání nových dat a nemusí se přepočítávat celý model od začátku, ale pouze ho lehce upravit. Strojové učení má široké uplatnění. Můžeme ho použít například pro rozpoznávání obrazů, elektrických a akustických signálů, v lékařství pro diagnostiku onemocnění, filtrování spamu, kompresi dat. [4]

V následujících kapitolách budou probrány principy konkrétních metod pro lokalizaci objektů v různých knihovnách pro programovací jazyk Python.

2.1 Viola-Jones detektor

Tento detektor využívá metody a algoritmy pro detekci objektů v obraze. Jako první s tímto přístupem přišli *P. Viola a M. Jones*, a publikovali to ve své práci s názvem *Rapid Object Detection using a Boosted Cascade of Simple Features* [5] v roce 2001. Hlavní myšlenkou této metody je, že klasifikátor postupně prochází náš obraz pomocí malých obdélníkových oken, tato metoda se nazývá *sliding window*[1], a u každého obdélníkového okna se snaží zjistit, jestli se tam náš hledaný objekt nachází či nikoliv. Snímky jsou procházeny ve stupních šedi a při trénování používá sadu snímků pozitivní a negativní, to znamená, že na pozitivních jsou zaznačeny správné detekce objektů a na negativních by se objekty neměly nacházet.

Při zjišťování prochází několika úrovněmi a snaží se co nejrychleji zahodit ty části podokna[2], které nesedí detekci. Pokud klasifikátor rozpozná určité vlastnosti objektu, tak se dostane do další úrovně a následně může být podokno označeno jako hledaný objekt. Pokud ne, tak bude podokno vyřazeno a klasifikátor předpokládá, že se tam hledaný objekt nenachází.

Tento detektor využívá příznaky typu Haar, pomocí kterých se snaží detekovat objekt. Dále využívá algoritmus AdaBoost pro trénování detektoru a vytvoření silného klasifikátoru. Také používá kaskádu klasifikátorů a integrální obraz pro rychlejší nacházení detekcí. V praktické části jsem pro tento detektor použil knihovnu OpenCV [6] v kombinaci s jazykem Python.

Kaskádové klasifikátory nejčastěji využívají přístup pomocí příznaků typu Haar, HOG a LBP. V této práci se budu detailněji zaobírat příznaky typu Haar a HOG. Tyto klasifikátory se dají použít například pro detekci obličejů, lidí, rozpoznávání písmen a čísel.

Vědecká práce, která se zabývala detektorem chodců založeným na principech této metody, se jmenuje *Detecting Pedestrians Using Patterns of Motion and Appearance* [7] a na obrázku 2.1 můžeme vidět prvních 5 příznaků typu Haar, které byly naučeny pro jejich detektor chodců.

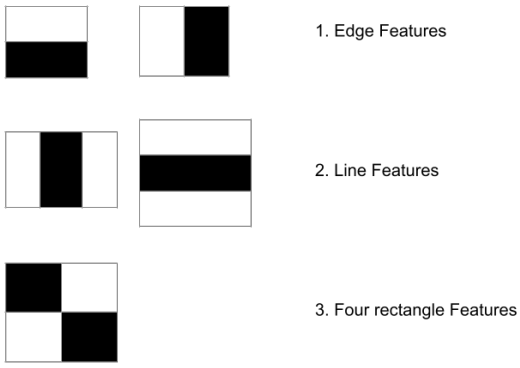
Obrázek 2.1: Ukázka prvních 5 příznaků typu Haar z publikace *Detecting Pedestrians Using Patterns of Motion and Appearance* [7]

2.1.1 Příznaky typu Haar

Na začátku potřebuje detektor spoustu pozitivních i negativních obrázků, aby se detektor mohl na nich natrénovat a vygenerovat si příznaky. Autoři se rozhodli využít příznaky, protože jsou rychlejší

[1] Metoda, kde vstupní obraz je procházen menšími obdélníkovými okny různých velikostí
[2] Výřez obrazu během lokalizačního procesu pomocí techniky *sliding window*

než metoda založená pouze na pixelech. Tyto příznaky mají různou velikost obdélníku a každý obdélník může obsahovat jiný příznak typu Haar. Dělí se podle toho, jaký typ informace obsahují. Může se jednat o hranové příznaky, čárové příznaky a příznak čtyř obdélníků. Tyto typy příznaků jdou vidět na obrázku 2.2.



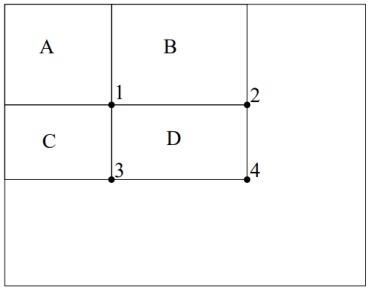
Obrázek 2.2: Ukázka příznaků typu Haar, kde (1) označuje hranové příznaky, (2) čárové příznaky a (3) příznak čtyř obdélníků [8]

Každé okno o rozměrech 24×24 obsahuje přes 180 000 příznaků. Pro každý výpočet by detektor musel najít součet pod bílými i černými obdélníky. Takové výpočty by zabraly nesmírně hodně času. Proto, aby vyřešili tento problém, tak autoři přišli se zavedením integrálního obrazu. To pomůže v tom, že jakkoliv je obrázek velký, omezí výpočty pro daný pixel na práci zahrnující pouze čtyři pixely. [9]

2.1.2 Integrální obraz

Před začátkem se vypočítá integrální obraz jako krok předběžného zpracování. Tím se detektor vyhne nadměrnému sčítání, a tudíž nám zrychlí celkový čas. Pro výpočet každého bodu se v obrázku počítá hodnota všech pixelů, které se nacházejí v obdélníku začínajícího v pravém horním rohu snímku. Obdélník má velikost až po bod, který chceme spočítat.

Když budeme vycházet z obrázku 2.3, tak součet pixelů v oblasti s písmenem D by se dal vypočítat tak, že by se od hodnoty s číslem 4 odečetla číselná hodnota 2 a 3 a následně by se přičetla číselná hodnota 1. Pro získání jednotlivých číselných hodnot z obrázku se použije součet, kde hodnotu 4 získáme součtem všech pixelů ze všech 4 oblastí, které jsou znázorněny písmeny v obrázku. Pro hodnotu 3 pomocí součtu pixelů v oblasti A a C, pro hodnotu 2 součtem pixelů v oblasti A a B a hodnotu 1 dostaneme součtem všech pixelů v oblasti A. [5]



Obrázek 2.3: Ukázka výpočtu součtu hodnot v integrálním obrazu [5]

2.1.3 AdaBoost

Adaptive Boosting (AdaBoost) je metoda použitá při trénování kaskádového klasifikátoru. Pro každý příznak se snaží najít nejlepší prahovou hodnotu. Hledá takovou hodnotu, která dosahuje co možná nejmenší chybovosti při posuzování a určování hledaného objektu, jestli se na určitém místě nachází či nikoliv. Při tomto procesu je každému obrázku na začátku přidělena stejná váha. Po každé klasifikaci se váhy špatně klasifikovaných obrázků zvyšují. Poté se proces provede znovu i s novými váhami. Toto se stále opakuje, dokud není dosaženo požadované přesnosti nebo chybovosti. Může to také skončit, když je nalezen požadovaný počet příznaků. Detektor, který byl představen autory, obsahoval okolo 6 000 příznaků. To znamená, že z původního počtu 180 000 příznaků byl tento počet pomocí algoritmu AdaBoost redukován na 6 000 příznaků. Původní příznaky se také nazývají slabé klasifikátory a výstup je binární, tudíž to znamená, že dokážou pouze určit, jestli poznal hledaný objekt nebo ne. Proto AdaBoost konstruuje silný klasifikátor, který se skládá kombinací slabých klasifikátorů, a tyto silné klasifikátory jsou schopny klasifikovat hledaný objekt. [5, 9]

2.1.4 Kaskáda klasifikátorů

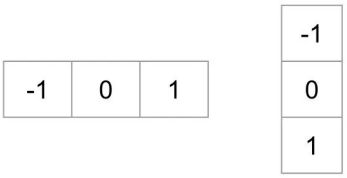
Předchozí metoda nám snížila počet příznaků o znatelnou část, čím dost urychluje detekci hledaného objektu, ale stále, kdyby detektor chtěl použít všech 6 000 příznaků najednou, by to bylo enormně časově náročné. Proto autoři zavedli koncept kaskády klasifikátorů. Ten je aplikován pouze na malou obdélníkovou oblast z celkového snímku, který je získán pomoci metody *sliding window*. Jedná se o seskupení příznaků do různých etap a každé podokno (výřez obrazu) je postupně procházeno několika etapami. V prvních etapách se nachází pouze pár seskupených příznaků a počet seskupení se zvyšuje s každou další etapou. Cílem je, co nejdříve se zbavit podoken, kde se nenachází náš hledaný objekt, aby se nedostal do dalších etap. Takže pokud podokno selže v první etapě detekce, tak ho detektor zahodí a jde na další podokno. Toto nám ušetří velké množství času při detekci, protože detektor nemusí aplikovat další příznaky. Pokud podokno projde první etapou, tak se dostane do další, kde je aplikováno více příznaků a postupuje neustále dál a dál, dokud neprojde všemi etapami. Jakmile projde všemi etapami, tak je tato část obrazu přijata a pravděpodobně se tam nachází hledaný objekt. [10]

Autoři v publikaci [5] uvádí, že detekční kaskáda obsahovala celkem 38 etap, kde bylo přes 6 000 příznaků. Počet příznaků v prvních pěti etapách bylo 1, 10, 25, 25 a 50. V každé další etapě pak bylo více a více příznaků. Autoři také poznamenali, že průměr použitých příznaků na jedno podokno byl 10.

2.2 Histogram orientovaných gradientů

Histogram orientovaných gradientů, dále jen HOG, je vizuální deskriptor používaný v počítačovém vidění za účelem detekce objektů. Tento detektor počítá směr a velikost gradientu v určitých částech obrazu. Tento koncept byl poprvé popsán *R. K. McConnelem*, ale bez použití výrazu HOG v roce 1986. Později roku 2005 byla tato metoda publikována ve vědecké publikaci od *N. Dalala a B. Triggsem* s názvem *Histograms of oriented gradients for human detection* [11], kde už byl výraz HOG použit a popsán. V publikaci popisují práci detektoru, který představují na detekci lidí. Nejčastěji využívané knihovny pro metodu HOG jsou OpenCV [6] a Dlib [12].

Aby detektor mohl detekovat hledaný objekt, musí se provést celá řada kroků. Nejdříve se provede předzpracování obrazu, kde dochází k normalizaci hodnot gamy a barev. Autoři, kteří publikovali tuto metodu, dodávají, že tento krok se dá vynechat, protože v následné normalizaci HOG deskriptoru dosáhneme stejného výsledku. Tudíž předzpracování má zanedbatelný dopad na výkon. Proto se spíše uvádí, že prvním krokem je výpočet hodnot gradientu, které se počítají pomocí kernelů 2.4 v horizontálním i vertikálním směru. Autoři testovali i jiné kernely jako je 3×3 nebo v diagonálním směru, ale obecně vedly k horší detekci. [13]



Obrázek 2.4: Kernely pro výpočet gradientu [10]

Autoři ve své publikaci [11] uvedli, že dále jsou informace o hranách obrazu (gradienty) zakódovány pomocí bloků a buněk. Blok může mít například velikost 16×16 pixelů a skládal by se ze čtyř buněk, kde jedna buňka má velikost 8×8 pixelů. Z těchto buněk se poté získávají histogramy, které se v rámci bloku normalizují. Histogram obsahuje 9 oblastí, také nazývané jako biny a jsou rozděleny po 20 od 0 do 160. Orientace gradientu určuje do jakého binu budou zapsány velikosti gradientu. Orientace gradientu je ve stupních v rozmezí od 0 do 360. Autoři pro zjednodušení i zrychlení výpočtu použili rozsah do 180 stupňů, což vedlo jenom k zanedbatelnému zhoršení.

Jako další krok autoři uvedli, že se provádí normalizace v rámci bloku. Z každého bloku je získán příslušný vektor příznaku. Kombinované příznaky jsou poté předány *Support Vector Machine* (SVM), která zajišťuje detekci objektů.

2.3 YOLO

YOLO se řadí mezi nejoblíbenější algoritmy pro detekci objektů v reálném čase. Využívá k tomu konvoluční neuronové sítě (CNN). Na rozdíl od kaskádového klasifikátoru dokáže YOLO v jednom kroku detekovat a lokalizovat objekt. Pro predikci objektů používá *anchor* boxy. Ty budou detailněji popsány v následující části této kapitoly 2.3.1. Díky své přesnosti a rychlosti detekovat více objektů se YOLO dostalo až do užívání v každodenním životě, jako je například jízdní asistent ve vozidlech, k detekci obličejů a osob a k detekci bezpečnostních hrozeb. S touto metodou poprvé přišli *X. Zhu, S. Lyu, X. Wang, Q. Zhao* v roce 2015 a publikovali to ve své práci s názvem *You Only Look Once: Unified, Real-Time Object Detection* [14]. Detekci chodců s využitím metody YOLO zveřejnili autoři *W. Lan, J. Dang, Y. Wang, S. Wang* ve své vědecké publikaci s názvem *Pedestrian Detection Based on YOLO Network Model* [15]. V této práci se detailněji budu zabývat verzí YOLOv7 v knihovně PyTorch.

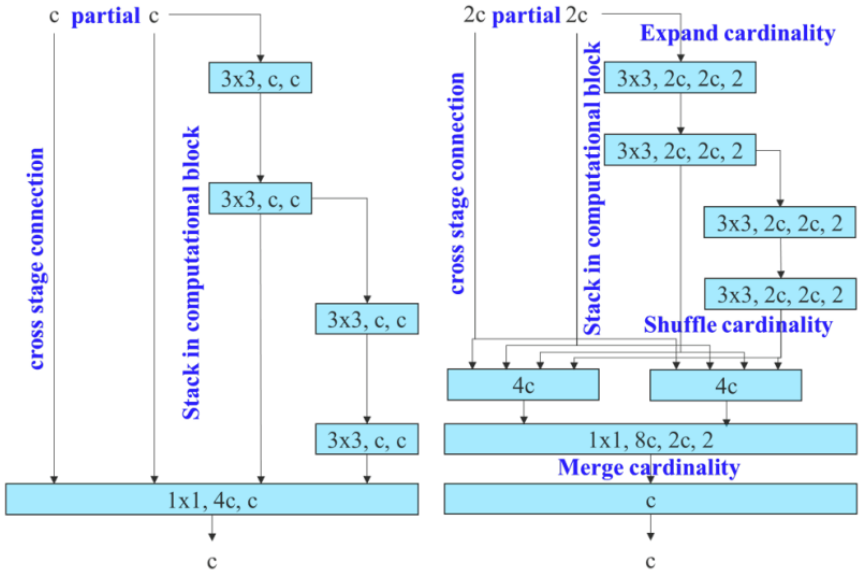
2.3.1 Anchor boxy

YOLO využívá *anchor* boxy, což jsou obdélníky, které jsou již předdefinované z tréninku modelu a mají různou velikost, tj. šířku a výšku, aby zachytily různě velké objekty s různým měřítkem a velikostí. U tréninku modelu se může nastavit několik velikostí *anchor* boxů, které budou využívány pro detekci. Nebo se také může zvolit možnost, že se velikosti *anchor* boxů budou volit samy podle nejčastějších pozitivních detekcí z tréninkových sad. Při detekci jsou *anchor* boxy rozloženy přes obraz. Síť předpovídá pravděpodobnost výskytu detekce a vypočítává *Intersection over Union* (IoU) pro každý *anchor* box, což udává hodnoty překrytí dvou boxů. IoU bude vysvětleno v kapitole 4.2. Síť přímo neurčí ohraničující boxy, ale určí pravděpodobnost a upřesní výskytu objektu. Vrací jedinečnou sadu předpovědí pro každý *anchor* box. Konečná mapa příznaků představuje detekci pro každou třídu. Výhoda těchto boxů je, že dokážou vypočítat predikci všech objektů najednou. To dokáže urychlit pracovní čas detektoru, a tudíž to bude daleko rychlejší než již dříve zmíněný *sliding window*, který počítá predikci pro každý potenciální výskyt detekce. [16]

2.3.2 Architektura YOLOv7

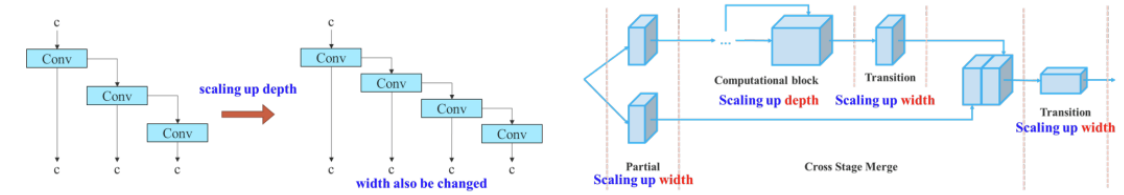
YOLOv7 představili autoři *C. Wang , A. Bochkovskiy a H. M. Liao* ve své práci *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors* [17] v roce 2022. V následující části popíšu techniky, které autoři využili v novém detektoru.

Podle autorů ve většině literatur je hlavními úvahami o efektivním navržení architektury počet parametrů, množství výpočtů a inferenční rychlost. Staví na výzkumu, který proběhl na toto téma. Potřebují, aby množství paměti, které je potřeba k udržení vrstev v paměti, spolu se vzdáleností, kterou potřebuje gradient k šíření mezi vrstvami, nebylo příliš velké, ale bylo efektivní. [18] Nakonec použili architekturu *Extended Efficient Layer Aggregation Network* (E-ELAN), která vychází z původní architektury ELAN. ELAN architektura měla za cíl návrh efektivní sítě řízením nejkratší a nejdelší gradientní cesty, aby se mohly hlubší sítě sbližovat a efektivně se učit. E-ELAN upravuje architekturu ve výpočetním bloku, aby došlo k lepším výsledkům a lepší schopnosti učení modelu bez zničení gradientní cesty z původního modelu. Ukázky obou architektur jdou vidět na obrázku 2.5. [19]



Obrázek 2.5: Architektury *ELAN* (vlevo) a *E-ELAN* (vpravo) [17]

Dalším důležitým konceptem u této metody je měřítko modelu. Díky změně měřítka lze zvětšit hloubku, změnit výšku a šířku. Hloubka měřítka znamená, že chceme zvýšit nebo snížit počet vrstev v modelu. Podobně šířka odpovídá škálování počtu kanálů v architektuře modelů. Faktory měřítka jsou již definované v souborech architektury modelu. YOLOv7 má architekturu, kde jsou vrstvy zřetězeny, proto při škálování hloubky výpočetního bloku, musí vypočítat změnu ve výstupních jádrech. Poté bude úprava šířky změněna o stejnou hodnotu, jako vypočítaná změna v jádrech. To zachová původní vlastnosti architektury a optimální strukturu. [19] Na obrázku 2.6 jde vidět, že u modelů (a) a (b) při hloubkové změně dochází k zvětšení výstupní šířky a jelikož jsou vrstvy zřetězeny, tak dojde i zvětšení vstupní šířky v následné vrstvě. Proto autoři navrhují model (c), kde mění hloubku pouze ve výpočetním bloku a zbývající část přenosové vrstvy se provádí s odpovídajícím měřítkem šířky.



Obrázek 2.6: Měřítka modelů pro zřetězené modely [17]

2.4 MobileNetSSD

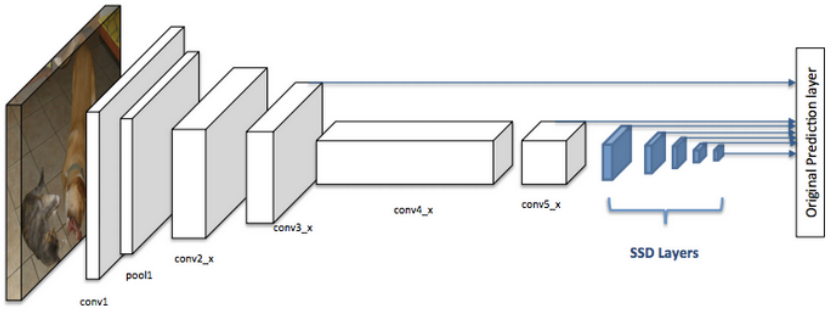
MobileNetSSD je model pro detekci objektů, který se snaží lokalizovat a rozpoznat třídu objektů ze vstupního obrazu. Tento model je speciálně optimalizován, aby mohl fungovat na mobilních zařízeních nebo tabletech, tudíž musí být velmi rychlý, přesný a paměťově nepříliš náročný. Skládá se z kombinace dvou částí. Jedna část je konvoluční neuronová síť zvaná MobileNet a druhá část je algoritmus pro detekci objektů s názvem *Single Shot Detector* (SSD). Algoritmus pro detekci byl představen ve vědecké práci s názvem *SSD: Single Shot MultiBox Detector* [20]. Původní SSD využívala konvoluční neuronovou síť s názvem VGG16 [21]. Dále v této práci budu zkoumat rychlejší verzi s názvem MobileNet. Použití modelu MobileNetSSD je velmi pestré. Může se jednat o detekci objektů, dopravních značek, rozpoznávání zvířat, precizní klasifikaci atd. V této práci budu zkoušet funkčnost detektoru v knihovně TensorFlow.

Detailní fungování sítě MobileNet je popsáno ve vědecké práci s názvem *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* [22], kde to kombinují s detekčními algoritmy SSD a Faster R-CNN. Zkráceně to je sada speciálních CNN, které jsou optimalizovány pro rychlou a vysokou přesnost. Toho se snaží docílit díky snížení parametrů, zmenšení výpočetní náročnosti a paměťového prostoru.

Algoritmus SSD je založený na dopředné konvoluční síti, která generuje soubor ohraničujících boxů tříd kolem instancí objektů a jejich skóre, které udává, s jakou pravděpodobností se jedná o daný objekt. Boxy obsahují hodnoty x, y, jejich šířku a výšku. Skóre obsahuje hodnoty pravděpodobností z každé kategorie objektů. Hodnota nula představuje pozadí. SSD obsahuje multireferenční techniky, které definují sadu *anchor* boxů různých velikostí a poměrů stran a na tomto základě předpovídají ohraničující box. Také obsahuje techniky s více rozlišeními, což napomáhá detekovat objekty ve více měřítkách a na různých síťových vrstvách. SSD implementuje algoritmus, který provádí úpravy v jednotlivých boxech, aby více odpovídaly skutečným tvarům objektu. [23] V následující části budou některé koncepty popsány.

Na obrázku 2.7 jde vidět prvních pár vrstev MobilNet, což tvoří páteř[3] detektoru a modré vrstvy představují hlavu[4] SSD.

[3] Anglicky *backbone*
[4] Anglicky *head*



Obrázek 2.7: Architektura konvoluční neuronové sítě s SSD detektorem [24]

2.4.1 Koncepty využívané u SSD

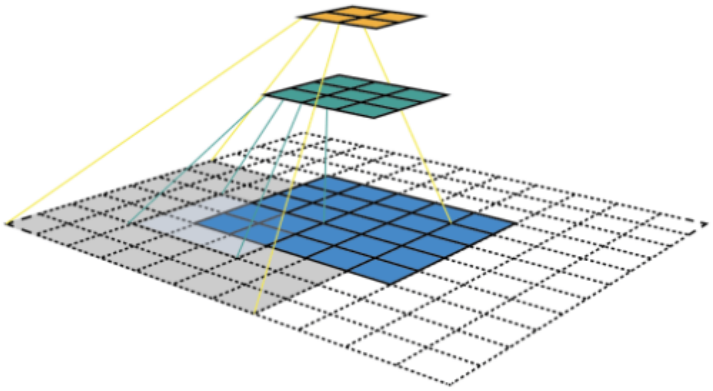
Na rozdíl od detektoru s příznaky typu Haar, který používal *sliding window*, tak SSD rozděluje obraz do mřížky a každá část je zodpovědná za detekci a předpovědění třídy v její oblasti. Pokud se v dané buňce nenachází žádný detekční objekt, tak je za třídu považováno pozadí a buňka bude ignorována. Může se stát, že hledaný objekt bude přes několik buněk mřížky nebo se v jedné buňce bude nacházet více objektů. O to se starají *anchor* boxy a receptivní pole.

Anchor boxy se starají o samotnou detekci a jsou zodpovědné za velikost a tvar hledaných objektů v mřížce. Můžeme přiřadit více těchto boxů v každé buňce nebo boxy můžou detekovat objekt přes více buněk. Termín *anchor* box byl již vysvětlen v kapitole 2.3.1 a u tohoto detektoru to funguje velmi obdobně.

Receptivní pole je oblast, která je ovlivněna konkrétním CNN příznakem. Díky konvolucím mají příznaky v různých vrstvách různou velikost oblasti na vstupním obrazu. Jakmile jdeme ve vrstvách hlouběji, tak se příznak zvětšuje. Na obrázku 2.8 jde vidět několik vrstev, kde je spodní vrstva modré bravy o velikosti 5×5. Když aplikujeme konvoluci, tak to vyústí ve střední zelenou vrstvu o velikosti 3×3, kde jeden pixel znázorňuje oblast 3×3 ze vstupní vrstvy. Po aplikaci další konvoluce na střední vrstvu dostaneme oranžovou o velikosti 2×2. U této vrstvy jeden pixel odpovídá velikosti 7×7 na vstupním obrázku. Tyto vrstvy se také nazývají příznakové mapy[5]. Příznaky na stejné mapě příznaků mají stejné receptivní pole a hledají stejný vzor, ale na různých místech. To tvoří prostorovou neměnnost CNN. [24]

Receptivní pole je hlavní premisou architektury SSD, protože dokáže detekovat objekty různých velikostí i různých poměrů stran. Například by se mohla použít mřížka o velikosti 4×4 pro nalezení malých objektů, mřížka o velikosti 2×2 pro nalezení středních objektů a mřížka o velikosti 1×1 pro objekty, které by se mohly vyskytovat přes celý obraz. [24]

[5] Anglicky *feature maps*



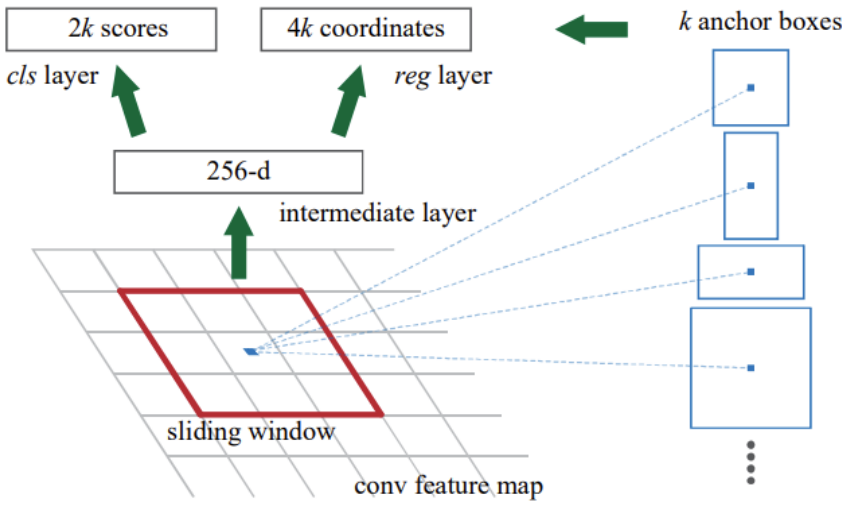
Obrázek 2.8: Vizualizace CNN příznakových map a receptivního pole [24]

2.5 Faster R-CNN

Další detektor, který jsem použil ke svým experimentům se jmenuje Faster R-CNN. Fungování detektoru jsem zkoušel v knihovně TensorFlow. Jak uvádějí autoři ve své vědecké publikaci s názvem *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [25], detektor se skládá ze dvou modulů. První modul je hluboká plně konvoluční síť, která navrhuje oblasti. Druhý modul je Fast R-CNN detektor, který používá navržené oblasti. Funguje to na principu, kde první modul *Region Proposal Network* (RPN) vyhledává oblasti a říká druhému modulu, kam se má podívat. Autoři také využili *anchor* boxy pro detekci objektů. Princip těchto boxů byl popsán v kapitole 2.3.1. Dokázali také zkrátit dobu výpočtu, díky sdílení konvolučních výpočtů napříč oběma používanými moduly. Faster R-CNN je rozšíření předchozího detektoru Fast R-CNN, který byl vytvořen jedním z autorů. Ten se také podílel na původní verzi R-CNN, ale tato veze byla velmi pomalá, tudíž se nedala aplikovat v reálném čase.

2.5.1 Region Proposal Network

První modul, který byl zmíněn v úvodu je RPN. Ten bere obrázky jakkoliv velkých velikostí jako vstup a jako výstup vygeneruje sadu obdélníkových návrhů objektů a každá obsahuje skóre předpověděné třídy. Tento proces vytváří díky plně konvoluční síti. Podle autorů byl největší cíl při budování tohoto detektoru, aby se výpočty sdílely mezi touto sítí a Fast R-CNN detekční síti objektů. Aby mohli vygenerovat návrh regionů, tak přesouvali *sliding window* přes výstup příznakových map z poslední konvoluční sítě sdílené s Fast R-CNN. U této metody je *sliding window* založené na vstupu obdélníkového okna o velikosti $n \times n$. Každé okno je mapováno na příznak z nižší dimenze. Tento příznak je poté předán do dvou sousedních plně propojených vrstev, z nichž se jedna nazývá regresní vrstva (reg) a druhá klasifikační vrstva (cls). [25] Na obrázku 2.9 lze vidět ukázku RPN se *sliding window* $n = 3$, které bylo použito autory v jejich vědecké práci.



Obrázek 2.9: *Region Proposal Network* (RPN) [25]

2.5.2 Skóre objektů

O předpovědění třídy se stará klasifikační vrstva (cls). Tato vrstva má výstupní vektor o dvou prvcích pro každý předpověděný region. Pokud první prvek je 1 a druhý 0, tak se jedná o pozadí. V opačném případě to znamená, že v regionu se nachází určitý objekt. Pro trénování RPN jsou poskytnuty *anchor* boxy s pozitivním nebo negativním skóre založené na metodě *Intesection over Union* (IoU). Vysvětlení fungování IoU bude vysvětleno v pozdější kapitole 4.2. Hodnoty se pohybují v rozmezí od 0 po 1, kde 1 znamená maximální překrytí 2 boxů a 0 znamená, že nemají spolu žádnou shodu. [26]

Pro určení, jestli detekci bude přiřazeno pozitivní nebo negativní skóre, rozhodují 4 kritéria. Především se to odvíjí od hodnoty IoU. První kritérium říká, že pokud v detekci je překrytí 2 *anchor* boxů větší než 0.7, tak je překrytí velmi vysoké a může se jednat o určitý objekt. Je označen jako pozitivní a může být použit pro trénování detektoru. Druhé kritérium říká, že pokud se v detekci nenachází IoU větší než 0.7, ale jsou tam detekce s IoU větší než 0.5, tak se tyto detekce také označí jako pozitivní. Třetí kritérium říká, že detekce s IoU pod 0.3, tak budou objekty označeny jako negativní a jedná se tak o falešnou detekci. Poslední kritérium říká, že pokud je skóre detekce mezi 0.3 a 0.5, tak detekce objektu není zařazena ani mezi negativní ani pozitivní. [26]

Kapitola 3

Datasety

3.1 EuroCity Persons dataset

Jeden z datasetů, který jsem použil ve své práci se jmenuje EuroCity Persons dataset [27], zkráceně ECP dataset. Byl publikován v roce 2019 a jejíž autoři jsou *Braun, Markus and Krebs, Sebastian and Flohr, Fabian B. and Gavrila, Dariu M.* ECP dataset se skládá z více než 238 200 záznamů osob, které byly ručně anotovány na více než 47 300 fotkách. Snímky byly pořizovány napříč celou Evropou ve 12 zemí a v 31 městech, především ve velkých evropských metropolích jako je například Berlín, Praha, Amsterdam, Barcelona. Objekty, které byly anotovány v tomto datasetu, tak jsou chodci, řidiči a jejich vozidla. Pro tuto práci jsem využil pouze chodce. Dále autoři uvádí, že pokud osoba nebyla celá viditelná, tak byl odhadnut celý rozsah osoby. Autoři pořizovali snímky jak ve dne, tak i v noci, aby mohli analyzovat na větším spektru případů. Do své práce jsem použil pouze snímky ze dne a vybral jsem si náhodné snímky z různých měst, abych na nich zkusil natrénovat některé modely detektorů. Dataset jsem použil i pro srovnání modelů mezi sebou.

3.2 Cityscapes dataset

Další dataset, který jsem použil pro trénování detektorů a následné ověřování funkčnosti a přesnosti je dataset s názvem Cityscapes dataset [28]. Ten byl publikován v roce 2016 a autoři jsou *M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, a B. Schiele.* Skládá se z velmi rozmanité sady video snímků z ulic, z více než 50 různých měst. Je zde anotováno přes 25 000 snímků. V datasetu se nachází 30 tříd objektů, ale pro účely této práce budu využívat pouze třídu s chodci. Záběry jsou pořizovány z různých ročních období, za dobrých povětrnostních podmínek a obsahují denní záběry. Z tohoto datasetu jsem využíval snímky jak k trénování, tak i k srovnávání jednotlivých modelů.

Obrázek 3.1: Ukázka snímku z ECP datasetu, město Praha [27]

Obrázek 3.2: Ukázka snímku z Cityscapes datasetu, město Hamburg [28]

3.3 Person Re-identification in the Wild dataset

Třetí dataset se jmenuje Person Re-identification in the Wild dataset, zkráceně PRW dataset. Je od autorů *Zheng, Liang and Zhang, Hengheng and Sun, Shaoyan and Chandraker, Manmohan and Tian, Qi.* Je z práce s názvem *Person Re-identification in the Wild* [29], která vyšla v roce 2016. Z tohoto datasetu jsem použil pár snímků pro testování funkčnosti metod a pro trénování modelů. Jedná se o snímky z oblasti před supermarketem na univerzitě Tsinghua v Číně. Obsahuje přes 11 000 snímků. Dataset se zaměřuje především na detekci lidí. Snímky jsou pořizovány ze šesti různých kamer, takže se zde nachází snímky z jedné lokace ze šesti různých úhlů. Jedná se o statické prostředí, takže s kamerou, která zaznamenává snímky, není pohybováno, tudíž není velká diverzita okolního prostředí.

Obrázek 3.3: Ukázka snímku z datasetu PRW [29]

Kapitola 4

Experimentální část

Tato kapitola se bude experimentálně zabývat funkčností jednotlivých metod na zmíněných datových sadách, konkrétně jejich přesností a rychlostí. Dále zde bude zmíněno, jakým principem jsem hodnotil jejich úspěšnost a následně zde budou výsledky jednotlivých metod a porovnání mezi sebou.

Jak jsem již zmiňoval v této práci, tak ověřování detektorů jsem prováděl na datových sadách ECP datasetu a Cityscapes datasetu. Pro tyto účely jsem si vybral celkem 300 snímků, které jsem si náhodně vybíral z obou datasetů a různých měst, abych zkoumal větší diverzitu a nejednalo se pouze o jednu lokalitu. Pro trénování jsem použil již zmíněné oba datasety a k tomu navíc jsem zkusil natrénovat pár modelů i na třetím datasetu PRW. Trénování jsem prováděl u metody YOLO, u zbylých metod jsem použil již natrénované modely od autorů. Zkoumání, trénování a zjišťování rychlostí jsem prováděl na stroji, který obsahuje tento hardware: procesor Intel Core i5-7500, grafickou kartu Nvidia GeForce GTX 1060 6GB a paměti RAM 16 GB. Pro ověření přesností detektorů jsem použil metodu *Mean Average Precision* (mAP), která se skládá z několika částí. Hlavní části této metody tvoří matici záměn[1] a *Intersection over Union* (IoU). V následující části této kapitoly bude vysvětleno fungování těchto částí.

4.1 Matice záměn

Matice záměn je takový typ matice, kde se zaznamenávají čtyři kategorie. První z nich je *true positive* (TP), to znamená, když detektor predikuje pozici objektu na místě, kde se skutečně nachází. Druhá je *true negative* (TN). V této práci se s touto kategorií nepracuje, protože se nevyskytuje u metrik, které byly použity pro porovnávání modelů. Zbývající dvě kategorie jsou *false positive* (FP) a *false negative* (FN). První znamená, že detektor na nějakém místě detekoval objekt, ale na daném místě se nenachází hledaný objekt. Druhá znamená opak, že se právě na nějakém místě nachází náš hledaný objekt, ale detektor ho neoznačil. Díky této matici můžeme vypočítat *Precision* a *Recall*, což jsou

[1] Anglicky *Confusion Matrix*

klasifikační metriky použité v této práci. *Precision* udává kolik skutečně pozitivních detekcí detektor našel ze všech pozitivních detekcí. Vypočítá se jako:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

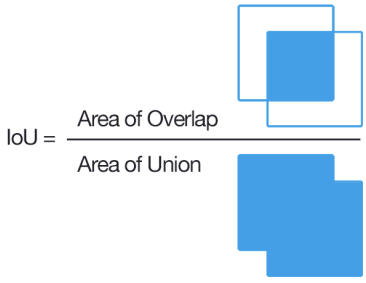
Recall uvádí kolik skutečně pozitivních detekcí detektor našel ze všech detekcí. Vypočítá se jako:

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

Aby se dalo určit, jestli se jedná o skutečně pozitivní nebo falešně pozitivní detekci, tak se musí vypočítat míra překrytí a nastavit nějaký práh[2]. O to se stará další část metody s názvem *Intersection over Union* (IoU).

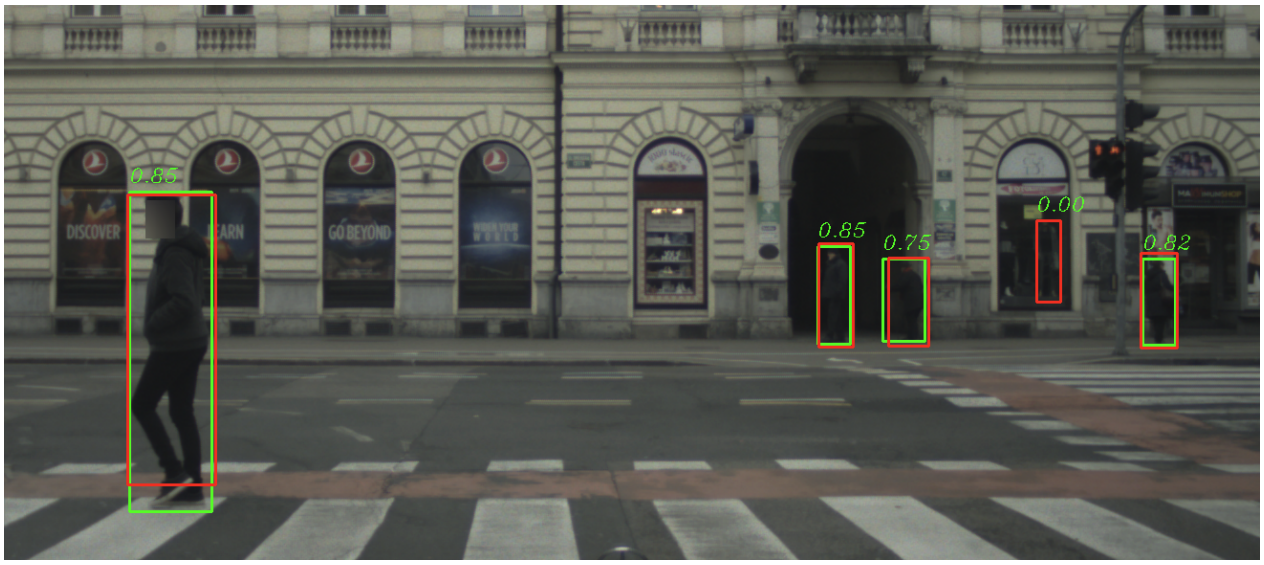
4.2 Intersection over Union

Intersection over Union, dále je IoU, je vyhodnocovací metrika, která slouží k určení přesnosti mezi dvěma ohraničujícími boxy. Jeden ohraničující box je určen jakýmkoliv detektorem, který označil nějakou část obrazu s domněnkou, že se jedná o hledaný objekt. Druhý ohraničující box označuje přesnou detekci hledaného objektu. IoU se poté spočítá jako společná plocha překrytí těchto dvou boxů a vydělí se s celkovou hodnotou ploch obou boxů. Jednoduchá ukázka vzorce jde vidět na obrázku 4.1. Pro trénování se za dobrou hodnotu IoU bere 0.5 a vyšší. V této práci při hodnocení přesnosti jednotlivých detektorů beru jako dobrou hodnotu od 0.3. Protože jestli detektor označil nějakou část obrazu jako výskyt chodce, tak je určitá pravděpodobnost, že se tam opravdu nachází. Pro trénování jsem použil hodnotu 0.5 a vyšší.



Obrázek 4.1: Ukázka výpočtu *Intersection over Union* [30]

[2]Anglicky *threshold*



Obrázek 4.2: Ukázka predikce na snímku z ECP datasetu s ohraničujícími boxy YOLOv7 detektorem (červená barva) a skutečnými ohraničujícími boxy chodce (zelená barva). Hodnoty nad detekcemi jsou hodnoty IoU [27]

4.3 Vytvoření vlastních modelů YOLOv7

U metody YOLO jsem si natrénoval vlastní modely, u kterých jsem poté zkoumal jejich rychlost a přesnost detekcí. Snažil jsem se zjistit, jestli modely natrénované na datových sadách, které jsou použity i pro testy, budou dosahovat lepších výsledků než modely natrénované na obecných sadách. Pro trénování jsem si vybral všechny tři datasety, které byly zmíněné v předchozí kapitole 3.

Pro anotaci snímků jsem využil program *LabelImg* [31], ve kterém jsem označil pozitivní detekce chodců. Výsledek z programu byl textový soubor ve formátu YOLO. Anotoval jsem snímky jak pro trénink, tak pro testování ze všech tří datasetů.

Samotné trénování probíhalo na grafické kartě Nvidia GeForce GTX 1060 6GB. Byly vyzkoušeny různé modely s různými parametry trénování. Měnily se i vstupní datové sady u trénování a u modelů bylo zkoumáno, jestli to bude mít na ně nějaký vliv. Jako výsledek trénování vznikly vlastní modely se svými váhami a některé z nich byly použity pro následné porovnání s modely jiných metod.

Jako první byly vyzkoušené tři modely, kde byly nastavené všechny parametry stejně a měnila se pouze trénovací množina. První z modelů s názvem YOLOv7-Berlin100 byl trénován pouze na datasetu ECP. K tomu bylo použito 100 náhodných fotek z města Berlín a validace byla prováděna na městě Hamburk. Druhý model s názvem YOLOv7-Tübingen100 byl natrénován na dalším datasetu s názvem Cityscapes. Tady také bylo vybráno 100 náhodných snímků z města Tübingen. K validaci byly snímky z města Jena. Třetí dataset s názvem YOLOv7-Mix220 byl natrénován na kombinaci 220 snímků z obou datasetů na třech různých městech jako je Norimberk, Florencie a Curych. U prvních dvou modelů trénování zabralo každému 70 minut a třetímu modelu 145 mi-

nut. Parametry pro trénování jsou vidět na obrázku 4.3. Jako první se nastavovala možnost *batch size*, u které jsem zvolil velikost 8. Při větší velikosti by bylo potřeba použít lepší hardware. *Batch size* udává počet trénovacích vstupů v jednom průběhu trénování. Jako další se dal nastavit počet epoch, který jsem zvolil na 100. Epochy udávají celkový počet průchodů celým datasetem. Dala se také nastavit velikost snímků, která je základně nastavená na 640×640. Toto nastavení bylo ponecháno, aby všechny YOLO modely měly stejné podmínky pro naučení. Také se daly použít různé předtrénované modely s váhami pro trénovaní. Byl použit předtrénovaný model s názvem *YOLOv7.pt*, který je k dispozici od autorů metody YOLOv7. Autoři metody YOLOv7 také uvedli, že se dají použít různě poupravené architektury pro různé modely YOLOv7. Při trénování byla použita stejná architektura jako pro základní model YOLOv7, kde byl pozměněn počet tříd na jednu. Jako poslední se daly nastavit hyperparametry. U těchto tří modelů bylo nastavení hyperparametrů ponecháno na předdefinovaných hodnotách základního modelu YOLOv7. Ty byly pozměněné až v dalších modelech.

```
python train.py --workers 1 --device 0 --batch-size 8 --epochs 100 --data data/custom_data.yaml
--img 640 640 --cfg cfg/training/yolov7-custom.yaml --weights 'yolov7.pt' --hyp data/hyp.scratch
.custom.yaml --name yolov7-custom
```

Obrázek 4.3: Ukázka tréninkového skriptu

Pro další pokusy se trénovalo ze všech tří datasetů. Jako první bylo vyzkoušeno nastavení *anchor* boxů. V základním nastavení je nastaveno, že velikosti boxů se budou určovat automaticky. Nejčastěji si tréninkový proces vybírá takové velikosti, jaké se vyskytovaly nejčastěji v trénovacích sadách. Druhý model měl již přednastavené velikosti boxů. Navíc u obou modelů k tomu byly pozměněny hyperparametry, kde byl snížen argument u konvoluční vrstvy z 512 na 128. To mělo za následek snížení počtu parametrů z celkového počtu 37 milionů na 30 milionů. Zrychlilo to trénovací proces a také výsledný model pro detekci zabral méně místa na disku. Na neštěstí se to projevilo i ve výsledcích. U varianty bez automatických *anchor* boxů to dopadlo trochu lépe, kde průměrná hodnota IoU byla kolem 0,23. Model se snažil udělat i více detekcí, ale to také vedlo k většímu počtu chyb. U druhého modelu byla průměrná hodnota IoU 0,22.

Další pokusy spočívaly k navýšení počtu epoch oproti předchozím modelům. Hyperparametry byly navráceny na základní hodnotu. Bylo vyzkoušeno trénování s 512 a 768 epochami. U modelu s menším počtem epoch proces trénování trval kolem 360 minut a u s větším počtem 540 minut. Bylo zjištěno, že při enormnímu navýšení počtu epoch se model přetrénoval, což se ukázalo v testovací fázi. Model s větším počtem epoch dosahoval horších výsledků než model s menším počtem. Při trénování je důležité najít rovnováhu v počtu epoch, aby model nebyl málo natrénován nebo právě přetrénován. Přetrénování se projevovalo tak, že model označoval pouze ty detekce, kde si byl skoro naprosto jistý. To mělo za následek, že udělal méně detekcí, a to i těch pozitivních, než druhý model.

V neposlední řadě se dalo využít trénovaní vah od nuly[3]. Nepoužily se váhy od autorů, jak v

[3]Anglicky *from scratch*

předchozích modelech, takže tréninkový proces nemohl použít již částečně předtrénovaný model, ze kterého by vycházel. V tomto případě také hrál roli počet epoch. Nejdříve bylo použito 50 epoch na trénování. To se ukázalo jako nedostatečné, a proto musel být počet navýšen. Další pokus byl se 100 epochami. Ten dopadl už o něco lépe a našel některé správné detekce. Průměrná hodnota IoU byla 0,12. Následně bylo zkoušeno 150 a 300 epoch. Bylo vypozorováno, že s větším počtem epoch se zvyšuje přesnost a modely dosahují lepších výsledků. Ale jak bylo zmíněno v předchozí části, měla by se najít rovnováha mezi dostatečným trénováním a přetrénováním.

Pro testování byly vybrány tři modely YOLOv7-Berlin100, YOLOv7-Tübingen100 a YOLOv7-Mix220. Budou srovnány v následující kapitole 4.4 s modely jiných metod.

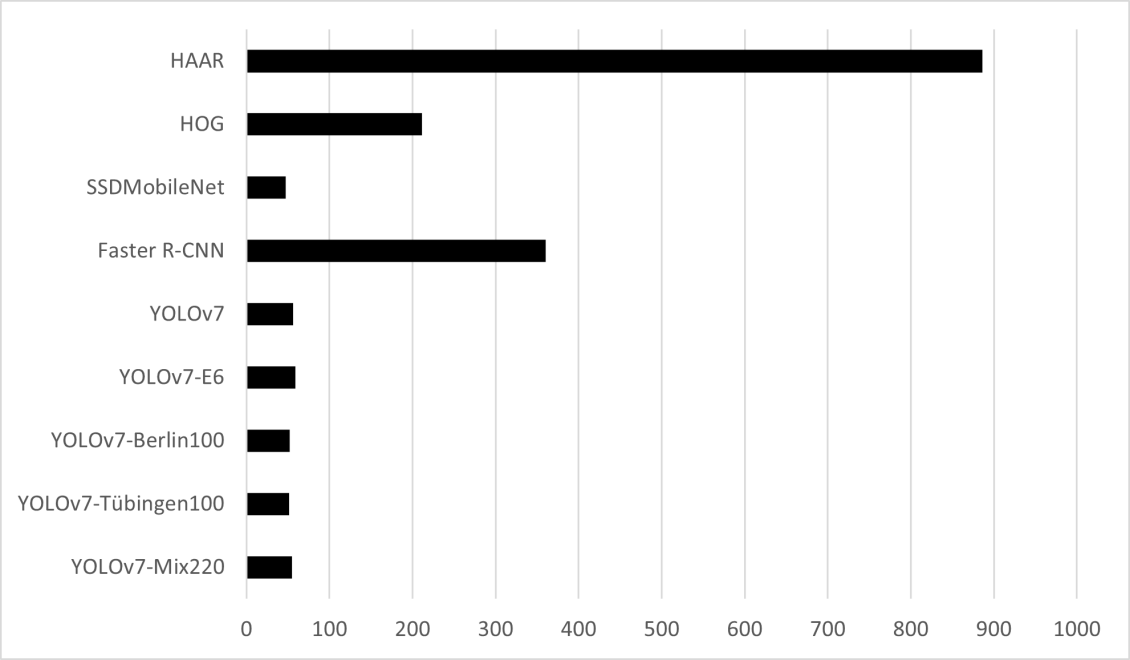
4.4 Srovnání modelů

Jak již v práci bylo zmíněno, srovnání mezi jednotlivými metodami jsem prováděl na celkem 300 snímcích z různých měst z různých států. Jednalo se o města Bratislava, Lublaň, Stuttgart a Turín. Velikosti vstupních obrazů byly 1920×1024 z datasetu ECP a 2048×1024 z Cityscapes datasetu. Všechny metody měly za úkol najít a správně lokalizovat všechny chodce z obrazu. U těchto metod jsem zkoumal celkovou rychlost, průměrnou přesnost IoU, *Precision* a *Recall*. Toto srovnání jsem dělal mezi šesti různými modely, kde jsem použil již natrénované modely. K tomu jsem si navíc natrénoval svoje vlastní modely vycházející z architektury YOLOv7 a porovnal jsem je mezi sebou. Detailnější informace ohledně trénovaní vlastních modelů jsou v předchozí kapitole 4.3.

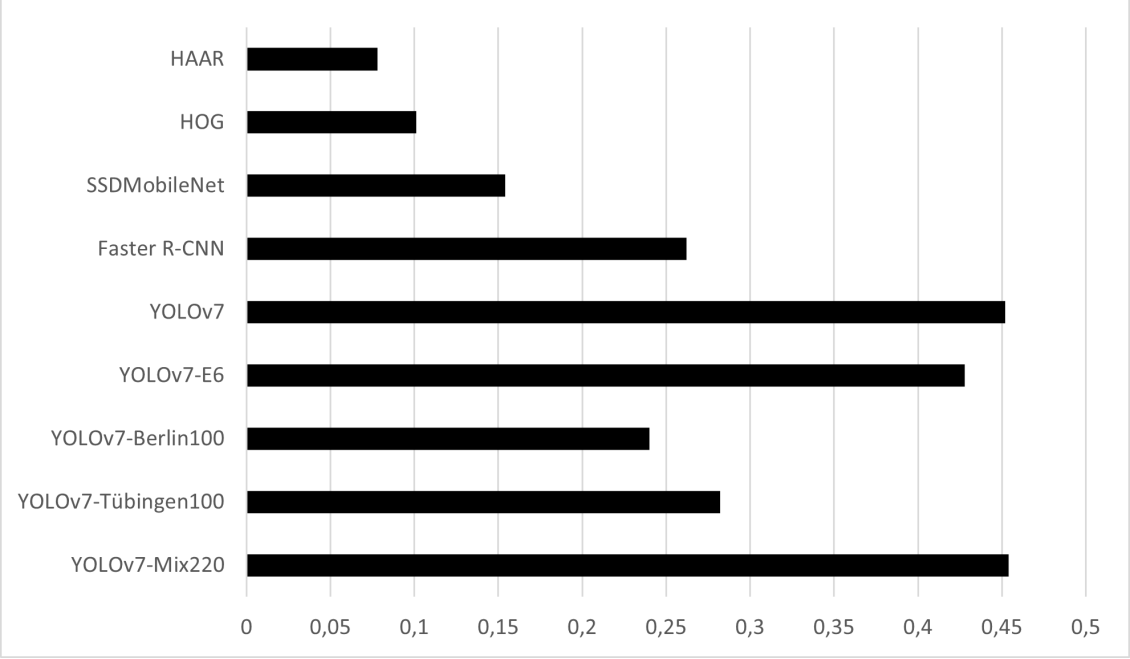
Na prvním grafu 4.4 jde vidět celkový čas jednotlivých detektorů ve vteřinách, kolik jim kompletní detekce všech 300 snímků zabrala. Jde vidět, že detektoru s příznaky typu Haar to trvalo nejdéle a to 886 vteřin. Mohlo to být způsobeno podrobnějším prohledáváním obrazu, protože jsem zvolil takové parametry, aby se detektor snažil najít raději více detekcí i za cenu pár špatných navíc. Při méně podrobném vyhledávání objektů detektor našel jen pár detekcí a ty by stejně nezaručovaly, že by se skutečně mohlo jednat o chodce. Poté následoval v pořadí s největším časem detektor Faster R-CNN a HOG. U zbylých modelů byl čas velmi podobný okolo 55 vteřin, což v průměru vychází na 5.45 snímků za vteřinu, takže by tyto detektory šly použít i pro živou detekci na videu.

Na dalším grafu 4.5 lze vidět průměrná přesnost IoU. To znamená, jak dobře se překrývaly ohraničující boxy mezi predikcí detektoru a skutečným boxem kolem chodce. Z grafu je patrné, že nejlépe ohraničoval chodce YOLO detektor. Dokázal tak nejlépe z vybraných metod určovat skutečnou velikost chodců. Také lze pozorovat, že detektor naučený na podobném prostředí, jako je prostředí testovací, dokáže být lepší než detektor, který byl naučený obecně na *Common Objects in Context* (COCO) datasetu [32]. Jedná se sice o nepatrný rozdíl, ale detektor YOLOv7-Mix220 byl natrénován pouze na 220 snímcích, zatímco originální detektor byl natrénován na daleko rozsáhlejší sadě. Naopak nejhůře na tom byly detektory s příznaky typu Haar a HOG.

Na tabulce 4.1 lze vidět matice záměn jednotlivých modelů. Model s příznaky typu Haar a HOG se snažil dělat, co nejvíce detekcí i za cenu špatných. Proto mají velká čísla ve FP. Naopak YOLO



Obrázek 4.4: Celkový čas pro detekci na 300 snímcích ve vteřinách



Obrázek 4.5: Průměrná přesnost IoU jednotlivých modelů

natrénované na COCO datasetu [32] má FP velmi nízko, takže pokud detekovalo objekt, tak si bylo skoro naprosto jisté, že se jedná o chodce. Naprosto nejvíce chodců našlo YOLO natrénované na mixu z obou datasetů, což můžeme pozorovat na vysoké hodnotě TP.

Tabulka 4.1: Matice záměn jednotlivých modelů

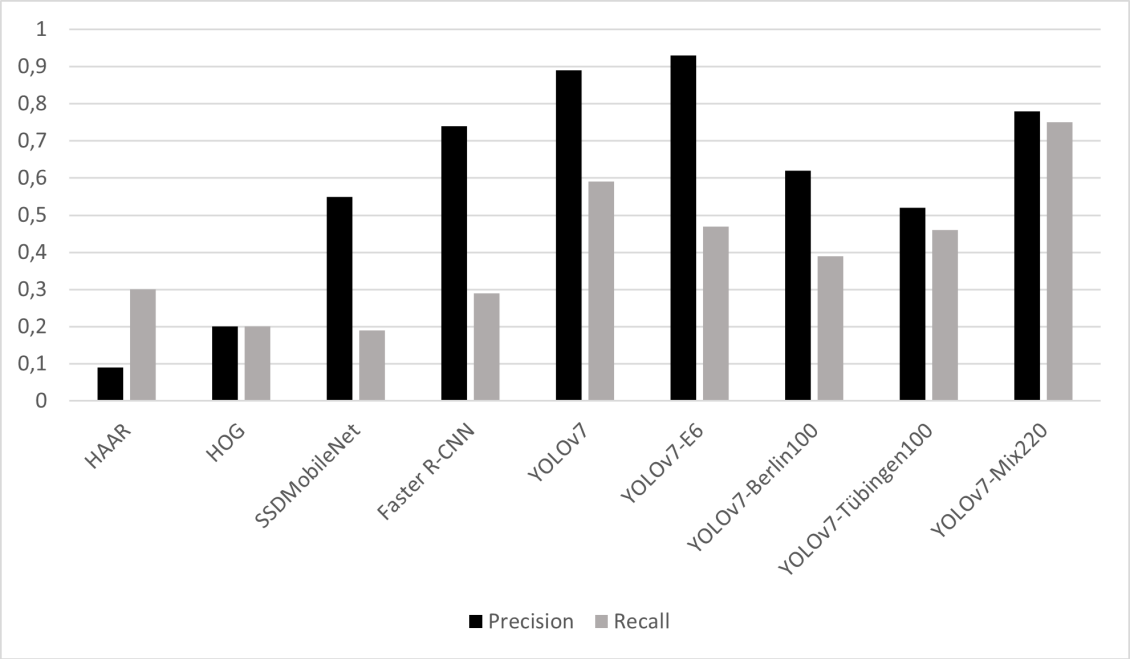
Názvy modelů	TP	FP	FN
Haar	145	1425	345
HOG	153	618	601
SSDMobileNet	190	153	822
Faster R-CNN	309	109	744
YOLOv7	632	77	445
YOLOv7-E6	518	39	589
YOLOv7-Berlin100	381	236	598
YOLOv7-Tübingen100	435	395	510
YOLOv7-Mix220	762	219	252

Jako další srovnání je přehled klasifikačních metrik *Precision* a *Recall* na grafu 4.6. Jak již bylo zmíněno, tak *Precision* se zaměřuje na přesnost v určených detekcích detektorem, v kolika případech se jednalo o chodce. Na rozdíl od něj se *Recall* zaměřuje na to, kolik našel skutečných detekcí ze všech svých pokusů o označení chodce a ke všem těm pokusům se ještě připočítává všechen počet chodců, který detektor neoznačil. Z pohledu detektoru, který se zajímá o chodce, a především v dopravě, tak je důležitější hodnota *Recall* než *Precison*. Protože, kdyby byl takový detektor jako jeden z jízdních asistentů, raději by měl označit větší počet detekcí i za cenu špatných, než aby detekoval jen ty objekty, kde si je jistý. Z grafu lze vyčíst, že nejlépe na tom dopadly modely YOLO, které mají větší *Recall* než ostatní modely. Také lze pozorovat, že model Faster R-CNN má vysokou přesnost, ale v poměru s *Recall* to ve výsledku není, co by se očekávalo od takového detektoru. Nejmenší hodnoty mají detektory s příznaky typu Haar a HOG.

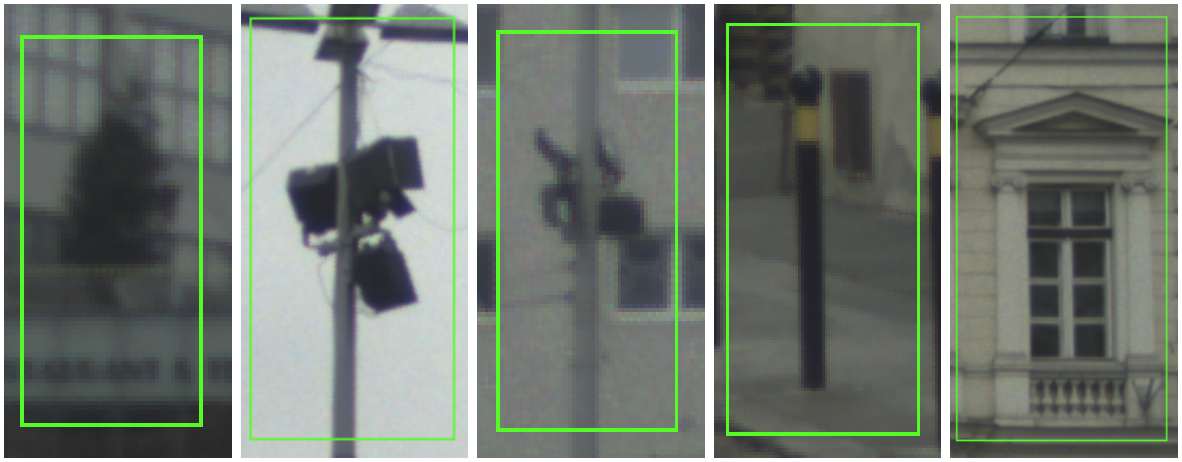
4.5 Nejčastější detekční chyby

Jak jsem již zmiňoval v úvodu, tak správná detekce chodců není úplně jednoduchá. Občas se může stát, že malé objekty mohou být zaměňovány za chodce a naopak. Na tabulce 4.1, která byla zmíněná v předchozí sekci, jde vidět, že nejčastější chybné detekce dělaly metody HOG a s příznaky typu Haar. Na následujícím snímku 4.7 jdou vidět nejčastější falešné detekce těchto dvou metod.

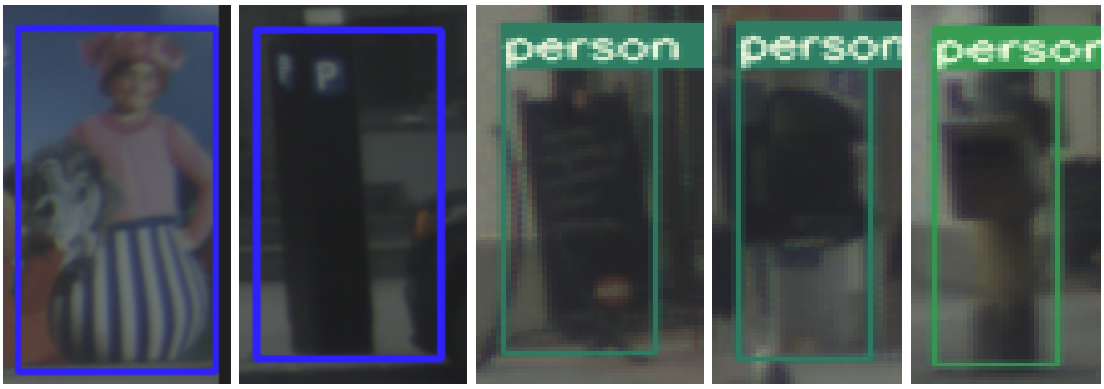
Zbylé metody byly přesnější, ale také nebyly zcela bez falešných detekcí. Na obrázku 4.8 jdou vidět nejčastější chybné detekce metod MobileNetSSD, Faster R-CNN a YOLO.



Obrázek 4.6: *Precision* a *Recall* jednotlivých modelů



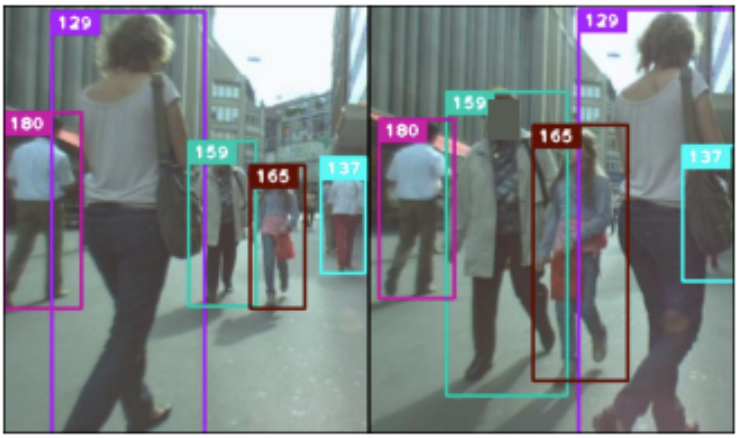
Obrázek 4.7: Ukázka typických detekčních chyb metod HOG a s příznaky typu Haar [27]



Obrázek 4.8: Ukázka typických detekčních chyb metod MobileNetSSD, Faster R-CNN a YOLO [27]

4.6 Vylepšení pro videosekvenci

V této práci se zaměřuji na snímky, které jsou náhodně vystřižené z videosekvencí. Nemají na sebe žádnou návaznost. Kdybychom chtěli použít detektor na živé video, tak by bylo dobré ho zkombinovat se sledovacím algoritmem. Sledovací algoritmus by měl za úkol přidělovat jedinečný identifikátor každému objektu, který by detekoval a v rámci sledování snímků ve videu by si ho zachovával. Díky tomuhle přidělení a sledování by mohl získat prostorovou i časovou vlastnost a mohl by tak předpovídat polohy sledovaných objektů. Jako sledovací algoritmus by mohl být použitý Deep SORT [33] a jako detekční modul by mohl být použit zmíněný detektor YOLOv7. [34] Ukázka kombinace jde vidět na obrázku 4.9.



Obrázek 4.9: Ukázka využití detekčního algoritmu se sledovacím algoritmem [33]

Algoritmus Deep SORT by pomohl řešit celou řadu problémů, které samotný detektor neřeší. Například bychom sledovali nějaký objekt v rámci videosekvence a částečně by se schoval za nějaký předmět. Detektor by objekt již nedetekoval jako hledaný objekt, zatímco sledovací algoritmus by stále sledoval objekt a mohl by ho tak predikovat a sledovat. Mezi další výhody při uplatnění tohoto

algoritmu může být již zmíněná jedinečná identifikace. Detektor předpovídá pouze umístění objektů. Při klasickém výstupu bychom dostali souřadnice umístění a třídu objektu, ale nevěděli bychom, o jaký konkrétní objekt se jedná. Například jestli se jedná o chodce A nebo chodce B. Uplatnění této kombinace by se dalo použít k sledování provozu, ve sportu nebo by mohlo být monitorováno chování lidí. [34]

Kapitola 5

Závěr

V této bakalářské práci byly popsány jednotlivé metody pro detekci chodců v dopravě a následně ověřeny jejich přesnosti a rychlosti na testovací sadě. Bylo použito pět různých architektur detektorů, ze kterých bylo otestováno devět modelů s různými trénovacími množinami.

Z výsledků srovnání všech modelů mezi sebou jsou odvozeny následující závěry. Je patrné, že jako nejlepší se jeví architektura YOLO. Ta měla velmi dobrý průměrný čas na detekci snímku, také velmi dobrou průměrnou přesnost IoU a také vysoké hodnoty *Precision* a *Recall*. Tento detektor by se zcela jistě dal použít v reálném životě jako třeba jeden z jízdních asistentů. Mělo by se podotknout, že vliv na určité detektory může mít, jak kvalita obrazů, tak množství trénovací sady. Myslím si, že je velmi důležité, aby se v trénovacích sadách objevila velká rozmanitost pozitivních detekcí, aby detektory mohly být co nejlépe natrénovány.

Také nesmíme opomenout, že u tohoto typu detektorů, který se zaměřuje na detekci lidí v situacích, kdy by jejich neodhalení mohlo vést k nějaké kritické situaci, je důležitější se zaměřit spíše na *Recall* než na *Precision*. Je žádoucí, aby se detektor snažil odhalit co možná nejvíce detekcí i za cenu pár špatných detekcí. Kdybychom se soustředili pouze na přesnost, tak by to detekovalo pouze jen ty případy, když by si byl detektor skoro jistý. Je mnohem lepší, aby auto občas nečekaně zastavilo a mohlo se stát, že by do něj někdo naboural, než aby například autonomní vozidlo nedetekovalo lidskou osobu a mohlo ji srazit. Samozřejmě je důležité najít rovnováhu mezi těmito metrikami, aby auto nezastavovalo co pár metrů.

Domnívám se, že toto téma patří mezi důležité a v budoucnu se jistě můžeme těšit na zdokonalení těchto detektorů, objevení zcela nových metod pro detekci chodců v dopravě a obecně pro detekci jakýchkoliv objektů.

Literatura

1. JIRKOVSKÝ Jaroslav, Humusoft s. r. o. Počítačové vidění: od hledání vzoru po Deep Learning. *Automa*. 2018, roč. 24, č. 4, s. 42–43. ISSN 1210-9592.
2. ZHANG, Liliang; LIN, Liang; LIANG, Xiaodan; HE, Kaiming. Is Faster R-CNN Doing Well for Pedestrian Detection? *CoRR*. 2016, roč. abs/1607.07032. Dostupné z arXiv: 1607.07032.
3. *Digitální zpracování obrazu* [online]. 2021-08-06 [cit. 2023-04-07]. Dostupné z: https://cs.wikipedia.org/wiki/Digit%C3%A1ln%C3%AD_zpracov%C3%A1n%C3%AD_obrazu.
4. *Strojové učení* [online]. 2023-03-05 [cit. 2023-04-07]. Dostupné z: https://cs.wikipedia.org/wiki/Strojov%C3%A9_u%C4%8Den%C3%AD.
5. VIOLA, P.; JONES, M. 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*. 2001, sv. 1, s. I–I. Dostupné z DOI: 10.1109/CVPR.2001.990517.
6. BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000.
7. VIOLA; JONES; SNOW. Detecting pedestrians using patterns of motion and appearance. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 734–741 vol.2. Dostupné z DOI: 10.1109/ICCV.2003.1238422.
8. TYAGI, Mrinal. *Viola Jones Algorithm and Haar Cascade Classifier* [online]. 2021-07-13 [cit. 2023-04-07]. Dostupné z: https://towardsdatascience.com/viola-jones-algorithm-and-haar-cascade-classifier-ee3bfb19f7d8.
9. *Cascade Classifier* [online]. 2021-12-25 [cit. 2023-04-07]. Dostupné z: https://docs.opencv.org/4.5.5/db/d28/tutorial_cascade_classifier.html.
10. BHADANI, Goutam. *Haar Cascade Classifier vs Histogram of Oriented Gradients(HOG)* [online]. 2020-05-13 [cit. 2023-04-07]. Dostupné z: https://medium.com/@goutam0157/haar-cascade-classifier-vs-histogram-of-oriented-gradients-hog-6f4373ca239b.
11. DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, sv. 1, 886–893 vol. 1. Dostupné z DOI: 10.1109/CVPR.2005.177.

12. KING, Davis E. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*. 2009, roč. 10, s. 1755–1758.

13. *Histogram of oriented gradients* [online]. 2023-01-28 [cit. 2023-04-07]. Dostupné z: `https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients`.

14. REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, roč. abs/1506.02640. Dostupné z arXiv: `1506.02640`.

15. LAN, Wenbo; DANG, Jianwu; WANG, Yangping; WANG, Song. Pedestrian Detection Based on YOLO Network Model. In: *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2018, s. 1547–1551. Dostupné z DOI: `10.1109/ICMA.2018.8484698`.

16. *Anchor Boxes for Object Detection* [online]. 2023 [cit. 2023-04-07]. Dostupné z: `https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html`.

17. WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. Dostupné z arXiv: `2207.02696 [cs.CV]`.

18. SOLAWETZ, Jacob. *What is YOLOv7? A Complete Guide.* [online]. 2022-07-17 [cit. 2023-04-07]. Dostupné z: `https://blog.roboflow.com/yolov7-breakdown/`.

19. MUNAWAR, Muhammad Rizwan. *YOLOv7 Architecture Explanation* [online] [cit. 2023-04-07]. Dostupné z: `https://www.cameralyze.co/blog/yolov7-architecture-explanation`.

20. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott E.; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, roč. abs/1512.02325. Dostupné z arXiv: `1512.02325`.

21. SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*. 2014-09.

22. HOWARD, Andrew G.; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; ADAM, Hartwig. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. Dostupné z arXiv: `1704.04861 [cs.CV]`.

23. CHOUDHARY, Anurag Singh. *Object Detection Using YOLO And Mobilenet SSD* [online]. 2022-08-22 [cit. 2023-04-07]. Dostupné z: `https://www.analyticsvidhya.com/blog/2022/09/object-detection-using-yolo-and-mobilenet-ssd/`.

24. *How single-shot detector (SSD) works?* [online] [cit. 2023-04-07]. Dostupné z: `https://developers.arcgis.com/python/guide/how-ssd-works/`.

25. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. Dostupné z arXiv: 1506.01497 [cs.CV].

26. GAD, Ahmed Fawzy. *Faster R-CNN Explained for Object Detection Tasks* [online]. 2021 [cit. 2023-04-07]. Dostupné z: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/.

27. BRAUN, Markus; KREBS, Sebastian; FLOHR, Fabian B.; GAVRILA, Dariu M. EuroCity Persons: A Novel Benchmark for Person Detection in Traffic Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2019, s. 1–1. ISSN 0162-8828. Dostupné z DOI: 10.1109/TPAMI.2019.2897684.

28. CORDTS, Marius; OMRAN, Mohamed; RAMOS, Sebastian; REHFELD, Timo; ENZWEILER, Markus; BENENSON, Rodrigo; FRANKE, Uwe; ROTH, Stefan; SCHIELE, Bernt. The Cityscapes Dataset for Semantic Urban Scene Understanding. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

29. ZHENG, Liang; ZHANG, Hengheng; SUN, Shaoyan; CHANDRAKER, Manmohan; TIAN, Qi. Person Re-identification in the Wild. *CoRR*. 2016, roč. abs/1604.02531. Dostupné z arXiv: 1604.02531.

30. ROSEBROCK, Adrian. *Intersection over Union (IoU) for object detection* [online]. 2022-04-30 [cit. 2023-04-14]. Dostupné z: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

31. TZUTALIN. *LabelImg* [Free Software: MIT License]. 2015. Dostupné také z: https://github.com/tzutalin/labelImg.

32. LIN, Tsung-Yi et al. Microsoft COCO: Common Objects in Context. *CoRR*. 2014, roč. abs/1405.0312. Dostupné z arXiv: 1405.0312.

33. WOJKE, Nicolai; BEWLEY, Alex; PAULUS, Dietrich. Simple Online and Realtime Tracking with a Deep Association Metric. *CoRR*. 2017, roč. abs/1703.07402. Dostupné z arXiv: 1703.07402.

34. SANYAM. *Understanding Multiple Object Tracking using DeepSORT* [online]. 2022-06-21 [cit. 2023-04-17]. Dostupné z: https://learnopencv.com/understanding-multiple-object-tracking-using-deepsort/.

Příloha A

Obsah přílohy

Příloha je ve formátu .zip a obsahuje následující složky a soubory:

- **Fotky** - Složka se 6 ukázkami snímků pro detekce.
- **Predictfasterrcnn** - Složka obsahuje detekce provedené metodou Faster R-CNN v textové i grafické podobě. (zelené boxy označují reálné boxy, červené boxy označují predikce chodců metodou)
- **Predictfotek** - Složka obsahuje detekce chodců z fotek v textové i grafické podobě.
- **Predicthaar** - Složka obsahuje detekce provedené metodou s příznaky typu Haar v textové i grafické podobě. (zelené boxy označují reálné boxy, červené boxy označují predikce chodců metodou)
- **Predicthog** - Složka obsahuje detekce provedené metodou HOG v textové i grafické podobě. (zelené boxy označují reálné boxy, červené boxy označují predikce chodců metodou)
- **Predictmobilenetssd** - Složka obsahuje detekce provedené metodou MobileNetSSD v textové i grafické podobě. (zelené boxy označují reálné boxy, červené boxy označují predikce chodců metodou)
- **XML** - Složka obsahuje XML soubor pro metodu s příznaky typu Haar.
- **Yolov7** - Složka obsahuje ukázky skriptů tréninků a detekcí metodou YOLOv7, soubor requirements.txt s potřebnými knihovnami pro spuštění skriptů, modely s příponou .pt s natrénovanými váhami, složky s predikcemi 4 modelů v textové i grafické podobě (zelené boxy označují reálné boxy, červené boxy označují predikce chodců modely), soubor github.txt, který obsahuje odkaz na github od autorů metody YOLOv7, upravené soubory s příponou .yaml, soubor intersectionoverunion.py pro porovnáváni přesnosti detekcí vytvořených modely, složku s ukázkami 6 snímků pro detekci, složku s detekcemi chodců z fotek v textové i grafické podobě a

soubor demo.mp4 s video ukázkou fungování YOLOv7 metody s následnou ukázkou úspěšnosti detekce.

- **datasety.txt** - Obsahuje odkazy na stažení datasetů.
- **intersectionoverunion.py** - Soubor pro porovnáváni přesnosti detekcí vytvořených modely.
- **jsontotxt.py** - Soubor pro převod anotace z ECP datasetu na formát YOLO.
- **knihovny.txt** - Knihovny potřebné pro spuštění souborů.
- **libopencv.py** - Soubor obsahuje kód pro vytvoření detekcí metodami s příznaky typu Haar a HOG.
- **libtensorflow.py** - Soubor obsahuje kód pro vytvoření detekcí metodami Faster R-CNN a MobileNetSSD.
- **vysledkymodelu.txt** - Soubor obsahuje výsledky jednotlivých modelů zmíněných v práci na testovací sadě 300 snímků.